isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

DEFINITIVO

Desenvolvimento de um API RESTful para suporte a aplicações móveis de caracterização de condução

Autor

Jorge Manuel Marques Pereira Lopes Pinheiro

Orientador

Anabela Borges Simões

Coimbra, julho, 2020

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Desenvolvimento de um API RESTful para suporte a aplicações móveis de caracterização de condução

Relatório de Estágio de Natureza Profissional para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

Jorge Manuel Marques Pereira Lopes Pinheiro

Orientador

Anabela Borges Simões

Supervisor na empresa — Sentilant

Bruno Cabral

Coimbra, julho, 2020

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

AGRADECIMENTOS

Agradeço ao Instituto Superior de Engenharia de Coimbra, aos seus docentes, sem os quais não teria sido possível percorrer esta etapa de forma tão gratificante.

À Sentilant, empresa onde fiz o estágio, que me permitiu realizar o estágio e ter flexibilidade no horário para que eu pudesse dar aulas no ISEC.

À Professora Anabela Simões, minha orientadora de estágio, que sempre se mostrou disponível para me ajudar.

Aos meus amigos e familiares próximos que me acompanharam durante este percurso.

À Joana, por estar sempre presente e por tudo.

À minha mãe, aos meus irmãos, às minhas cunhadas, aos meus sobrinhos e às minhas sobrinhas, pelo apoio, por estarem sempre presentes e por tudo.

RESUMO

A segurança rodoviária é um ponto crítico na sociedade, sendo imperativo melhorar os hábitos de condução. Para tal, a Sentilant, empresa onde decorreu o estágio, oferece aplicações móveis de caracterização de condução, ou seja, aplicações que permitem medir os hábitos de condução.

Para expandir o seu negócio, a Sentilant propôs ao estagiário a criação de uma API RESTful, em *back-end*, que permita a utilização das funcionalidades que a Sentilant dispõe nos seus produtos a aplicações móveis de terceiros.

Para cumprir o objetivo, foi criada uma loja de forma a que os clientes possam adquirir e incorporar nas suas aplicações os produtos desenvolvidos pela Sentilant. Estes podem ser adquiridos, individualmente ou por subscrição, para serem incorporados nas suas aplicações, devendo os clientes depois da aquisição gerar chaves de acesso para que os seus utilizadores possam aceder ao produto adquirido.

Antes da criação deste produto, foi feito o estudo de quais as tecnologias e técnicas mais adequadas para o desenvolvimento. Por indicação da Sentilant, foi utilizada a linguagem Python, tendo sido feita a comparação entre duas *frameworks* para esta linguagem, Django e Flask, de forma a ver qual seria a mais indicada para o desenvolvimento, tendo sido também estudada e utilizada a plataforma de pagamentos Braintree, com a qual a Sentilant já trabalhou, e que permitiu criar os planos para os clientes fazerem as subscrições e pagamentos.

A conclusão deste estudo, mostrou que o mais indicado seria utilização da *framework* Django, pois tem incorporado um painel de administração, *forms, templates* e suporte para SQLite, utilizado neste desenvolvimento, tendo também uma excelente documentação e uma comunidade grande. Tendo, assim, sido optado por esta *framework*.

Os objetivos que diziam respeito ao estagiário, foram todos realizados sucesso, podendo a Sentilant incorporar o que foi desenvolvido de forma a poder disponibilizar o seu produto.

Palavras-chave: API, REST, Python, Django.

ABSTRACT

Road safety is a critical point in society, it is imperative to improve driving habits. To this end, Sentilant, the company where the internship took place, offers mobile driving characterization applications, applications that measure driving habits.

To expand its business, Sentilant proposed to the trainee the creation of a *back-end* RESTful API which allows the use of the functionalities that Sentilant has in its products to third-party mobile applications.

To fulfill the objective, a store was created so customers can purchase and incorporate the products developed by Sentilant in their applications. These can be purchased, individually or by subscription, to be incorporated into their applications, and after the purchase, customers must generate access keys so that their users can access the purchased product.

Before the creation of this product, a study was made of which technologies and techniques are most suitable for development. As indicated by Sentilant, the Python language was used, and a comparison was made between two frameworks for this language, Django and Flask, in order to see which would be the most suitable for development. The payment platform Braintree was also studied and used, with which Sentilant has already worked, and has enabled the creation of plans for customers to make subscriptions and payments.

The conclusion of this study showed that the most suitable would be to use the Django, as it has incorporated an administration panel, forms, templates and support for SQLite, used in this development, also because Django have an excellent documentation and a large community. Thus, this framework was chosen.

The objectives concerning the trainee were all successful, and Sentilant can now incorporate what was developed in order to make its product available.

Keywords: API, REST, Python, Django.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIATURAS E ACRÓNIMOS

API – *Application Programming Interface*.

APP – *Application*.

AVS – *Address Verification System*.

CRUD – *Create, Read, Update* e *Delete*.

CSRF – *Cross-site request forgery*.

CVV – *Credit Verification Value*.

DCU – *Diagrama de Casos de Uso*.

HTML – *HyperText Markup Language*.

HTTP – *HyperText Transfer Protocol*.

HTTPS – *Hyper Text Transfer Protocol Secure.*

ID – *Identity*.

IP – *Internet Protocol*.

JSON – *JavaScript Object Notation*.

Regex – *Regular expression*.

REST – *Representational State Transfer*.

REST API – *Representantional State Transfer.*

SDK – *Software development kit*.

SQL – *Structured Query Language*.

URI – *Uniform Resource Identifier*.

URL – *Uniform Resource Locator*.

WWW – *World Wide Web*.

XML – *eXtensible Markup Language*.

CAPÍTULO 1

1 INTRODUÇÃO

O presente relatório enquadra-se na Unidade Curricular Estágio inserida no Mestrado em Informática e Sistemas, Ramo de Desenvolvimento de Software, do Instituto Superior de Engenharia de Coimbra e tem como objetivo proporcionar aos alunos a integração nas empresas, possibilitando aos estagiários desenvolver um produto que não só seja útil para a empresa, como também para o estagiário, de forma a que este possa adquirir conhecimentos relevantes na área da engenharia informática.

Tendo em conta os objetivos da Unidade Curricular Estágio, foi feita a proposta, por parte da empresa Sentilant que tem produtos para aplicações móveis que medem os hábitos de condução, para que o estagiário desenvolva, em *back-end*, uma API RESTful. Mais concretamente, o estagiário tem de criar um sistema de compras que permita aos utilizadores comprarem a API, também desenvolvida pelo estagiário, para que possam incorporar nas suas aplicações os produtos desenvolvidos pela Sentilant.

De seguida, neste capítulo, será apresentada a empresa onde decorreu o estágio, para depois ser feita a contextualização do problema, onde se salienta a importância do estágio, seguindo-se a descrição os objetivos do estágio e, no último subcapítulo, a organização do documento.

1.1 A Sentilant

Sediada na incubadora Instituto Pedro Nunes[1], em Coimbra, a Sentilant[2], que nasceu no ano de 2013, "desenvolve produtos na área de telemática para a condução, aplicáveis à área da segurança rodoviária e seguros"[3], e tem produtos como *Drivian*, o seu primeiro produto, uma aplicação móvel disponível também para *web*, que tem como objetivo "promover uma condução mais segura e económica" (Inovação, 2015) ao notificar os condutores quando estes cometem alguma falha na sua condução, de forma a que depois consigam perceber as suas falhas e melhorar a sua condução (Inovação, 2015), *AutoBund,* um produto que também pretende analisar os hábitos de condução *e Driviantasks*[4], "um sistema inteligente e de baixo custo, que combina a última tecnologia em sistemas web com soluções de mobilidade acessíveis para planear e controlar operações, operacionais e recursos em tempo real." (DrivianTasks Gestão inteligente de mobilidade, 2016). Tendo também o *GeoTasks*, uma "solução

[1] Para mais informações sobre o Instituto Pedro Nunes, uma instituição privada sem fins lucrativos, cf. https://www.ipn.pt/ (consultado a 28/05/2020).
[2] O *site* da Sentilant encontra-se em https://sentilant.com/ (consultado a 28/05/2020).
[3] Cf. Anexo A, proposta de estágio.
[4] Para mais informações sobre o DrivianTasks pode ser consutlado o *site*: https://www.driviantasks.com (consultado a 28/05/2020).

informática para as empresas planearem, otimizarem e preverem as tarefas dos trabalhadores que andam na estrada" (Larguesa, 2018) e que tem como principal concorrência as empresas Jobber, Synchroteam, CoreSystems e ClickSoftwarer, primando a Sentilant pela a sua simplicidade e preço de subscrição (Larguesa, 2017).

Entre os produtos da Sentilant, encontra-se a APP *Ganha LOGO*, uma aplicação de segurança rodoviária e que foi útil ao estagiário para este compreender o que a empresa faz. Esta aplicação dá *feedback* aos utilizadores sobre a condução destes através de um sistema de pontos. Pontos estes que são o resultado das métricas: velocidade, ter uma velocidade moderada; atenção, não estar ao telemóvel; fadiga, não conduzir muitas horas seguidas; eficiência, ter atenção às travagens, curvas e velocidade; condução defensiva, ter uma condução adequada ao tipo de estrada e condições metrológicas, não só no momento da viagem, mas também antes da viagem, pois também influenciam o estado da estada.[5]

Esta APP, para além de dar pontuação e da inserção, edição e eliminação de veículos e visualização de viagens, tem também, por exemplo, a possibilidade de ver o progresso das viagens, uma parte social, onde é possível adicionar *amigos* (pessoas) que tenham a aplicação, a hipótese dos condutores completarem desafios, individuais ou em equipa (caso se tenha *amigos* com a aplicação), e receber *medalhas* virtuais depois de os concluir.

1.2 Contextualização do problema

A Sentilant, dentro dos produtos que desenvolve, tem produtos para a segurança rodoviária, relacionados com os hábitos de condução. No momento atual, a empresa sentiu a necessidade de ampliar a sua oferta permitindo que aplicações móveis de terceiros comprem um API RESTful desenvolvido pela Sentilant e que permite às empresas que adquiram o produto aceder às funcionalidades da Sentilant.

Assim, neste estágio, foi proposto ao estagiário criar, em *back-end*, uma loja que permita a aquisição de algumas funcionalidades disponibilizadas pela Sentilant, quer seja ao adquirir um produto único ou fazer a subscrição de um plano, previamente definido pela Sentilant. Adquiridos os produtos, a empresa que os adquire pode criar chaves (*API-keys*) para esses produtos, possibilitando assim o uso destas funcionalidades por parte dos seus utilizadores.

Com a criação desta API RESTful, a Sentilant amplia o seu negócio, pois não só continua com o negócio atual, onde vende as suas soluções, nomeadamente sob a forma de aplicações móveis que permitem a medir os hábitos de condução, como também permite que os seus produtos sejam adquiridos através desta API, uma possibilidade de projeção da empresa, ou seja, uma mais valia para o negócio e algo fundamental dos dias de hoje, onde a concorrência é cada vez maior.

[5] Adaptado da secção "informação" disponível na aplicação Ganha Logo.

De seguida, no subcapítulo 1.3, são apresentados os objetivos do trabalho para que se compreenda melhor o propósito do estágio.

1.3 Objetivos do estágio

Com o propósito de perceber melhor os objetivos do trabalho, este subcapítulo foi dividido em objetivos gerais e objetivos específicos.

1.3.1 Objetivos gerais

De acordo com a proposta apresentada pela Sentilant, os objetivos gerais foram os seguintes:

- "Desenho de uma API RESTful para suporte a aplicações de caracterização de condução da Sentilant.
- Desenvolvimento do API projetado.
- Integração dos desenvolvimentos no produto da empresa."[6]

1.3.2 Objetivos específicos

Os objetivos específicos dizem respeito ao trabalho desenvolvido, em *back-end*, no âmbito do produto e são os seguintes:

1. Possibilitar que os utilizadores, sem estarem registados, visualizem os produtos e planos de subscrição disponibilizados pela Sentilant;
2. Permitir que os administradores da Sentilant insiram, editem e eliminem produtos, assim como atualizem a lista de planos disponíveis;
3. Permitir que os utilizadores se registem na plataforma para poderem adquirir o(s) produto(s)/plano(s);
4. Disponibilizar o *token*[7] de acesso à plataforma após o registo;
5. Possibilitar que os utilizadores, com registo efetuado, entrem na plataforma;
6. Possibilitar a edição e visualização do perfil;
7. Permitir que os utilizadores possam mudar a *password*;
8. Viabilizar a hipótese de se alterar o *username*/*email*;
9. Conceder a possibilidade de recuperar a *password*;
10. Possibilitar que o cliente elimine a sua conta;

[6] Os objetivos gerais apresentados são os disponibilizados na proposta de estágio. Cf. Anexo A.
[7] Os *tokens* são *chaves* que possibilitam o acesso a informação específica.

11. Possibilitar adicionar produtos a uma encomenda;
12. Assegurar a possibilidade que o cliente possa visualizar as suas encomendas;
13. Permitir que o cliente edite ou elimine uma encomenda ainda não transacionada;
14. Permitir que o cliente faça uma encomenda de um ou mais produtos;
15. Conceder a possibilidade de visualização das transações efetuadas;
16. Possibilitar que o cliente faça subscrições;
17. Possibilitar que o cliente adicione cartões de crédito para proceder ao pagamento das subscrições, assim como editar os cartões ou eliminar;
18. Assegurar que é possível ver as subscrições adquiridas;
19. Viabilizar a hipótese de editar e cancelar uma subscrição;
20. Permitir que o cliente gere chaves para que outros utilizadores tenham acesso ao produto/plano adquirido;
21. Possibilitar a visualização das chaves criadas que permitem o acesso a uma parte ou totalidade da aplicação da Sentilant;
22. Assegurar que o cliente tem a possibilidade de eliminar chaves de acesso à aplicação Sentilant;
23. Permitir fazer o registo na aplicação da Sentilant;
24. Possibilitar fazer *login* na aplicação da Sentilant;
25. Permitir a visualização do perfil e edição do mesmo;
26. Conceder a possibilidade de inserir, editar e eliminar veículos;
27. Permitir que a visualização das especificações dos veículos;
28. Permitir a inserção, edição e eliminação de viagens;
29. Facultar a possibilidade de o utilizador visualizar a sua pontuação geral.

1.4 Organização do documento

O presente relatório está dividido em seis capítulos, que a seguir se descrevem.

O primeiro capítulo tem como objetivo fazer uma breve introdução, de forma a que se compreenda qual o âmbito do estágio, perceber o que faz a Sentilant, qual o propósito do estágio e também a motivação que levou o estagiário a escolher a empresa Sentilant para realizar o seu estágio, definindo ainda, neste capítulo, os objetivos gerais e específicos do estágio.

No segundo capítulo, é apresentado o plano inicial para o estágio, aquele que foi proposto ao estagiário e o plano final, que demonstra como realmente decorreu o

estágio, sendo feita também uma comparação entre os dois planos. Neste capítulo é ainda feita uma breve descrição sobre a metodologia usada ao longo do estágio.

No que respeita ao terceiro capítulo, é feita uma análise do estado da arte no que respeita às tecnologias existentes no mercado para desenvolver uma API RESTful, assim como as melhores técnicas, de forma a que a escolha para o seu desenvolvimento permita que a Sentilant continue a emergir cada vez mais no mercado.

No quarto capítulo, são apresentadas as ferramentas utilizadas no decorrer do estágio, com uma breve descrição para compreender o motivo da sua utilização.

O quinto capítulo diz respeito ao desenho da solução, onde é incluída a informação sobre a descrição dos *stakeholders*[8] e utilizadores, os requisitos funcionais e não funcionais e as funcionalidades da API, dividida por grupos para uma melhor compressão, sendo esses grupos os seguintes: produtos, planos, registo, perfil, alterar *password,* recuperar *password*, *username*, encomendas, transações, cartões, subscrições, chaves, autenticação na APP, perfil na APP, veículos e viagens. Sendo cada grupo composto por diversos subcapítulos, adaptados de quando o estagiário realizou o seu estágio da licenciatura: objetivo, premissas, estratégia, requisitos funcionais, casos de uso, estrutura, implementação e testes unitários.

No último capítulo, o sexto, são apresentadas as conclusões finais, onde é feita uma reflexão sobre como correu o estágio e também o trabalho futuro.

Acresce ainda a estes seis capítulos, as referências bibliográficas e os seguintes anexos:

- Anexo A – Proposta de Estágio
- Anexo B – Sentilant API - Swagger

[8] Indivíduo ou organização que é parte interessada.

CAPÍTULO 2

2 PLANEAMENTO E METODOLOGIA

Neste capítulo é apresentado, em primeiro lugar, o plano de desenvolvimento do estágio, onde é feita uma breve reflexão sobre o mesmo, de forma a compreender melhor como correu o estágio a nível da realização das atividades propostas e, de seguida, é feita uma breve descrição da metodologia utilizada no estágio.

2.1 Plano de desenvolvimento

Para compreender melhor como decorreu o estágio, são apresentados, de seguida, dois planos: o plano inicial, que se encontra na Tabela 1 e é adaptado da proposta de estágio[9], e o plano final, apresentado na Tabela 2, que mostra como decorreu, efetivamente, o estágio.

É ainda apresentado um diagrama de Gantt, na Figura 1, que mostra a comparação entre os dois planos, seguida de uma breve explicação das diferenças entre os dois planos.

- Planeamento inicial

Tabela 1 - Planeamento inicial do estágio.

Tarefas	Atividade	Início	Término	Duração (semanas)	Duração (dias úteis)
Tarefa 1	Definição de requisitos	22/10/2019	31/10/2019	2	8
Tarefa 2	Estado da arte	01/11/2019	31/12/2019	9	41
Tarefa 3	Programação	01/12/2019	30/04/2020	22	106
Tarefa 4	Integração	01/03/2020	31/05/2020	13	63
Tarefa 5	Validação	01/03/2020	30/06/2020	17	83
Tarefa 6	Escrita do relatório	01/05/2020	10/07/2020	10	48

[9] Cf. Anexo A – Proposta de Estágio.

- Planeamento final

Tabela 2 - Planeamento final do estágio.

Tarefas	Atividade	Início	Término	Duração (semanas)	Duração (dias úteis)
Tarefa 1	Definição de requisitos	22/10/2019	11/12/2019	7	36
Tarefa 2	Estado da arte	01/11/2019	31/1/2020	13	63
Tarefa 3	Programação	01/02/2020	25/05/2020	17	79
Tarefa 4	Integração	01/05/2020	25/05/2020	4	16
Tarefa 5	Validação	17/06/2020	17/06/200	0	1
Tarefa 6	Escrita do relatório	22/05/2020	10/06/2020	3	20

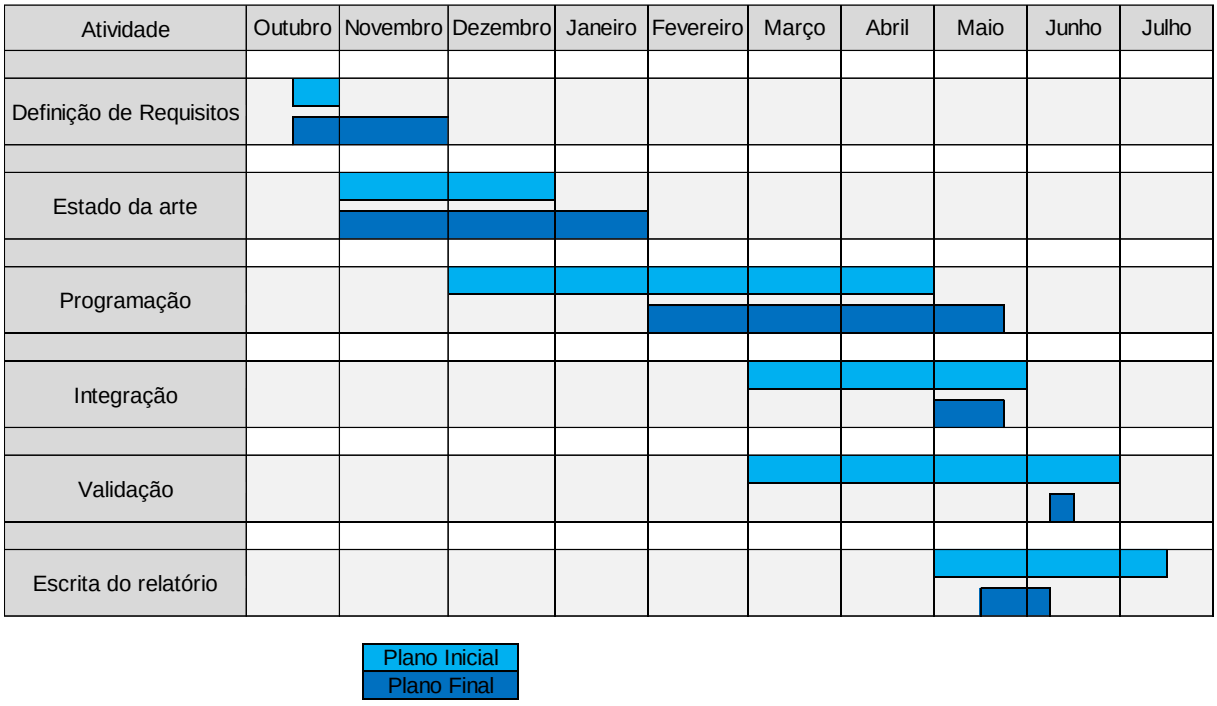


Figura 1 - Diagrama de Gantt: Plano Inicial e Plano Final

Como se pode observar pela Figura 1, os objetivos foram todos cumpridos, apesar de não terem seguido exatamente a duração prevista no plano inicial.

O facto de a definição de requisitos ter demorado mais tempo que o previsto, deveu-se, nomeadamente, ao facto de uma semana, em tempo parcial, ser pouco para os definir, uma vez que o estagiário não conhecia a empresa nem o seu produto, pontos fundamentais para a definição dos requisitos. Assim, nesta fase, o estagiário explorou também a aplicação móvel da Sentilant (produto final, não o código) de forma a

conseguir perceber melhor o que faz a Sentilant para definir os requisitos para a API RESTful. Requisitos estes que foram discutidos nas reuniões com a equipa da Sentilant, uma vez que no início ainda existiam reuniões semanais.

O facto de a definição de requisitos ter demorado mais que o previsto, levou a que o estado da arte também tivesse uma duração maior, pois este último ponto começou quando ainda decorria o anterior. Acrescido do facto do estagiário desde sensivelmente o meio de dezembro ter trabalhado sem a ajuda das reuniões com a equipa da Sentilant, o que dificultou muito a tarefa deste ao longo do estágio.

A parte de desenvolvimento, apesar de ter começado mais tarde que o previsto, como o estagiário fez um estudo prévio no estado da arte de forma a perceber bem o funcionamento das ferramentas acabou por facilitar e levar a um término sensivelmente no tempo previsto, assim como a integração, também pelo facto do estagiário ter trabalhado desde 17 de Fevereiro pelo menos 56 horas semanais no estágio, por iniciativa própria.

Sobre a validação, foi realizada no mês de junho, numa sessão via Zoom, onde o estagiáro demonstrou o trabalho final, tendo o mesmo sido aprovado e validado, sendo dispensado o documento assinado pela Senliant a comprovar a validação.

Quanto ao relatório, o tempo de escrita foi consideravelmente inferior ao suposto, uma vez que o estagiário tem facilidade no processo de escrita e organizou tudo durante o estágio para que fosse fácil a sua escrita.

2.2 Metodologia

No início do estágio, quando era em tempo parcial, aquando da definição dos requisitos funcionais e não funcionais, foi definido que todas as semanas ia decorrer uma reunião onde era referido o que foi feito na semana anterior e o que estava previsto fazer para a próxima semana.

Contudo, sensivelmente a meio de dezembro, depois de cerca de sete reuniões, a metodologia mudou. Isto é, deixou de existir. Acabaram as reuniões semanais e o estagiário teve de fazer, nomeadamente, a parte do estado da arte e desenvolvimento da API RESTful sozinho, sem nenhuma orientação, o que acabou por tornar o estágio ainda numa tarefa mais complexa que inicialmente era previsto, pois é crítico não existir uma metodologia, nenhum apoio. Não obstante este ponto, o estagiário ultrapassou todas as dificuldades e com as bases que tinha e aprendendo tudo de forma autodidata, conseguiu chegar ao objetivo de ter a API a funcionar, cumprindo assim o objetivo.

Na Tabela 3, são referidos os pontos que foram cumpridos por parte da Sentilant e os que não foram no que respeita ao desenvolvimento e documentação das funcionalidades devolvidas.

Tabela 3 - Proposta da Sentilant para metodologia e se foi aplicada pela empresa.

Proposta da Sentilant[10] \ Realizado pela Sentilant? (Sim/Não)	Sim	Não
Integração do estagiário no processo de desenvolvimento de software interno.		X
Utilização de uma metodologia ágil para gestão do desenvolvimento baseada em SCRUM.		X
Recurso a técnicas de estimação para planeamento detalhado de sprints.		X
Reuniões de início e fim de sprint para definição e avaliação de objetivos.		X
Escrita de documentação inerente às funcionalidades desenvolvidas, de acordo com os requisitos da empresa.		X

Como se pode constar, nenhum dos pontos da Tabela 3 foi aplicado pela Sentilant, o que acabou por dificultar, de forma muito significativa, como referido, a tarefa do estagiário.

Perante este facto, fica como sugestão para que no futuro a empresa aplique, efetivamente, a adaptação da metodologia ágil proposta, cumprindo assim com o proposto.

[10] A proposta da Sentilant apresentada nesta tabela é que se encontra no Anexo A, *ipsis verbis*.

CAPÍTULO 3

3 ESTADO DA ARTE

Neste capítulo é feita uma breve análise da API RESTful[11], com o intuito de perceber melhor os seus princípios e boas práticas, para depois ser feito um estudo comparativo entre Django Rest Framework e o Flask, as duas *frameworks* para Python mais utilizadas, de forma a perceber qual a melhor opção para a API a desenvolver, e será feita ainda uma análise ao Braintree, a plataforma de serviço de pagamentos que a Sentilant já utiliza.

3.1 API REST

Uma API REST é "uma API WEB que esteja em conformidade com o estilo de arquitetura REST" (Massé, 2012, p. 5), sendo a API WEB "a face de um serviço da Web, diretamente ouvindo e respondendo às solicitações do cliente" (Massé, 2012, p. 5), como exemplificado na Figura 2.

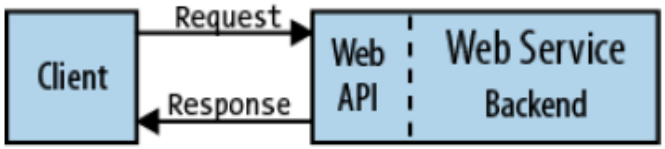


Figura 2 - Web API.[12]

Para a criação de uma API REST, foram tidos em consideração alguns pontos fundamentais, tais como perceber quais são os princípios REST, saber como definir os recursos (*resources*), utilizar os métodos HTTP *request* de forma correta*,* escolher o método do tipo de retorno (*e.g.* XML, JSON), retornar o código de *status* correto, ter uma versão da API consistente e ter documentação. (Mota, 2019) (Joshi, 2017)

Existem cinco princípios REST, ou seja, cinco práticas que devem ser seguidas para que uma API REST tenha uma arquitetura estruturada. Tem de haver uma separação clara entre cliente-servidor, isto é, estes devem poder trabalhar de forma separada, a interface do cliente tem de ter a possibilidade de ser trabalhada em multiplataformas, independentemente do servidor. Outro princípio é o facto de a interface ter de ser uniforme, ou seja, para que a comunicação entre cliente e servidor seja feita de forma eficiente, tem de haver um contrato bem definido, que não mude a sua lógica, existir um URI que retorne o conteúdo previsto, sem se alterar o URI com o tempo. A API REST tem de ser também *statele*ss, ou seja, os *requests* do cliente têm de ter toda a informação necessária para o processamento da operação; outro principio é a cache,

[11] RESTful é o ato de fazer REST.
[12] Fonte: Massé, M. (2012). REST API Design Rulebook. Sebastopol: O'Reilly. p. 6.

como os *requests* do cliente são *statele*ss, o que leva a uma perda de performance, a cache permite aumentar essa performance, reduzindo a latência entre as interações, ao armazenar, com autorização do cliente, a informação. Por fim, a API REST tem de ser construída por camadas, ou seja, o cliente não pode saber se o seu *request* está a fazer uma conexão direta ao servidor final ou passa por outros até ao final, não consegue saber para além da camada para o qual faz o pedido. (Simpson, 2020) (REST Architectural Constraints, s.d.) (Biswas, 2019) (De, 2017, pp. 29-31)

A nível dos recursos, é importante que sigam todos o mesmo padrão, sejam simples e intuitivos. Devendo estes seguir algumas regras, nomeadamente utilizar o plural, ou seja, “se o domínio é /vehicle, deve-se usar /vehicles” (Mota, 2019), acontecendo o mesmo com os sub-recursos, se se pretender saber algo do veículo, deve ser no plural, como por exemplo “/vehicles/specifications”.

Na Tabela 4 são apresentados os métodos HTTP utilizados nas APIs, indicando a função de cada um deles. De salientar que tanto o método PUT como o método PATCH permitem atualizar, contudo, enquanto normalmente o PUT é utilizado para atualizar todos os campos, o método PATCH é utlizado para fazer uma atualização parcial, isto é, alterar apenas alguns campos.

Tabela 4 - Métodos HTTP.

Método HTTP	CRUD
POST	*CREATE* (criar)
GET	*READ* (ler)
PUT	*UPDATE* (atualziar)
PATCH	*UPDATE* (atualizar)
DELETE	*DELETE* (eliminar)

A nível de respostas, foi optado por utilizar JSON e não XML, não só pelo facto da Sentilant utilizar JSON, mas também pelo facto de o XML ser mais difícil de analisar, mais difícil de ler e mais detalhada, ou seja, mais complexa e também mais lenta (Sahni, s.d.) (Difference Between JSON and XML, s.d.) (Joshi, 2017), além do JSON ser mais utilizado. Um exemplo indicativo da tendência de utilização do JSON são as pesquisas que são feitas no Google, com recurso ao Google Trends pode-se observar que ao fazer a comparação entre XML API e JSON API a nível de pesquisas nos últimos cinco anos, que a diferença é significativa, o JSON tem um número consideravelmente superior ao XML, como se pode observar na Figura 3.

Figura 3 - Google Trends: XML API vs JSON API, últimos 5 anos.[13]

Quanto aos códigos de resposta (*status code*), na Tabela 5 são apresentados alguns dos mais utilizados, estando presentes nesta lista códigos também utilizados na API desenvolvida, sendo que quando o código começa com 2, significa que o pedido foi feito com sucesso, quando começa com 4, o pedido não foi feito devido a um erro do lado do cliente e, por fim, nesta lista, quando começa com 5, indica que ocorreu um erro do lado do servidor, não tendo sido, por esse motivo, a operação concretizada.

Tabela 5 - Alguns *Status Code* mais utilizados.

Status Code	Nome	Descrição
200	*OK*	O pedido foi feito com sucesso.
201	*Created*	O recurso foi criado.
202	*Accepted*	O pedido foi aceite para processamento.
204	*No Content*	O pedido correu bem, mas não houve conteúdo na resposta.
400	*Bad Request*	Algo correu mal no pedido, mas o servidor não conseguiu perceber o motivo.
401	*Unauthorized*	O pedido não foi autorizado.
403	*Forbidden*	O pedido foi efetuado, mas as credenciais não conferem acesso.
404	*Not Found*	O URI não existe.
405	*Method Not Allowed*	O método que é enviado no pedido não é aceite.
409	*Conflit*	Existe um conflito no pedido.

[13] Fonte: https://trends.google.pt/trends/explore?date=today%205-y&q=xml%20api,json%20api, consultado a 15/05/2020.

429	*Too Many Requests*	Foram enviados mais pedidos que os permitidos.
500	*Internal Server Error*	Ocorreu um erro no servidor que impede a concretização do pedido.

A nível da versão, a presente API apresenta uma versão 1.0, uma vez que se encontra em condições para ir para desenvolvimento, podendo ser adicionados mais recursos que a Sentilant queira disponibilizar na API de uma forma simples. De salientar que a versão 1 só aparece na API que é adquirida pelo cliente, não na loja da Sentilant.

Para gerar documentação foram analisadas algumas ferramentas como OpenAPI GUI [14], ReDOC [15] e Swagger [16], tendo sido optado por esta última para o desenvolvimento da API, uma vez que o estagiário já tinha conhecimento da ferramenta e é a mais utilizada a nível da criação de documentação no que respeita às APIs, sendo também uma das sugeridas pela *framework* Django Rest Framework[17].

De seguida, será feita uma breve reflexão sobre as diferenças entre Django e Flask, de forma a perceber qual a ferramenta que melhor se ajusta para a API RESTful.

3.2 Django vs Flask

Apesar da Sentilant utilizar a *framework* Django[18], o estagiário achou pertinente fazer uma comparação desta com outra *framework* para Python[19], muito utilizada também, o Flask[20]. O motivo deste estudo prendeu-se, nomeadamente, para ver se existiam mais vantagens em desenvolver a API em Flask e se assim fosse expor a situação de forma a ver da disponibilidade da Sentilant para fazer a sua alteração, uma vez que é importante utilizar as ferramentas mais adequadas às tarefas que são realizadas, estando sempre na vanguarda do que é utilizado.

Assim, na Tabela 6, são apresentadas algumas diferenças entre o Django e o Flask que foram tidas em conta para a decisão final de escolher a *framework* a utilizar.

[14] O *site* pode ser consultado em https://mermade.github.io/openapi-gui/ (consultado a 11/11/2019).
[15] O *site* pode se consultado em https://redocly.github.io/redoc/ (consultado a 11/11/2019).
[16] A descrição desta ferramenta encontra-se no capítulo 4, ponto 4.8 Swagger Editor.
[17] Cf. https://www.django-rest-framework.org/topics/documenting-your-api/ (consultado a 11/11/2019).
[18] A documentação do Django pode ser consultada em https://www.djangoproject.com/ (consultado a 11/11/2019).
[19] Mais informação sobre o Python pode ser consultada em https://www.python.org/ (consultado a 11/11/2019).
[20] A documentação do Flask, pode ser consultada no site https://flask.palletsprojects.com/en/1.1.x/ (consultado a 11/11/2019).

Tabela 6 - Django vs Flask.

Tópicos	Django	Flask
Criado	2005	2010
Curva de aprendizagem	Ligeiramente mais difícil	Ligeiramente mais fácil
Painel de administração	Tem	Não tem
Base de dados	SQLite, PostgreSQL e MySQL	NoSQL ou sem Base de dados
Tamanho do projeto	Indicado para maiores	Indicado para menores
Autenticação	Inclui	Não inclui
Suporte de bibliotecas	Menos que o Flask	Mais que o Django
Templates	Incorporado	Jinja2 (baseado do *template* do Django)
Forms	Incluído	Não incluido
Performance	Ligeiramente pior	Ligeiramente melhor
Segurança	Semelhante	Semelhante
Testes	Unittest framework do Python	Unittest framework do Python
Documentação	Excelente	Muito boa
Comunidade	Muito grande	Grande
Clientes conhecidos	Instagram, Dropbox, Udemy	Linkedin, Netflix, Uber

Para além destas diferenças, foi também analisada frequência de pesquisas das *frameworks* no google. Assim, recorrendo ao Google Trends, verificou-se que nos últimos 5 anos a utilização do Django tem tido uma procura no google consideravelmente maior que o Flask, como se pode verificar na Figura 4.

Figura 4 - Google Trends: Django vs Flask, últimos 5 anos.[21]

[21] Fonte: https://trends.google.pt/trends/explore?date=today%205-y&q=%2Fm%2F0dgs72v,%2Fm%2F06y_qx, consultado a 15/05/2020.

Como se pode observar na Tabela 6, em comparação com o Flask, o Django tem mais cinco anos de existência, o que leva a que também tenha uma comunidade maior. Não obstante, como se pode observar na Figura 4, a tendência continua a ser para a procura continuar a ser por Django, visto que o número de pesquisas dos últimos cinco anos ter pendido, nitidamente, para Django, o que é um ponto a favor deste último.

A nível da curva de aprendizagem, o Flask tem um ponto a seu favor, uma vez que é mais fácil adquirir conhecimentos sobre esta tecnologia (Anderson, 2019), contudo, para o desenvolvimento da API este ponto não foi muito tido em conta, pois é dever do estagiário trabalhar com a melhor tecnologia, independentemente da sua dificuldade de aprendizagem, pois um dos objetivos do estágio é que o estagiário possa adquirir novos conhecimentos.

Ainda na Tabela 6, podemos observar alguns pontos a favor do Django, como o facto de este já vir com um painel de administração que pode ser customizado, facilitando assim as tarefas dos *developers*, já o Flask não tem, possibilitando, contudo, a adição de extensões que permitem essa funcionalidade (Herman, 2019), porém fica-se a depender de terceiros, o que é uma desvantagem.

Outros pontos a favor do Django é o facto de este ter um sistema de autenticação incluído, assim como *templates*, "mecanismos de modelo que permitem injetar informações do *back-end* dinamicamente numa página" (Herman, 2019), e *forms*, suporta "formulários de criação, validação de dados e validação de CSRF token" (Kostrzewa, 2019). O facto de estar incluído no Django, vem facilitar muito o trabalho a quem desenvolve o produto, não necessitando assim de recorrer a bibliotecas externas como teria de ser com a *framework* Flask (Singh, 2020). A nível da segurança, se por um lado entre as duas *frameworks* é semelhante, o facto do Flask recorrer mais a bibliotecas externas vem comprometer a sua segurança, uma vez que é necessário que os *developers* estejam sempre atentos a essas extensões.

É ainda uma ventagem do Django, o facto de esta ter suporte para a base de dados que é utilizada na API desenvolvida pelo estagiário, o SQLite, ao contrário do Flask, que teria de recorrer a terceiros (Django vs Flask in 2020, 2019).

Todavia, se se pensar em termos de performance e suporte de bibliotecas, o Flask tem uma melhor performance ligeiramente melhor, ponto que é muito importante, e tem um maior suporte para bibliotecas, o que também poderá ser prejudicial, pois pode-se tornar mais complexa a sua gestão (Senij & Stempniak, 2019).

No que respeita a documentação, o Flask oferece uma documentação bem organizada, contudo, ainda longe da documentação que o Django oferece, pois, a deste último, está muito bem detalhada e estruturada.

Perante estes dados, achou-se que era mais indicado continuar a trabalhar com a *framework* Django, uma vez que no geral é mais indicada, tendo também, naturalmente, um peso o facto de a empresa já trabalhar com esta ferramenta.

Depois de se ter optado pelo Django, foi decidido utilizar também o Django Rest Framework[22], um conjunto de ferramentas, desenvolvidas em Django, para construir Web APIs, de forma a aproveitar as suas funcionalidades e facilitar o desenvolvimento da API, como por exemplo ao fazer uso do seu recurso *Throttling*, que permite determinar se um pedido deve ser aceite.

Esta política de *Throttling* deve ser definida nos *settings* do projeto, podendo depois ser reescrita. Na Figura 5, apresenta-se um exemplo dessa configuração base, onde é exemplificado a limitação para utilizadores anónimos e utilizadores identificados, sendo que o limite do exemplo para os utilizadores anónimos são 100 *requests* diários, e para os utilizadores identificados, 1000 *requests* diários. Ou seja, se o utilizador anónimo fizer mais de 100 requests num dia, não poderá aceder mais ao conteúdo nesse dia, voltando o contador a 0 no dia seguinte.

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ],
    'DEFAULT_THROTTLE_RATES': {
        'anon': '100/day',
        'user': '1000/day'
    }
}
```

Figura 5 - Política de *Throttling*.[23]

Definida a *framework*, foi analisada a plataforma de serviços Braintree, permitindo assim ao estagiário compreender a forma como são processados os pagamentos e são feitas as subscrições para depois poder implementar aquando da criação da loja da Sentilant e API.

3.3 Braintree

Para fazer o estudo da tecnologia que permita aos clientes fazerem os pagamentos e as subscrições, foi indicado ao estagiário que devia analisar o *site* da plataforma *Braintree*[24], "uma solução inovadora para tecnologia de pagamentos [e] soluções escaláveis" (Braintree, s.d.), de forma a perceber a melhor maneira de aproveitar este serviço, uma vez que a Sentilant já trabalha com esta empresa.

Perante estas indicações, foi estudada a plataforma *Braintree*, não só para compreender os produtos oferecidos, mas nomeadamente para perceber como esta

[22] O *site do* Django Rest Framework pode ser consultado em https://www.django-rest-framework.org/ (consultado a 12/12/2019).
[23] Fonte: https://www.django-rest-framework.org/api-guide/throttling/, consultado a (15/02/2020).
[24] O *site* da Braintree pode ser consultado em https://www.braintreepayments.com/ (consultado a 30/012/2019).

pode ser implementada na API que se está a desenvolver. Assim, a nível da sua inclusão na API, o *site* da plataforma Braintree demonstra quais os passos que esta ferramenta segue, como se pode constatar na Figura 6.

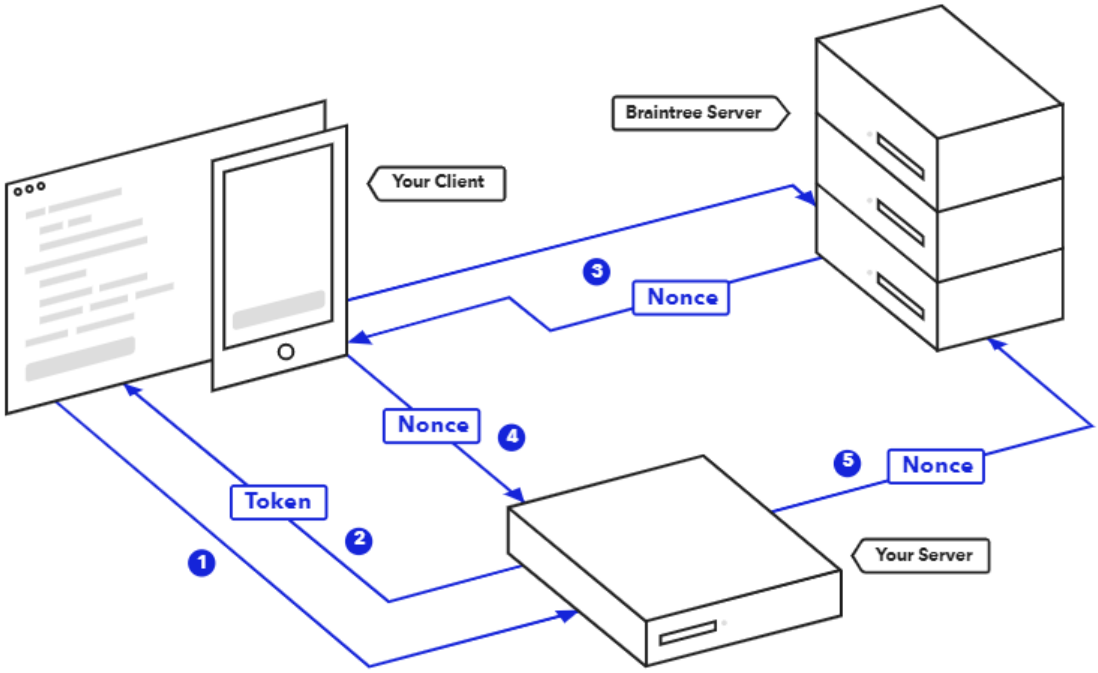


Figura 6 - Interação entre Cliente, Servidor e Braintree.[25]

Como se pode observar na Figura 6, esta interação entre cliente, servidor e Braintree pode ser divida em duas fases. A primeira, que engloba os pontos de 1 a 4 da Figura 6, diz respeito ao *setup* do cliente e outra fase, que se refere ao *setup* do servidor e que compreende os pontos 1, 2, 4 e 5 da mesma figura (Braintree, Overview, s.d.).

Este processo começa com uma interação entre cliente e servidor (ponto 1 e 2). É nesta primeira interação que o utilizador vai fazer um pedido ao servidor (ponto 1) para que este conceba um *token*, utilizando o Braintree SDK, e o envie ao cliente (ponto 2). Já com o *token*, o cliente SDK, disponíveis para Android, iOS e JavaScript, vai submeter a sua informação de pagamento (ponto 3) ao servidor Braintree que vai retornar as formas de pagamento *nonce*[26] e de seguida o *front-end* com a informação recebida é enviada para o servidor da Sentilant que vai usar o SDK da Braintree, disponível para a linguagem Python, entre outras, para criar a transação, enviando essa informação para o Braintree (ponto 5). (Braintree, Overview, s.d.).

A nível do servidor, a parte que interessa mais ao estagiário, uma vez que está a trabalhar com a parte de *back-end*, a sua instalação passa por passos como indicar

[25] Fonte: https://developers.braintreepayments.com/start/overview, consultado a 12/12/2019.
[26] N*once* “é um número arbitrário destinado a um único uso” (Computer Hope - Nonce, 2019).

as configurações e credenciais da API, Figura 7, gerar o *token* do cliente, Figura 8 e permitir a criação de transações, Figura 9.

```
import braintree

gateway = braintree.BraintreeGateway(
  braintree.Configuration(
    braintree.Environment.Sandbox,
    merchant_id="use_your_merchant_id",
    public_key="use_your_public_key",
    private_key="use_your_private_key"
  )
)
```

Figura 7 - Braintree: Configurações base.[27]

```
client_token = gateway.client_token.generate({
  "customer_id": a_customer_id
})
```

Figura 8 - Braintree: Gerar *token* do cliente.[28]

```
result = gateway.transaction.sale({
  "amount": "10.00",
  "payment_method_nonce": nonce_from_the_client,
  "device_data": device_data_from_the_client,
  "options": {
   "submit_for_settlement": True
  }
})
```

Figura 9 - Braintree: Criar uma transação.[29]

A estes pontos de configuração base, acresce também a criação de novos clientes, Figura 10, e fazer subscrições, Figura 11.

[27] Fonte: https://developers.braintreepayments.com/start/hello-server/python, consultado a 12/12/2019.
[28] *Ibidem.*
[29] *Ibidem.*

```
result = gateway.customer.create({
  "first_name": "Jen",
  "last_name": "Smith",
  "company": "Braintree",
  "email": "jen@example.com",
  "phone": "312.555.1234",
  "fax": "614.555.5678",
  "website": "www.example.com"
})
```

Figura 10 - Braintree: Criar novos clientes.[30]

```
result = gateway.subscription.create({
    "payment_method_token": "token",
    "plan_id": "Trips",
    "price": price,
    "never_expires": True,
})
```

Figura 11 - Braintree: Fazer uma subscrição.[31]

No que respeita ao modo como é feito o processamento das transações e subscrições, este é apresentado na Figura 12, para as transações, e na Figura 13 para as subscrições.

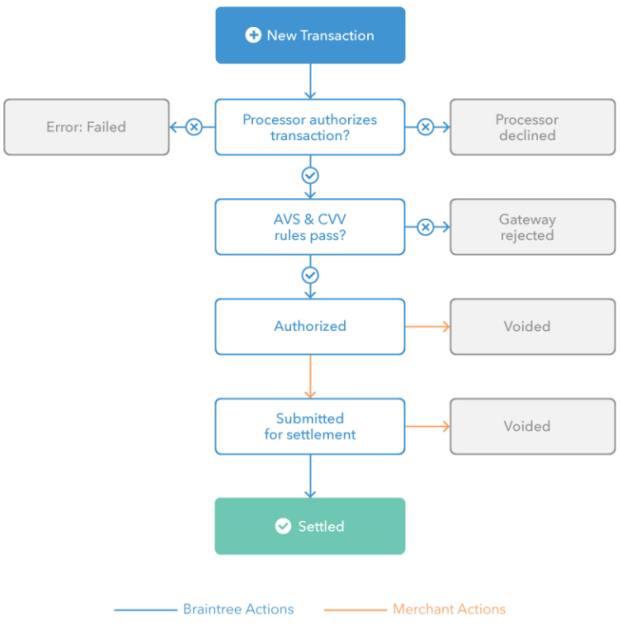


Figura 12 - Braintree: Transações.[32]

[30] Fonte: https://developers.braintreepayments.com/reference/request/customer/create/python, consultado a 12/12/2019.

[31] Para mais informações sobre as subscrições, cf. https://developers.braintreepayments.com/reference/ request/subscription/create/python, consultado a 12/12/2019.

[32] Fonte: https://developers.braintreepayments.com/guides/transactions/python, consultado a 12/12/2019.

A nível das transações, a Figura 12 mostra como é que estas são tratadas internamente, sendo o seu processo simples, antes da transação ser estabelecida (*Settled*), é feita a verificação se o banco ao qual pertence o cartão do cliente aceita (*processor authorizes transaction?*), caso seja autorizado, assim como o AVS ou o CVV (caso sejam necessários), a transação é submetida para que seja removido o dinheiro da conta do cliente (*Submitted for settlement*) e finalmente, se for retirado o dinheiro, o processo fica concluído satisfatoriamente (*Settled*).

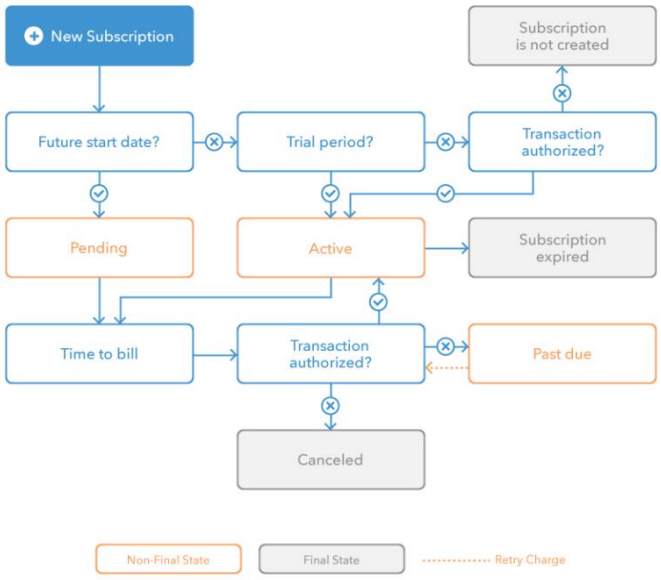


Figura 13 - Braintree: Estados da Subscrição.[33]

Ligeiramente mais complexo que as transações são as subscrições, como se pode observar na Figura 13. Sucintamente, o cliente solicita uma subscrição (*New Subscription*) e é feita a verificação, no Braintree, se o plano para o qual o cliente pretende fazer a subscrição tem período de experimentação ou não (*Trial*). Caso exista um período de experimentação, então o produto fica ativo até que acabe esse período. Findado este período (*Subscription expired*), ou não existindo o mesmo, é feito o pagamento (*Time to bill*), onde é feita a verificação se os dados introduzidos são autorizados (*Transacton authorized?*). Se não for aceite (*Past due* ou *Canceled)*, então o cliente é avisado do sucedido, caso contrário, se for aceite, é feita a subscrição e o cliente pode aceder ao produto.

Para facilitar o trabalho a quem pretende implementar o Briantree, esta plataforma de serviços de pagamento oferece a possibilidade de os utilizadores criarem uma *Sandbox*, que permite testar a plataforma antes de entrar em produção (Test Everything Braintree, 2019), o que possibilitou ao estagiário criar uma conta e fazer testes de forma a que tudo fosse desenvolvido sem ser em ambiente de produção, uma mais valia.

[33] Fonte: https://developers.braintreepayments.com/guides/recurring-billing/overview, consultado a 12/12/2019.

Importa ainda salientar que o Braintree permite criar *Webhooks*, “um sistema de notificações automatizadas que indicam que um evento ocorreu no *gateway*. Em vez de exigir que o cliente extraia as informações por meio da nossa [Brintree] API, os *webhooks* enviam informações para o destino quando ocorrem eventos importantes” (Webhooks, Overview, 2019).

Para criar um *webhook* no Braintree é solicitado, naturalmente, o URL de destino, que existia que fosse um protocolo HTTPS e como o produto estava a ser testado com recurso a um protocolo HTTP, foi criado um túnel seguro para o servidor local através do serviço ngrok[34]. Para além do URL de destino, é indicado que tipo de notificações se pretende receber como por exemplo: cancelamento, pagamentos com sucesso e sem sucesso, subscrições que deixaram de estar dentro da validade.

Neste ponto, assim que existe alguma alteração no Braintree que seja pertinente, essa informação é enviada para a API desenvolvida pelo estagiário e a informação é guardada na base de dados da API, de forma a que esteja sempre disponível essa informação sem ter de se recorrer ao Braintree sempre que se pretende saber algo sobre as alterações. Ou seja, caso seja feito um cancelamento, essa informação é enviada para a API, guardada na base de dados e o cliente deixa de ter acesso a essa subscrição.

[34] Cf. Capítulo 4, Ferramentas utilizadas, ponto 4.6 ngrok.

CAPÍTULO 4

4 FERRAMENTAS UTILIZADAS

De seguida são apresentadas, de forma sucinta, as ferramentas utilizadas pelo estagiário no decorrer do estágio, sendo mencionado o propósito de cada uma.

4.1 GitHub

A plataforma GitHub[35] foi utilizada pelo estagiário para como repositório Git do projeto que desenvolveu. A escolha desta ferramenta deve-se ao facto de o estagiário já ter trabalhado com ela e a achar muito satisfatória para o objetivo pretendido, dentro das ferramentas semelhantes que são gratuitas.

4.2 GitLab

O repositório que foi facultado ao estagiário no mês de maio encontra-se no GitLab[36], uma ferramenta que cumpre funções semelhantes ao GitHub e que o estagiário também já conhecia. Contudo, não tinha tantos conhecimentos comparativamente com o GitHub, sendo que apenas necessitou de aceder ao GitLab para consultar o código realizado pela equipa da Sentilant, de forma a compreender melhor o que estava desenvolvido do lado da Sentilant para poder fazer os *requests*.

4.3 Slack

O Slack[37] permite que os membros de uma equipa de trabalho comuniquem entre si, quer seja, por exemplo, via mensagem (*chat*), chamada ou vídeo.

No decorrer do estágio, revelou ser muito importante quando no final do desenvolvimento da API o estagiário necessitou de comunicar com a equipa de desenvolvimento da Sentilant para saber qual era o erro que estava a aparecer nos *logs* da APP da Sentilant, uma vez que não tinha acesso a estes e estava a ter um erro na inserção de viagens.

[35] Cf. o *site* oficial https://github.com/ (último acesso a 24/05/2020).
[36] Cf. o *site* oficial https://about.gitlab.com/ (último acesso a 20/05/2020).
[37] Cf. o *site* oficial https://slack.com/ (último acesso a 05/11/2019).

4.4 DB Browser for SQLite

O DB Browser for SQLite[38] é uma ferramenta *open source* que permite criar, editar e consultar base de dados SQLite e foi utilizada pelo estagiário, que já tinha conhecimento sobre esta ferramenta, nomeadamente para consultar a base de dados que estava a construir por forma a verificar se estava a ficar tudo conforme pretendido, uma vez que esta ferramenta é bastante eficaz e intuitiva.

4.5 Postman

A nível do envio de HTTP *requests* dos *endpoints*[39] à API, para ir testando aquando do desenvolvimento em *back-end*, foi utilizado o Postman[40], uma ferramenta já utilizada pela equipa da Sentilant e que não só o estagiário já conhecia por ter tido experiência a nível de trabalho, como também foi explicada na Unidade Curricular de Testes e Qualidade de Software do presente mestrado, uma mais valia, pois permitiu ao estagiário aprofundar conhecimentos.

4.6 ngrok

Para criar um túnel seguro durante o desenvolvimento para o servidor local foi utilizado o ngrok[41] uma ferramenta muito simples e intuitiva. Mais concretamente, o objetivo da sua utilização prendeu-se com o facto de o *webhook* da Braintree necessitar de aceder a um protocolo HTTPS e o que estava a ser utilizado para desenvolvimento era um protocolo HTTP. Com esta ferramenta foi possível, assim, fazer a ligação do *webhook* da Braintree e a aplicação que estava a ser desenvolvida.

4.7 Atom

Para a implementação da API foi utilizado o editor de texto gratuito Atom[42], uma ferramenta simples e intuitiva, muito utilizada quando se programa em Python/Django e que o estagiário já conhecia, uma mais valia na curva de aprendizagem.

[38] Cf. o *site* oficial https://sqlitebrowser.org/ (último acesso a 01/03/2020).
[39] Um *endpoint* é "o ponto de entrada para a API. Ele define a URL que os clientes precisam usar para invocar a API" (De, 2017, p. 62).
[40] Cf. o *site* oficial https://www.postman.com/ (último acesso a 24/05/2020).
[41] Cf. o *site* oficial https://ngrok.com/ (último acesso a 03/05/2020).
[42] Cf. o *site* oficial https://atom.io/ (último acesso a 24/03/2020).

4.8 Swagger Editor

Para gerar a documentação para a equipa de *front-end* realizar a sua parte, foi utilizada a ferramenta Swagger, mais concretamente o seu editor online, Swagger Editor[43]. De salientar que, inicialmente, foi testada a ferramenta Django REST Swagger que gera automaticamente a documentação (Django REST Swagger, s.d.), facilitando assim o trabalho. Contudo, o estagiário não ficou satisfeito com a ferramenta, uma vez que o facto de gerar automaticamente a documentação limita a forma como essa documentação é apresentada, isto é, não permite a sua customização, razão pela qual se optou por criar manualmente a documentação no Swagger Editor.

[43] Cf. o *site* oficial https://editor.swagger.io/ (último acesso a 24/05/2020).

CAPÍTULO 5

5 DESENHO DA SOLUÇÃO

Neste capítulo é feita uma análise de todas as tarefas realizadas ao longo do estágio. Inicialmente começou por ser feita a identificação dos *stakeholders* e utilizadores, no subcapítulo 5.1, para depois, no subcapítulo 5.2, serem descritos os requisitos funcionais e não funcionais da API desenvolvida pelo estagiário.

No subcapítulo 5.3, é apresentado o modelo conceptual e físico de dados, de forma a perceber como foi construída a base de dados. Sendo que nestes modelos encontram-se também os parâmetros criados de forma automática pelo Django, estando as tabelas criadas pelo estagiário com a designação "core_" seguido do nome do modelo criado no projeto.

A nível da implementação, no subcapítulo 5.4, é descrito, para cada funcionalidade, qual o objetivo, a estratégia, as premissas, os requisitos funcionais, os casos de uso, a estrutura, a implementação e também os testes unitários[44] de todas as tarefas realizadas.[45]

De salientar que, para além dos testes unitários, para verificar que as funcionalidades implementadas retornavam o conteúdo pretendido e estavam todas operacionais, aquando da implementação o conteúdo foi sendo testado com recurso ao Postman de forma a ver se tudo corria bem, sendo também assegurado o mesmo quando, no final, se providenciou o conteúdo para a equipa de *front-end* trabalhar, utilizando a ferramenta Swagger. A demonstração de que o procedimento correu conforme esperado, pode ser encontrada no Anexo B, onde estão todas[46] as funcionalidades testadas e com o resultado esperado.

No que respeita limite de *requests* (*Trhottling),* foram estabelecidos limites para os veículos, viagens e pacotes completos, as funcionalidades previstas para esta fase. Sendo que o limite de *requests* é comprado pelo cliente, tendo sido reescrita a classe *ScopedRateThrottle* que é utilizada para limitar o acesso a determinadas partes da API, uma vez que algumas partes dizem respeito às viagens e outras aos veículos. Para utilizar o *ScopedRateThrottle*, foi adicionado nos *settings* no *'DEFAULT_THROTTLE_CLASSES': ['rest_framework.throttling.ScopedRateThrottle',]* e em cada classe da API foi

[44] Pode ser consultada informação sobre os testes unitários em Python no *site* https://docs.python.org/3/library /unittest.html, consultado a 15/05/20200.

[45] Nas funcionalidades, quando é referido um *endpoint*, o domínio aparece sob a designação de *{{api_url}}*, uma vez que este, aquando da sua implementação, é diferente do final, ou seja, do domínio de produção. De salientar ainda que no Anexo B estão presentes os *requests* a todos os *endpoints* desenvolvidos, apresentado os seus resutlados.

[46] Exceto as que interagem com a aplicação da Sentilant, uma vez que essas foram removidas do relatório a pedido da empresa.

adicionado *throttle_scope = 'vehicles',* para o caso de se tratar de uma classe que diga respeito aos veículos (tendo sido feito o mesmo procedimento para as viagens), e depois é chamada a classe que foi reescrita *throttle_classes = (ScopedRateThrottle,)* de forma a verificar os requests desse utilizador.

5.1 Descrição de *stakeholders* e utilizadores

Na Tabela 7 é feita uma breve descrição das partes envolvidas nesta API, sendo indicado também as responsabilidades de cada uma.

Tabela 7 - Descrição de *stakeholders* e utilizadores.

Nome	Descrição	Responsabilidades
Developer	Tem acesso à API adquirida pelo cliente e integra-a num sistema.	Implementação da API criada pela Sentilant num produto.
Sentilant	Entidade responsável pela criação, gestão e manutenção das APIs, assim como o produto disponibilizado pelas mesmas.	*Stakeholder*.
Administrador	Responsável da Sentilant por fazer a gestão da API.	*Stakeholder.*
APP	Aplicação que executa as funcionalidades desenvolvidas pela empresa Sentilant.	Executar as funcionalidades desenvolvidas pela empresa Sentilant.
Braintree	Empresa que faz a gestão das subscrições, assim como dos pagamentos das transações e dos cartões.	*Stakeholder*.
Cliente	Entidade que se regista na loja da Sentilant e compra a API.	*Stakeholder*.
Utilizador	Tem acesso às funcionalidades autorizadas da API que foram integradas no sistema do cliente.	Utilizador final.

5.2 Requisitos

Os requisitos são “uma especificação do que deve ser implementado. São descrições de como o sistema se deve comportar, ou de uma propriedade ou atributo do sistema. Estes podem ser uma restrição no processo de desenvolvimento do sistema” (Sommerville & Sawyer, 1997, p. 4), sendo classificados, normalmente, em funcionais e não funcionais.

A nível da elaboração dos requisitos funcionais, que dizem respeito a algo que o sistema deve fazer, depois de ter sido feita a identificação dos *stakeholders* e utilizadores (*vide* subcapítulo *5.1*) foi desenhado um diagrama de casos de usos, apresentado no capítulo 5.2.1, de forma a identificar os requisitos funcionais, pois "cada caso de uso define um requisito funcional para o sistema" (Hull, Dick, & Jackson, 2005, p. 53).

Desta forma, a identificação dos requisitos funcionais é feita com recurso ao diagrama de casos de uso no subcapítulo 5.2.1, sendo estes depois descritos, individualmente, de forma detalhada, dentro de cada funcionalidade do subcapítulo 5.4, onde consta, por exemplo, a descrição sumária dos casos de uso de cada um dos requisitos funcionais.

No que respeita aos requisitos não funcionais, que se encontram no subcapítulo 5.2.2, que são restrições ao modo de como os requisitos funcionais são implementados, foram definidos os requisitos de produto e os requisitos externos. Os primeiros, os de produto, incluem os requisitos de portabilidade, fiabilidade, usabilidade, segurança, desempenho e performance. Os requisitos externos, incluem definidos os requisitos legais, éticos e de interoperabilidade.

5.2.1 Requisitos Funcionais

Como referido no subcapítulo 5, a descrição dos casos de uso, de forma pormenorizada, é feita no subcapítulo 5.4, contudo, como é importante também disponibilizar não só o diagrama de casos de uso na sua totalidade, para se ter um panorama geral do trabalho desenvolvido, como também os requisitos funcionais, estão presentes na Figura 14 sob a forma de casos de uso, pois estes últimos definem os requisitos funcionais, como referido.

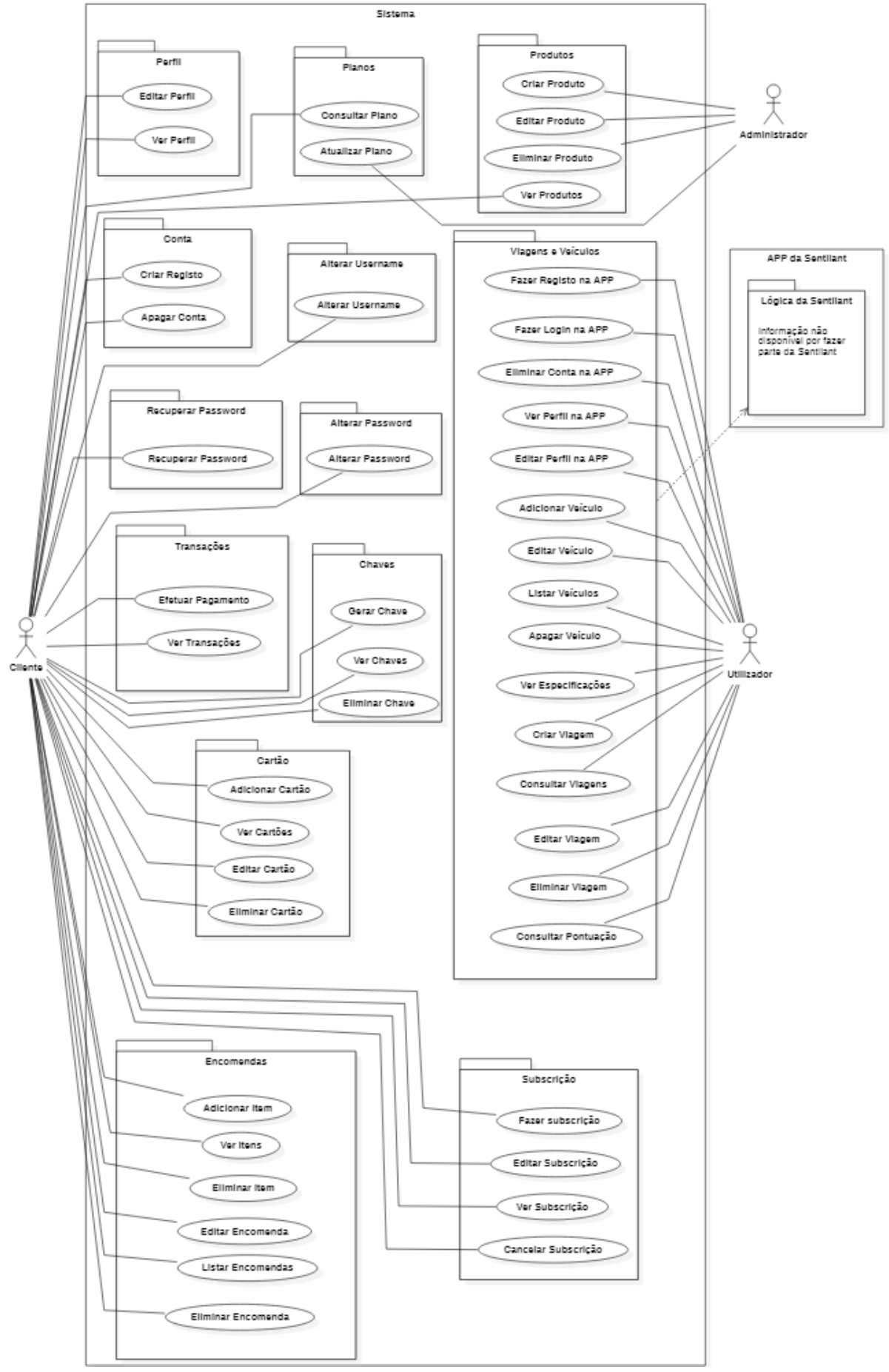


Figura 14 - Diagrama de Casos de Uso.

5.2.2 Requisitos não funcionais

Sommerville divide os requisitos não funcionais em três grupos, requisitos de produto, que especificam o comportamento do *software*, como requisitos de usabilidade, eficiência, fiabilidade e segurança, requisitos organizacionais, relacionados com restrições impostas pelo cliente e por quem desenvolve o produto, e requisitos externos, que são requisitos externos ao sistema, como requisitos éticos e legais. (Sommerville I. , 2011, p. 88)

De seguida, são apresentados os requisitos não funcionais para o presente trabalho.

- Requisitos de produto:
  - Requisitos de usabilidade: Os clientes devem conseguir utilizar sistema sem ser necessário instruções para o seu uso, devendo estas ser simples e intuitivas.
  - Requisitos de eficiência: A nível de desempenho, o sistema deve possibilitar processar um número de transações por segundo que não impeça a sua utilização de forma eficaz e eficiente, devendo assim o sistema “minimizar as comunicações entre os componentes” (Sommerville I. , 2011, p. 87).
  - Requisitos de fiabilidade: O serviço deve ter uma disponibilidade de 99%.
  - Requisitos de portabilidade: A interface do sistema deve estar disponível de forma a que seja possível a sua utilização nos *browsers*: Chrome, Firefox, Edge, Safari e Opera.
  - Requisitos de segurança: Só pode adquirir o produto quem estiver registado; apenas podem aceder às funcionalidades da APP da Sentilant, a lógica do produto, quem tiver uma chave de acesso da API; toda a informação disponibilizada é confidencial e os contratos REST devem ser feitos com recurso ao protocolo HTTPS.
- Requisitos organizacionais:
  - Requisitos de desenvolvimento: O sistema deve ser escrito na linguagem Python, utilizando a *framework* Django.
- Requisitos externos:
  - Requisitos legais: O sistema desenvolvido deve estar em conformidade com a legislação portuguesa.
  - Requisitos éticos: O Sistema desenvolvido não deve mostrar a informação privada a terceiros, assim como deve estar desenvolvido de forma a que seja aceite pela ética nas organizações e sociedade.

5.3 Modelos de dados

Neste subcapítulo é apresentado o modelo conceptual (Figura 15) e o modelo físico de dados (Figura 16), por forma a permitir uma compreensão mais clara da base de dados criada para a loja da Sentilant e que permite também o acesso ao conteúdo desenvolvido pela empresa Sentilant, que, naturalmente, não é apresentado por questões de confidencialidade.

Como se pode constar na Figura 15 e na Figura 16, existem diversas entidades presentes que são geradas de forma automática pelo Python/Django, sendo a descrição que se segue destas figuras centrada apenas nos modelos criados pelo estagiário, que nas figuras começam por “core_”, uma vez que se encontram na pasta *core* do projeto:

- core_usermanager: Permite criar um cliente, assim como um administrador.
- core_user: É onde são feitas as configurações base do utilizador, visíveis também no painel de Administração do Django.
- core_profile: Tem como objetivo ter a informação do perfil do utilizador.
- core_product: Este modelo permite armazenar a informação sobre os produtos.
- core_plan: Permite guardar a informação sobre os planos da Sentilant, de forma a que os clientes os possam visualizar sem ser feita sempre uma chamada ao Braintree.
- core_orderItem: Tem como objetivo guardar todos os itens que se encontram nas encomendas dos utilizadores, estejam estas realizadas ou não.
- core_order: Permite guardar as encomendas, realizadas ou não, pelos clientes.
- core_transaction: Possibilita que sejam guardadas todas as transações feitas pelos clientes.
- core_key: Esta tabela permite armazenar as chaves de forma a que seja possível fazer as chamadas para a app da Sentilant. Chaves estas que são criadas apenas por clientes que tenham adquirido o produto.
- core_braintree: Tem como objetivo armazenar a informação do Braintree, de forma a que sempre que o cliente necessite de alguma informação relacionada com as subscrições, não tenha de chamar sempre a plataforma Braintree, agilizando assim o processo. Importa referir que sempre que acontece algo de importante no Braintree, essa informação é adicionada esta tabela, uma vez que foi criado um *webhook*.

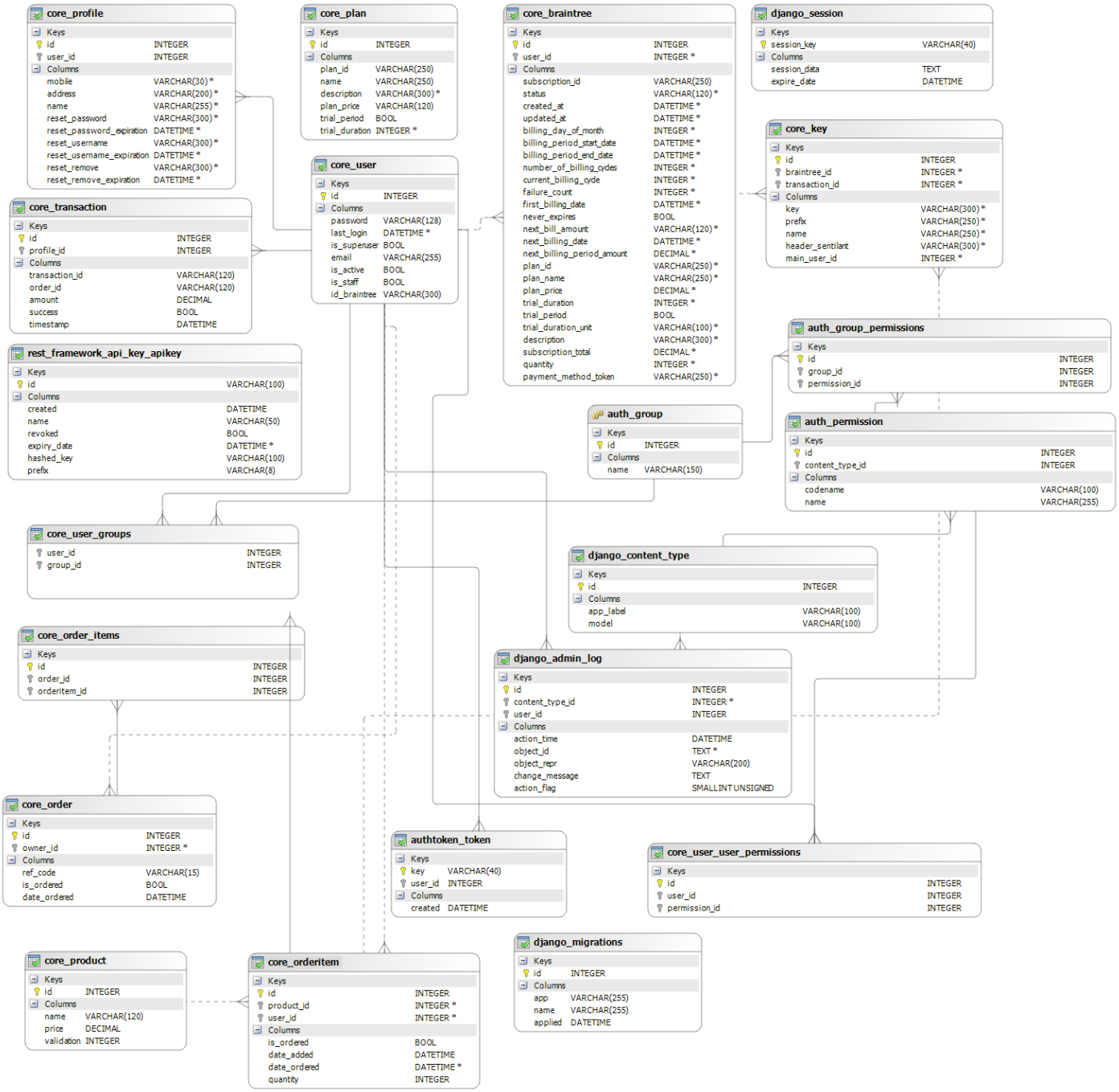


Figura 15 - Modelo Conceptual de Dados.

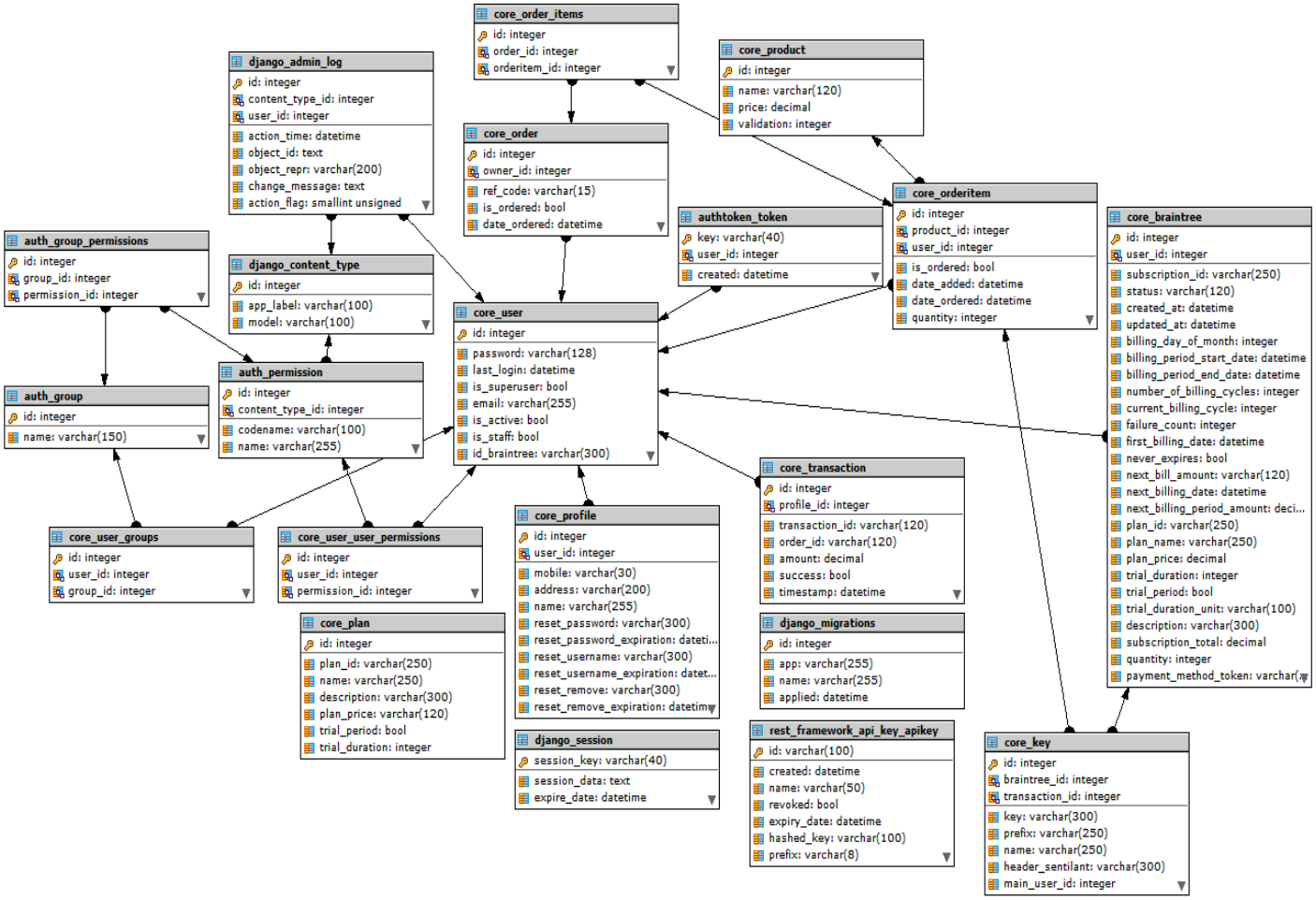


Figura 16 - Modelo físico de Dados.

5.4 Funcionalidades

Neste subcapítulo são identificadas as funcionalidades realizadas ao longo do estágio, sendo apresentado o seu objetivo, as premissas, os requisitos funcionais, os casos de uso, a sua estrutura, assim como é explicada de forma sucinta a implementação e os testes unitários.

5.4.1 Produtos

5.4.1.1 Objetivo

A tarefa ver produtos tem como objetivo não só possibilitar que os clientes visualizem os produtos da Sentilant, como também permitir que os administradores da Sentilant consigam criar, editar ou eliminar produtos sem recorrer ao painel de administração do Django, agilizando, desta forma, o processo.

5.4.1.2 Premissas

Para criar, editar e eliminar produtos:

- O administrador da Sentilant tem de estar registado.
- Será definida a forma como o administrador pode criar, editar e eliminar produtos.
- Estas funções administrativas, que só a equipa da Sentilant pode manipular, podem ser efetuadas na loja da Sentilant, entrando com conta de administrador, ou com recurso ao painel de administração do Django.

Para visualizar os produtos:

- O cliente não tem de estar registado na loja da Sentilant.
- Será definida a forma como o cliente pode visualizar e interagir com os produtos.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.1.3 Estratégia

Para criar, editar e eliminar produtos, foi definida a seguinte estratégia:

- O administrador tem de estar autenticado na loja.
- Para inserir um novo produto é necessário inserir os seguintes campos:
    - Nome.
    - Preço.
    - Validade do produto.
- Os campos passíveis de ser editados são os mesmo da criação de um novo produto, podendo ser alterados todos os campos de uma vez só ou então apenas um campo.
- Para eliminar um produto é necessário indicar o seu id.

Para a ver produtos foi concebida a seguinte estratégia:

- O cliente não tem de estar registado na loja.
- Ao visualizar os produtos, deve ser apresentada a seguinte informação:
    - Nome.
    - Preço.
    - Validade do produto.

5.4.1.4 Requisitos

Na Tabela 8, são apresentadas as tarefas relativamente aos produtos, com indicação da história de quem pode realizar essa operação.

Tabela 8 - Requisitos: Produtos.

Título	História do Administrador e Cliente
Criar Produto	Como administrador da Sentilant, quero ter a possibilidade criar um produto.
Editar Produto	Como administrador da Sentilant, quero ter a hipótese de editar um produto.
Eliminar Produto	Como administrador da Sentilant, quero ter a opção de eliminar um produto.
Ver Produtos	Como cliente, quero ter a possibilidade de visualizar os produtos da Sentilant.

5.4.1.5 Casos de uso

A Figura 17 apresenta diagrama de casos de uso no que respeita aos produtos, de forma a identificar as tarefas possíveis e os seus atores.

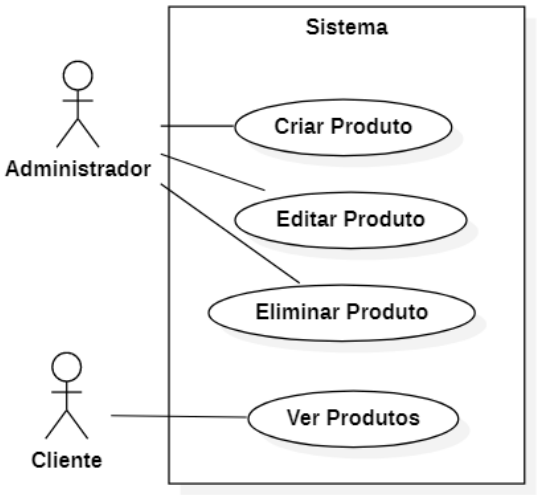


Figura 17 - DCU: Produtos.

De seguida, é apresentada a descrição sumária para cada caso de uso: criar produto, editar produto, eliminar produto e ver produtos.

- **Criar Produto**
  - **Nome**: Criar Produto.
  - **Ator**: Administrador.
  - **Objetivo**: Permitir que um administrador da Sentilant crie um novo produto.
  - **Pré-Condições**: O administrador tem de estar autenticado.
  - **Pós-Condições**: A informação sobre o produto fica registada na base de dados.
  - **Descrição**: Neste caso de uso, o sistema permite ao administrador da Sentilant adicionar um novo produto.
  - **Cenário principal**:
    1. O administrador indica que pretende criar um produto.
    2. O sistema apresenta um formulário.
    3. O administrador preenche o formulário para criação de um produto, inserindo a informação sobre o preço, validade e nome.
    4. O sistema guarda a informação na base de dados.
    5. O caso de uso termina com sucesso.
  - **Cenário alternativo:**

    3.a. Alternativamente, o administrador preenche um campo de forma errada.

    3.b. O sistema avisa que o campo está preenchido de forma errada.

    3.c. Volta para 2.

    3.c.1. O caso de uso termina sem sucesso.

    4.a. Alternativamente, o nome do produto inserido já exise.

    4.b. O sistema avisa o administrador.

    4.c. Volta para 2.

    4.c.1. O caso de uso termina sem sucesso.

- **Editar Produto**
  - **Nome**: Editar Produto.
  - **Ator**: Administrador.
  - **Objetivo**: Possibilitar que um administrador da Sentilant edite um produto.
  - **Pré-Condições**: O administrador tem de estar autenticado.

- **Pós-Condições**: A informação sobre o produto fica registada na base de dados.
- **Descrição**: O sistema, neste caso de uso, permite ao administrador da Sentilant editar um produto, sendo possível editar os campos nome, preço e validade.
- **Cenário principal**:
  1. O administrador indica que pretende editar um produto.
  2. O sistema apresenta os campos passíveis de serem alterados.
  3. O administrador altera os campos e submete a informação.
  4. O sistema guarda a informação na base de dados.
  5. O caso de uso termina com sucesso.
- **Cenário alternativo**:

  3.a. Alternativamente, o administrador preenche um campo de forma errada.

  3.b. O sistema avisa que o campo está preenchido de forma errada.

  3.c. Volta para 2.

  3.c.1. O caso de uso termina sem sucesso.

➢ **Eliminar Produto**

- **Nome**: Eliminar Produto.
- **Ator**: Administrador.
- **Objetivo**: Permitir que um administrador da Sentilant elimine um produto.
- **Pré-Condições**: O administrador tem de estar autenticado.
- **Pós-Condições**: O sistema elimina o produto.
- **Descrição**: Neste caso de uso, o sistema possibilita a um administrador da Sentilant eliminar um produto da loja.
- **Cenário principal**:
  1. O administrador indica que pretende eliminar um produto.
  2. O sistema elimina o produto.
  3. O caso de uso termina com sucesso.

- **Ver Produtos**
  - **Nome**: Ver Produtos.
  - **Ator**: Cliente.
  - **Objetivo**: Visualizar os produtos disponibilizados pela Sentilant.
  - **Pré-Condições**: Não tem.
  - **Pós-Condições**: O cliente vê o leque de produtos disponibilizados pela Sentilant.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente ver os produtos disponibilizados pela Sentilant.
  - **Cenário principal**:
    1. O cliente indica que pretende consultar os produtos.
    2. O sistema mostra os produtos.
    3. O caso de uso termina com sucesso.

5.4.1.6 Estrutura

Para criar, editar e eliminar produtos foram criados dois *endpoints*. O *endpoint* {{api_url}/api/admins/products/ que pode ser acedido recorrendo ao método POST e que permite adicionar um novo produto e outro *endpoint,* {{api_url}/api/admins/products/{id}, que pode ser chamado através dos métodos PUT, PATCH e DELETE.

Os métodos PUT e PATCH permitem a edição dos produtos, sendo apresentados os parâmetros na Tabela 9 passíveis de serem alterados – de salientar que poderia ser feito apenas o método PATCH, que permite a edição de um ou mais campos da referida tabela, ao contrário do método PUT que exige a alteração de todos os campos, contudo foi decidido fazer os dois métodos para que a equipa de *front-end* possa fazer a decisão final quando tiver oportunidade. Quanto ao método DELETE, este permite que se elimine o produto indicado através do id.

Tabela 9 - Parâmetros dos *endpoints* Criar e Editar Produtos.

Chave	Tipo
Name	string
Price	decimal
validation	int

Para possibilitar a visualização dos produtos são apresentados dois *endpoints* recorrendo ambos ao método GET. O *endpoint* {{api_url}/api/products/ apresenta a

lista completa de produtos da Sentilant, com a informação destes todos, e o *endpoint* {{api_url}/api/products/{id} apresenta a informação apenas do produto selecionado.

Na Tabela 10 são apresentados os parâmetros dos *endpoints* que são enviados em formato JSON para a equipa de *front-end*, de forma a que estes os possam implementar a nível da interface.

Tabela 10 - Parâmetros dos *endpoints* Listar Produtos.

Chave	Tipo
id	int
name	string
price	decimal
validation	int

5.4.1.7 Implementação

Para ver os produtos, foi criada a classe *ProductViewSet*. Esta classe permite que os utilizadores visualizem os produtos disponíveis na sua totalidade sem necessitarem de estar registados, uma vez que não foi imposta nenhuma restrição a esse nível, conforme previsto.

Para que os utilizadores visualizem o conteúdo de todos os produtos, uma vez que não é necessário nenhum tipo de configuração particular, apenas foi necessário na classe indicar nos parâmetros da classe que se está a trabalhar com *ListModelMixin*, que, por defeito, cria um método *list* que permite listar conteúdos quando se tem um *request* GET sem parâmetros opcionais, e indicar o conteúdo a mostrar dentro da classe: o *queryset*, que recebe o conteúdo dos produtos, queryset = Product.objects.all(), e a classe de serialização, serializer_class = serializers.ProductSerializer.

Para que os utilizadores tenham a possibilidade de ver apenas o conteúdo de um produto específico, foi inserido nos parâmetros da classe o *RetrieveModelMixin*, e depois foi reescrito, nesta classe, o método que vem por defeito para o *RetrieveModelMixin* que é o *retrieve* e que possibilita que os utilizadores visualizem apenas um determinado produto escolhido, uma vez que vai fazer a procura pelos produtos existentes na base de dados da API pelo id do produto submetido: Product.objects.filter(id=kwargs.get('id')), sendo depois enviado como resposta o conteúdo do produto encontrado.

No que diz respeito às tarefas do administrador, de criar, editar e eliminar um produto, foi criada a classe *AdminProductViewSet*. Esta, só permite que utilizadores autenticados e administradores façam alterações: permission_classes = (IsAuthenticated, IsAdminUser,).

Foram ainda reescritos quatro métodos, o método *create*, que permite criar produtos, mediante a informação que é enviada: preço, validação e nome. O administrador submete a informação e depois esta classe vai verificar se a informação inserida é passível de ser inserida na base de dados, verificando alguns parâmetros como por exemplo se o preço é maior que zero, se é decimal, se a data de validação (número de meses) é maior que um e inteiro, ou se o nome do produto já existe na base de dados. Caso se verifique o produto a ser inserido não cumpre os requisitos, então o administrador é avisado, caso contrário o produto é inserido e o utilizador recebe a informação que foi criado o produto.

Para eliminar um produto, foi reescrito o método *destroy*, onde vai procurar, mediante o id do produto que recebe, queryset = Product.objects.filter(id=kwargs.get('id')), se este existe ou não. Se existir é eliminado self.perform_destroy(queryset).

Para editar um produto, foi reescrito o método *update* e *partial_update*, o primeiro para o caso do *request* ser um PUT e o segundo ser um PATCH. Nestes dois métodos são analisados os campos inseridos pelo utilizador de forma a verificar se estão em condições de serem alterados. Caso esteja, é feita a sua alteração, caso contrário, o administrador é avisado do problema.

5.4.1.8 Testes Unitários

- Criar Produto:
  - test_add_product: é verificado se o que retorna da adição de um produto é o pretendido, retornando um *status* 201 caso tenha corrido conforme previsto.
  - test_add_product_price_string: onde o preço do produto é submetido sob a forma de string, retornando um *status* 400.
  - test_add_product_validation_string: caso a data de validade do produto seja submetida como string, então é retornado *status* 400.
  - test_add_product_name_int: Caso o nome submetido seja um inteiro e não uma string, como suposto, é retornado um *status* 400.
  - test_add_product_all_empty: caso todos os campos submetidos para adicionar um produto estejam vazios, é retornado um *status* 400.
  - test_add_product_price_empty: caso o campo do preço esteja vazio, então retorna um *status* 400.
  - test_add_product_validation_empty: se a validação for submetida nem nenhum conteúdo, é retornado um *status* 400.
  - test_add_product_name_empty: sendo o campo nome obrigatório, se este for vazio é retornado um *status* 400.

- test_add_product_no_admin: Se for tentado adicionar um produto sem ser administrador da Sentilant, será retornado um *status* 403.

- Editar Produto:
  - test_patch_product: se todos os campos preenchidos estiverem em conformidade com o solicitado, é retornado um *status* 201 e é feita a verificação se a data foi introduzida.
  - test_patch_product_incomplete: uma vez que se trata de um PATCH, se o administrador não introduzir um dos campos, como o nome, é retornado um *status* 200.
  - test_patch_product_empty: se não for submetido nenhum conteúdo nos campos, é retornado um *status* 400.
  - test_patch_product_price_string: é retornado um *status* 400 se o administrador introduzir uma *string* no campo do preço.
  - test_patch_product_validation_string: se no campo da validação for introduzida uma string, é retornado um *status* 400.
  - test_patch_product_name_int: é retornado um *status* 400 se o administrador introduzir um inteiro no campo do nome.
  - test_put_product: se o administrador adicionar um produto em conformidade com os tipos de dados solicitados, é feita a verificação se a data que é retornada tem aquela e é retornado o *status* 201.
  - test_put_product_incomplete: se algum dos campos não for preenchido, é retornado um *status* 400.
  - test_put_product_empty: caso um dos campos for submetido sem conteúdo, será retornado um *status* 400.
  - test_put_product_price_string: é retornado um *status* 400 se o administrador introduzir uma *string* no campo do preço.
  - test_put_product_validation_string: caso o administrador insira uma *string* no campo da validação, é retornado um *status* 400.
  - test_put_product_name_int: é retornado um *status* 400 se o administrador introduzir um inteiro no campo do nome.

- Eliminar Produto
  - test_delete_product: é feita a verificação se aquando da eliminação de um produto se é retornado um *status 204*, assim como se o produto é, efetivamente, eliminado.
  - test_delete_product_wrong_id: caso seja enviado um id errado, que não exista, é retornado um *status* 400.

- test_delete_product_string_id: é retornado um *status* 400 se for submetido uma *string* no campo do id.

- Ver Produtos: Sendo estas operações publicas, isto é, que não necessitam de autenticação, o seu teste foi criado dentro da classe *PublicCartApiTests(TestCase)*:
  - test_list_products: verifica se a listagem de produtos corre conforme previsto.
  - test_list_product_no_products: quando não existem produtos inseridos, verifica se a listagem de produtos apresenta o resultado pretendido.
  - test_retrieve_product: permite verifica aquando da solicitação da informação de um produto específico, se essa informação é apresentada.
  - test_retrieve_product_wrong_id: caso seja indicado um id errado, confirma que é retornado um HTTP 400.
  - test_retrieve_product_string_id: se o id enviado for uma string, então terá de retornar um HTTP 400.

5.4.2 Planos

5.4.2.1 Objetivo

Esta tarefa tem como objetivo possibilitar que o cliente da Sentilant possa consultar os planos disponibilizados, assim como os administradores da Sentilant tenham a hipótese de atualizar a lista de planos, fazendo assim a sincronização com os planos definidos na plataforma Braintree.

5.4.2.2 Premissas

- Para visualizar os planos, o cliente não tem de estar registado na loja da Sentilant.
- Será definida a forma como o cliente poderá visualizar os planos.
- Para o administrador atualizar a lista de planos, este tem de estar autenticado na loja.
- Será definido a forma como a função administrativa de atualizar a lista de planos será realizada. As restantes funções administrativas são realizadas na plataforma Braintree e no painel de administração do Django.

5.4.2.3 Estratégia

Foi definida a seguinte estratégia para atualizar os planos:

- O administrador tem de estar autenticado na loja.
- Apenas é necessário que o administrador corra o *endpoint* definido para este efeito.

Para visualizar os planos, a estratégia foi a seguinte:

- O cliente não tem de estar registado na loja.
- Ao visualizar os planos, deve ser apresentada a seguinte informação para cada plano:
    - Nome.
    - Descrição.
    - Preço.
    - Período *Trial,* caso exista.
    - Duração do *Trial*, caso tenha *Trial*.

5.4.2.4 Requisitos

Na Tabela 11, são apresentadas as tarefas relativas aos planos, com indicação da história de quem pode realizar essa operação.

Tabela 11 - Requisitos: Planos.

Título	História do Administrador
Consultar Plano	Como administrador da Sentilant, quero ter a possibilidade criar um produto.
Atualizar Plano	Como administrador da Sentilant, quero ter a hipótese de atualizar a lista de planos.

5.4.2.5 Casos de uso

Na Figura 18 é apresentado o diagrama de casos de uso para os planos, de forma a compreender as tarefas possíveis e quais os seus atores.

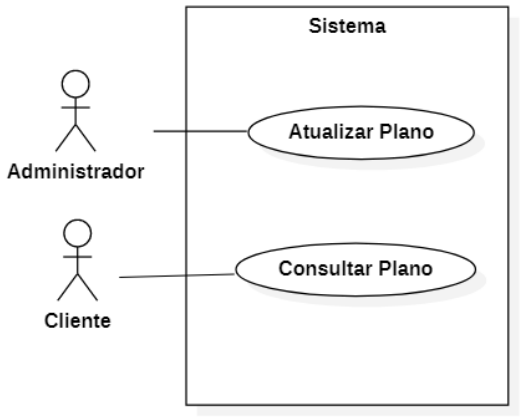


Figura 18 - DCU: Planos.

De seguida, é apresentada a descrição sumária para cada caso de uso: atualizar plano e consultar plano.

- **<u>Atualizar Plano</u>**
  - **Nome**: Atualizar Plano.
  - **Ator**: Administrador.
  - **Objetivo**: Permitir que um administrador da Sentilant atualize os planos que são mostrados aos clientes ao fazer uma sincronização com os planos criados na plataforma Braintree.
  - **Pré-Condições**: O administrador tem de estar autenticado.
  - **Pós-Condições**: A informação sobre os planos fica guardada na base de dados.
  - **Descrição**: Neste caso de uso, o sistema permite ao administrador da Sentilant atualizar a lista de planos que são disponibilizados ao cliente.
  - **Cenário principal**:
    1. O administrador corre o *endpoint* destinado a fazer esta atualização.
    2. O sistema atualiza a base de dados com a informação proveniente do Braintree sobre os planos.
    3. O caso de uso termina com sucesso.
  - **Cenário alternativo**:

    2.a. Alternativamente, ocorre um problema ao aceder ao Braintree.

    2.b. O sistema avisa que não foi possível fazer a atualização.

    2.b.1. O caso de uso termina sem sucesso.

- **Consultar Plano**
  - **Nome**: Consultar Plano.
  - **Ator**: Cliente.
  - **Objetivo**: Visualizar os planos disponibilizados pela Sentilant.
  - **Pré-Condições**: Não tem.
  - **Pós-Condições**: O cliente vê os planos disponibilizados pela Sentilant.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente ver os planos disponibilizados pela Sentilant.
  - **Cenário principal**:
    1. O cliente indica que pretende consultar os planos.
    2. O sistema mostra os planos.
    3. O caso de uso termina com sucesso.

5.4.2.6 Estrutura

Para atualizar os planos foi criado o *endpoint* {{api_url}}/api/admins/plans/ que pode ser acedido recorrendo ao método POST.

Para consultar o plano, foram criados dois *endpoints*. Um que permite visualizar a lista de todos os planos da Sentilant com a informação dos mesmos: {{api_url}/api/plans/ e outro que permite visualizar a informação de apenas um produto: {{api_url}/api/plans/{id}, sendo que ambos são acedidos com o método GET.

Na Tabela 12 são apresentados os parâmetros dos *endpoints* para consultar um plano que são enviados em formato JSON para a equipa de *front-end*, de forma a que estes os possam implementar a nível da interface.

Tabela 12 - Parâmetros dos *endpoints* Consultar Plano.

Chave	Tipo
id	int
plan_id	string
name	string
plan_price	string
trial_period	Boolean
trial_duration	int

5.4.2.7 Implementação

Para que o utilizador possa visualizar os planos e os seus detalhes, foi criada a classe *PlanViewSet*, que tem o parâmetro *ListModelMixin*, que permite ver o conteúdo dos planos, depois de se definir o *queryset* na classe e indicar que este recebe todos os planos que se encontram na base de dados: queryset = Plan.objects.all(), assim como a classe de serialização: serializer_class = serializers.PlanSerializer. Posto isto, não é necessário adicionar nenhum método, uma vez que internamente ao adicionarmos o parâmetro *ListModelMixin* o Django já tem um método *list* que faz essa seleção dos produtos.

No que respeita ao *request* GET que solicita a visualização de apenas um produto, foi inserido o parâmetro *RetrieveModelMixin* na classe e reescrito o método *retrieve* de forma a mostrar o pretendido. É feita a procura pelo id do produto, Plan.objects.filter(id=kwargs.get('id')), e depois é mostrado ao utilizador os dados do plano.

Relativamente às tarefas do administrador no que respeita aos planos e sua atualização, foi criada a classe *AdminPlanViewSet* que tem também como parâmetro o *CreateModelMixin*, de modo a que seja possível receber o *request* para criar os planos. Neste caso, optou-se por um método POST, sendo o procedimento apagar os planos existentes na base de dados local relativa aos planos, *Plan.objects.all().delete()*, e depois preencher esse modelo com os planos que se encontram na plataforma Braintree, ficando desta forma atualizada.

5.4.2.8 Testes Unitários

- Atualizar plano:
    - test_fill_plan: verificar se é feita a atualização da lista de planos, remetendo um *status* 201.
    - test_add_plan_no_admin: foi verificado que caso se tente adicionar um plano sem ser administrador, é retornado um *status* 403.
- Consultar plano:
    - test_list_plans: é feita a verificação se a lista de planos disponível é apresentada, retornando um *status* 200 e verificado se o conteúdo apresentado é o suposto.
    - test_list_plan: verifica aquando da indicação de um plano específico se é mostrado o conteúdo desse plano, retornando um *status* 200 também.
    - test_list_plan_wrong_id: caso seja submetido um id errado, que não existe, é detornado um *status* 400.

5.4.3 Conta

5.4.3.1 Objetivo

A nível da conta, foram criadas duas funcionalidades. Uma para que o cliente possa fazer registo na loja da Sentilant e outra para que este a possa eliminar.

5.4.3.2 Premissas

- Será definida a forma como o cliente pode fazer o registo e eliminar a conta.
- As funções administrativas são realizadas através do painel de administração do Django.
- Para criar uma conta na loja da Sentilant:
    - O cliente não tem, naturalmente, de estar registado na loja da Sentilant.
- Para eliminar a conta de cliente na loja da Sentilant:
    - O cliente para realizar esta tarefa tem de estar autenticado na loja da Sentilant.

5.4.3.3 Estratégia

Na tarefa de criar registo foi definida a seguinte estratégia:

- O cliente não tem de estar registado na loja.
- Devem existir três campos para preenchimento: um para *email*, que será o *username* do cliente, outro para a *password* e, por fim, um campo para o nome.
- O *email* (*username*), terá de ser um *email* válido.
- A *password* terá de cumprir as políticas de *password* definidas pela Sentilant.

Para que o cliente possa remover a sua conta foi definida a seguinte estratégia:

- O cliente tem de estar autenticado no sistema para fazer a solicitação de remoção da conta.
- Devido às consequências desta tarefa, a eliminação a conta do cliente, esta tarefa envolve dois passos:
    - Num primeiro passo, o cliente solicita a remoção da conta, inserindo o seu *username* e a sua *password* e a repetição desta última.
    - De seguida, o cliente recebe um *email* com uma hiperligação, que tem uma validade limitada, para que possa apagar a sua conta, inserindo o *username* e depois a sua *password* duas vezes, por questões de segurança.

5.4.3.4 Requisitos

Na Tabela 13, é apresentada a tarefa que diz respeito ao registo e eliminação da conta, com indicação da história de quem pode realizar essas operações.

Tabela 13 - Requisitos: Conta.

Título	História do Cliente
Criar Registo	Como futuro cliente da loja da Sentilant, quero ter a possibilidade de fazer o registo na loja.
Apagar Conta	Como cliente da loja da Sentilant, quero ter a possibilidade de apagar a minha conta.

5.4.3.5 Casos de uso

No que respeita aos casos de uso, na Figura 19 é apresentado o diagrama para o registo e eliminação de conta, onde consta também o ator que interage com as funcionalidades.

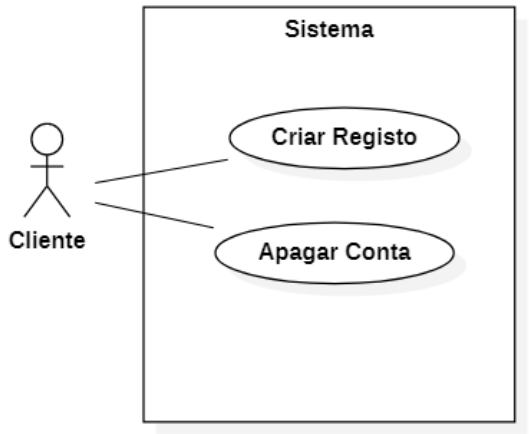


Figura 19 - DCU: Conta.

De seguida, é apresentada a descrição sumária para os dois casos de uso: criar registo e apagar conta.

- **<u>Criar Registo</u>**
  - **Nome**: Criar Registo.
  - **Ator**: Cliente.
  - **Objetivo**: Criar uma conta de cliente na loja Sentilant.
  - **Pré-Condições**: O cliente tem de ter um *email*.
  - **Pós-Condições**: O cliente fica registado, podendo aceder a esta loja da Sentilant.

- **Descrição**: Neste caso de uso, o sistema permite ao cliente fazer o registo na loja da Sentilant para que tenha acesso a esta. Para fazer este registo, o cliente tem de inserir a seguinte informação: *email*, que será o seu *username*, *password* e nome.
- **Cenário principal**:
  1. O sistema disponibiliza uma opção para o cliente se registar.
  2. O cliente preenche os campos *email* (*username)*, *password* e nome.
  3. O sistema faz o registo e guarda a informação na base de dados.
  4. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  2.a. Alternativamente, um dos campos preenchidos está incorreto: ou o *email* não cumpre os requisitos para ser um *email* válido, ou *password* não cumpre as políticas de *password* definidas pela Sentilant.

  2.b. O sistema avisa o cliente do erro.

  2.c. Volta para 1.

  2.c.1. O caso de uso termina sem sucesso.

➢ **<u>Apagar Conta</u>**

- **Nome**: Apagar Conta.
- **Ator**: Cliente.
- **Objetivo**: Possibilitar que o cliente elimine a sua conta.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: A conta de cliente é eliminada, impossibilitando que este entre na mesma.
- **Descrição**: Neste caso de uso, o cliente, inicialmente, faz a solicitação de eliminação da sua conta, recebendo, posteriormente, um *email* com uma hiperligação que tem uma validade limitada e que permite que este faça, efetivamente a eliminação da sua conta de cliente.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de solicitar a eliminação da conta de cliente.
  2. O cliente submete a informação solicitada, *username*, *password* e repetição desta última.

3. O sistema envia um *email* ao cliente que tem uma hiperligação para o cliente prosseguir com o processo de remoção de conta.
4. O cliente acede ao endereço que recebeu via *email*.
5. O cliente, no endereço que recebeu, preenche os campos *username*, *password* e repetir *password* e submete a informação.
6. A conta do cliente é eliminada.
7. O caso de uso termina com sucesso.

- **Cenário Alternativo**:

  2.a. Alternativamente, a informação inserida está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c.1 Volta a 1.

  4.a. O tempo definido para aceder a esta hiperligação passou de prazo.

  5.a. Alternativamente, o preenchimento dos campos está incorreto.

  5.a.1 Volta para 5.

  2.c.1./ 4.a.1./ 5.a.1. O caso de uso termina sem sucesso.

5.4.3.6 Estrutura

É possível realizar a tarefa de criar um registo através do método POST, tendo sido criado para esse efeito o *endpoint* {{api_url}}/api/users/creates/ que tem os parâmetros presentes na Tabela 14.

Tabela 14 - Parâmetros do *endpoint* Criar Registo.

Chave	Tipo
email	string
password	string
name	string

A tarefa de apagar a conta de cliente envolve dois *endpoints*, ambos com o método POST. O *endpoint* {{api_url}}/api/users/removes/, onde o cliente pode fazer a solicitação da remoção da sua conta, e o *endpoint* {{api_url}}/api/users/removes/{id}, que possibilita, efetivamente, que o cliente elimine a sua conta.

Na Tabela 15 encontram-se os parâmetros destes dois *endpoints.*

Tabela 15 - Parâmetros dos *endpoints* solicitar eliminar conta e apagar conta.

Chave	Tipo
current_username	string
password	string
password_repeated	string

5.4.3.7 Implementação

A nível da implementação da criação da conta de cliente, que recebe um *resquest* POST, foi utilizada a funcionalidade disponibilizada pelo Django, sendo apenas necessário fazer o *import* do *get_user_model*, *from django.contrib.auth import get_user_model*, e na classe *CreateUserView (generics.CreateAPIView),* recorrendo ao serializer_class = UserSerializer. Na classe *UserSerializer*, foi criado o método *create,* onde é feita a criação do utilizador: user = get_user_model().objects.create_user(**validated_data) e depois gravar a operação: user.save().

Os parâmetros do "**validated_data", dizem respeito aos parâmetros que são recebidos e que são necessários para criar a conta, nomeadamente, o *email* e a *password.*

Para que o cliente possa apagar a sua conta, que é feito através do método POST, foi criada a classe *RemoveAccountViewSet*, que tem um parâmetro *CreateModelMixin* para que seja possível receber o método POST. De seguida, foi indicado que para que se tenha acesso é necessário que o cliente tenha um token, *authentication_classes = (TokenAuthentication,)*, e também que esteja autenticado, *permission_classes = (IsAuthenticated,)*, tendo sido, posteriormente, reescrito o método *create,* de forma a ser possível apagar a conta de utilizador com configurações personalizadas. Este método recebe os parâmetros inseridos pelo utilizador e confirma se os mesmo estão corretos. Caso estejam, não elimina logo a conta de utilizador, é enviado um *email* ao cliente com uma hiperligação para que o cliente possa prosseguir com a sua operação. Isto é feito por questões de segurança, uma vez que se trata de uma operação muito sensível.

Nessa hiperligação que o utilizador recebe via *email*, está incluída uma *hash* que é gerada e tem um tempo predefinido pela Sentilant, ou seja, passado esse tempo a hiperligação deixa de ter validada. De salientar ainda que a hash criada é única na base de dados, pois aquando da sua criação é feita essa verificação.

Quando o cliente recebe o *email* e entra na hiperligação, via método POST, é então chamado o método criado *RemoveAccount*, que tem uma notação @api_view(['POST']) e recebe um *request* e o *id*, que é a *hash*. De seguida, faz a verificação para confirmar que a *hash* ainda está dentro da validade e se estiver, então vai verificar se os campos introduzidos pelo utilizador para a confirmação da

eliminação da *password* estão corretos. Estando tudo conforme previsto, então a conta é eliminada.

5.4.3.8 Testes Unitários

- Criar registo:
    - test_create_user: foi testar a criação de um utilizador com a inserção dos parâmetros corretos, verificando que tudo corria conforme suposto e que a password não aparecia no resultado também.
    - test_create_user_email_caps: Caso o utilizador preencha o campo de email com caps, então foi verificado, corretamente, que a conta foi adicionada, retornando um *status* 201, e que a password não consta no resultado mostrado.
    - test_user_exists: caso se tente criar uma conta com um *username* que já existe, a conta não é criada e é retornado um *status* 400.
    - test_creat_user_with_no_password: Se se tentar criar uma conta e não submeter a password, então irá receber um *status* 400, sendo feita também a verificação que a conta não é criada.
    - test_create_user_with_short_password: é verificado que não é criada uma conta quando a password não tem o número mínimo de caracteres, recebendo o *status 400.* Sendo feita também a verificação que não foi criada a conta.
    - test_create_user_with_wrong_email: Caso se introduza o *email* (*username*) inválido, que não tenha, por exemplo um "@", é verificado que é retornado um *status* 400 e que a conta não foi criada.
    - test_create_user_email_limit_plus1: caso o utilizador tente criar uma conta onde passe o limite máximo de caracteres definidos pela empresa para o email, então a conta não é criada e é retornado um *status* 400.
    - test_create_user_with_no_email: Caso não se introduza o *email* (*username*), então é verificado que é retornado um *status* 400 e que a conta não foi criada.
    - test_create_user_all_empty: foi verificado que é retornado um *status* 400 caso os campos para a criação de conta estejam todos vazios. Foi feita a verificação que caso aconteça este procedimento, que a conta não é criada.
- Apagar conta:
    - test_remove_user: foi testado que é retornado um *status* 200 quando o cliente solicita a eliminação da conta, preenchendo todos os dados corretos.

- test_remove_user_wrong_user: caso o cliente insira o seu *username* incorreto é retornado um *status* 400, não sendo feita a solicitação.
- test_remove_user_empty_fields: se na solicitação foram enviados vários campos vazios, que têm de ser preenchidos, isto é, obrigatórios, então é retornado um *status* 400.
- test_remove_user_empty_pass_solicitaiton: se o campo da *password* for vazio, foi testado que é retornado um *status* 400.
- test_remove_user_empty_repeated: caso o campo da de repetição da *password*, aquando da solicitação para apagar a conta, for vazio, foi testado que é retornado um *status* 400.
- test_remove_user_wrong_pass: se o cliente inserir a *password* errada, não é feita a solicitação e é retornado um *status* 400.
- test_remove_user_wrong_repeat: foi testado também a possibilidade de o cliente inserir a repetição da *password* errada. Caso aconteça, não é feita a solicitação e é retornado um *status* 400.
- test_remove_user_pass_caps: se o cliente introduzir a *password* em letras maiúsculas, não é feita a solicitação e é retornado um *status* 400.
- test_remove_user_repeat_caps: na eventualidade de o cliente introduzir a repetição da *password* em letras maiúsculas, não é feita a solicitação retornado um *status* 400.
- test_remove_user_wrong_pass_and_repeated: se a *password* e a repetição da *password* estiverem mal, não é feita a solicitação da eliminação da conta e é retornado um *status* 400.
- test_remove_user_hash: é feita a verificação que se o cliente introduzir a *hash* correta dentro da sua validade e os campos solicitados corretos, que é feita a eliminação da conta, sendo retornado o *status* 200.
- test_remove_user_hash_out_of_date: foi feita a verificação que se o cliente introduzir uma *hash* para eliminar a sua conta que já não esteja dentro da validade, que a sua conta não é eliminada e é retornado um *status 400*.
- test_remove_user_wrong_hash: se for introduzida uma *hash* errada, é apresentado um *status* 400.
- test_remove_user_wrong_user: se a *hash* para eliminação da conta for a correta, mas o cliente introduzir mal o seu *username*, não é feita a eliminação e é retornado um *status* 400.
- test_remove_user_empty_pass_and_empty_repetead: aquando da confirmação da eliminação da conta se os campos da *password* e

repetição da *password* forem vazios, não é feita a eliminação e retornado um *status* 400.

- test_remove_user_empty_repeated: se na confirmação da eliminação conta a repetição da *password* for enviada sem ser preenchida, não é eliminada a conta e é retornado um *status* 400.
- test_remove_user_empty_pass: se o campo da *password* for vazio, não é feita a eliminação da conta, sendo retornado um *status* 400.
- test_remove_user_pass_caps: caso a *password* seja submetida com letras maiúsculas, não é eliminada a conta, sendo o *status* 400.
- test_remove_user_repeat_caps: foi testado o caso da repetição da *password* ser submetida com letras maiúsculas. Caso aconteça, a conta não é eliminada, sendo o *status* 400.
- test_remove_user_pass_repeat_caps: se a *password* e a repetição da *password* forem submetidas com letras maiúsculas, não é eliminada a conta, sendo o *status* 400.

5.4.4 Perfil

5.4.4.1 Objetivo

A tarefa perfil tem como objetivo possibilitar que os clientes editem e visualizem o seu perfil da loja Sentilant.

5.4.4.2 Premissas

- O cliente tem de autenticado na loja da Sentilant.
- Será definida a forma como o cliente pode visualizar e editar o perfil.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.4.3 Estratégia

Na tarefa de editar e visualizar o perfil foi definida a seguinte estratégia:

- O cliente tem de estar autenticado na loja.
- Existem três campos passíveis de serem editados e visualizados no perfil são:
  - Nome.
  - Número de telemóvel.
  - Morada

5.4.4.4 Requisitos

Na Tabela 16, são identificadas as tarefas relativas ao perfil, com indicação da história de quem pode realizar essas operações.

Tabela 16 - Requisitos: Perfil.

Título	História do Cliente
Editar Perfil	Como cliente da loja da Sentilant, quero ter a possibilidade de editar o meu perfil.
Ver Perfil	Como cliente da loja da Sentilant, quero ter a hipótese de visualizar meu perfil.

5.4.4.5 Casos de uso

Aa Figura 20 apresenta o diagrama de casos de uso para o perfil, onde consta as funcionalidades deste tópico, assim como o ator que as realiza.

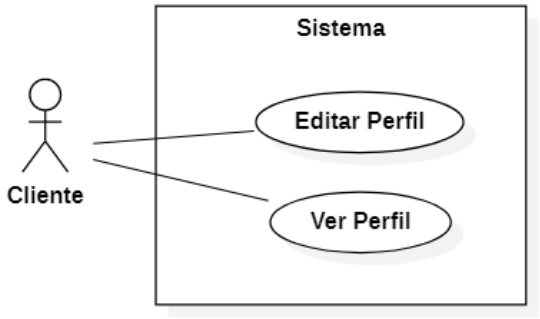


Figura 20 - DCU: Perfil.

De seguida, é apresentada a descrição sumária para os dois casos de uso: editar perfil e ver perfil.

- **<u>Editar Perfil</u>**
  - **Nome**: Editar Perfil.
  - **Ator**: Cliente.
  - **Objetivo**: Permite ao cliente editar o seu perfil da loja Sentilant.
  - **Pré-Condições**: O cliente tem de estar autenticado na loja da Sentilant.
  - **Pós-Condições**: As alterações ficam registadas na base de dados.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente editar o seu perfil da loja da Sentilant, sendo possível alterar os campos nome, morada e telemóvel.

- **Cenário principal**:
  1. O cliente seleciona a opção de alterar o perfil.
  2. O sistema apresenta o perfil com os campos passíveis de serem editados.
  3. O cliente altera os dados e submete a informação.
  4. A informação fica registada na base de dados.
  5. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  3.a. Alternativamente, um dos campos preenchidos está incorreto, não corresponde aos critérios definidos pela Sentilant.

  3.b. O sistema avisa o cliente do erro.

  3.c. Volta para 1.

  3.c.1. O caso de uso termina sem sucesso.

➢ **Ver Perfil**

- **Nome**: Ver Perfil.
- **Ator**: Cliente.
- **Objetivo**: Permite que o cliente visualize o seu perfil da loja Sentilant.
- **Pré-Condições**: O cliente tem de estar autenticado na loja Sentilant.
- **Pós-Condições**: O cliente vê o seu perfil da loja Sentilant.
- **Descrição**: Neste caso de uso, o sistema permite ao cliente ver o seu perfil da loja Sentilant.
- **Cenário principal**:
  1. O cliente indica que pretende consultar o seu perfil.
  2. O sistema mostra o perfil do cliente.
  3. O caso de uso termina com sucesso.

5.4.4.6 Estrutura

Para esta tarefa foi criado o *endpoint* {{api_url}}/api/users/profiles/, que pode ser acedido recorrendo ao método GET para ver o perfil e para editar o perfil com o método PUT, para quando se faz alterações dos campos todos, e com o método PATCH, para quando se altera pelo menos um dos campos. Os parâmetros presentes na Tabela 17 dizem respeito a este *endpoint*.

Tabela 17 - Parâmetros do *endpoint* Perfil.

Chave	Tipo
name	string
address	string
mobile	string

5.4.4.7 Implementação

A nível da implementação, no que respeita às funcionalidades do perfil, edição e visualização por parte dos clientes, foi criado uma classe, *ManageProfile* com o parâmetro *generics.RetrieveUpdateAPIView*, que permite a edição e visualização do perfil.

Como para que seja possível aos clientes fazerem as operações permitidas nestas funcionalidades têm de ter um token e estar autenticados, foi adicionada, nesta classe, esse requisito: *authentication_classes = (authentication.TokenAuthentication,)* e *permission_classes = (permissions.IsAuthenticated,)*, respetivamente.

No request com o método GET para visualizar o perfil, é feito um filtro pelo user, Profile.objects.filter(user=self.request.user), e mostrado o perfil deste. Isto feito no método get desta classe.

Quanto ao método PATCH e PUT, foi reescrito o método que permite estas operações na classe: patch e put, respetivamente. Sendo que não era necessário, na perspetiva do estagiário, o método PUT, uma vez que o PATCH cumpre o objetivo, foi criado também para que a equipa da Sentilant devida se o pretende utilizador ou não.

A nível da edição, é verificado também a expressão regular para verificar se o número de telefone tem entre 9 e 14 dígitos, para o caso de o utilizador decidir introduzir o indicativo de Portugal (00351). Sendo a verificação feita com o comando: *re.compile("^\d{9,14}$"),* depois de se fazer *import* ao *re*.

5.4.4.8 Testes Unitários

- Editar perfil:
    - test_patch_fields_empty: caso os dados para edição (PATCH) do perfil estejam todos vazios, é retornado um *status* 400.
    - test_put_fields_empty: se os dados para edição (PUT) forme todos vazios, é retornado um *status* 400.
    - test_update_all_profile: se o cliente introduzir os dados conforme suposto (PATCH), o *status* é 200, sendo feita a verificação do conteúdo que é alterado.

- test_update_2_profile_patch: caso o cliente pretenda editar apenas alguns campos com recurso ao PATCH, é retornado o *status* 200, sendo feita a alteração e a sua verificação.
- test_update_2_profile: se se recorrer ao método PUT e não se introduzir todos os campos, o *status* apresentado é o 400 e não é feita a alteração.
- test_update_address_too_big: no método PUT, se o campo do endereço for maior que o permitido na base de dados, é retornado um *status* 400 e a edição não é feita.
- test_update_name_too_big: no método PUT, quando o nome é maior que o permitido na base de dados, é retornado um *status* 400 e a edição não é feita.
- test_update_address_mobile_too_big: no método PUT, quando o número de telemóvel é maior que o permitido na base de dados, o *status* é 400 e a edição não é feita.
- test_update_address_mobile_too_short: Tendo o número de telemóvel o número mínimo, quando se recorre ao método PUT, quando o número é menor que o permitido na base de dados, é retornado um *status* 400 e a alteração não é feita.
- test_update_address_too_big_patch: no método PATCH, quando o endereço é maior que o permitido na base de dados, o *status* é 400 e a edição não é feita.
- test_update_name_too_big_patch: no método PATCH, se o campo do nome for maior que o estipulado na base de dados, é retornado um *status* 400.
- test_update_address_mobile_too_big_patch: no método PATCH, quando o campo do endereço e do número de telemóvel forem maiores que o permitido na base de dados, é retornado um *status* 400.
- test_update_mobile_too_short_patch: no método PATCH, foi verificado que é retornado um *status* 400 quando o número de telemóvel é de tamanho inferior ao definido pela Sentilant, 9 a 14 dígitos.
- test_patch_profile_unauthorized: se o cliente não tiver feito autenticação e tentar fazer a edição (PATCH) é retornado um *status* 401.
- test_put_profile_unauthorized: se o cliente não tiver feito autenticação e tentar fazer a edição (PUT) é retornado um *status* 401.

- Ver perfil:
  - test_retrive_profile: é feita a verificação que se estiver tudo conforme previsto, quando o cliente solicita a visualização do seu perfil recebe o conteúdo suposto, sendo o *status* 200.

- test_retrieve_user_unauthorized: se o cliente não tiver feito autenticação e tentar ver o perfil é retornado um *status* 401.

5.4.5 Alterar *password*

5.4.5.1 Objetivo

O objetivo desta tarefa é permitir que os clientes tenham a possibilidade de alterar a sua *password*.

5.4.5.2 Premissas

- O cliente para realizar esta tarefa tem de estar autenticado no sistema.
- Será definida a forma como o cliente pode alterar a sua *password*.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.5.3 Estratégia

Na tarefa de alterar a *password* foi definida a seguinte estratégia:

- O cliente tem de estar autenticado no sistema.
- Devem existir três campos para preenchimento: um para a *password* atual, outro para a nova *password* e, finalmente, um campo para que o cliente repita a nova *password*.
- Após a operação de alterar a *password* ter sido realizada com sucesso, é enviado um *email* ao cliente a informar que esta foi alterada.

5.4.5.4 Requisitos

No que respeita ao modo como se altera a *password*, na Tabela 18 são apresentados dois requisitos do cliente para essa tarefa, com indicação da história de quem pode realizar as operações.

Tabela 18 - Requisitos: Password.

Título	História do Cliente
Alterar *password*	Como cliente da loja da Sentilant, quero ter a possibilidade de alterar a minha *password*.
Notificação de *email*	Como cliente da loja da Sentilant, pretendo receber uma notificação via *email* quando a *password* é alterara.

5.4.5.5 Casos de uso

Na Figura 21 é apresentado o diagrama de casos de uso para a funcionalidade de alterar a password, com o ator que pode realizar a operação.

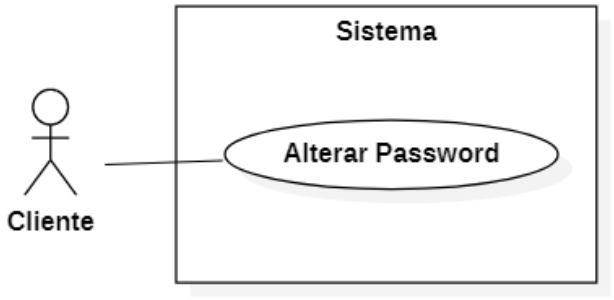


Figura 21 - DCU: Alterar *password*.

De seguida, é apresentada a descrição sumária do caso de uso alterar *password*.

- **<u>Alterar password</u>**
  - **Nome**: Alterar *Password*.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente altere a sua *password*.
  - **Pré-Condições**: O cliente tem de estar autenticado.
  - **Pós-Condições**: A *password* do cliente é alterada e este recebe uma notificação via email a informar do sucedido.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente alterar a sua *password*. Para tal, insere num campo a *password* atual noutro campo a nova *password* e, por questões de segurança e ter a certeza que é a que pretende, insere novamente a nova *password* noutro campo. Caso a operação tenha decorrido com sucesso, o cliente recebe um email com a informação do sucedido.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de alterar a *password*.
    2. O cliente preenche os campos *password* atual, nova *password* e repetir nova *password*.
    3. O sistema guarda a informação na base de dados.
    4. O sistema envia um *email* ao cliente a informar da alteração.
    5. O caso de uso termina com sucesso.

- **Cenário Alternativo**:

  2.a. Alternativamente, um dos campos preenchidos está incorreto: ou a *password* atual está incorreta, ou a nova *password* não cumpre as políticas de *password,* ou o conteúdo do campo nova *password* e repetir nova *password* é diferente.

  2.b. O sistema avisa o cliente do erro.

  2.c. Volta para 1.

  2.c.1. O caso de uso termina sem sucesso.

5.4.5.6 Estrutura

A tarefa de alterar a *password* é possível ser concretizada através do método POST, tendo sido criado o *endpoint* {{api_url}}/api/users/passwords/ para este procedimento.

Os parâmetros do *endpoint* para alterar a *password* estão presentes na Tabela 19.

Tabela 19 - Parâmetros do *endpoint* Alterar *Password*.

Chave	Tipo
current_password	string
new_password	string
repeat_password	string

5.4.5.7 Implementação

Para alterar a *password* foi criado uma classe, *PasswordViewSet*, que tem como parâmetro o *CreateModelMixin* que permite receber um método POST, possibilitando assim a alteração da *password*.

Para que seja possível alterar a password foi reescrito o método *create*, de forma a poder verificar se os campos introduzidos pelo utilizador são válidos. Inicialmente é verificado o request feito, *profile = Profile.objects.select_related('user'). get(user=request.user)*, depois e analisada a data enviada pelo utilizador. Na *password* atual, é necessário verificar se esta coincide com a atual, *profile.user.check_password(Current_password)*, se tal acontecer, então é verificado se o campo da nova password cumpre os requisitos para a criação de uma *password*, como ter pelo menos seis caracteres, e, de seguida, é feita a verificação se o campo da repetição da *password* coincide com a nova *password*. Se se verificar, então a *password* é alterada: *profile.user.set_password(new_password)*, sendo depois gravado na base de dados. Se eventualmente algum campo não estiver correto, o utilizador é avisado do sucedido.

É importante ainda salientar que para que o utilizador poder alterar a *password* tem de estar *autenticado permission_classes = (IsAuthenticated,)* e tem de ter um *token* de aceso: *authentication_classes = (TokenAuthentication,)*.

5.4.5.8 Testes Unitários

- Alterar *password*:
    - test_change_password: foi feito o teste para o caso do cliente insere os campos de forma correta quando pretende alterar a sua *password*. Neste caso, a *password* é alterada e o reebido *status* é 200.
    - test_change_password_wrong_current: quando o cliente tenta alterar a *password* e insere a *password* atual de forma errada, o *status* é 400 e a *password* não é alterada.
    - test_change_password_all_empy: se os campos submetidos forem vazios, o *status* é 400 e a *password* não é alterada.
    - test_change_password_repeat_empy: caso a repetição da *password* não seja preenchida, é retornado um *status* 400 e a *password* não é alterada.
    - test_change_password_new_empy: se o campo da nova *password* estiver vazio, o *status* é 400 e a *password* não é alterada.
    - test_change_password_diferent: foi testado também o caso da repetição da *password* se apresentar vazia, tendo sido verificado que o resultado é o esperado: *status* é 400 e esta a *password* é alterada.
    - test_change_password_one_caps1: na eventualidade da *password* estar errada, com letras maiúsculas e não ser assim, o *status* é 400 e a *password* não é alterada.
    - test_change_password_one_caps2: se a repetição da password for em letras maiúsculas, não tendo a *password* maiúsculas, o status é 400 e a *password* não é alterada.

5.4.6 Recuperar *Password*

5.4.6.1 Objetivo

Esta tarefa tem como objetivo possibilitar que os clientes recuperem a sua *password*, caso se tenham esquecido da mesma.

5.4.6.2 Premissas

- O cliente para realizar esta tarefa não tem, naturalmente, de estar autenticado no sistema.

- Será definida a forma como o cliente pode recuperar a sua *password*.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.6.3 Estratégia

Na tarefa de recuperar a *password* foi definida a seguinte estratégia:

- O cliente não tem de estar autenticado no sistema.
- Por esta tarefa envolver segurança extra, não fará parte do perfil do cliente, tendo um campo separado.
- Deverá existir um campo para o cliente digitar o seu *email*.
- Verificar se o email existe na base de dados da loja da Sentilant. Se existir, o cliente receberá um *email* com a hiperligação que remete o cliente para uma página onde o cliente terá dois campos de preenchimento, um para a nova password e outro para repetir a mesma. Caso não exista, o cliente é informado que o processo correu conforme esperado. por questões de segurança, para que não se consiga saber os *emails* registados. Naturalmente, neste último caso, não será enviado nenhum *email* para o endereço submetido.

5.4.6.4 Requisitos

Um dos requisitos do cliente é a recuperação da *password*, assim, na Tabela 20, é apresentada essa tarefa, com indicação da história de quem pode realizar essa operação.

Tabela 20 - Requisitos: Recuperar *Password*.

Título	História do Cliente
Recuperar *password*	Como cliente da loja da Sentilant, quero ter a possibilidade de recuperar a minha *password*.

5.4.6.5 Casos de uso

A Figura 22 Figura 17apresenta o diagrama de casos de uso para a recuperação da password, onde está também visíveil o interviente nesta operação.

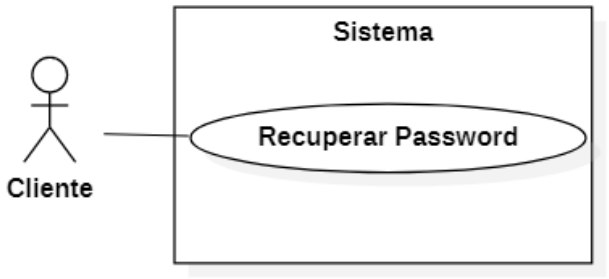


Figura 22 - DCU: Recuperar *password*.

De seguida, é apresentada a descrição sumária para o caso de uso recuperar *password*.

- **<u>Recuperar password</u>**
  - **Nome**: Recuperar *Password*.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente recupere a sua *password*.
  - **Pré-Condições**: Não existem.
  - **Pós-Condições**: A *password* do cliente é alterada.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente recuperar a sua *password*. Para tal, insere o seu *email* num campo e submete a informação. De seguida, irá receber um *email* com uma hiperligação para uma página onde poderá definir uma nova *password* preenchendo dois campos: um para a nova *password* e o outro para a repetição da mesma, para que o cliente tenha a certeza que é a *password* pretendida.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de recuperar a *password*.
    2. O cliente insere o seu *email* e submete a informação.
    3. O sistema envia um *email* ao cliente onde consta uma hiperligação para o cliente fazer a alteração da *password*.
    4. O cliente define uma nova *password* e submete a informação.
    5. A nova password fica registada na base de dados.
    6. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    2.a. Alternativamente, o *email* inserido pelo cliente não consta na base de dados da Sentilant.

2.b. O sistema avisa que tudo correu conforme previsto.

2.b.1. O caso de uso termina sem sucesso.

5.4.6.6 Estrutura

A tarefa de recuperar a *password* é possível ser concretizada através do método POST, tendo sido criados dois *endpoints*. O *endpoint* {{api_url}}/api/users/passwords/recoveries/ que permite ao cliente inserir o seu email para fazer a solicitação de recuperação da *password* e outro, {{api_url}}/api/users/passwords/recoveries/{id}, que permite ao cliente altear a *password*.

O parâmetro do *endpoint* para fazer a solicitação da alteração da *password* está presente na Tabela 21 e os da alteração de *password* estão na Tabela 22.

Tabela 21 - Parâmetros do *endpoint* Recuperar *password*.

Chave	Tipo
username	string

Tabela 22 - Parâmetros do *endpoint* Alterar a *password*.

Chave	Tipo
password	string
repeat_password	string

5.4.6.7 Implementação

Para que seja possível recuperar a *password*, o cliente não tem de estar autenticado no sistema, tendo apenas de fazer a solicitação da *password* recorrendo *endpoint* {{api_url}}/api/users/passwords/recoveries/, com um método POST que chama a classe *RecoveryViewSet* e que tem o parâmetro *CreateModelMixin*, que permite o método POST. Assim, na no método *create* que vai receber o request POST, vai verificar se o *email* introduzido pelo utilizador para recuperar a password é válido, isto é, se está na base de dados: user = *User.objects.filter(email=request.data. get('username').lower()).first()*.

Se for, efetivamente cliente da loja da Sentilant, então é gerada uma *hash* que não existe na base de dados da loja da Sentilant e é enviado um *email* ao cliente com utilizador com uma hiperligação onde consta essa *hash* no URL, {{api_url}}/api/users/passwords/recovers/{id}, e que é válido apenas por um tempo determinado, para que depois o utilizador possa aceder e escolher uma nova *password*. Este último *endpoint*, vai chamar o método *PassRecover* que recebe o request e o id, que é a *hash*. De seguida, se estiver dentro da validade e for uma *hash*

válida, então o utilizador pode definir uma nova *password*, inserindo num campo a nova *password* e depois noutro a repetição da mesma. Se estes campos forem iguais e se cumprirem as políticas da password estipuladas pela Sentilant, como ter pelo menos seis caracteres, então a password é alterada *profile.user.set_password(password)* e é gravada na base de dados.

Importa salientar que se aquando da solicitação da alteração da *password* o *email* não constar na base de dados, a responsa será um HTTP 200, por questões de segurança, de forma a que não seja possível identificar que email tenha conta ou não – naturalmente, se o *email* introduzido não estiver na base de dados não será enviado nenhum *email* para essa pessoa.

5.4.6.8 Testes Unitários

- Recuperar *password*:
    - test_pass_recovery: quando os dados para a recuperação da *password* estão corretos, é enviado um *email* ao utilizador e o *status* 200.
    - test_pass_recovery_empty: se o campo do *username* estiver vazio, o status é *400*e não é enviado email para recuperar a *password.*
    - test_pass_recovery_wrong_user: Se o *username* estiver errado, não é enviado o *email* para alterar a *password* e é retornado um *status* 400.
    - test_pass_recovery_email_caps: caso o *username* esteja escrito em letras maiúsculas, corre tudo conforme previsto, sendo o *status* 200.
    - test_recover: o *status* é 200, indicando que tudo correu conforme previsto, quando o utilizador acede a uma hiperligação que recebe no mail e altera a *password* dentro do limite de tempo possível para fazer este procedimento.
    - test_recover_out_of_date: caso o utilizador tente alterar a *password* depois de ter solicitado a sua recuperarão e a *hash* já tenha passado de prazo, é recebido um *status* 400, não sendo feita a alteração.
    - test_recover_wrong_hash: se a *hash* introduzida para a alteração da *password* estiver errada, é retornado um *status* 400.
    - test_recover_empy_fields: depois de solicitar a recuperação da *password*, quando se procede a esta alteração, se os campos forem vazios, a alteração não é feita e é retornado um *status* 400.
    - test_recover_repeat_empty: se aquando da alteração a repetição da *password* estiver vazia, não é feita a alteração e o *status* é 400*.*
    - test_recover_pass_empty: se o campo da *password* estiver vazio quando se pretende fazer a sua alteração, não é feita esta operação e o *status* é 400.

- test_recover_caps_pass_only: se o campo da *password* for em maiúsculas e a sua repetição não, o *status* é 400 e não é feita a alteração.
- test_recover_caps_passrecov_only: se a repetição da *password* for em maiúsculas e a *password* não for só constituído por maiúsculas, a alteração não é feita e é retornado um *status* 400.
- test_recover_no_min: se a *password* inserida não tiver o número mínimo de caracteres, não é feita a alteração da password e o *status* é 400.

5.4.7 Alterar *Username*

5.4.7.1 Objetivo

Esta tarefa tem como objetivo facultar a possibilidade aos clientes de alterem o seu *username* (*email*).

5.4.7.2 Premissas

- O cliente para realizar esta tarefa tem de estar autenticado na loja da Sentilant.
- Será definida a forma como o cliente pode alterar o seu *username*.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.7.3 Estratégia

Para alterar o *username* definida a seguinte estratégia:

- O cliente tem de estar autenticado no sistema para fazer a solicitação da alteração do *username*.
- Esta tarefa, por ser um campo sensível, que envolve segurança extra, não fará parte do perfil do cliente.
- O processo de alteração de username envolve dois passos, por questões de segurança.
  - Inicialmente deverá ser feita a solicitação da alteração do *username*. Para tal, deve existir um campo para o cliente digitar o seu *username* atual e outro para a sua password.
  - Depois de fazer a solicitação da alteração do *username*, o cliente receberá um *email* com uma hiperligação para fazer a alteração do *email* onde constam três campos para o cliente preencher: novo *username, password* atual e repetição da *password* atual, para confirmar que é o cliente que está a alterar o *username*.

5.4.7.4 Requisitos

Na Tabela 23, é apresentada o requisito relativo ao modo de alterar o username, com indicação da história do cliente.

Tabela 23 - Requisitos: Alterar *Username*.

Título	História do Cliente
Alterar *Username*	Como cliente da loja da Sentilant, quero ter a possibilidade de alterar o meu *username*.

5.4.7.5 Casos de uso

No que respeita aos casos de uso, na Figura 23 é apresentado o diagrama de casos de uso da alteração do *username*, onde é apresentado também o ator que pode realizar a operação.

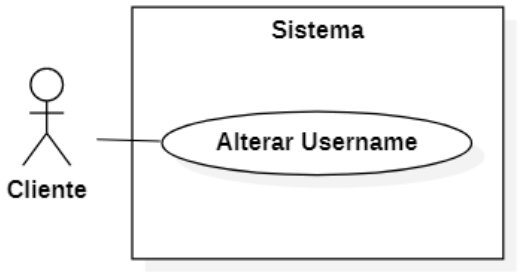


Figura 23 - DCU: Alterar *username*.

De seguida, é apresentada a descrição sumária do caso de uso alterar *username*.

- **Alterar Username**
  - **Nome**: Alterar *Username*.
  - **Ator**: Cliente.
  - **Objetivo**: Possibilitar que o cliente altere o seu *username*.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O *username* é alterado.
  - **Descrição**: Neste caso de uso, o sistema permite ao cliente alterar o seu *username*. Inicialmente faz a solicitação da alteração do *username* e, de seguida, irá receber um *email* onde consta uma hiperligação para poder prosseguir com a alteração do *username*. Nesta hiperligação, que tem uma duração limitada, terá de preencher três campos, uma para o novo *usesrname*, um para a *password* e, finalmente, outro para a repetição da password, aumentado assim a segurança neste procedimento.

- **Cenário principal**:
  1. O sistema disponibiliza a opção de solicitar a alteração do *username*.
  2. O cliente insere o seu *username* atual e a sua *password* e submete a informação.
  3. O sistema envia um *email* ao cliente onde consta uma hiperligação para o cliente fazer a alteração do *username*.
  4. O cliente acede ao endereço que recebeu via *email*.
  5. O cliente, no endereço que recebeu, define um novo *username* e submete a informação.
  6. O novo *username* fica registado na base de dados.
  7. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  2.a. Alternativamente, o cliente insere algum campo com informação incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c.1 Volta a 1.

  4.a. O tempo definido para aceder a esta hiperligação passou de prazo.

  5.a. Alternativamente, o cliente insere informação incorreta.

  5.a.1 Volta para 5.

  2.c.1./ 4.a.1./ 5.a.1. O caso de uso termina sem sucesso.

5.4.7.6 Estrutura

A tarefa de atualizar o *username* é possível ser efetuada recorrendo a dois *endpoints*, ambos com o método POST. O *endpoint* {{api_url}}/api/users/usernames/, onde o cliente pode fazer a solicitação da alteração do *username*, inserindo os parâmetros que se encontram na Tabela 24, e o *endpoint* {{api_url}}/api/users/usernames/{id}, que tem os parâmetros que constam na Tabela 25.

Tabela 24 - Parâmetros do *endpoint* Solicitar a alteração do username.

Chave	Tipo
current_username	string
password	string

Tabela 25 - Parâmetros do *endpoint* Alterar o *username*.

Chave	Tipo
password	string
username	string
repeat_username	string

5.4.7.7 Implementação

O cliente para alterar o *username (email)*, por questões relacionadas com segurança, tem de fazer a sua solicitação primeiro e só depois, mediante um *email* que irá receber, é que o poderá alterar. Assim, na parte da implementação quando se solicita através do *endpoint* {{api_url}}/api/users/usernames/, com o método POST, é chamada a classe criada com o nome de *UsernameViewSet*, que tem um parâmetro que permite receber o método POST: *CreateModelMixin*.

Nesta classe, é feita, inicialmente, a verificação que o utilizador tem um *token* e que está autenticado: *authentication_classes = (TokenAuthentication,)* e *permission_classes = (IsAuthenticated,)*, respetivamente, para depois no método *create* da classe que vai receber a data enviada pelo utilizador, vai verificar se esta está em conformidade com o solicitado, ou seja, verifica se o email atual solicitado é, efetivamente, o email do cliente, e também se a password inserida é a do utilizador. Se o cliente introduziu algum campo de forma incorreta, será informado do sucedido, caso contrário, vai ser criada uma *hash* única, isto é, que não existe na base de dados da loja da Sentilant, para depois a incluir na hiperligação que o cliente irá receber no *email*. Essa hiperligação, que tem como *endpoint* {{api_url}}/ api/users/usernames/{id}, sendo o id a *hash* e que tem uma validade limitada e faz uso do método POST, vai permitir que o utilizador altere o seu *username*, não necessitando para o efeito de estar autenticado nem de nenhum dado extra. Os campos que terá de inserir para a alteração ser feita, e que são recebidos no método *UsernameRecover*, são: a *password,* o novo *username* (*email)* e a repetição deste último campo, para confirmar que quer efetivamente alterar para esse *username*.

5.4.7.8 Testes Unitários

- Alterar *username*:
    - test_change_username: quando o cliente faz a solicitação para alterar o *username*, é retornado um *status* 200 e enviado um *email* para o utilizador.
    - test_change_username_wrong_email: se o cliente inserir um email errado, não é enviado o email para fazer a alteração e o *status* é 400.

- test_change_username_wrong_pass: quando o cliente introduz a *password* errada, não é feita a solicitação para alterar o *username* e o *status* recebido é 400.
- test_change_username_all_emtpy: se todos os campos enviados estiverem vazios, o *status* recebido é 400, não sendo feita a solicitação de alteração.
- test_change_username_emtpy_user: quando o cliente deixa o camo do *username* atual vazio, não é feita a solicitação e é retornado um *status* 400.
- test_change_username_emtpy_pass: assim como acontece com o campo da *password* vazio, retornar um *status* 400, o mesmo acontece quando o campo para a repetição desta fica vazio.
- test_change_username_pass_caps: se a *password* atual estiver em maiúsculas, não é feita a solicitação e o *status* é 400.
- test_username_update: quando o cliente recebe o *email* com a *hash* e corre tudo conforme previsto, é feita a alteração e o *status* é 200.
- test_username_update_time_out_of_date: se a *hash* introduzida se encontra fora da validade, é recebido um *status* 400.
- test_username_update_wrong_hash: quando a *hash* é inválida, é retornado um *status* 400.
- test_username_update_empy_fields: se todos os campos submetidos para a alteração do *username* estiverem vazios, não é feita a alteração desta e o *status* é 400.
- test_username_update_empy_user_and_repeat: quando a repetição da *password* e o *username* estão vazios aquando da submissão, não é feita a sua alteração e o *status* 400.
- test_username_update_empty_repeat: se a repetição da *password* estiver a vazio, não é feita a alteração e o *status* é 400.
- test_username_update_dif_user: quando a *password* e a repetição desta forem diferentes, o retorno será um *status 400*, não sendo feita a sua alteração.
- test_username_update_worng_mail: se o email atual introduzido pelo utilizador não for válido, a sua alteração não é feita, sendo retornado um *status* 400.
- test_username_update_user_repeat_caps: caso o campo da repetição da *password* esteja em letra maiúsculas, não se verificando o mesmo na *password*, o *status* é 400 e não é feita a alteração.

- test_username_update_user_caps: se o novo *username* estiver em letras maiúsculas, este é alterado, sendo recebido um *status* 200.
- test_username_update_repeat_caps: na eventualidade da repetição do *username* estar em letras maiúsculas, a sua alteração é feita normalmente, retornando um *status* 200.
- test_username_update_user_too_big: se o *username* introduzido for muito grande, maior que o permitido pelo definido pela empresa Sentilant, não é feita a alteração e o retorno é *status* 400.

5.4.8 Encomendas

5.4.8.1 Objetivo

Objetivo é possibilitar ao cliente fazer a gestão das suas encomendas, possibilitando a este adicionar itens a uma encomenda, visualizar os itens, eliminar os mesmos, listar encomendas e eliminar as encomendas.

5.4.8.2 Premissas

- O cliente para realizar esta tarefa tem de estar autenticado na loja da Sentilant.
- Será definida a forma como o cliente pode adicionar um item, visualizar os itens, eliminar um item, listar as encomendas ou eliminar as mesmas.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.8.3 Estratégia

A estratégia para que o cliente possa fazer a gestão das encomendas foi a seguinte:

- O cliente:
  - Tem de estar autenticado no sistema.
  - Para adicionar um item a uma encomenda, tem de se inserir o número de *requests* pretendidos.
  - Pode visualizar um ou mais itens, assim como encomendas.
  - Tem a possibilidade de editar uma encomenda, alterando a quantidade pretendida do número de *requests*.
  - Pode eliminar um item ou encomendas, desde que estas não tenham sido transacionadas.

5.4.8.4 Requisitos

Na Tabela 26, são apresentadas as tarefas relativas às encomendas, com indicação da história do cliente.

Tabela 26 - Requisitos: Encomendas.

Título	História do Cliente
Adicionar Item	Como cliente da loja da Sentilant, quero ter a possibilidade de adicionar um item a uma encomenda.
Ver Itens	Como cliente da loja da Sentilant, quero ter a hipótese de visualizar os itens.
Eliminar Item	Como cliente da loja da Sentilant, quero ter a possibilidade de eliminar um item que ainda não foi adquirido.
Editar Encomenda	Como cliente da loja da Sentilant, quero ter a possibilidade de editar a minha encomenda.
Listar Encomendas	Como cliente da loja da Sentilant, quero ter a possibilidade de visualizar as encomendas.
Eliminar Encomenda	Como cliente da loja da Sentilant, quero ter a possibilidade de eliminar uma encomenda ainda não transacionada.

5.4.8.5 Casos de uso

Na Figura 24 é apresentado o diagrama de casos de uso para as encomendas, de forma a perceber quais as funcionalidades disponíveis e também o ator que pode realizar estas operações.

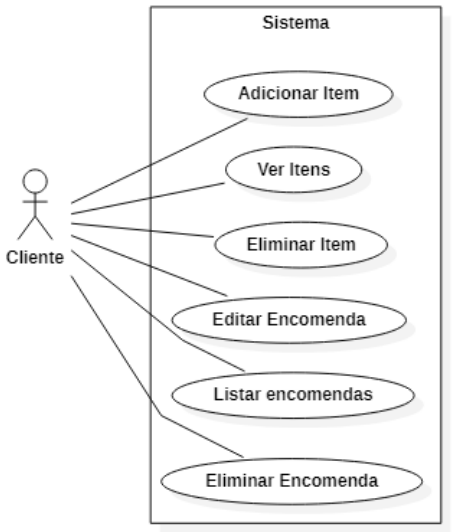


Figura 24 - DCU: Encomendas.

De seguida, é apresentada a descrição sumária para os casos de uso: adicionar item, ver itens, eliminar item, editar encomenda, listar encomendas e eliminar encomenda.

- **Adicionar Item**
  - **Nome**: Adicionar Item.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente adicione um item ao seu carrinho de compras da loja Sentilant.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O item fica adicionado ao carrinho de compras.
  - **Descrição**: Neste caso de uso, o cliente adiciona o item que pretende adquirir ao carrinho de compras, indicando a quantidade de *requests* que pretende desse item.
  - **Cenário principal**:

1. O sistema disponibiliza a opção de adicionar um item a uma encomenda.
2. O cliente seleciona o item e indica a quantidade pretendida de *requests*.
3. O item é adicionado ao carrinho de compras.
4. O caso de uso termina com sucesso.

  - **Cenário Alternativo**:

    2.a. Alternativamente, a informação inserida está incorreta.

    2.b. O sistema avisa que ocorreu um erro.

    2.c. Volta a 1.

    2.c.1. O caso de uso termina sem sucesso.

- **Ver Itens**
  - **Nome**: Ver Itens.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente visualize os itens que foram adquiridos ou estão no carrinho de compras, com a informação dos mesmos.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: É apresentada a lista de itens ao cliente, caso tenha algum item.

- **Descrição**: Neste caso de uso, o cliente visualiza os itens que foram adquiridos ou que estão no carrinho de compras.
- **Cenário principal**:
  1. O sistema disponibiliza a opção visualizar os itens adquiridos e os que estão no carrinho.
  2. O cliente indica que pretende visualizar os itens.
  3. O sistema mostra os itens.
  4. O caso de uso termina com sucesso.

➢ **Eliminar Item**

- **Nome**: Eliminar Item.
- **Ator**: Cliente.
- **Objetivo**: Permitir que o cliente possa eliminar um item que ainda não tenha sido adquirido.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: O item do carrinho de compras é removido do mesmo.
- **Descrição**: Neste caso de uso, o cliente tem a possibilidade de eliminar um item que consta no carrinho de compras, ou seja, um item que ainda não foi adquirido.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de eliminar um item do carrinho de compras.
  2. O cliente indica o item que pretende eliminar.
  3. O sistema remove do carrinho de compras o item selecionado.
  4. O caso de uso termina com sucesso.

➢ **Editar Encomenda**

- **Nome**: Editar Encomenda.
- **Ator**: Cliente.
- **Objetivo**: Possibilitar que o cliente edite um item da encomenda que pretende realizar.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.

- **Pós-Condições**: O item é editado e a informação fica registada na base de dados.
- **Descrição**: Neste caso de uso, o cliente edita um item que se encontra no carrinho de compras, ou seja, um item da encomenda que ainda não foi efetuada.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de editar um item da encomenda que se encontra no carrinho de compras.
  2. O cliente edita o item pretendido indicado a quantidade de *requests* pretendida.
  3. O sistema guarda a informação na base de dados.
  4. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  2.a. Alternativamente, a informação inserida pelo cliente não está correta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  2.c.1. O caso de uso termina sem sucesso.

➢ **Listar Encomendas**

- **Nome**: Listar Encomendas.
- **Ator**: Cliente.
- **Objetivo**: Permitir que o cliente visualize as encomendas efetuadas e as que estão por efetuar.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: A lista de encomendas com a informação das mesmas é apresentada ao cliente.
- **Descrição**: Neste caso de uso, o cliente visualiza as suas encomendas, quer estejam por processar ou processadas, podendo visualizar os dados de cada uma.
- **Cenário principal**:
  1. O sistema disponibiliza a opção para o cliente visualizar as encomendas.
  2. O cliente visualiza as encomendas que fez e as que estão por fazer.
  3. O caso de uso termina com sucesso.

- **Eliminar Encomenda**
  - **Nome**: Eliminar Encomenda.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente elimine uma encomenda por processar.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: A encomenda é eliminada do carrinho de compras.
  - **Descrição**: Este caso de uso permite que o cliente elimine uma encomenda que se encontra por processar, deixando de estar assim disponível no carrinho de compras.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de eliminar uma encomenda do carrinho de compras.
    2. O cliente indica a encomenda que pretende eliminar.
    3. O caso de uso termina com sucesso.

5.4.8.6 Estrutura

Neste ponto estão presentes 4 *endpoints*:

- {{api_url}}/api/orders/, que pode ser utilizado com recurso a 2 métodos:
  - POST: Permite adicionar um item a uma encomenda e os seus parâmetros encontram-se na Tabela 27.

Tabela 27 - Parâmetros do *endpoint* Adicionar Item.

Chave	Tipo
id	int
quantity	int

  - GET: Possibilita a visualização das encomendas, efetuadas e por efetuar, e os parâmetros que são enviados para a equipa de *front-end* em formato JSON são os que se encontram na Tabela 28.

Tabela 28 - Parâmetros do *endpoint* Ver Encomendas.

Chave	Tipo
id	int
owner	int

is_ordered	boolean
date_ordered	date
Itens > id	int
Itens > is_ordered	boolean
Itens > date_added	date
Itens > date_ordered	date
Itens > quantity	int
Itens > user	int
Itens > product	int

- {{api_url}}/api/orders/{id}, que pode ser usado com os métodos:
  - GET: Para ver uma encomenda específica, sendo que os dados enviados para a equipa de *front-end* encontram-se na Tabela 28.
  - DELETE: Para eliminar uma encomenda.
- {{api_url}}/api/items/, que recorre de um método:
  - GET: Visualizar os itens que já adquiriu e os que estão no carrinho de compras, sendo enviados os parâmetros que se encontram na Tabela 29 para a equipa de *front-end* em formato JSON.

Tabela 29 - Parâmetros do *endpoint* Ver Itens.

Chave	Tipo
id	int
is_ordered	boolean
date_added	date
date_ordered	date
quantity	int
user	int
product	int

- {{api_url}}/api/items/{id}, que possível ser acedido com três funções diferentes, consoante o método pretendido:
  - GET: Visualizar a informação de um item específico, sendo que a informação que é enviada, em formato JSON, para a equipa de *front-end* encontra-se na Tabela 29.

- PATCH: Para editar um item de uma encomenda, é necessário o parâmetro que se encontra na Tabela 30.

Tabela 30 - Parâmetros do *endpoint* Editar Item.

Chave	Tipo
quantity	int

- DELETE: Eliminar um item de uma encomenda que se encontra no carrinho de compras.

5.4.8.7 Implementação

Para a implementação de todos os pontos deste tópico, o cliente tem de estar autenticado na loja da Sentilant e tem de ter um *token* de acesso, para tal, em todas as classes que forem descritas neste ponto, é feita a verificação: *authentication_classes = (TokenAuthentication,)*, para o *token*, e *permission_classes = (IsAuthenticated,)*, para a autenticação.

A classe *OrderViewSet*, que tem nos parâmetros, *ListModelMixin*, *CreateModelMixin*, *DestroyModelMixin*, *RetrieveModelMixin,* que permite que sejam reescritos os métodos list, create, destrói e retrieve, respetivamente.

Ao reescrever o método create, que é chamado através do *endpoint* {{api_url}}/api/orders/ é feita a parte da lógica que permite a adição de um item (produto) a uma encomenda, sendo verificado o id do produto solicitado Product.objects.filter(id=self.request.data.get('id')).first(), para depois ser feita a verificação do campo introduzido pelo utilizador, a quantidade de *requests* que este pretende para esse produto. Sendo que esse número, naturalmente, tem de sair maior que um, sendo essa verificação feita.

Ainda com o último *endpoint* referido, se for utilizador o método GET, é chamado o método *list*, permitindo assim que o utilizador visualize as suas encomendas, uma vez que é feito neste método uma seriação pelas suas encomendas: *Order.objects.filter(owner=self.request.user).*

Se o utilizador pretender visualizar os dados de uma encomenda específica, utilizando o método GET com o *endpoint* {{api_url}}/api/orders/{id}, onde o id é o id do produto, é chamado é método desta classe *retrieve*, onde é feita a seriação desse produto, dentro das encomendas do utilizador, estejam adquiridas ou não: *Order.objects.filter(owner=self.request.user, id=kwargs.get('id')).*

Com acesso ao mesmo ao mesmo *endpoint*, mas utilizando o método DELETE, o utilizador chama o método *destroy*, que vai eliminar a encomenda que tem o id da URL. Em primeiro lugar, é feita a identificação desse produto, *queryset = Order.objects.filter(owner=self.request.user, id=kwargs.get('id')).first()*, para depois se

a encomenda não tiver sido ainda transacionada, ser eliminada. Caso tenha sido transacionada, então é apresentada um HTTP 409, uma mensagem de conflito, pois não é permitido eliminar encomendas efetuadas.

Na classe *ItemViewSet*, que recebe os parâmetros que a classe *OrderViewSet,* menos o *CreateModelMixin*, uma vez que não existe nenhuma intenção de criar algo nesta classe, ou seja, não permite métodos POST, é possível através do método *list*, que é chamado com o método GET e *endpoint* {{api_url}}/api/items/, visualizar todos os itens que o utilizador ou tem no carrinho de compras ou adquiriu, sendo feita a nível de implantação a sua triagem: *OrderItem.objects.filter(user=self.request.user)*.

Se o cliente utilizar o *endpoint* {{api_url}}/api/items/{id}, com o método GET, é possível visualizar um item (produto) específico que adquiriu ou está no carrinho, sendo chamado o método *retrieve*, onde é feito a seleção para mostrar apenas esse produto e desse utilizador: OrderItem.objects.filter(user=self.request.user, id=kwargs.get('id')).

Já se o utilizador utilizar o método PATCH, vai chamar o método partial_update, que vai permitir atualizar a quantidade de itens que pretende adquirir, sendo também feita a verificação se se trata já de um produto adquirido e nesse caso não pode altear a quantidade, obtendo um HTTP 409, um conflito, e também se o número é maior que 1.

Se o cliente optar por eliminar um item de uma encomenda, então com o mesmo *endpoint*, {{api_url}}/api/items/{id} e com recurso ao método DELETE, é chamado o método *destroy*, que vai eliminar esse item se este ainda não tiver sido encomendado.

5.4.8.8 Testes Unitários

- Adicionar item:
    - test_add_order: foi feita a verificação quando o cliente introduz os campos corretos de forma a adicionar um item a uma encomenda, sendo verificado se esse item foi adicionado e o retorno da *status* como 200.
    - test_add_order_negative_qty: caso se tente adicionar um item com uma quantidade de requests negativos, não é permitido, sendo retornado um *status* 400.
    - test_post_order_no_data: se na adição não for enviada nenhuma informação, é retornado um *status* 400, não sendo, naturalmente, nada adicionado.
    - test_add_order_string_qty: se a quantidade for sob a forma de string, este item não é adicionado, sendo retornado um *status* 400.

- Ver itens:
    - test_list_items_from_user: se o cliente pretender ver os seus itens e correr tudo conforme previsto é retornado um *status* 200. Sendo feita também a verificação que a informação a ser apresentada é a correta.
    - test_retrieve_a_item_from_user: caso o cliente não tenha encomendas ainda, é retornado um status 200.
    - test_retrieve_a_item_from_user_wrong_id: caso o id não exista ou seja de outro cliente, no que respeita ao item, é apresentado um *status* 400.
    - test_retrieve_a_item_from_user_string_id: se o id enviado for uma string, é considerado como inválido e retornado um *status* 400.
- Eliminar itens
    - test_remove_item: é apresentado um *status* 204 quando a nível da eliminação de itens corre conforme previsto.
    - test_remove_item_wrong_id: quando se tenta eliminar um item e o id não existe ou é de outra pessoa, o item não é eliminado, sendo recebido um *status* 400.
    - test_remove_item_string_id: se o id for submetido sob a forma de *string*, não é aceite e é retornado um *status* 400.
- Editar encomendas
    - test_update_item: caso se pretenda atualizar uma encomenda e tudo corra conforme previsto, foi feita a verificação que os dados são alterados e o *status* retornado é 200.
    - test_update_item_negative_quantity: se na edição a quantidade inserida for negativa, é retornado um *status* 400 e não é feita a alteração.
    - test_update_item_string_qty: na eventualiadde da quantidade ser submetida como uma *string*, foi feita a verificação que não é admitido e feita a alteração, sendo retornado o *status* 400.
- Listar encomendas:
    - test_list_orders_from_user: se o cliente quiser ver a lista das suas encomendas e tudo correr conforme previsto, essa verificação foi feita, sendo confirmada a informação que é apresentada e o *status* retornado é 200.
    - test_retrieve_a_order_from_user: é retornado um *status* 200 caso o cliente pretenda ver uma encomenda específica, quando tudo corre conforme previsto, sendo também, naturalmente, feita a confirmação que a informação apresentada é a suposta.

- test_retrieve_a_order_from_user_wrong_id: se o id enviado for um id que não exista ou que não pertença ao cliente, é retornado um *status* 400.
- test_retrieve_a_order_from_user_id_string: quando o id enviado é uma string, o que não é suposto, é retornado um *status* 400.

- Eliminar encomenda:
  - test_remove_order: quando o utilizador tenta remover uma encomenda que ainda não foi transacionada, esta é eliminada, sendo recebido um *status* 204.
  - test_remove_order_id_do_not_exists: se o id da encomenda a eliminar não existir ou pertencer a outro cliente, este não é, naturalmente, eliminado e o *status* retornado é 400.
  - test_remove_order_id_string: se o id da encomenda a eliminar for apresentado sob a forma de *string*, esta não é reconhecida e é retornado o *status* 400.
  - test_remove_transacted_order: é retornado um *status* 409 quando o utilizador tenta eliminar uma encomendada que já foi transacionada, uma vez que não é permitido.

5.4.9 Transações

5.4.9.1 Objetivo

Esta tarefa tem como objetivo que o cliente possa fazer um pagamento de uma encomenda, assim como visualizar as transações efetuadas.

5.4.9.2 Premissas

- O cliente para realizar esta tarefa tem de estar autenticado na loja da Sentilant.
- Será definida a forma como o cliente pode fazer uma transação, assim como visualizar a lista das transações efetuadas.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.9.3 Estratégia

Para que o cliente da loja da Sentilant possa fazer um pagamento e visualizar as transações efetuadas foi a seguinte:

- O cliente tem de estar autenticado no sistema.

- O cliente tem de ter um cartão de crédito para que possa efetuar um pagamento, inserindo os dados do mesmo. A saber: número, data de validade e cvv.

5.4.9.4 Requisitos

Na Tabela 31, são apresentadas as tarefas relativas às transações, com indicação da história do cliente.

Tabela 31 - Requisitos: Transações.

Título	História do Cliente
Efetuar Pagamento	Como cliente da loja da Sentilant, quero ter a hipótese de fazer um pagamento de uma encomenda.
Ver Transações	Como cliente da loja da Sentilant, quero ter a possibilidade de ver as minhas transações.

5.4.9.5 Casos de uso

A Figura 25 apresetna o diagrama de casos de uso no que respeita às transações, identificando assum as tarefas possíveis e o seu ator.

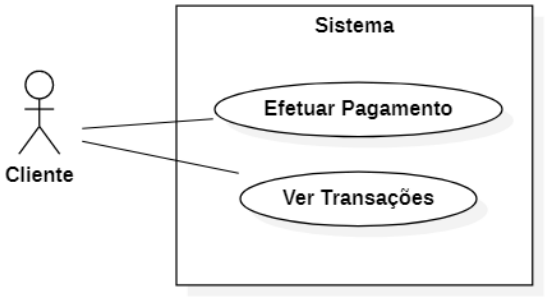


Figura 25 - DCU: Transações.

De seguida, é apresentada a descrição sumária para os dois casos de uso: efetuar pagamento e ver transações.

- **<u>Efetuar Pagamento</u>**
  - **Nome**: Efetuar Pagamento.
  - **Ator**: Cliente.
  - **Objetivo**: Efetuar o pagamento de uma encomenda para poder ter acesso aos serviços disponibilizados pela Sentilant que se encontram na encomenda.

- **Pré-Condições**: Estar autenticado na loja da Sentilant e ter um cartão de crédito.
- **Pós-Condições**: O cliente fica com acesso aos produtos que adquiriu.
- **Descrição**: Neste caso de uso, o cliente efetua o pagamento da encomenda, podendo assim, posteriormente, ter acesso aos produtos adquiridos. Para tal, insere os dados do seu cartão: data de validade, cvv e número.
- **Cenário principal**:
  1. O sistema disponibiliza um formulário para inserir os dados do cartão.
  2. O cliente insere os dados do seu cartão: número, data de validade e cvv.
  3. O sistema verifica os dados com recurso aos serviços do Braintree.
  4. O valor é descontado no cartão do cliente.
  5. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  2.a. Alternativamente, a informação inserida está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  3.a. O sistema não consegue comunicar com a Braintree ou não tem saldo suficiente.

  3.b. O sistema avisa o cliente do sucedido.

  3.c. Volta para 1.

  2.c.1./3.c.1. O caso de uso termina sem sucesso.

➢ **<u>Ver Transações</u>**

- **Nome**: Ver Transações.
- **Ator**: Cliente.
- **Objetivo**: Permitir que o cliente visualize as transações que fez.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: É apresentada a lita de transações que o cliente fez.
- **Descrição**: Neste caso de uso, o cliente visualiza as transações que já fez.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de visualizar as transações.

2. O cliente indica que pretende visualizar as transações.
3. O sistema mostra as transações que o cliente fez.
4. O caso de uso termina com sucesso.

5.4.9.6 Estrutura

Neste ponto existem dois *endpoints,*{{api_url}}/api/transactions/, que pode ser acedido através dos métodos:

- POST: Permite fazer uma transação e os campos que o cliente tem de inserir estão na Tabela 32.

Tabela 32 - Parâmetros do *endpoint* Adicionar Cartão.

Chave	Tipo
number	string
expiration_date	string, formato: “mm/yyyy” (*e.g.*: “10/2022”)
cvv	string

- GET: Permite que o cliente veja as transações que efetuou, estando os dados que são enviados para a equipa de *front-end*, em formato JSON*,* na Tabela 33, fazendo depois a equipa de front-end a seriação do que será mostrado ao cliente.

Tabela 33 - Parâmetros do *endpoint* Ver Cartões.

Chave	Tipo
id	string
transaction_id	string
order_id	string
amount	string
success	string
timestamp	string
profile	string

O *endpoint* {{api_url}}/api/transactions/{id}, que pode ser acedido com recurso ao método GET. Este método permite que o cliente visualize os dados de apenas um cartão selecionado pelo cliente. Os elementos que são enviados para a equipa de *front-end* em formato JSON, encontram-se na Tabela 33.

5.4.9.7 Implementação

Para que o cliente possa fazer um pagamento ou ver as suas transações tem de estar autenticado na loja da Sentilant e tem de ter um *token*, para tal no início da classe que permite estas operações, *TransactionViewSet*, é feita a verificação da autenticação, *permission_classes = (IsAuthenticated,)* e verificado se o cliente tem um *token* válido, *authentication_classes = (TokenAuthentication,)*.

Esta classe é chamada para efeitos de fazer uma transação através do *endpoint* {{api_url}}/api/transactions/, com método POST, pelo que no parâmetro da classe tem a indicação para receber um POST com a indicação do *CreateModelMixin*, e na classe existe um método *create* que vai receber os parâmetros enviados pelo cliente e fazer a verificação se os dados estão corretos e fazer a comunicação com o Braintree.

No início é feita a verificação se a encomenda para qual o cliente pretende fazer a transação ainda não foi transacionada, *Order.objects.filter(owner=user_profile, is_ordered=False)*, se o total do carrinho é positivo e é também gerado um token recorrendo ao Braintree:

```
gateway = braintree.BraintreeGateway(
   braintree.Configuration(
      environment=info.BT_ENVIRONMENT,
      merchant_id= info.BT_MERCHANT_ID,
      public_key= info.BT_PUBLIC_KEY,
      private_key= info.BT_PRIVATE_KEY
   )
)[47]
```

e depois para gerar o *token: gateway.client_token.generate()*.

De seguida, é feita a verificação da validade da data para que depois seja feita a feita a transação enviado os dados no parâmetro *options*: *gateway.transaction.sale(options)*. Caso a transação seja efetuada com sucesso, é feita a atualizada na base de dados da loja da Sentilant relativa às encomendas.

Ainda com o último *endpoint* referido, se o cliente pretender ver a lista de transações efetuadas, este processo é feito com recurso ao método GET e chama esta classe, contudo para que fosse permitido a chamada com o método GET, foi inserido no parâmetro da casse o *ListModelMixin*, sendo feita depois a seriação no método *list* da classe pelas transações do utilizador: *Transaction.objects.filter(profile=self. request.user)*.

Se o cliente pretender visualizar uma transação em específico, então pode fazer recorrendo ao método GET e com o *endpoint* {{api_url}}/api/transactions/{id}, sendo o id a identificação da transação. A nível da implementação, é feita a seriação para

[47] O *braintree* é o *import* do *Braintree* e o *info* é um nome fictício, de forma a não dizer onde se encontram as informações da Sentilant sobre a sua conta de utilizador da Braintree.

mostra só esse produto do respetivo utilizador: *Transaction.objects.filter(profile=self.request.user, id=kwargs.get('id'))*.

5.4.9.8 Testes Unitários

- Efetuar pagamento:
    - test_transaction_visa: é verificado que se todos os campos forem preenchidos de forma correta e o cartão o permitir, é feito o pagamento da encomenda, sendo retornado um *status* 200.
    - test_transaction_visa_expired: caso a data do cartão tenha expirado, é retornado um *status* 400 e não é feito o pagamento.
    - test_transaction_Processor_Declined: caso o cartão não esteja em condições, o Braintree irá apresentar essa informação, sendo depois retornado um *status* 400.
    - test_transaction_visa_wrong_info_int: caso a data de validade seja enviada como inteiro e não *string*, não é aceite, sendo apresentado um *status* 400.
    - test_transaction_visa_wrong_cvv: se o cvv for enviado como *string*, este não é válido, sendo retornado um *status* 400.
    - test_transaction_visa_wrong_number: se o número do cartão for enviado como *string*, não é válido e é retornado um *status* 400.
- Ver transações:
    - test_get_a_transaction_and_all_mastercard: é verificado que se o cliente solicitar ver todas as transações que fez, e estiver tudo em conformidade, é retornado um *status* 200 e verificado que o conteúdo apresentado é o pretendido.
    - test_get_a_transaction_empty_transactions: é retornado um *status* 400 se o cliente tentar ver uma transação específica que não existe.
    - test_get_all_empty_transactions: caso o cliente pretenda ver todas as transações e não tiver efetuado nenhum pagamento, então recebe um *status* 200, apesar de não ter conteúdo.
    - test_get_a_transaction_string_id_transactions: se for enviado um id como *string* este não é válido, sendo recebido um *status* 400.

5.4.10 Cartões

5.4.10.1 Objetivo

Esta tarefa tem como objetivo que o cliente tenha a possibilidade de fazer a gestão dos seus cartões de crédito: adicionar, ver, editar ou eliminar um cartão.

5.4.10.2 Premissas

- O cliente para realizar esta tarefa tem de estar autenticado na loja da Sentilant.
- Será definida a forma como o cliente pode adicionar, visualizar, editar ou eliminar os seus cartões da loja da Sentilant.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.10.3 Estratégia

A estratégia para que o cliente possa fazer a gestão dos cartões foi a seguinte:

- O cliente:
    - Tem de estar autenticado no sistema.
    - Para adicionar um cartão, terá de inserir os dados do mesmo: número, data de expiração, cvv, e nome, sendo este último opcional.
    - Para editar um cartão, o cliente pode alterar o seu nome, cvv ou data de validade.
    - Pode visualizar um ou mais cartões e os seus detalhes possíveis, não apesentado a informação toda do cartão, e apagar os cartões, indicando o cartão que pretende remover.

5.4.10.4 Requisitos

Para se compreender melhor os requisitos para os cartões, na Tabela 34 são apresentadas as tarefas pera este requisito, com a descrição da história do cliente.

Tabela 34 - Requisitos: Cartões.

Título	História do Cliente
Adicionar Cartão	Como cliente da loja da Sentilant, quero ter a possibilidade de adicionar um cartão.
Ver Cartões	Como cliente da loja da Sentilant, quero ter a hipótese de visualizar os cartões.
Editar Cartão	Como cliente da loja da Sentilant, quero ter a hipótese de editar um cartão.
Eliminar Cartão	Como cliente da loja da Sentilant, quero ter a possibilidade de eliminar um cartão.

5.4.10.5 Casos de uso

Na Figura 26 é apresentado o diagrama de casos de uso no que respeita aos cartões, de forma a identificar as tarefas possíveis para este tópico o ator que as pode realizar.

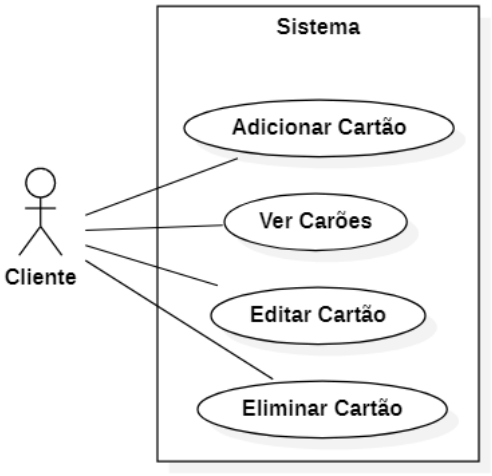


Figura 26 - DCU: Cartões.

De seguida, é apresentada a descrição sumária para cada caso de uso: adicionar cartão, ver cartões, editar cartão e eliminar cartão.

- **<u>Adicionar Cartão</u>**
  - **Nome**: Adicionar Cartão.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente adicione um cartão.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O cartão é adicionado.
  - **Descrição**: Neste caso de uso, o cliente adiciona um cartão que poderá ser utilizado para efetuar uma subscrição.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de adicionar um cartão.
    2. O cliente preenche a informação solicitada.
    3. O cartão é adicionado.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    2.a. Alternativamente, a informação inserida está incorreta.

2.b. O sistema avisa que ocorreu um erro.

2.c. Volta a 1.

3.a. O sistema não consegue comunicar com a Braintree e o cartão não é adicionado.

3.b. O sistema avisa o cliente do sucedido.

3.c. Volta para 1.

2.c.1./3.c.1. O caso de uso termina sem sucesso.

- **Ver Cartões**

  - **Nome**: Ver Cartões.
  - **Ator**: Cliente.
  - **Objetivo**: Possibilitar que o cliente visualize alguns dados dos seus cartões de crédito.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: É apresentada a lita de cartões que o cliente tem.
  - **Descrição**: Neste caso de uso, o cliente visualiza alguns dados dos seus cartões de crédito, como os últimos 4 dígitos do cartão, o tipo de cartão, o nome que está neste, data de criação, data de atualização e mês e ano de expiração.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de visualizar alguns dados dos cartões de crédito do cliente.
    2. O cliente indica que pretende visualizar os cartões.
    3. O sistema mostra os cartões e informação dos mesmos.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    3.a. Alternativamente, o sistema não consegue comunicar com o Braintree e não são mostrados os cartões.

    3.b. O sistema avisa do sucedido.

    3.c. Volta a 1.

    3.c.1. O caso de uso termina sem sucesso.

- **Editar Cartão**
  - **Nome**: Editar Cartão.
  - **Ator**: Cliente.
  - **Objetivo**: Este caso de uso permite que o cliente edite alguns dados de um cartão de crédito.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O cartão é editado.
  - **Descrição**: Neste caso de uso, o cliente tem a hipótese de editar um cartão de crédito, como o nome, o cvv e a data de expiração.
  - **Cenário principal**:
    1. O sistema disponibiliza três campos para edição do cartão: nome, cvv e data de expiração.
    2. O cliente preenche os dados que pretende alterar e submete a informação.
    3. O atualiza a informação.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    2.a. Alternativamente, a informação preenchida pelo cliente está incorreta.

    2.b. O sistema avisa que ocorreu um erro.

    2.c. Volta a 1.

    3.a. O sistema não consegue comunicar com o Braintree e o cartão não é editado.

    3.b. O sistema avisa que o cartão não foi editado.

    3.c. Volta a 1.

    2.c.1./ 3.c.1. O caso de uso termina sem sucesso.

- **Eliminar Cartão**
  - **Nome**: Eliminar Cartão.
  - **Ator**: Cliente.
  - **Objetivo**: Possibilitar que o cliente elimine um cartão de crédito.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O cartão de crédito é eliminado da base de dados da Braintree.

- **Descrição**: Neste caso de uso, o cliente elimina um cartão de crédito e este é removido da base de dados da Braintree.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de eliminar um cartão de crédito.
  2. O cliente indica que pretende eliminar um cartão.
  3. O sistema elimina o cartão.
  4. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  3.a. Alternativamente, o sistema não consegue comunicar com o Braintree e o cartão não é eliminado.

  3.b. O sistema avisa que o cartão não foi eliminado.

  3.c. Volta a 1.

  3.c.1. O caso de uso termina sem sucesso.

5.4.10.6 Estrutura

Neste ponto existem dois *endpoints,*{{api_url}}/api/transactions/cards/, que pode ser acedido através dos métodos:

- POST: Permite adicionar um cartão e os seus parâmetros encontram-se na Tabela 35.

Tabela 35 - Parâmetros do *endpoint* Adicionar Cartão.

Chave	Tipo
number	string
expiration_date	string, formato: “mm/yy” (*e.g.*: “10/22”)
cvv	string
cardholder_name	string

- GET: Possibilita a visualização dos cartões, sendo enviado parâmetros da Tabela 36, em formato JSON, para a equipa de *front-end*.

Tabela 36 - Parâmetros do *endpoint* Ver Cartões.

Chave	Tipo
bin	string
card_type	string

cardholder_name	string
created_at	string
expiration_month	string
expiration_year	string
last_4	string
token	string
updated_at	string

- PUT e PATCH: Possibilita que o cliente edite alguns parâmetros do cartão, sendo que no método PATCH pode alterar um ou mais campos e no método PUT tem de alterar todos os campos solicitados – o campo PUT não seria necessário, mas foi realizado para o caso da equipa de *front-end* achar necessário. Os parâmetros passíveis de serem editados são os que estão presentes na Tabela 37.

Tabela 37 - Parâmetros do *endpoint* Editar Cartão.

Chave	Tipo
id	string (apenas para a equipa de front-end saber qual é o cartão a editar)
cardholder_name	string
cvv	string
expiration_date	string, formato: “mm/yy” (*e.g.*: “10/22”)

e ainda o *endpoint* {{api_url}}/api/transactions/cards/{id}, que pode ser acedido com recurso aos métodos:

- GET: Permite visualizar os dados de um cartão específico, estando os elementos presentes que são enviados para a equipa de *front-end* em formato JSON na Tabela 36.
- DELETE: Possibilita que um cliente elimine um cartão de crédito.

5.4.10.7 Implementação

A nível da implementação dos cartões, que é feita na classe *CreateCardViewSet*, e é chamada com recurso a dois *endpoints*. O *endpoint* {{api_url}}/api/transactions/cards/, onde é permitido chamar o método POST, GET e DEELTE e o *endpoint* {{api_url}}/api/transactions/cards/{id}, que pode se acedido através do método GET, sendo o id a identificação do cartão.

No primeiro *endpoint*, para que o método POST seja possível, foi inserido no parâmetro da classe *CreateModelMixin* e depois foi criado o método *create* para a inserção do cartão mediante os dados fornecidos pelo utilizador. Inicialmente é feita a verificação se o cliente já tem uma conta no Braintree, se não tiver é criada, ficando o id da conta do Braintree registada na base de dados da Sentilant, de forma a que seja possível fazer a relação entre o Cliente da Sentilant e a sua conta do Braintree. De seguida, é feita a procura pelo cliente na Braintree *customer = gateway.customer.find(…)* para depois, se este for encontrado, ser criado o cartão customer.gateway.payment_method_nonce.gateway.credit_card.create({….}). Sendo feita, naturalmente, a verificação se todos os campos inseridos pelo utilizador cumprem os requisitos estipulados pela Sentilant, como por exemplo a data estar no formato “mm/yyyy”, onde mm é o mês, *e.g.* 12 e o “yyyy” é o ano, *e.g.* 2022.

Neste *endpoint*, é possível também editar alguns parâmetros do seu cartão, para tal, com recurso ao método PATCH, e com o parâmetro *UpdateModelMixin* na classe, foi criado o método *patch*. Neste, é feita a verificação se os dados introduzidos pelo utilizador estão todos corretos e, caso estejam, é feita a sua atualização na plataforma Braintree: *gateway.credit_card.update(…)*.

Ainda neste *endpoint*, é possível que o utilizador visualize todos os seus cartões e respetivos dados através do método GET. Para que fosse possível esta chamada, foi inserido na classe o parâmetro *ListModelMixin* e criado o método *list*, sendo feita a procura pelo cliente na Braintree (gateway.customer.find(…), para depois mostrar os dados do cartão, retornando os mesmos em formato JSON.

Para eliminar um cartão com recurso ao mesmo *endpoint*, com o método DELETE, foi inserido o parâmetro *DestroyModelMixin* na classe *CreateCardViewSet* e criado o método dentro desta o método *destroy*, que faz inicialmente uma procura pelo id do cliente na *Braintree customer = gateway.customer.find(…)*, para depois procurar pelo cartão que o utilizador indicou que pretende eliminar, *customer.gateway.payment_method_nonce.gateway.credit_card.find(…)*. Caso seja encontrado, então é eliminado e o utilizador informado. Se não for encontrado, naturalmente o cliente também é informado do sucedido.

Caso o utilizador pretenda visualizar apenas os dados de um cartão apenas, então pode fazer recorrendo ao *endpoint* {{api_url}}/api/transactions/cards/{id} e método GET. Sendo que, ao fazer este pedido, é chamada a classe *CreateCardViewSet* e o método dentro desta *retrieve*, sendo que no parâmetro da classe tem de estar *RetrieveModelMixin* para que seja possível a sua chamada. Neste método é feita a procura pelo cliente na plataforma Braintree e depois pelo cartão, *customer = gateway.customer.find(…) e credit_card = customer.gateway.payment_method_nonce.gateway .credit_card.find(…)* e, de seguida, é feita a conversão da informação do cartão para JSON para que o utilizador a possa visualizar.

5.4.10.8 Testes Unitários

- Adicionar cartão:
    - test_create_card_and_find_all: é feita a verificação quando se adiciona um cartão se corre tudo bem, caso corra, é retornado um status *200* e feita procura dos cartões que existam, confirmando a sua informação.
    - test_create_card_and_find_it: é testada a criação de um cartão e feita a procura desse cartão, sendo retornado um *status* 200 e feita a verificação do que é retornado.
    - test_create_card_wrong_date: se a data inserida tiver passado, não é aceite e é retornado um *status* 400.
    - test_create_card_empty_name: o campo nome, como é opcional, caso esteja vazio, é aceite e retornado um status *200.*
    - test_create_card_empty_fields: quando se tenta criar um cartão e os campos de preenchimento estão vazios, a solicitação não é aceite e é retornado um *status* 400.
    - test_create_card_empty_expiration: ao tentar criar um cartão com o campo da data de validade vazio, não é aceite, sendo retornado um *status* 400.
    - test_create_card_empty_number: ao tentar ser criado um cartão com o campo do núemero do cartão vazio, este não é aceite. É apresentado um *status* 400.
- Ver cartão:
    - test_get_all_empty_cards: caso o cliente pretenda ver os dados de um cartão e este não exista, é retornado um *status* 400.
- Editar cartão:
    - test_put_card: recorrendo ao método PUT, quando se tenta altear os parâmetros do cartão todos e são submetidos conforme exigem as regras, é retornado um status 200, sendo feita a sua verificação que os dados são alterados.
    - test_put_card_empty_all: no método PUT, se os campos do cartão forem vazios, não é aceite e apresentado um *status* 400.
    - test_put_card_token_ok_empty_rest: no método PUT, se o *token* do cartão a alterar estiver correto e o resto dos parâmetros vazios, não é aceite e o *status* a apresentar é 400.
    - test_put_card_token_ok_no_expiration: no método PUT, se o campo de expiração do cartão for vazio, não é aceite e é retornado um *status* 400.

- test_patch_card: no método PATCH, quando a altearção dos dados cumpre os requisitos é retornado um *status* 200 e verificado que os dados alterados estão corretos.
- test_patch_card_empty_all: no método PATCH, se os campos submitidos estiverem vazios, não é aceite a alteração, sendo retornado um *status* 400.
- test_patch_card_token_ok_empty_rest: quando utilizado o método PATCH, e o *token* do cartão estiver correto e o resto da informação estiver vazia, é retornado um status 400, pois não é feita a alteração.
- test_patch_card_token_ok_no_expiration: no método PATCH, se o campo da validade do cartão não for alterado, o cartão é alterado na mesma, sendo recebido um *status* 200, sendo feita essa verificação no teste.

- Eliminar cartão:
  - test_delete_card: quando se pretende eliminar um cartão e a informação está toda em conformidade, este é eliminado sendo recebido um *status* 200*,* sendo feita também a verificação que o cartão já não existe, ao procurar pelo mesmo e ser apresentado um *status* 400*.*
  - test_delete_card_wrong_id: caso se tente eliminar um cartão com um id que não existe, este não é eliminado, sendo recebido um *status* 400.

5.4.11 Subscrições

5.4.11.1 Objetivo

O objetivo deste ponto é que o cliente possa gerir as suas subscrições, podendo fazer, ver, editar e eliminar subscrições.

5.4.11.2 Premissas

- O cliente para realizar as operações de criar, editar e eliminar subscrições tem de estar autenticado na loja da Sentilant.
- O cliente para criar uma subscrição tem de ter um cartão de crédito.
- Será definida a forma como o cliente pode fazer, editar e eliminar as subscrições.
- As funções administrativas são realizadas através do painel de administração do Django e do Braintree.

5.4.11.3 Estratégia

A estratégia para que o cliente possa fazer, visualizar, editar e eliminar subscrições foi a seguinte:

- O cliente:
    - Tem de estar autenticado na loja da Sentilant.
    - Para fazer uma subscrição o cliente tem de ter um cartão de crédito adicionado na loja da Sentilant, indicando o mesmo, referir o plano pretendido, a quantidade de *requests* que pretende e se prende que o plano não tenha um prazo de limite de utilização, ou se pretender que tenha uma data de validade, indicar o número de ciclos pretendidos. Passados esses ciclos, deixa de ser cobrado no cartão associado e deixa de ter acesso ao plano.
    - Para editar uma subscrição, o cliente pode escolher o cartão, se a subscrição nunca expira ou caso contrário, se expirar, escolher a sua duração.
    - Para visualizar uma subscrição o cliente indica a subscrição pretendida ou então pode visualizar todas as que tem e estas são mostradas, indicando se está ativa, a data de criação, atualização, dia do mês em que é feito o débito, dia do mês em que começou a ser cobrada, número de ciclos de validade, caso tenha uma data de validade, quantidade de *requests* adquiridos, nome do plano adquirido, informação sobre o período *Trial*, descrição da subscrição, total que o cliente paga e preço base do plano.
    - No que respeita ao cancelamento de uma subscrição, o cliente indica qual a subscrição que pretende cancelar.

5.4.11.4 Requisitos

Na Tabela 38, são apresentadas as tarefas relativas às descrições, com indicação da história do cliente.

Tabela 38 - Requisitos: Subscrições.

Título	História do Cliente
Fazer Subscrição	Como cliente da loja da Sentilant, quero ter a possibilidade de fazer uma subscrição.
Ver Subscrições	Como cliente da loja da Sentilant, quero ter a hipótese de ver as minhas subscrições e sua informação.
Editar Subscrição	Como cliente da loja da Sentilant, quero ter a hipótese de editar uma subscrição.

Cancelar Subscrição	Como cliente da loja da Sentilant, quero ter a possibilidade de cancelar uma subscrição.

5.4.11.5 Casos de uso

Os casos de uso para a tarefa das subscrições, estão identificados na Figura 27, assim como o ator que as pode realizar.

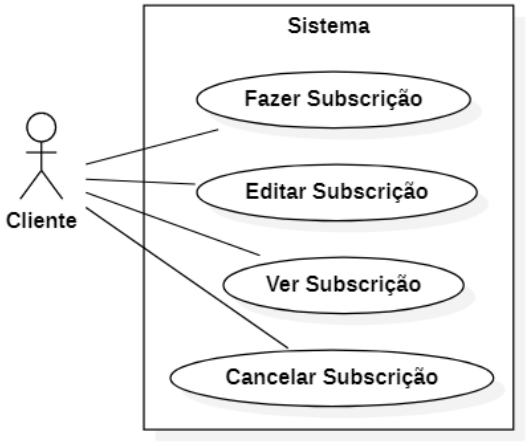


Figura 27 - DCU: Subscrições.

De seguida, é apresentada a descrição sumária para cada caso de uso: fazer subscrição, editar subscrição, ver subscrição e cancelar subscrição.

- **Fazer Subscrição**
  - **Nome**: Fazer Subscrição.
  - **Ator**: Cliente.
  - **Objetivo**: Possibilitar que o cliente faça uma subscrição.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant e ter previamente adicionado um cartão na loja da Sentilant.
  - **Pós-Condições**: O cliente fica com acesso ao serviço adquirido da Sentilant.
  - **Descrição**: Neste caso de uso, o cliente solicita uma subscrição e o Braintree verifica se o cliente está em condições, de ter acesso ao serviço da Sentilant, verificando os dados do cartão.
  - **Cenário principal**:
    1. O sistema apresenta um formulário para o cliente preencher.
    2. O cliente preenche o formulário e solicita a subscrição de um produto.

3. A Sentilant com recurso ao Braintree verifica se o cliente está em condições de subscrever e efetua o pagamento.
4. O caso de uso termina com sucesso.

- **Cenário Alternativo**:

  2.a. Alternativamente, a informação inserida está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  3.a. O sistema não consegue comunicar com a Braintree e não é feita a subscrição, ou o cliente não tem valor monetário para fazer o pagamento.

  3.b. O sistema avisa o cliente do sucedido.

  3.c. Volta para 1.

  2.c.1./3.c.1. O caso de uso termina sem sucesso.

➢ **<u>Ver Subscrição</u>**

- **Nome**: Ver Subscrição.
- **Ator**: Cliente.
- **Objetivo**: Possibilitar que o cliente visualize as suas subscrições.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: São apresentadas as subscrições e as informações destas ao cliente.
- **Descrição**: Neste caso de uso, o cliente visualiza as suas subscrições, com a informação destas: se está ativa, as datas de criação, atualização, validade, pagamento, o número de *requests* adquiridos, o nome do plano, caso tenha um período *Trial* a sua duração, o valor do pagamento, o preço base do plano, o número de ciclos de validade, caso tenha escolhido uma data limite para ser cobrada, e ainda a descrição da subscrição.
- **Cenário principal**:

  1. O sistema disponibiliza ao cliente a possibilidade de este visualizar as suas subscrições.
  2. O cliente indica que pretende visualizar as subscrições.
  3. O sistema mostra as subscrições.
  4. O caso de uso termina com sucesso.

- **Cenário Alternativo**:

  3.a. Alternativamente, o sistema não consegue comunicar com o Braintree e não são mostradas as subscrições.

  3.b. O sistema avisa do sucedido.

  3.c. Volta a 1.

  3.c.1. O caso de uso termina sem sucesso.

➢ **<u>Editar Subscrição</u>**

- **Nome**: Editar Subscrição.
- **Ator**: Cliente.
- **Objetivo**: Este caso de uso permite que o cliente edite uma subscrição.
- **Pré-Condições**: Estar autenticado na loja da Sentilant.
- **Pós-Condições**: A subscrição é editada.
- **Descrição**: Neste caso de uso, o cliente pode editar uma subscrição, sendo permitido alterar para se pretende não ter um prazo de termino da subscrição, ou se pretender o número de ciclos (meses) em que será cobrada a subscrição.
- **Cenário principal**:
  1. O sistema disponibiliza um formulário para o cliente preencher, com a opção alterar o cartão, se pretende que a subscrição tenha limite e caso seja afirmativo, o número de ciclos (meses) em que será cobrada a subscrição.
  2. O cliente preenche os dados que pretende alterar e submete a informação.
  3. O atualiza a informação.
  4. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  2.a. Alternativamente, a informação preenchida pelo cliente está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  3.a. O sistema não consegue comunicar com o Braintree e a subscrição não é editada.

  3.b. O sistema avisa que o cartão não foi editado.

  3.c. Volta a 1.

2.c.1./ 3.c.1. O caso de uso termina sem sucesso.

- **Cancelar Subscrição**
  - **Nome**: Cancelar Subscrição.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir ao cliente cancelar uma subscrição.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: O cliente fica com a subscrição cancelada, deixando de ter acesso ao plano que adquiriu.
  - **Descrição**: Neste caso de uso, o cliente cancela uma subscrição, não tendo mais acesso ao seu conteúdo, não sendo mais cobrada a subscrição.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de cancelar uma subscrição.
    2. O cliente indica que pretende cancelar uma subscrição.
    3. O sistema comunica com a Braintree para cancelar a subscrição e esta é cancelada.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    3.a. Alternativamente, o sistema não consegue comunicar com o Braintree e a subscrição não é cancelada.

    3.b. O sistema avisa que a subscrição não foi cancelada.

    3.c. Volta a 1.

    3.c.1. O caso de uso termina sem sucesso.

5.4.11.6 Estrutura

Neste ponto existem três *endpoints.* O endpoint{{api_url}}/api/transactions/ subscriptions/, que pode ser acedido através dos métodos:

- POST: Possibilita que o cliente faça uma subscrição de um plano previamente definido pela Sentilant. Os parâmetros necessários encontram-se na Tabela 39, sendo que o cliente só insere a quantidade de *requests* pretendidos, se tem uma data de validade e caso tenha o número de ciclos (meses) que pretende ser cobrado, a restante informação é enviada pelo *front-end* para fazer a sua triagem internamente.

Tabela 39 - Parâmetros do *endpoint* Fazer Subscrição.

Chave	Tipo
card_token	string
product_id	string
quantity	int
never_expires	boolean
cycles	int

- GET: Permite que o cliente consiga ver as suas subscrições, sendo enviado os parâmetros da Tabela 40 e Tabela 36, em formato JSON, para a equipa de *front-end*.

Tabela 40 - Parâmetros do *endpoint* Ver Subscrições.

Chave	Tipo
id	string
subscription_id	string
status	string
created_at	string
updated_at	string
billing_day_of_month	string
billing_period_start_date	string
billing_period_end_date	string
number_of_billing_cycles	string
current_billing_cycle	string
first_billing_date	string
never_expires	string
next_bill_amount	string
next_billing_date	string
next_billing_period_amount	string
plan_id	string
plan_name	string
plan_price	string

trial_duration	string
trial_period	string
trial_duration_unit	string
description	string
subscription_total	string
quantity	string
payment_method_token	string
user	string

- PATCH: Possibilita que o cliente altere a opção de a subscrição ter ou não uma validade e caso tenha decida o número de ciclos (meses) desta, cartão, estando estes parâmetros presentes na Tabela 41 – os restantes campos são para a equipa de desenvolvimento fazer a identificação da subscrição internamente.

Tabela 41 - Parâmetros do *endpoint* Editar Subscrição.

Chave	Tipo
id	int
card_token	string
subscription_id	string
never_expires	boolean
cycles	int

O *endpoint* {{api_url}}/api/transactions/subscriptions/{id}, que pode ser acedido com recurso ao método GET, permitindo este que o cliente visualize os dados de uma subscrição específica apenas, estando os parâmetros que são enviados para a equipa de *front-end*, em formato JSON, na Tabela 40.

Por fim, está disponível o *endpoint* {{api_url}}/api/transactions/subscriptions/, co o método POST, que permite que o cliente elimine uma subscrição, sendo enviado nos seus parâmetros o id da subscrição (o cliente apenas terá de selecionar a subscrição, o id será enviado ao selecionar o mesmo).

5.4.11.7 Implementação

Para que o cliente possa criar uma subscrição foi criado o *endpoint* {{api_url}}/api/transactions/subscriptions/ que pode ser acedido com o método POST

para este efeito, uma vez que tem nos seus parâmetros *CreateModelMixin*, que possibilita a chamada do POST e tem a classe *create* que foi implementada e recebe os dados como o id do plano pretendido, a quantidade do número de *requests* introduzida pelo utilizador, o cartão para o qual será descontado o valor, e se este pretende que o seu plano adquirido não tenha expiração ou caso contrario, se pretender que tenha, a duração a nível de meses deste. Estes dados são todos verificados neste método, sendo enviados que se estiverem em conformidade com o previsto, são enviados para a plataforma Braintree de forma a criar a subscrição: *gateway.subscription.create({…})*.

Caso este procedimento corra com sucesso, alguma informação é também guardada na base de dados da loja da Sentilant de forma a que seja possível depois aceder a essa informação aquando a visualização das subscrições sem ser necessário chamar o Braintree, o que iria demorar mais tempo a nível de *requests*. Assim, para que o utilizador possa visualizar todas as suas subscrições, ainda com recurso a ao *endpoint* {{api_url}}/api/transactions/subscriptions/, mas com o método GET, o utilizador pode consultar as mesmas, sendo que para que isso fosse possível foi inserido o parâmetro *ListModelMixin* na classe e criado o método *list*. Método este que vai permitir mostrar as subscrições do utilizador, depois de ser feita a seriação dentro do modelo Braintree para mostrar só os dados do utilizador: *Braintree.objects.filter(user=self.request.user)*.

Para que fosse possível o cliente editar as suas subscrições, foi inserido o parâmetro *UpdateModelMixin* na classe *CreateModelMixin* e pode ser acedido com recurso ao método PATCH com o endpoint {{api_url}}/api/transactions/subscription/, tendo sido criado o método patch nesta classe, onde faz a verificação dos dados introduzidos pelo utilizador e se estes foram válidos, então é feita a alteração na plataforma Braintree e também no *model* Brainteree da loja da Sentilant.

De salientar que sempre que ocorre uma alteração na plataforma da Braintree a nível das subscrições, essa informação é enviada para a Sentilant através de um *Webhook* que chama o *endpoint* {{api_url}}/api/transactions/braintree/webhooks/. Este *endpoint* que recebe um método POST, vai chamar o método *webhook*, que recebe essa informação e faz a atualização na base de dados da loja da Sentilant, estando assim a base de dados da Sentilant sempre atualizada.

Caso o cliente pretenda visualizar apenas uma subscrição, esse processo é feito com recurso ao método GET e *endpoint* {{api_url}}/api/transactions/subscriptions/{id}, sendo o id a identificação da subscrição. Este *endpoint* vai chamar a classe *CreateModelMixin*, e o método *retrieve* que faz a filtragem pelo id da subscrição e pelo utilizador, recorrendo ao *model* Braintree da loja da Sentilant, de forma a mostrar a informação pretendida e é possível ser acedido por se ter inserido nos parâmetros da classe *RetrieveModelMixin*.

Se o cliente tiver intenção de cancelar uma subscrição, pode ser feita com o método POST e o *endpoint* {{api_url}}/api/transactions/subscriptions/, que chama a classe *CancelSubscriptionViewSet*, onde é feita também a verificação se o utilizador está

autenticado (*permission_classes = (IsAuthenticated,)*) e se tem um token *(authentication_classes = (TokenAuthentication,))* e que tem como parâmetro *CreateModelMixin*, que permite receber o método POST. Ao ser feito a solicitação de cancelamento, é chamado o método dentro desta classe *create*, que faz o cancelamento recorrendo ao Braintree: *gateway.subscription.cancel(id)*. Caso tenha corrido com sucesso, essa informação também é alterada na base de dados da loja da Sentilant.

5.4.11.8 Testes Unitários

- Fazer subscrição:
    - test_create_subscription: é confirmado que é retornado um *status* 200 quando se faz uma subscrição e tudo corre conforme previsto, sendo verificada que a informação foi adicionada.
    - test_create_subscription_empty_get_with_card: caso não seja apresentado nada no campo do cartão ao criar uma subscrição, a operação não é válida e é retornado um *status* 400.
    - test_create_subscription_empy_fields: caso se tente criar uma subscrição com campos vazios, é retornado um *status* 400, não sendo aceite a operação.
    - test_create_subscription_negative_qty: se a quantidade inserida na criação de um cartão for negativa, não é feita subscrição, sendo retornado um *status* 400.
    - test_create_subscription__qty_0: foi testada a quantidade inserida a 0, tendo sido confirmado que o resultado era o esperado, não sendo aceite e um *status* 400.
- Editar subscrição:
    - test_update_subscription: caso o utilizador pretenda fazer uma atualização do conteúdo do cartão e corra tudo bem, foi feita essa verificação, ou seja, que a alteração é feita e que é recebido um *status* 200.
    - test_update_subscription_string_never_expires: se for enviada uma *string* com conteúdo aleatório, por exemplo, no campo da expiração, essa operação não é válida e é retornado um *status* 400.
    - test_update_subscription_wrong_card: caso os dados do cartão submetidos sejam invádlios, não é aceite a operação e retornado um *status* 400.

- test_update_subscription_wrong_subscription_id: na eventualidade de ser enviado um id da subscrição para atualizar errado, a operação não é concretizada, como esperado, e é retornado um *status* 400.

- Ver subscrição:
  - test_create_2_subscriptions_get_all: foi testado a subscrição de duas operações. Como espectável o resultado é um *status* 200, tendo sido inseridos os parâmetros corretos. De seguida, foi verificado as subscrições existentes do cliente, confirmando que eram as espectáveis, sendo retornado também um *status* 200.
  - test_create_subscription_empty_get_all_no_cards: caso o cliente ainda não tenha subscrições e pretender visualizar as mesmas, é retornado um *status* 200, pois apesar de não existirem é retornado um campo vazio, ou seja, não ocorreu nenhum procedimento errado.
  - test_create_subscription_wrong_id_get: se se pretender ver uma subscrição específica e o id enviado for o errado, enão não é válida a operação e é retornado um *status* 400.
- Cancelar subscrição:
  - test_cancel_subscription: se corer tudo conforme previsto, o cancelamento é efetuado e apresentado um *status* 200.
  - test_cancel_subscription_wrong_id: se ao cancelar for enviado um id errado, a operação não é válida e é retornado um *status* 400.

5.4.12 Chaves

5.4.12.1 Objetivo

O objetivo deste ponto é permitir que o cliente faça a gestão das chaves API, podendo adicionar, visualizar e eliminar as chaves. Estas chaves permitem acesso produto da Sentilant a que essa chave pertence, desde que, naturalmente, seja válida.

5.4.12.2 Premissas

- O cliente para realizar as operações de gerar e visualizar as chaves tem de autenticado na loja da Sentilant e ter adquirido um produto, estando o mesmo dentro da validade.
- Será definida a forma como o cliente pode fazer, criar, visualizar e eliminar as chaves.
- As funções administrativas são realizadas através do painel de administração do Django.

5.4.12.3 Estratégia

A estratégia para que o cliente possa criar, visualizar e eliminar chaves foi a seguinte:

- O cliente:
    - Tem de estar autenticado na loja da Sentilant.
    - Tem de ter adquirido o produto para o qual quer gerar a chave e o produto tem de estar dentro da validade.
    - Para gerar uma chave o cliente indica o número de chaves que pretende gerar de um determinado produto que adquiriu, seja através de subscrição ou não.
    - Aquando da solicitação para visualizar as chaves criadas, este visualiza as chaves e também a que plano/produto pertence, podendo ver também as chaves que criou apenas para um determinado produto/plano.
    - Para eliminar uma chave, o cliente indica a chave a eliminar.

5.4.12.4 Requisitos

Na Tabela 42, são apresentadas as tarefas relativas às chaves, com indicação da história do cliente.

Tabela 42 - Requisitos: Chaves.

Título	História do Cliente
Gerar Chave	Como cliente da loja da Sentilant, quero ter a possibilidade de gerar chaves.
Ver Chaves	Como cliente da loja da Sentilant, quero ter a hipótese de ver as chaves criadas.
Eliminar Chave	Como cliente da loja da Sentilant, quer ter a hipótese de eliminar uma chave que tenha criado.

5.4.12.5 Casos de uso

Na Figura 28 é apresentado o diagrama de casos de uso para as chaves, identificando as suas funcionalidades e o ator que as pode executar.

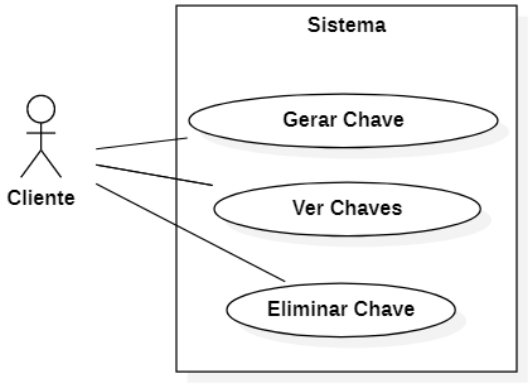


Figura 28 - DCU: Chaves.

De seguida, é apresentada a descrição sumária para os três casos de uso: gerar chave, ver chave e eliminar chave.

- **Gerar Chave**
  - **Nome**: Gerar Chave.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente gere uma chave API para que os seus utilizadores possam aceder ao conteúdo adquirido.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant e ter previamente adquirido um produto/plano da Sentilant.
  - **Pós-Condições**: O número de chaves que o cliente indicou são geradas.
  - **Descrição**: Neste caso de uso, o cliente indica o número de chaves que pretende ver geradas para um determinado produto que adquiriu na loja da Sentilant.
  - **Cenário principal**:
    1. O sistema apresenta um formulário para o cliente preencher, onde pode selecionar o produto para o qual pretende gerar chaves e o número de chaves a gerar.
    2. O cliente preenche o formulário e submete a informação.
    3. O sistema gera as chaves.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    2.a. Alternativamente, a informação inserida está incorreta.

    2.b. O sistema avisa que ocorreu um erro.

2.c. Volta a 1.

3.a. O produto não é válido e o sistema avisa o cliente do sucedido.

3.b. Volta para 1.

2.b.1./3.c.1. O caso de uso termina sem sucesso.

- **Ver Chaves**
  - **Nome**: Ver Chaves.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente visualize as chaves geradas.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: São apresentadas as chaves que o cliente gerou.
  - **Descrição**: Neste caso de uso, o cliente tem a opção visualizar as chaves API que gerou, para que as possa incluir no seu código de forma a que os seus clientes tenham acesso ao produto adquirido.
  - **Cenário principal**:
    1. O sistema apresenta ao cliente a possibilidade de visualizar as suas chaves geradas.
    2. O cliente indica que pretende ver as chaves.
    3. O sistema mostra as chaves.
    4. O caso de uso termina com sucesso.

- **Eliminar Chave**
  - **Nome**: Eliminar Chave.
  - **Ator**: Cliente.
  - **Objetivo**: Permitir que o cliente elimine uma chave gerada.
  - **Pré-Condições**: Estar autenticado na loja da Sentilant.
  - **Pós-Condições**: A chave é eliminada da base de dados.
  - **Descrição**: Neste caso de uso, o cliente tem a opção eliminar uma chave que gerou.
  - **Cenário principal**:
    1. O sistema apresenta ao cliente a possibilidade eliminar uma chave.
    2. O cliente indica a chave que pretende eliminar.

3. O sistema elimina a chave da base de dados.
4. O caso de uso termina com sucesso.

- **Cenário alternativo**:

  3.a. Alternativamente, o sistema não encontra a chave.

  3.b. O sistema avisa que ocorreu um erro.

  3.c. Volta a 1.

  3.c.1. O caso de uso termina sem sucesso.

5.4.12.6 Estrutura

Neste ponto existem dois *endpoints.* O *endpoint* {{api_url}}/api/users/keys/, que pode ser acedido através dos métodos:

- POST: Permite que o cliente gere uma chave, selecionando o produto para o qual pretende criar as chaves e o número de chaves pretendidas. Na Tabela 43 são apresentados os parâmetros deste *endpoint*.

Tabela 43 - Parâmetros do *endpoint* Gerar Chave.

Chave	Tipo
subscription	boolean
id	string
number	int

- GET: Este método permite que o cliente visualize as chaves todas que foram geradas, sendo apresentadas na Tabela 44 os parâmetros que são enviados para a equipa de *front-end*, por forma a que a equipa possa inserir a informação na interface. De salientar que ou é enviado o nome braintree ou item, conforme se trate de um produto adquirido individualmente ou de um plano, respetivamente.

Tabela 44 - Parâmetros do *endpoint* Ver Chaves.

Chave	Tipo
key	string
braintree/item	string

DELETE: Caso o cliente pretenda eliminar uma chave que tenha gerado, sendo apenas necessário indicar a chave que pretende eliminar.

O outro *endpoint* deste ponto, {{api_url}}/api/users/keys/{id}, pode ser acedido com recurso ao método GET, possibilitando ao cliente ver a lista de chaves criadas para um produto específico, sendo que a informação enviada para a equipa de *front-end* é a mesma que se encontra na Tabela 44.

5.4.12.7 Implementação

Para a concretização da implementação das chaves que permitem que os utilizadores acedam ao conteúdo da lógica da Sentilant, foi criada a classe *ApiKeyViewSet* que pode ser acedida, para criação das chaves, visualização e eliminação das mesmas, através do *endpoint* {{api_url}}/api/users/keys/ com o método POST para criar, o método GET para visualizar e o DELETE para eliminar uma chave, sendo que neste último apenas é necessário enviar a chave que se pretende remover. Para que fosse possível a criação de chaves e receber método o POST, foi inserido o parâmetro *CreateModelMixin* na classe e criado o método *create* dentro da classe.

Este método, que recebe a indicação se se trata de um plano ou um produto individual, assim como o id do mesmo e o número de chaves pretendidas para esse produto/plano, faz uma verificação se esses parâmetros são válidos e estão dentro da validade, assim como se o produto foi, efetivamente, encomendado. Caso se confirme, então são geradas as chaves, que são únicas, ficando essa informação registada na base de dados da Sentilant e podem ser consultadas com recurso ao mesmo *endpoint,* mas com o método GET. Para que este método fosse válido, foi introduzido na classe *ApiKeyViewSet* o parâmetro *ListModelMixin* e criado o método dentro da classe *list*, onde é feita a procura de chaves desse utilizador e retornado em formato JSON a chave e se pertence a uma subscrição ou a um produto, como nome do mesmo.

Se o cliente tiver intenção de visualizar as chaves para um produto/plano específico, pode ser feito com recurso ao método GET e *endpoint* {{api_url}}/api/users/keys/{id}, onde id é o id do produto/plano selecionado. Na classe que é chamada, *ApiKeyViewSet,* foi introduzido o parâmetro *RetrieveModelMixin* e o método *retrieve*, de forma a que fosse possível fazer a chamada e mostrar os resultados, que são enviados para a equipa de *front-end* em formato JSON. Assim, no método *retrieve,* é feita a verificação se se trata de uma subscrição ou um produto e mostradas as chaves, apenas desse utilizador, naturalmente, para o id pretendido.

Importa ainda salientar que para que seja possível aceder a esta classe, *ApiKeyViewSet,* o cliente tem de estar autenticado (*permission_classes = (IsAuthenticated,)*) e ter um token (*authentication_classes = (TokenAuthentication,*).

5.4.12.8 Testes Unitários

- Gerar chaves:
    - test_create_key: foi testada a criação de chaves com todos os parâmetros e condições favoráveis e correu conforme previsto, tendo sido retornado um *status* 200.
    - test_create_key_wrong_id: caso seja enviado um id da subscrição ou produto errado, é retornado um *status* 400, não sendo a operação válida.
    - test_create_key_string_number: se no campo do número for enviado uma string, é retornado um *status* 400, não sendo a operação permitida.
    - test_create_key_sbuscription_not_boolean: se no campo da subscrição, que é *Boolean*, for uma *string*, a operação não é válida e é retornado um *status* 400.
    - test_create_key_sbuscription_empty_data: é retornado um *status* 400 caso seja submetida informação necessária vazia.
    - test_create_key_negative_number: se o número de chaves a gerar for negativo, esta operação não é concretizada e é retornado um *status* 400.
    - test_create_key_zero_number: a operação não é validade se for submetido no número de chaves o valor de 0, sendo retornado um *status* 400.
- Ver chaves:
    - Para verificação de que tudo corre conforme previsto na visualização das chaves, foi testado aquando da criação das chaves (nos testes do ponto anterior), tendo a resposta sido positiva, ao ser retornado um *status* 200 e a verificação do conteúdo.
    - test_list_a_key_for_x_product_wrong_id: caso seja inserido um id errado, a operação não é validada e é retornado um *status* 400.
    - test_list_a_key_for_x_product_string_id: se for enviado uma string no lugar do id, é retornado um *status* 400, pois não é esperado que tal aconteça.

5.4.13 Autenticação na APP[48]

5.4.13.1 Objetivo

O objetivo deste ponto é permitir ao utilizador fazer não só o registo na APP da Sentilant como também permitir que este faça *login* e elimine a sua conta na APP.

[48] A pedido da Sentilant, toda a informação específica que envolve informação da Sentilant, com código ou informação específica de acesso, foi removida do relatório.

5.4.13.2 Premissas

- Para que o utilizador crie uma conta na APP da Sentilant, possa entrar nesta e eliminar a mesma, é necessário que este tenha uma chave válida.
- Será definida a forma como a APP vai permitir ao utilizador criar uma conta de utilizador, entrar e eliminar a mesma, mediante também a informação da parte da lógica da APP.
- As funções administrativas são realizadas através do painel de administração do Django e na APP da Sentilant.

5.4.13.3 Estratégia

A estratégia para que o utilizador faça o registo, *login* e eliminação da conta da APP da Sentilant foi:

- Verificar a lógica da APP da Sentilant e perceber qual o conteúdo que deve ser enviado pelo utilizador, que por motivos de confidencialidade não são descritos neste relatório.
- Apenas é permitido o acesso aos *endpoints* mediante uma API-key válida.

5.4.13.4 Requisitos

Na Tabela 45, são apresentadas as tarefas relativas ao processo de registo, autenticação e eliminação da conta da APP da Sentilant, com indicação da história do utilizador.

Tabela 45 - Requisitos: Autenticação.

Título	História do Utilizador
Fazer Registo na APP	Como utilizador, quero ter a possibilidade de fazer o registo na APP da Sentilant.
Efetuar *Login* na APP	Como utilizador, quero ter a hipótese efetuar o *login* na APP.
Eliminar Conta na APP	Como utilizador, quero ter a possibilidade de eliminar a conta da APP da Sentilant.

5.4.13.5 Casos de uso

A Figura 29 apresenta o diagrama de casos de uso para a autenticação da APP da Sentilant, assim como o seu ator.

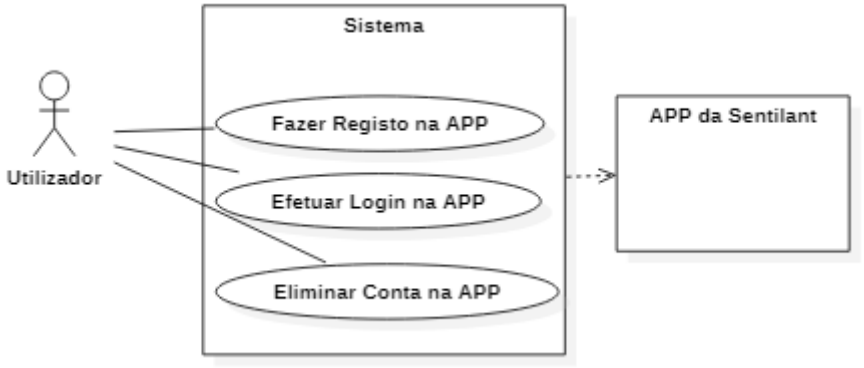


Figura 29 - DCU: Autenticação.

De seguida, é apresentada a descrição sumária para os três casos de uso: fazer registo na APP, efetuar *login* na APP e eliminar conta na APP.

- **Fazer Registo na APP**
  - **Nome**: Fazer Registo na APP.
  - **Ator**: Utilizador.
  - **Objetivo**: Possibilitar que o utilizador faça um registo na APP.
  - **Pré-Condições**: Ter uma chave de acesso (API-key).
  - **Pós-Condições**: O utilizador fica com uma conta na APP.
  - **Descrição**: Neste caso de uso, o utilizador criar uma conta de utilizador na APP da Sentilant, recorrendo ao endpoint disponibilizado pela API criada pelo estagiário.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção para o utilizador fazer o registo na APP.
    2. O utilizador indica que pretende fazer um registo.
    3. O sistema apresenta um formulário.
    4. O utilizador insere a informação solicitada para fazer o registo.
    5. O sistema comunica com a APP da Sentilant e faz o registo na APP.
    6. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    4.a. Alternativamente, a informação inserida está incorreta.

    4.b. O sistema avisa que ocorreu um erro.

    4.c. Volta a 1.

4.b.1. O caso de uso termina sem sucesso.

- **Efetuar Login na APP**
  - **Nome**: Efetuar *Login* na APP.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador entre na sua conta de utilizador da APP.
  - **Pré-Condições**: Ter uma chave de acesso (API-key).
  - **Pós-Condições**: O utilizador fica com o *login* efetuado.
  - **Descrição**: Neste caso de uso, o utilizador faz o *login* na APP da Sentilant.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção para fazer o *login* na APP da Sentilant.
    2. O utilizador indica que pretende efetuar o *login*.
    3. O sistema apresenta o formulário de *login*.
    4. O utilizador insere a informação solicitada.
    5. O sistema comunica com a APP da Sentilant e esta permite o acesso ao utilizador dos produtos adquiridos.
    6. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    4.a. Alternativamente, a informação inserida está incorreta.

    4.b. O sistema avisa que ocorreu um erro.

    4.c. Volta a 1.

    4.b.1. O caso de uso termina sem sucesso.

- **Eliminar conta na APP**
  - **Nome**: Eliminar Conta na APP.
  - **Ator**: Utilizador.
  - **Objetivo**: Possibilitar que o utilizador elimine a sua conta da APP.
  - **Pré-Condições**: Ter uma chave de acesso (API-key).
  - **Pós-Condições**: O utilizador deixa de ter acesso àquela que era a sua conta.
  - **Descrição**: Neste caso de uso, o utilizador elimina a sua conta da APP da Sentilant.

- **Cenário principal**:
    1. O sistema disponibiliza a opção para o eliminar a sua conta de utilizador da APP da Sentilant.
    2. O utilizador indica que pretende eliminar a conta.
    3. O sistema comunica com a APP da Sentilant e esta elimina a conta do utilizador.
    4. O caso de uso termina com sucesso.

5.4.13.6 Estrutura

Neste ponto existem três *endpoints.* O *endpoint* {{api_url}}/api/v1/drivings/new/, que pode ser acedido através do método POST.

O segundo *endpoint*, {{api_url}}/api/v1/drivings/logins/, pode ser acedido com recurso ao método POST e permite que o utilizador faça *login* na APP da Sentilant, enviado os parâmetros solicitados.

Quanto ao terceiro *endpoint,* {{api_url}}/api/v1/drivings/users/, pode ser acedido através do método POST e permite eliminar a conta do utilizador na APP.

5.4.13.7 Implementação

Para que o utilizador possa fazer um registo na APP da Sentilant, este tem que ter uma chave (API-key) válida gerada na loja da Sentilant, pelo que essa verificação é feita no início da classe *NewUserViewSet,* que é chamada com recurso ao método POST no *endpoint* {{api_url}}/api/driving/new/, através da seguinte operação: *permission_classes = [HasAPIKey]*, sendo feita também a gestão a nível dos pedidos com a operação *throttle_classes = (ScopedRateThrottle,)*. Operações estas que também são feitas nas duas classes que se seguem.

Para que fosse possível fazer POST, foi introduzido na classe o parâmetro *CreateModelMixin* e criado nesta classe o método *create* que recebe a informação introduzida pelo utilizador e vai fazer a sua verificação e fazer o pedido na APP da Sentilant para efetuar o seu registo: *requests.post(…)*.

Para que o utilizador efetue o *login*, através do método POST e do *endpoint* {{api_url}}/api/v1/drivings/logins/, foi criada a classe LoginViewSet com o parâmetro *CreateModelMixin*, para permitir receber o método POST, e depois foi criado o método *create* na classe onde vai fazer o login com a informação que o utilizador introduz*: response = requests.post(…)*. Se correr tudo conforme previsto, então é guardada a chave de acesso para que este possa entrar nas restantes partes adquiridas da aplicação.

Se o utilizador pretender remover a conta, pode fazer com recurso ao método POST e o *endpoint* {{api_url}}/api/v1/drivings/users/, sendo depois chamada a classe

UserViewSet que tem como um dos parâmetros *CreateModelMixin*, para que seja possível o método POST. Dentro da classe, foi criado o método *create* onde é feita a ligação com a APP da Sentilant para eliminar a conta de utilizador nessa APP através do pedido *requests.delete(…)*.

5.4.13.8 Testes Unitários

Nestes testes unitários são apresentados os testes que dizem respeito a ligações feitas a subscrições, sendo a parte final do nome do método “subs”. Os mesmos testes foram verificados para as compras de produtos via encomendas, tendo os métodos o mesmo nome, mas sem o “subs”.

- Fazer registo na APP:
    - test_new_user_sentilant_and_already_exists_subs: na eventualidade do utilizador tentar criar uma conta na APP com um *username* que já exista, o resultado é o esperado, não é permitido, sendo retornada essa informação para que o utilizador saiba que já existe um utilizador, tendo sido feito o teste de comparação a nível da *string* que é retornada.

      Neste método, foi também verificada a criação normal de uma conta, sendo retornado um *status* 201. A razão deste procedimento ser feito neste método deve-se ao facto de para tentar criar uma conta de utilizador que já exista, ter de existir um utilizador com esse nome. Assim foi criada primeiro uma conta e depois tentada criar outra com esse nome de utilizador.
- Efetuar *login* na APP:
    - test_login_user_subs: este método verifica que se o utilizador inserir a informação correta, é feito o *login* e recebido um *status*200.
    - test_login_user_wrong_info_subs: Se o utilizador tentar entrar com a informação de *login* errada, é retornado um *status* 401, não sendo permitida a operação.
- Eliminar conta na APP:
    - test_delete_user_subs: foi verificado que se o utilizador pretender eliminar a sua conta e estiver tudo em conformidade, esta é eliminada e recebido um *status* 200.

5.4.14 Perfil na APP[49]

5.4.14.1 Objetivo

O objetivo deste ponto possibilitar ao utilizador visualizar e alterar o seu perfil da APP da Sentilant.

5.4.14.2 Premissas

- Para que o utilizador visualize e edite o seu perfil da APP, é necessário que este tenha uma chave válida.
- Será definida a forma como o utilizador pode ver e editar o seu perfil da APP, mediante também a informação da parte da lógica da APP.
- As funções administrativas são realizadas através do painel de administração do Django e na APP da Sentilant.

5.4.14.3 Estratégia

A estratégia para que o utilizador visualize e edite o seu perfil da conta na APP da Sentilant foi:

- Os dados enviados para editar o perfil são: país, nome e nif, sendo os mesmos campos passíveis de serem editados.
- Apenas é permitido o acesso aos *endpoints* mediante uma API-key válida.

5.4.14.4 Requisitos

Os requisitos que dizem respeito ao perfil do utilizador na APP da Sentilant estão presentes na Tabela 46, com a descrição da história do utilizador.

Tabela 46 - Requisitos: Perfil na APP.

Título	História do Utilizador
Ver Perfil na APP	Como utilizador, quero ter a possibilidade de visualizar o meu perfil na APP.
Editar Perfil na APP	Como utilizador, quero ter a hipótese de editar o meu perfil na APP.

5.4.14.5 Casos de uso

A Figura 30 apresenta a diagrama de casos de uso para o perfil da APP da Sentilant, identificando as tarefas possíveis e o ator que as pode executar.

[49] A pedido da Sentilant, toda a informação específica que envolve informação da Sentilant, com código ou informação específica de acesso, foi removida do relatório.

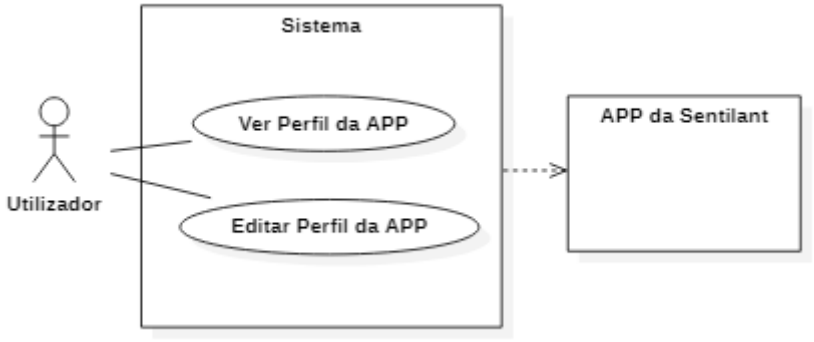


Figura 30 - DCU: Perfil na APP.

De seguida, é apresentada a descrição sumária para os dois casos de uso: ver perfil da APP e editar perfil da APP.

- **Ver Perfil na APP**
  - **Nome**: Ver Perfil na APP.
  - **Ator**: Utilizador.
  - **Objetivo**: Possibilitar que o utilizador visualize o seu perfil da APP.
  - **Pré-Condições**: Ter uma chave de acesso (API-key).
  - **Pós-Condições**: O utilizador visualiza o seu perfil da APP.
  - **Descrição**: Neste caso de uso, o utilizador visualiza o seu perfil, onde constam o país, o nome e o nif.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção para o utilizador ver o seu perfil da APP.
    2. O utilizador indica que pretende ver o perfil.
    3. O sistema comunica com a APP da Sentilant e apresenta o perfil.
    4. O caso de uso termina com sucesso.

- **Editar Perfil na APP**
  - **Nome**: Editar Perfil na APP.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador edite o seu perfil da APP.
  - **Pré-Condições**: Ter uma chave de acesso (API-key).

- **Pós-Condições**: A APP guarda a informação inserida pelo utilizador.
- **Descrição**: Neste caso de uso, o utilizador edita o seu perfil da APP, mediante a inserção de pelo menos um dos seguintes campos: pais, nome e nif.
- **Cenário principal**:
  1. O sistema disponibiliza a opção para editar o perfil da APP.
  2. O utilizador indica que pretende editar o perfil.
  3. O sistema apresenta o formulário de edição.
  4. O utilizador insere a informação solicitada.
  5. O sistema comunica com a APP da Sentilant e esta guarda a informação na base de dados.
  6. O caso de uso termina com sucesso.
- **Cenário Alternativo**:

  4.a. Alternativamente, a informação inserida está incorreta.

  4.b. O sistema avisa que ocorreu um erro.

  4.c. Volta a 1.

  4.b.1. O caso de uso termina sem sucesso.

5.4.14.6 Estrutura

Neste ponto existem o *endpoint* {{api_url}}/api/v1/drivings/users/, que pode ser acedido através dos métodos GET e PATCH. O método GET, permite que o utilizador visualize os campos do perfil e o método PATCH permite aos utilizadores alterarem um dos campos do perfil.

5.4.14.7 Implementação

Para que o utilizador possa visualizar o perfil quando faz o *request* com o método GET ao utilizar o *endpoint* {{api_url}}/api/v1/drivings/users/, é chamada a classe *UserViewSet*, que tem o parâmetro *ListModelMixin* que permite receber o GET e depois foi criado dentro desta classe o método *list* para que seja feito o *request requests.get(…)* para receber a informação da APP da Sentilant.

Se o utilizador pretender editar o perfil, com recurso ao mesmo *endpoint* mas com o método PATCH, a classe que chamada também é a *UserViewSet,* onde consta nos parâmetros *UpdateModelMixin* para que possa receber o PATCH, tendo sido depois criado o método nesta classe *patch* que vai fazer o pedido com a informação introduzida pelo utilizador: *requests.patch(…)*.

Importa ainda salientar que para ver o perfil ou editar é necessário ter uma API-Key válida, pelo que foi introduzido no início da classe essa verificação: *permission_classes = [HasAPIKey]*, sendo feita ainda a verificação dos *requests: throttle_classes = (ScopedRateThrottle,)*.

5.4.14.8 Testes Unitários

- Ver e editar perfil da APP:
    - test_patch_and_get__profile_user_subs: foi testada a edição do perfil com recurso ao método PATCH inserindo os campos espectáveis, tendo sido retornado um *status* 200 e feita a verificação que o conteúdo foi alterado, retornando o conteúdo através do método GET.
    - test_patch_profile_empty_data_subs: caso se envie informação sem dados, é recebido um *status* 400*,* não sendo feita nenhuma alteração.

5.4.15 Veículos[50]

5.4.15.1 Objetivo

O objetivo deste ponto é permitir ao utilizador criar, editar, ver e apagar veículos, assim como ver as especificações destes.

5.4.15.2 Premissas

- Para que o utilizador possa criar, editar, ver e apagar veículos, tem de ter uma chave de acesso válida.
- Será definida a forma como a APP permitirá criar, editar, apagar e ver as especificações dos veículos, mediante também a informação da parte da lógica da APP.
- As funções administrativas são realizadas através do painel de administração do Django e na APP da Sentilant.

5.4.15.3 Estratégia

A estratégia para que o utilizador possa realizar as operações relativas aos veículos foi:

- A APP permite que os utilizadores adicionem veículos, mediante a lógica da APP da Sentilant.

[50] A pedido da Sentilant, toda a informação específica que envolve informação da Sentilant, com código ou informação específica de acesso, foi removida do relatório.

- A APP permite que os utilizadores visualizem e editem a informação dos veículos.
- A APP para mostrar a especificação dos veículos aos utilizadores pode mostrar sem nenhum filtro ou com filtro.
- Apenas é permitido o acesso aos *endpoints* dos veículos mediante uma API-key válida.

5.4.15.4 Requisitos

Na Tabela 47, são apresentadas as tarefas relativas aos veículos, com a descrição da história da APP.

Tabela 47 - Requisitos: Veículos.

Título	História do Utilizador
Adicionar Veículo	Como utilizador, pretendo ter a hipótese de adicionar veículos.
Editar Veículo	Como utilizador, quero ter a possibilidade de editar os meus veículos.
Listar Veículos	Como utilizador, quero ter a possibilidade de listar os meus veículos.
Apagar Veículo	Como utilizador, quero ter a hipótese de apagar um veículo que tenha inserido.
Ver Especificações	Como utilizador, quero ter a hipótese de ver as especificações dos veículos.

5.4.15.5 Casos de uso

No que respeita aos casos de uso, a Figura 31 apresenta os casos de uso para os veículos e o ator que pode executar as tarefas.

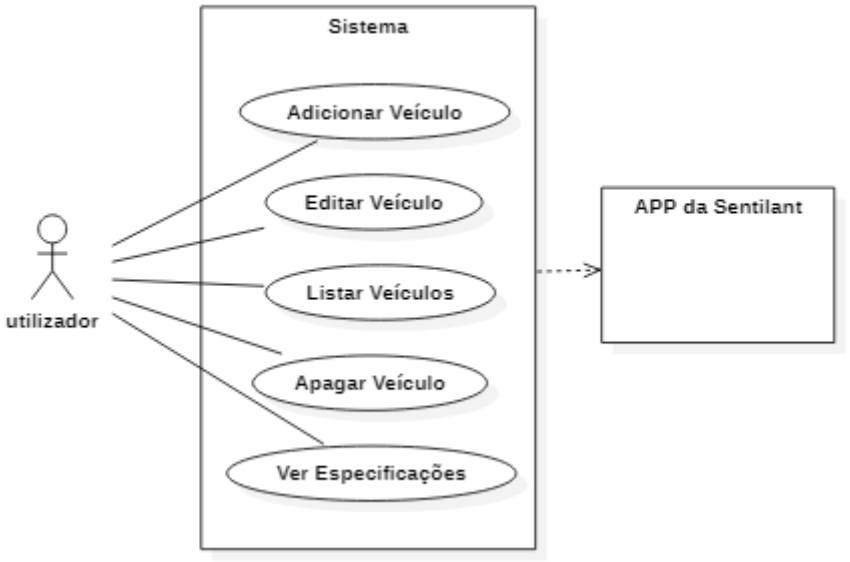


Figura 31 - DCU: Veículos.

De seguida, é apresentada a descrição sumária para cada caso de uso: Adicionar veículo, editar veículo, listar veículos, apagar veículo e ver especificações.

- **<u>Adicionar Veículo</u>**
  - **Nome**: Adicionar Veículo.
  - **Ator**: Utilizador.
  - **Objetivo**: Adicionar um Veículo.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: A informação adicionada fica registada na base de dados da Sentilant.
  - **Descrição**: Neste caso de uso, a APP adiciona um veículo sob solicitação de um utilizador, após este preencher os dados solicitados.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção adicionar um veículo.
    2. O utilizador preenche e submete a informação.
    3. O sistema comunica com a APP da Sentilant e esta adiciona o veículo.
    4. O caso de uso termina com sucesso.
  - **Cenário Alternativo**:

    2.a. Alternativamente, a informação inserida está incorreta.

    2.b. O sistema avisa que ocorreu um erro.

2.c. Volta a 1.

2.c.1. O caso de uso termina sem sucesso.

- **<u>Listar Veículos</u>**
  - **Nome**: Listar Veículos.
  - **Ator**: Utilizador.
  - **Objetivo**: Listar os veículos do utilizador.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: São listados os veículos do utilizador.
  - **Descrição**: Neste caso de uso, a APP lista todos os veículos de um utilizador, com a informação dos mesmos.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de listar os veículos.
    2. O utilizador seleciona a opção de listar os veículos.
    3. O sistema comunica com a APP da Sentilant e esta retorna a lista os veículos.
    4. O caso de uso termina com sucesso.

- **<u>Editar Veículo</u>**
  - **Nome**: Editar Veículo.
  - **Ator**: Utilizador.
  - **Objetivo**: Editar um Veículo.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: A edição fica registada na base de dados da Sentilant.
  - **Descrição**: Neste caso de uso, a APP permite ao utilizador fazer a edição de um veículo podendo alterar as funcionalidades disponibilizadas.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de editar o veículo.
    2. O utilizador edita o veículo.
    3. O sistema comunica com a APP da Sentilant e esta guarda a informação na base de dados.
    4. O caso de uso termina com sucesso.

- **Cenário Alternativo**:

  2.a. Alternativamente, a informação inserida está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  2.c.1. O caso de uso termina sem sucesso.

➢ **Apagar Veículo**

- **Nome**: Apagar Veículo.
- **Ator**: Utilizador.
- **Objetivo**: Apagar um Veículo.
- **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
- **Pós-Condições**: O veículo indicado pelo utilizador é eliminado.
- **Descrição**: Neste caso de uso, a APP possibilita ao utilizador fazer a eliminação de um veículo.
- **Cenário principal**:
  1. O sistema disponibiliza a opção de eliminar um veículo.
  2. O utilizador indica qual o veículo a apagar.
  3. O sistema comunica com a APP da Sentilant e esta elimina esse veículo da base de dados.
  4. O caso de uso termina com sucesso.

➢ **Ver Especificações**

- **Nome**: Ver Especificações.
- **Ator**: Utilizador.
- **Objetivo**: Ver as especificações dos veículos que constam na base de dados da APP da Sentilant.
- **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
- **Pós-Condições**: As especificações são mostradas ao utilizador.
- **Descrição**: Neste caso de uso, a APP possibilita ao utilizador visualizar as especificações dos veículos, podendo o utilizador escolher alguns parâmetros para limitar os resultados.
- **Cenário principal**:

1. O sistema disponibiliza a opção de visualizar as especificações dos veículos.
2. O utilizador seleciona a opção de ver especificações.
3. O sistema mostra os campos que o utilizador pode preencher para limitar os resultados.
4. O utilizador preenche a informação.
5. O sistema comunica com a APP da Sentilant e esta retorna a lista de especificações que é mostrada ao utilizador.
6. O caso de uso termina com sucesso.

- **Cenário alternativo:**

  4.a. Alternativamente, a informação inserida está incorreta.

  4.b. O sistema avisa que ocorreu um erro.

  4.c. Volta a 1.

  4.c.1. O caso de uso termina sem sucesso.

5.4.15.6 Estrutura

Neste ponto existem três *endpoints:*

O *endpoint* {{api_url}}/api/v1/drivings/vehicles/users/, que pode ser acedido através dos métodos:

- POST: Possibilita que o utilizador adicione um veículo, introduzindo os parâmetros solicitados.
- GET: Permite que os utilizadores visualizem os dados dos seus veículos, que são enviados, em formato JSON, para a equipa de *front-end*, de forma a que sejam depois inseridos na interface da APP.

O segundo *endpoint* deste ponto, {{api_url}}/v1/api/drivings/vehicles/users/{id}/, pode ser acedido com os métodos:

- GET: Permite que os utilizadores visualizem a informação de um veículo específico.
- PATCH: Possibilita que o utilizador edite os seus veículos, podendo alterar os parâmetros que disponibilizados.
- DELETE: Permite que os utilizadores eliminem um dos seus veículos, indicando qual pretendem eliminar.

O último *endpoint* deste tópico, {{api_url}}/api/v1/drivings/vehicles_specifications/{Query_params}, faz uso do método GET e permite ver as especificações dos veículos conforme se encontram na APP da Sentilant, sendo a informação enviada para o *front-*

end, em formato JSON, de forma a que estes façam a triagem do que será mostrado ao utilizador final.

5.4.15.7 Implementação

No que respeita aos veículos, quando o utilizador faz um pedido através do *endpoint* {{api_url}}/api/v1/drivings/vehicles/users/, é chamada a classe *UserVehicleViewSet*, onde é feita a verificação se o utilizador tem um API-Key válida *(permission_classes = [HasAPIKey])*, e também o controlo do número de request feito pelo utilizador, só permitindo aqueles que este adquiriu. Essa verificação é feita através das operações: *throttle_scope = 'vehicles'*, indicando que se trata de um campo relativo aos veículos e depois com recurso a *throttle_classes = (ScopedRateThrottle,)*, para fazer esse controlo de *requests*.

Quando é utilizado o método POST no *endpoint* para inserir um veículo e é chamada a referida classe que tem o parâmetro *CreateModelMixin* para permitir a sua chamada e tem o método *create* que verifica também que a API-key enviada pertence ao leque que permite o acesso aos veículos, e depois é enviada a informação para a APP da Sentilant com o *request*: *requests.post(…)*.

Se no *endpoint* for utilizador o método GET, para visualizar a lista de veículos do utilizador, é chamada a mesma classe que tem o parâmetro *ListModelMixin* que permite receber o método GET. Dentro da classe foi criado o método *list,* que faz o *request* para a APP da Sentilant: *requests.get(…)* depois de verificar que a chave enviada também diz respeito a veículos.

Para que esta classe possa receber o método PATCH do referido *endpoint*, de forma a que possa ser possível editar os veículos*,* foi inserido o parâmetro na classe *UpdateModelMixin* e criado o método *partial_update*, que depois de verificar que a API-key diz respeito a veículos, envia os dados para a APP da Sentilant para fazer a edição através do método PATCH: *requests.patch(…)*.

Com o método DELETE é possível eliminar um veículo neste *endpoint.* Para que fosse possível esse método, na classe *UserVehicleViewSet*, foi inserido como um dos parâmetros *DestroyModelMixin* e criado o método *destroy*, onde depois de ser feita a verificação que a chave API-key diz respeito aos veículos, é feito o pedido para a eliminação do veículo e respetiva eliminação: response = requests.delete(…).

Se o utilizador pretender ver um veículo específico, pode fazer com o método GET e com recurso ao *endpoint* {{api_url}}/api/v1/drivings/vehicles/users/{id}, sendo o id a identificação do veículo. Quando feito o pedido, é chamada a classe *UserVehicleViewSet* que depois das verificações faz o pedido na APP da Sentilat: response = requests.get(…).

Na secção dos veículos, ainda existe a opção de ver as especificações dos carros, tendo sido criado o *endpoint* {{api_url}}/api/v1/drivings/vehicles_specifications/{Query_params} que pode ser acedido com o método GET. Este vai chamar a classe

VehiclesSpecsViewSet, que faz as mesmas verificações que a classe *UserVehicleViewSet* (verificar o pedido de *requests* e se se trata de uma chave válida e que diz respeito a veículos*)*, com o parâmetro *ListModelMixin*, para permitir o método GET, e criado, dentro da classe, o método *list* que vai fazer o pedido para que a APP da Sentilant mostre as especificações: *requests.get(…).* Neste pedido, são enviados também os campos que o utilizador pode inserir para fazer um filtro na pesquisa.

5.4.15.8 Testes Unitários

- Editar veículos:
    - test_patch_vehicle_subs: se for feita uma edição através do método PATCH com os dados corretos, é retornado um *status* 200 e verificada se a informação está editada.
    - test_patch_vehicle_wrong_inspection_date_subs: é retornado um *status* 500*,* apesar do estagiário achar que faria mais sentido um *status* 400, caso se envie um campo na data que não o suposto, como uma *string* com conteúdo aleatório.
    - test_patch_vehicle_wrong_id_subs: se se tentar editar um veículo com o id deste errado, é retornado um *status* 400, não sendo feita a sua edição.
- Listar veículos:
    - test_get_vehicle_subs: caso o utilizador pretenda visualizar os seus veículos e submete a informação de forma correta, é retornado um *status* 200 com a informação pretendida.
    - test_get_vehicle_specific_subs_all: é retornado um *status* 200 caso o utilizador pretenda ver os dados de um veículo específico, quando o id deste está correto.
    - test_get_vehicle_specific_wrong_id_subs: caso o id do veículo solicitado não pertença ao utilizador ou não exista, é retornado um *status* 400, não sendo realizada a operação.
    - test_get_vehicle_specific_wrong_id_string_subs: se o id que é enviado para ver o veículo for uma *string*, a operação não é permitida, sendo retornado um *status* 400.
- Adicionar veículos:
    - test_post_vehicle_subs: se o veículo for adicionado com os campos corretos, é retornado um *status* 200 e verificado que foi adicionado esse veículo, fazendo a comparação com os dados introduzidos e os existentes na base de dados.

- test_post_vehicle_data_in_insurance_subs: é retornado um *status* 500 caso o utilizador insira uma *string* aleatória num dos campos, contudo, o estagiário acha mais indicado que seja retornado um *status* 400.

- Apagar veículo:
  - test_delete_vehicle_subs: se for feito o procedimento de eliminação correto, é recebido o *status* 204, indicando que correu tudo conforme previsto, sendo o veículo eliminado.
  - test_delete_vehicle_wrong_id_subs: é recebido um *status* 404 quando se for enviado um id errado, não sendo a operação concretizada.
  - test_delete_vehicle_string_id_subs: caso se tente eliminar um veículo com um id enviado como *string*, a operação não é concretizada e retornado um *status* 404.
- Ver especificações:
  - test_get_vehicle_scpecs_subs: foi feita a verificação que caso seja feita solicitação normal de especificações de veículos, é retornada a informação solicitada e um *status* 200.
  - test_get_vehicle_scpecs_with_params_subs: é retornado um *status* 200 e a respetiva informação, caso se faça uma filtragem a nível dos parâmetros opcionais predefinidos para esta operação.

Foram realizados mais testes unitários, contudo, por questões de confidencialidade, estes foram removidos do relatório.

5.4.16 Viagens[51]

5.4.16.1 Objetivo

O objetivo deste ponto é permitir ao utilizador criar, editar, ver e apagar viagens, assim como ver a pontuação das mesmas.

5.4.16.2 Premissas

- A APP para permitir que o utilizador possa criar, editar, ver e eliminar viagens, tem de confirmar que estes têm uma chave de acesso válida. O mesmo acontece quando o utilizador pretende ver a sua pontuação.
- Será definida a forma como a APP permitirá criar, editar, apagar e ver a pontuação das viagens, mediante também a informação da parte da lógica da APP.

[51] Por indicação da Sentilant, toda a informação específica que envolve informação da Sentilant, com código ou informação específica de acesso, foi removida do relatório.

- As funções administrativas das viagens são realizadas através do painel de administração do Django e na APP da Sentilant.

5.4.16.3 Estratégia

A estratégia para que o utilizador realize as operações relativas às viagens foi:

- Para permitir ao utilizador faça a gestão das suas viagens, o sistema tem de enviar os dados que são solicitados pela parte da lógica da APP da Sentilant para este efeito.
- Apenas é permitido o acesso aos *endpoints* dos veículos mediante uma API-key válida.

5.4.16.4 Requisitos

Na Tabela 48, são apresentadas as tarefas relativas às viagens, com a descrição da história do utilizador.

Tabela 48 - Requisitos: Viagens.

Título	História do utilizador
Criar Viagem	Como utilizador, pretendo ter a possibilidade de criar viagens.
Consultar Viagens	Como utilizador, quero ter a possibilidade de consultar as minhas viagens.
Editar Viagem	Como utilizador, quero ter a hipótese de editar as minhas viagens.
Eliminar Viagem	Como utilizador, quero ter a possibilidade de eliminar uma viagem que tenha realizado.
Consultar Pontuação	Como utilizador, quero ter a hipótese de consultar a pontuação das minhas viagens.

5.4.16.5 Casos de uso

A Figura 32 apresenta o diagrama de casos de uso no que respeita às viagens, de forma a identificar as tarefas possíveis para este tópico e também o seu ator.

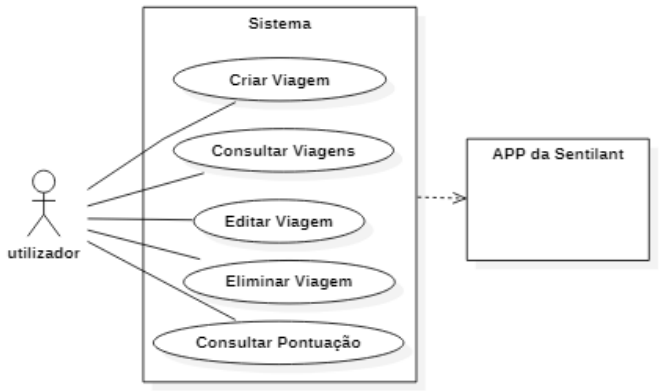


Figura 32 - DCU: Viagens.

De seguida, é apresentada a descrição sumária para cada caso de uso: criar viagens, consultar viagens, editar viagens, eliminar viagem e consultar pontuação.

- **Criar Viagem**
  - **Nome**: Criar Viagem.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador crie uma viagem.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: A viagem é criada e fica registada na base de dados da Sentilant.
  - **Descrição**: Neste caso de uso, o utilizador pode criar uma viagem.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção adicionar uma viagem.
    2. O utilizador adiciona a informação solicitada.
    3. O sistema comunica com a APP da Sentilant e esta adiciona a viagem.
    4. O caso de uso termina com sucesso.
  - **Cenário alternativo**:

    2.a. Alternativamente, a informação inserida está incorreta.

    2.b. O sistema avisa que ocorreu um erro.

2.c. Volta a 1.

2.c.1. O caso de uso termina sem sucesso.

- **<u>Editar Viagem</u>**
  - **Nome**: Editar Viagem.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador possa editar uma viagem.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: A informação editada fica registada na base de dados da Sentilant.
  - **Descrição**: Neste caso de uso, o utilizador pode fazer a edição de uma viagem que tenha realizado.
  - **Cenário principal**:

1. O sistema disponibiliza a opção editar uma viagem.
2. O utilizador edita a viagem.
3. O sistema comunica com a APP da Sentilant e esta guarda a informação na base de dados.
4. O caso de uso termina com sucesso.

- **Cenário alternativo**:

  2.a. Alternativamente, a informação inserida está incorreta.

  2.b. O sistema avisa que ocorreu um erro.

  2.c. Volta a 1.

  2.c.1. O caso de uso termina sem sucesso.

- **<u>Consultar Viagens</u>**
  - **Nome**: Consultar Viagens.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador consulte as suas viagens.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: São listadas as viagens efetuadas pelo utilizador.
  - **Descrição**: Este caso de uso permite ao utilizador ver as suas viagens.
  - **Cenário principal**:

1. O sistema disponibiliza a opção consultar viagens.
2. O utilizador seleciona a opção de consultar viagens.
3. O sistema comunica com a APP da Sentilant e esta mostra as viagens do utilzador.
4. O caso de uso termina com sucesso.

- **Eliminar Viagem**
  - **Nome**: Eliminar Viagem.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador elimine uma viagem.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: A viagem que o utilizador indicou é eliminada.
  - **Descrição**: Neste caso de uso, o utilizador pode eliminar uma viagem.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de eliminar uma viagem.
    2. O utilizador indica a viagem que pretende eliminar.
    3. O sistema comunica com a APP da Sentilant e esta elimina essa viagem da base de dados.
    4. O caso de uso termina com sucesso.

- **Consultar Pontuação**
  - **Nome**: Consultar Pontuação.
  - **Ator**: Utilizador.
  - **Objetivo**: Permitir que o utilizador visualize a pontuação das viagens.
  - **Pré-Condições**: O utilizador tem de estar autenticado no sistema.
  - **Pós-Condições**: O sistema mostra a pontuação das viagens.
  - **Descrição**: Neste caso de uso, o utilizador pode consutlar a pontuação das suas viagens.
  - **Cenário principal**:
    1. O sistema disponibiliza a opção de consultar a pontuação das viagens.
    2. O utilizador seleciona a opção de consultar a pontuação das viagens.

3. O sistema comunica com a APP da Sentilant e esta retorna a lista a informação sobre a pontuação.
4. O caso de uso termina com sucesso.

5.4.16.6 Estrutura

Neste ponto existem três *endpoints:*

O *endpoint* {{api_url}}/api/v1/drivings/trips/, que pode ser acedido através dos métodos:

- POST: Possibilita que o utilizador crie uma viagem.
- GET: Permite que os utilizadores visualizem os dados das suas viagens.

O segundo *endpoint* deste ponto, {{api_url}}/api/v1/drivings/trips/{id}/,pode ser acedido recorrendo aos métodos:

- GET: Permite que os utilizadores visualizem a informação de uma viagem específica.
- PATCH: que permite editar uma viagem.
- DELETE: Permite que os utilizadores eliminem uma viagem.

É apresentado ainda um *endpoint* que permite ver a pontuação: {{api_url}}/api/v1/ drivings/scores/. Este pode ser acedido através do método GET e vai receber a informação da pontuação geral do utilizador.

5.4.16.7 Implementação

Quando o utilizador pretende fazer uma operação com viagens a nível da sua criação, edição, eliminação ou visualização, faz uso do *endpoint* {{api_url}}/api/v1/drivings/trips/. Este endpoint vai chamar a classe *TripsViewSet* que faz a verificação se o utilizador tem uma API-key válida, *permission_classes = [HasAPIKey]*, e controla os requests a nível deste *endpoint*: *throttle_scope = 'trips'* e *throttle_classes = (ScopedRateThrottle,).* Tendo sido inseridos os parâmetros nesta classe *ListModelMixin*, *CreateModelMixin*, *DestroyModelMixin* e *UpdateModelMixin* para que seja possível receber os métodos GET, POST, DELETE e PATCH, respetivamente.

Se neste *endpoint* for utilizado um método POST, é chamada na classe *TripsViewSet* o método *create* que depois de fazer a verificação se efetivamente se trata de uma chave que permite o acesso às viagens, envia o conteúdo para a APP da Sentilant de forma a esta criar uma viagem: *requests.post(…)*.

Caso o utilizador pretenda visualizar os suas viagens é chamada o método *list,* que depois de fazer a verificação que a API-key diz respeito a viagens, vai fazer a chamada para que a APP da Sentilant devolva a lista dos seus veículos: requests.get(…).

Se o utilizador pretender fazer a edição dos veículos, é chamado o método *partial_update* que depois de fazer as verificações, faz o pedido de alteração na APP da Sentilant: requests.patch(…). Sendo o mesmo procedimento feito caso o utilizador pretenda eliminar uma viagem, ao ser chamado o método destroy e depois feito o pedido para a APP da Sentilant a eliminar, mas desta vez com recurso a um método *delete*: *requests.delete(…)*.

Se o utilizador tiver intenção de visualizar os dados de uma viagem específica, faz uso do *endpoint* {{api_url}}/api/v1/drivings/trips/{id}, sendo o id a identificação da viagem, com recurso ao método GET. Feito o pedido, é chamada a classe *TripsViewSet*, que tem como um dos parâmetros *RetrieveModelMixin*, de forma a permitir o método GET com parâmetros opcionais. Dentro desta classe encontra-se o método *retrieve* que vai fazer a solicitação da informação da viagem com recurso ao método GET: *requests.get(…)*.

Caso o utilizador pretenda ver a sua pontuação geral das viagens, faz uso do *endpoint* {{api_url}}/api/v1/drivings/scores/ recorrendo ao método GET. Esta operação vai chamar a classe *ScoresViewSet* que tem como parâmetro *ListModelMixin* para permitir a chamada do método GET.

Nesta classe, são feitas as mesmas verificações que na classe *TripsViewSet* para registar os *requests* e verificação da APi-Key. Sendo depois feito o pedido GET para que a APP da Sentilant devolva a pontuação geral através do pedido: *requests.get(…)*.

5.4.16.8 Testes Unitários

- Editar e consultar viagens:
    - test_patch_trips_user_list_a_trip_subs: caso se falta a alteração, com recurso ao método PATCH, normal da viagem, é verificado que esta é alterada e é retornado um *status* 200. Assim como é verificado o método GET retornado apenas a informação dessa viagem, já com os campos alterados, sendo retornado também um *status* 200.
    - test_list_trips_user_subs: é verificado que é retornado um *status* 200 quando o utilizador pretende ver as suas viagens, assim como o conteúdo destas é verificado que é apresentado.
    - test_patch_trips_wrong_id_subs: se for introduzido um id da viagem que não seja válido ou não pertença ao utilizador, a operação fica sem efeito, sendo retornado um *status* 400.
    - test_patch_trips_letter_id_subs: caso seja submetido uma letra no id da viagem é retornado um *status* 404.

- Criar viagens:
    - test_post_trips_user_subs: retorna um *status* 201, indicando que tudo correu conforme pretendido, caso o utilizador insira os campos previstos.
    - test_post_empty_data_trips_user_subs: é retornado um *status* 400, indicando que a informação enviada da parte do cliente está incorreta.
    - test_post_trips_user_id_wrong_subs: caso o campo do id do utilizador esteja incorreto, é retornado um *status* 400, não sendo realizada a operação.
- Eliminar viagens:
    - test_post_ok_and_delete_a_trip_wrong_id_subs: é verificado que caso o utilizador tente eliminar uma viagem e corre tudo bem o retorno será um *status* 200 e caso tente eliminar um veículo que já não existe, como aquele que acabou de ser eliminado, é retornado um *status* 400.
- Consultar pontuação:
    - test_get_scores_subs: é feita a verificação que caso o utilizador pretenda ver a pontuação geral, é retornado um *status* 200 e a informação da pontuação.

Foram realizados mais testes unitários, contudo, por questões de confidencialidade, estes foram removidos do relatório.

CAPÍTULO 6
6 CONCLUSÃO

Considera-se que o objetivo do estágio, construir uma API RESTful para suporte a aplicações móveis de caracterização de condução da Sentilant, isto é, aplicações que medem os hábitos de condução, em *back-end*, foi realizado com sucesso, tendo cumprido todos os objetivos propostos. Neste momento, o trabalho desenvolvido está pronto para a equipa de *front-end* poder desenvolver o seu trabalho e o produto ficar disponível.

O trabalho desenvolvido pelo estagiário consistiu, mais concretamente, em construir em *back-end* uma loja para que a Sentilant possa vender os seus produtos, como a gestão das viagens e dos veículos. Nessa loja, os clientes, depois de se registarem, podem ou adquirir um produto individualmente ou fazer uma subscrição de um plano que foi previamente definido pela Sentilant com recurso ao Braintree, uma plataforma de serviços de pagamentos. Na aquisição, os clientes escolhem o número máximo de *requests* por utilizador que querem comprar. Assim que os clientes fazem a compra de um determinado produto, podem gerar chaves de acesso para que possam permitir aos seus utilizadores terem acesso às funcionalidades adquiridas.

A nível de dificuldades do presente estágio, estas apresentaram-se inicialmente pelo facto de ser um tema novo para o estagiário, tendo este, de forma autónoma e autodidática, que investigar sobre o assunto, não só a nível teórico como a nível prático. Contudo, estas dificuldades foram, naturalmente, ultrapassadas com o decorrer do tempo, encontrando-se agora numa fase sólida de conhecimentos.

Outra dificuldade prendeu-se pelo facto de não existir uma metodologia por parte da empresa para com o estagiário. Como referido, a partir do dezembro, deixaram de existir reuniões semanais, e não foi implementada a metodologia prevista para a parte de desenvolvimento, algo que aumentou muito a dificuldade na realização das tarefas do estagiário, mas que este conseguiu ultrapassar com seriedade e dedicação.

Posto isto, importa salientar que o estágio permitiu ao estagiário adquirir conhecimentos fundamentais para o futuro, uma vez que não só desenvolveu um produto que permite ter muitas bases para o mercado de trabalho, visto que se trata de um tema que é muito atual, como também trabalhou com tecnologias muito utilizadas, como Python, uma das linguagens mais utilizadas, utilizando a framework Django, a *framework* mais utilizada para esta linguagem.

Este produto desenvolvido também mostra ser uma mais valia para a Sentilant, uma vez que permite aumentar o seu negócio, permitindo aos clientes ter mais uma possibilidade, a aquisição dos produtos da Sentilant recorrendo o trabalho desenvolvido pelo estagiário.

A nível de trabalho futuro, seria interessente ampliar os produtos disponibilizados pela Sentilant, inserindo também a parte social e a parte de gamificação, algo que será relativamente simples, uma vez que a loja desenvolvida pelo estagiário se encontra pronta para essa implementação, tendo de ser feita a ligação ao conteúdo já desenvolvido pela Sentilant que tem a parte da lógica. Fica ainda a sugestão, de, apesar de presumivelmente existir uma metodologia na Sentilant, que esta seja utilizada no desenvolvimento do estágio, pois o facto de não se utilizar uma metodologia complica muito a tarefa desenvolvida pelo estagiário, assim como para a própria empresa também não é benéfico.

Feitas estas considerações, não obstante os referidos obstáculos que foram aparecendo no decorrer do estágio, pode-se dizer que foi concluído com sucesso.

REFERÊNCIAS

Anderson, M. (21 de 08 de 2019). *Flask vs Django: A Comparative Guide to Python Frameworks*. Obtido em 16 de 05 de 2020, de https://www.iflexion.com/blog/python-flask-vs-django

Biswas, B. (14 de 04 de 2019). *How not to blow your REST interview*. Obtido em 2020 de 05 de 15, de https://medium.com/future-vision/the-principles-of-rest-6b00deac91b3

Braintree. (s.d.). Obtido em 30 de 12 de 2019, de Braintree: https://www.braintreepayments.com/gb?locale=gb

Braintree. (2019). *Webhooks, Overview*. Obtido em 12 de 12 de 2019, de Braintree: https://developers.braintreepayments.com/guides/webhooks/overview

Braintree, Overview. (s.d.). Obtido em 3 de 12 de 2019, de Braintree: https://developers.braintreepayments.com/start/overview

De, B. (2017). *API Management.* India: Apress.

Difference Between JSON and XML. (s.d.). Obtido em 02 de 02 de 2020, de EDUCBA: https://www.educba.com/json-vs-xml/

Django REST Swagger. (s.d.). Obtido em 15 de 03 de 2020, de Django REST Swagger: https://django-rest-swagger.readthedocs.io/en/latest/

Django vs Flask in 2020. (14 de 11 de 2019). Obtido em 15 de 02 de 2020, de https://asperbrothers.com/blog/django-vs-flask-how-are-they-different/

DrivianTasks Gestão inteligente de mobilidade. (14 de 03 de 2016). Obtido em 28 de 05 de 2020, de Transportes: http://www.transportesemrevista.com/Default.aspx?tabid=210&language=pt-PT&id=54164

Herman, M. (29 de 11 de 2019). *Django vs. Flask in 2019: Which Framework to Choose*. Obtido em 16 de 05 de 2020, de testdriven.io: https://testdriven.io/blog/django-vs-flask/

Hope, C. (10 de 07 de 2019). *Computer Hope - Nonce*. Obtido em 12 de 01 de 2020, de Computer Hope: https://www.computerhope.com/jargon/n/nonce.htm

Hull, E., Dick, J., & Jackson, K. (2005). *Requirements Engineering.* Oxford: Springer.

Inovação. (13 de 04 de 2015). *Aplicação ajuda a conduzir melhor*. Obtido em 27 de 05 de 2020, de Jorgal de Notícias: https://www.pressreader.com/portugal/jornal-de-noticias/20150413/281728383042172

Joshi, V. (18 de 01 de 2017). *Why you should be using JSON vs XML*. Obtido em 02 de 02 de 2020, de Cloud Elements: https://blog.cloud-elements.com/using-json-over-xml

Kostrzewa, K. (08 de 2019). *Django vs Flask: Which one should you choose for your project?* Obtido em 2020 de 02 de 15, de Sunscrapers: https://sunscrapers.com/blog/django-vs-flask/#Forms

Larguesa, A. (26 de 04 de 2017). *Sentilant: Ferramenta de Coimbra "mete papéis" no Panamá*. Obtido em 28 de 05 de 2020, de Jornal de Negócios: https://www.jornaldenegocios.pt/negocios-iniciativas/premio-flad-ey-buzz-usa/detalhe/ferramenta-de-coimbra-mete-papeis-no-panama

Larguesa, A. (08 de 01 de 2018). *Conheça os 15 projectos empresariais concorrentes ao prémio FLAD.EY BUZZ USA*. Obtido em 28 de 05 de 2020, de Jornal de Negócios: https://www.jornaldenegocios.pt/negocios-iniciativas/premio-flad-ey-buzz-usa/detalhe/conheca-os-15-projectos-empresariais-concorrentes-ao-premio-fladey-buzz-usa

Massé, M. (2012). *REST API Design Rulebook.* Sebastopol: O'Reilly.

Mota, A. (05 de 03 de 2019). *8 Tips For Designing Quality REST APIs*. Obtido em 20 de 11 de 2019, de NORDIC APIS: https://nordicapis.com/8-tips-for-designing-quality-rest-apis/

REST Architectural Constraints. (s.d.). Obtido em 15 de 15 de 2020, de REST API Tutorial: https://restfulapi.net/rest-architectural-constraints/#cacheable

Sahni, V. (s.d.). *Best Practices for Designing a Pragmatic RESTful API*. Obtido em 02 de 02 de 2020, de Vinay Sahni: https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#restful

Senij, W., & Stempniak, A. (31 de 05 de 2019). *Flask vs. Django: Which Python Framework Is Better for Your Web Development?* Obtido em 16 de 02 de 2020, de https://stxnext.com/blog/2019/05/31/flask-vs-django-comparison/

Simpson, J. (30 de 04 de 2020). *https://nordicapis.com/all-you-need-to-know-about-rest-api-design/*. Obtido em 15 de 05 de 2020, de NORDIC APIS: https://nordicapis.com/all-you-need-to-know-about-rest-api-design/

Singh, V. (28 de 04 de 2020). *Flask vs Django in 2020: Which Framework to Choose?* Obtido em 16 de 05 de 2020, de https://hackr.io/blog/flask-vs-django

Sommerville, I. (2011). *SOFTWARE ENGINEERING.* Boston: Pearson.

Sommerville, I., & Sawyer, P. (1997). *REQUIREMENTS ENGINEERING - A Good Practice.* England: Lancaster University .

Test Everything Braintree. (2019). Obtido em 12 de 12 de 2019, de Braintree: https://www.braintreepayments.com/pt/sandbox

ANEXOS

Anexo A – Proposta de Estágio

SENTILANT - CONSULTORIA E INOVAÇÃO, LDA

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS

PROPOSTA DE ESTÁGIO

Ano Letivo de 2019/2020

em Mestrado em Informática e Sistemas (Desenvolvimento de Software)

TEMA

Desenvolvimento de um API RESTful para suporte a aplicações móveis de caracterização de condução

SUMÁRIO

A Sentilant desenvolve produtos na área de telemática para a condução, aplicáveis à área da segurança rodoviária e seguros. Com o presente Estágio, a empresa pretende desenvolver um API RESTful de suporte a estes produtos, a validar e integrar na sua solução de driving telematics. O aluno dispõe, assim, da oportunidade de contribuir diretamente para uma base tecnológica chave para a empresa, bem como validar o seu trabalho em aplicações desenvolvidas na área pela empresa.

1. ÂMBITO

Atualmente sediada na Incubadora de Empresas do Instituto Pedro Nunes, a Sentilant é uma empresa *spin-off* da Universidade de Coimbra, que se dedica à investigação e desenvolvimento de soluções inteligentes multiplataforma. Alguns dos produtos atuais da Sentilant são vocacionados para a segurança rodoviária, sendo a sua tecnologia desenvolvida e adaptada à medida das necessidades das seguradoras. A solução tecnológica desenvolvida pela empresa suporta a criação de aplicações móveis desenhadas com o objetivo de suportar *gamification*, segurança rodoviária e a obtenção de informação telemétrica de atividades e estilos de condução. Neste contexto, a empresa pretende disponibilizar um API RESTful que permite a utilização da tecnologia da Sentilant em aplicações móveis de terceiros.

2. OBJECTIVOS

O presente projecto pretende atingir os seguintes objetivos genéricos:

- Desenho do API RESTful para suporte a aplicações de caraterização de condução da Sentilant.
- Desenvolvimento do API projetado.
- Integração dos desenvolvimentos no produto da empresa.

3. PROGRAMA DE TRABALHOS

O estágio consistirá nas seguintes atividades e respetivas tarefas:

- ***T1*** - *Definição de requisitos* - Definição de requisitos funcionais e não-funcionais.
- ***T2*** - *Estado da arte* - Estudo das tecnologias e técnicas para desenvolver APIs RESTful.
- ***T3*** - *Programação* - Desenvolvimento, validação e verificação do código fonte.
- ***T4*** - *Integração* - Integração da componente desenvolvida.
- ***T5*** - *Validação* - Avaliação e validação da implementação.
- ***T6*** - *Escrita do relatório*- Escrita do relatório final do Estágio.

4. CALENDARIZAÇÃO DAS TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida:

	Meses									
Tarefas	Oct-19	Nov-19	Dec-19	Jan-20	Feb-20	Mar-20	Apr-20	May-20	Jun-20	Jul-20
T1										
T2										
T3										
T4										
T5										
T6										
Metas		M1		M2		M3		M4	M5	M6

M1 Tarefa T1 terminada
M2 Tarefa T2 terminada
M3 Versão 0.1 do API
M4 Versão 1.0 do API
M5 Tarefa T5 terminada
M6 Tarefa T6 terminada

5. RESULTADOS

Os resultados do estágio serão consubstanciados num conjunto de documentos a elaborar pelo estagiário de acordo com o seguinte plano:

M1

R1.1: Documento de requisitos

M2:

R2.1: Documento do estado da arte

M3:

R3.1: Código fonte

M4:

R4.1: Código fonte

R4.2: Plano de testes

M5:

R5.1: Documento de validação assinado pela Sentilant

M6:

R6.1: Relatório de estágio

6. LOCAL DE TRABALHO

O trabalho decorrerá nas instalações da Sentilant na Incubadora de empresas do Instituto Pedro Nunes, Rua Pedro Nunes, 3030-199 Coimbra. Espera-se que o Estagiário cumpra um horário em regime de full-time na empresa.

7. METODOLOGIA

metodologia de trabalho a seguir terá em conta os seguintes aspetos:

- Integração do estagiário no processo de desenvolvimento de software interno;
- Utilização de uma metodologia ágil para gestão do desenvolvimento baseada em SCRUM;
- Recurso a técnicas de estimação para planeamento detalhado de sprints;
- Reuniões de início e fim de sprint para definição e avaliação de objetivos;
- Escrita de documentação inerente às funcionalidades desenvolvidas, de acordo com os requisitos da empresa.

SENTILANT - CONSULTORIA E INOVAÇÃO, LDA

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS

8. ORIENTAÇÃO

Empresa:

Pedro Costa (pacosta@sentilant.com)
Bruno Cabral (bcabral@sentilant.com)

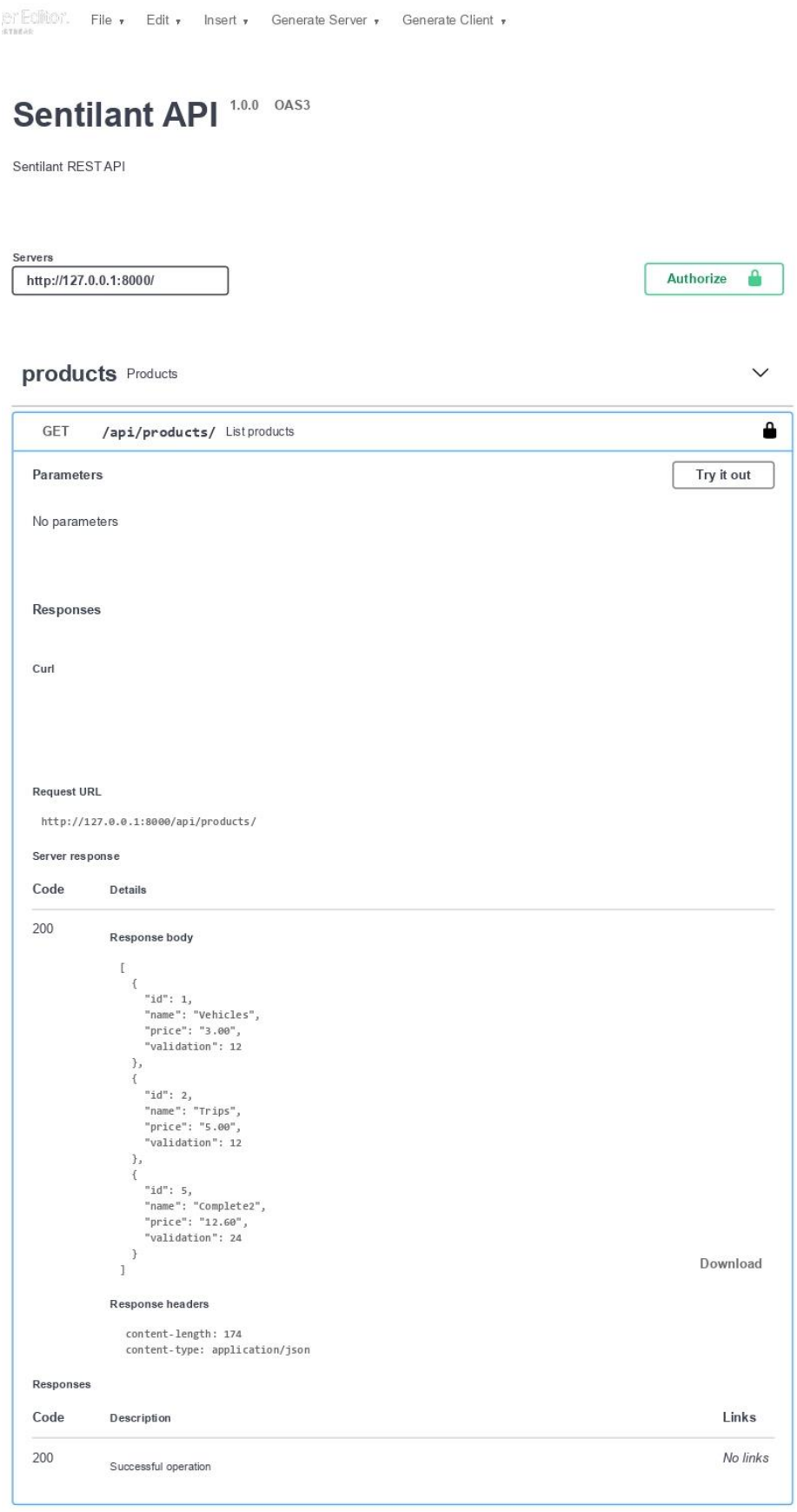
DEIS-ISEC:

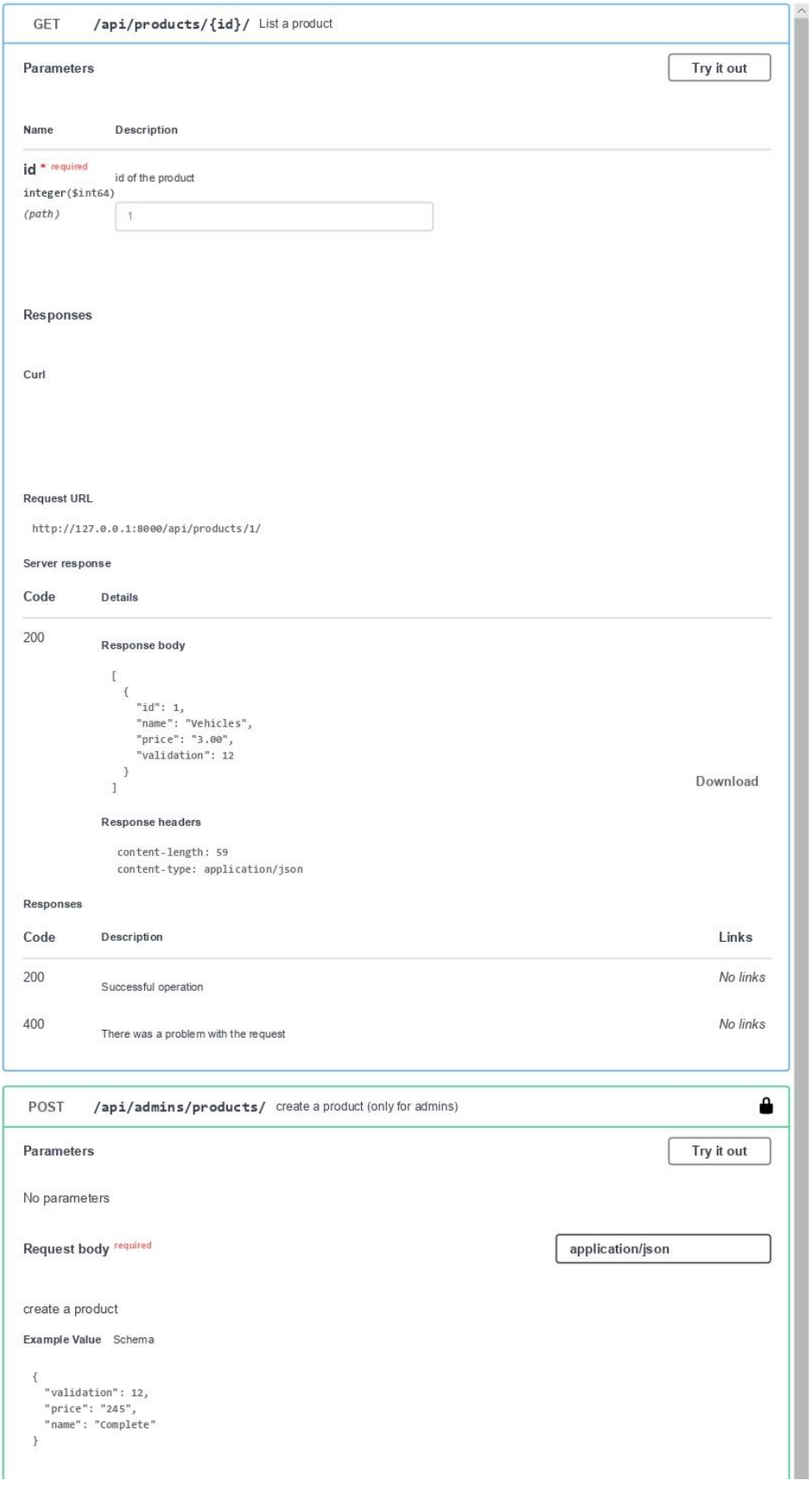
Nome (nome@isec.pt)
Categoria

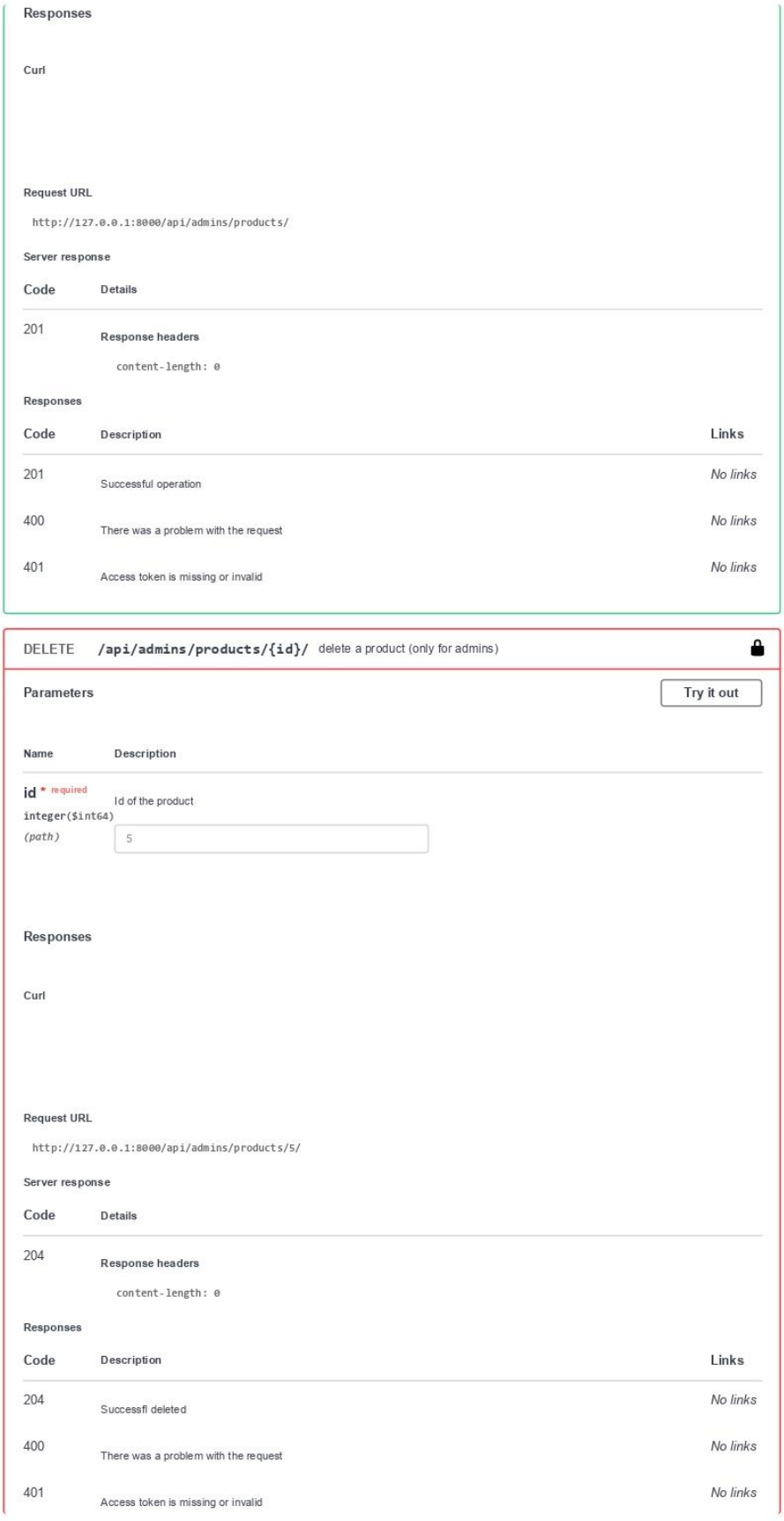
9. CARACTERIZAÇÃO E REMUNERAÇÃO

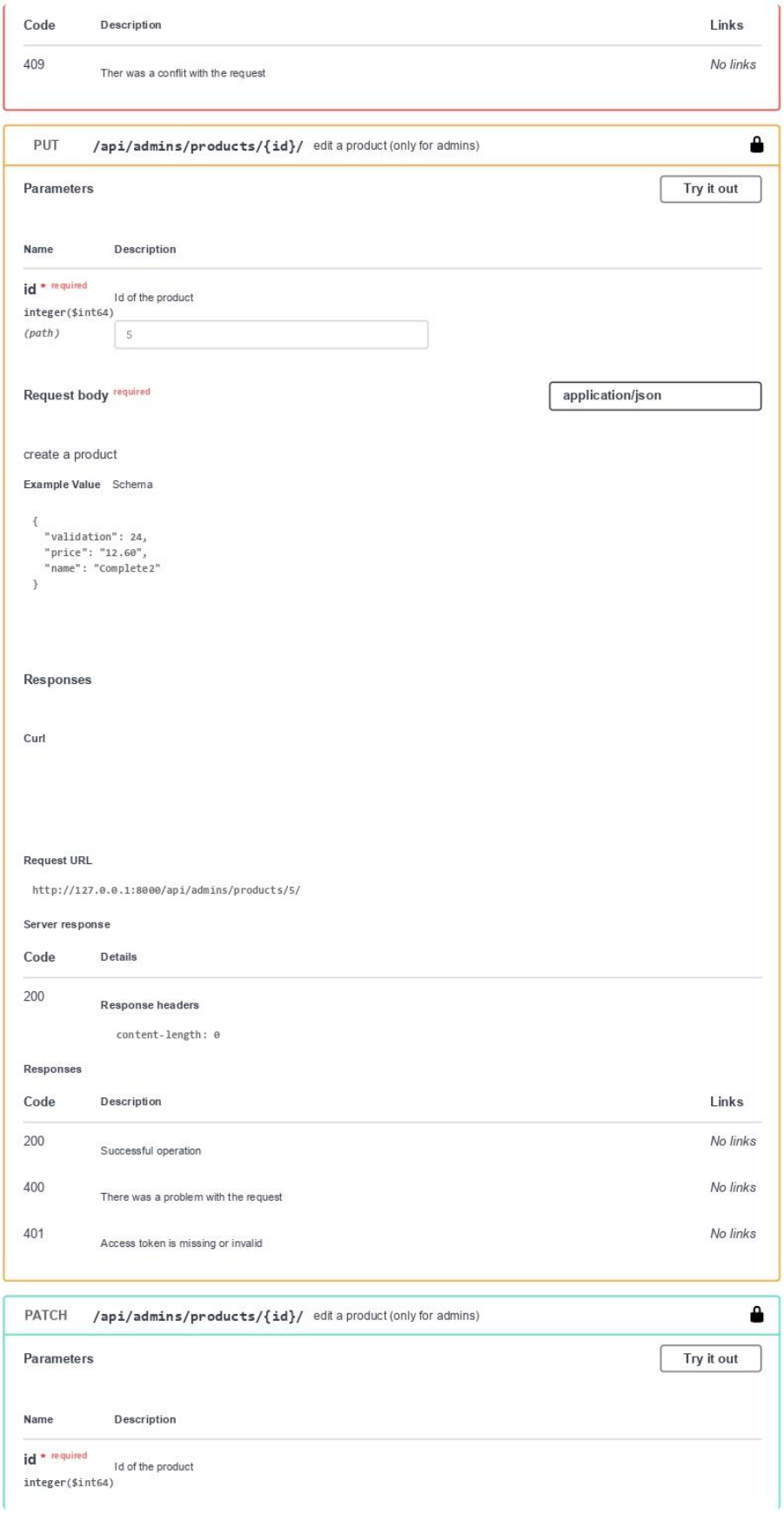
- Data de início: 22 de outubro 2019
- Data de fim: 10 de julho 2020
- Horário: 50% entre 22/10/2019 e 14/02/2020; 100% entre 17/02/2020 e 10/07/2020
- Tipo de formação oferecida aos estagiários: formação em trabalho e acompanhamento diário.
- Local: Sentilant, Incubadora do IPN, Coimbra

Anexo B – Sentilant API - Swagger


File
Edit
Insert
Generate Server
Generate Client
Sentilant API 1.0.0 OAS3
Sentilant REST API
Servers
http://127.0.0.1:8000/
Authorize
products Products
GET /api/products/ List products
Parameters
Try it out
No parameters
Responses
Curl
Request URL
http://127.0.0.1:8000/api/products/
Server response
Code
Details
200
Response body
[
{
"id": 1,
"name": "Vehicles",
"price": "3.00",
"validation": 12
},
{
"id": 2,
"name": "Trips",
"price": "5.00",
"validation": 12
},
{
"id": 5,
"name": "Complete2",
"price": "12.60",
"validation": 24
}
]
Download
Response headers
content-length: 174
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links


GET /api/products/{id}/ List a product
Parameters
Try it out
Name
Description
id * required
integer($int64)
(path)
id of the product
1
Responses
Curl
Request URL
http://127.0.0.1:8000/api/products/1/
Server response
Code
Details
200
Response body
[
{
"id": 1,
"name": "Vehicles",
"price": "3.00",
"validation": 12
}
]
Download
Response headers
content-length: 59
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
POST /api/admins/products/ create a product (only for admins)
Parameters
Try it out
No parameters
Request body required
application/json
create a product
Example Value
Schema
{
"validation": 12,
"price": "245",
"name": "Complete"
}


Responses
Curl
Request URL
http://127.0.0.1:8000/api/admins/products/
Server response
Code Details
201
Response headers
content-length: 0
Responses
Code Description Links
201 Successful operation No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links
DELETE /api/admins/products/{id}/ delete a product (only for admins)
Parameters
Try it out
Name Description
id * required
integer($int64)
(path)
Id of the product
5
Responses
Curl
Request URL
http://127.0.0.1:8000/api/admins/products/5/
Server response
Code Details
204
Response headers
content-length: 0
Responses
Code Description Links
204 Successfl deleted No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links



Code	Description	Links
409	Ther was a conflit with the request	*No links*

PUT **/api/admins/products/{id}/** edit a product (only for admins)

Parameters Try it out

Name	Description
id * required integer($int64) *(path)*	Id of the product 5

Request body required application/json

create a product

Example Value Schema

```
{
  "validation": 24,
  "price": "12.60",
  "name": "Complete2"
}
```

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/admins/products/5/
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

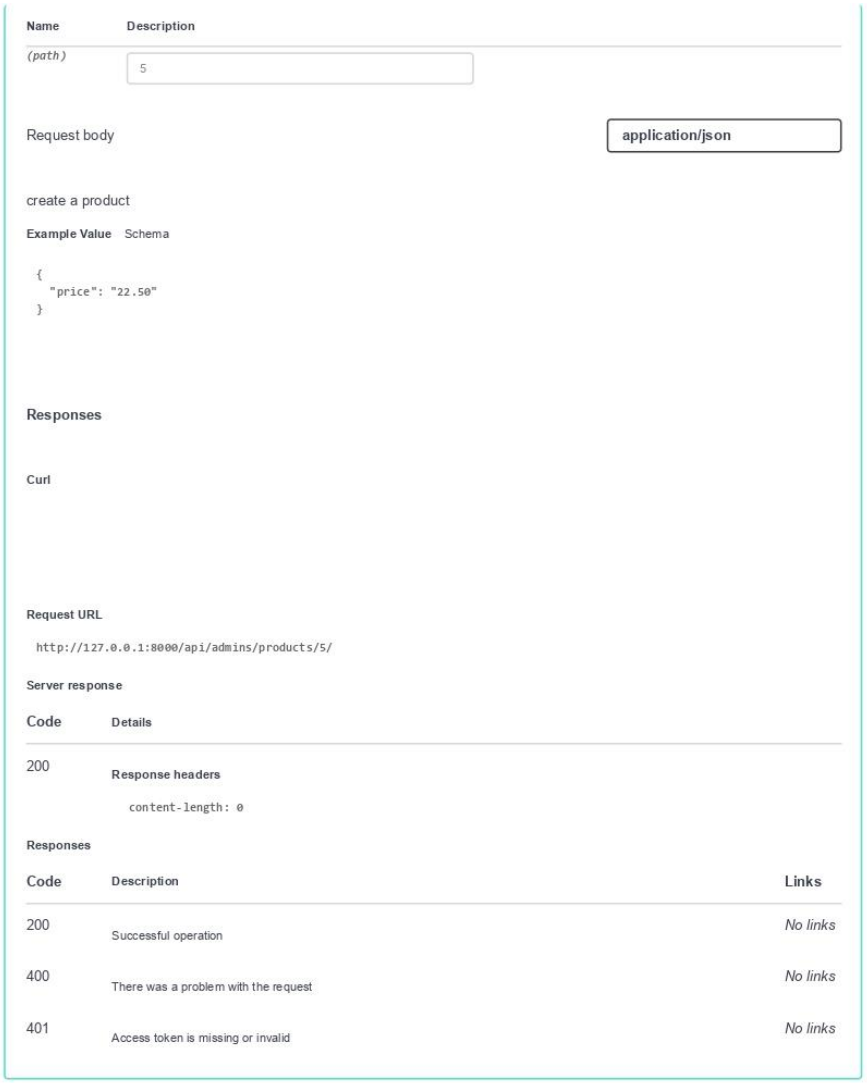
Responses

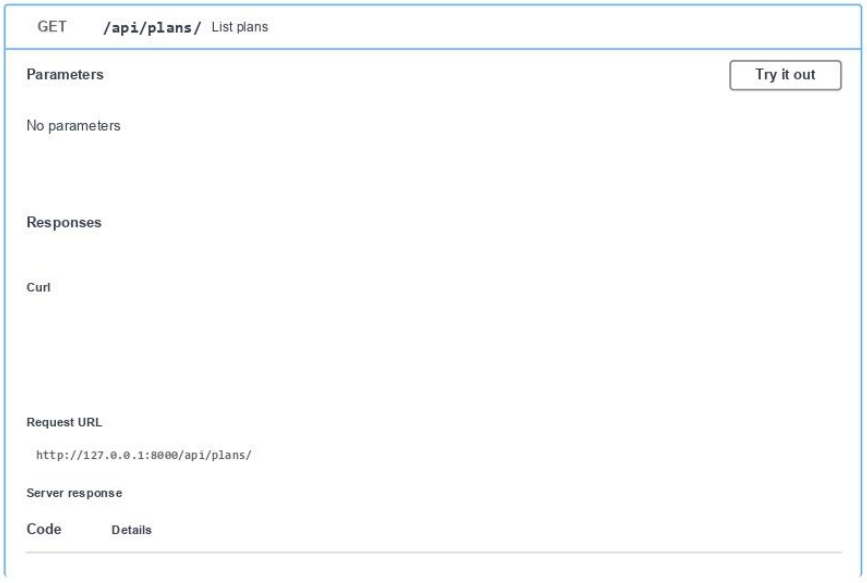
Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

PATCH **/api/admins/products/{id}/** edit a product (only for admins)

Parameters Try it out

Name	Description
id * required integer($int64)	Id of the product


Name
Description
(path)
5
Request body
application/json
create a product
Example Value
Schema
{
"price": "22.50"
}
Responses
Curl
Request URL
http://127.0.0.1:8000/api/admins/products/5/
Server response
Code
Details
200
Response headers
content-length: 0
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links


GET
/api/plans/
List plans
Parameters
Try it out
No parameters
Responses
Curl
Request URL
http://127.0.0.1:8000/api/plans/
Server response
Code
Details

Code Details

200 Response body


```
[
  {
    "id": 12,
    "plan_id": "6bhb",
    "name": "Complete",
    "description": "All products",
    "plan_price": "20.00",
    "trial_period": true,
    "trial_duration": 2
  },
  {
    "id": 13,
    "plan_id": "bcnb",
    "name": "Trips",
    "description": "",
    "plan_price": "12.00",
    "trial_period": true,
    "trial_duration": 2
  },
  {
    "id": 14,
    "plan_id": "tnb6",
    "name": "Vehicles".
```

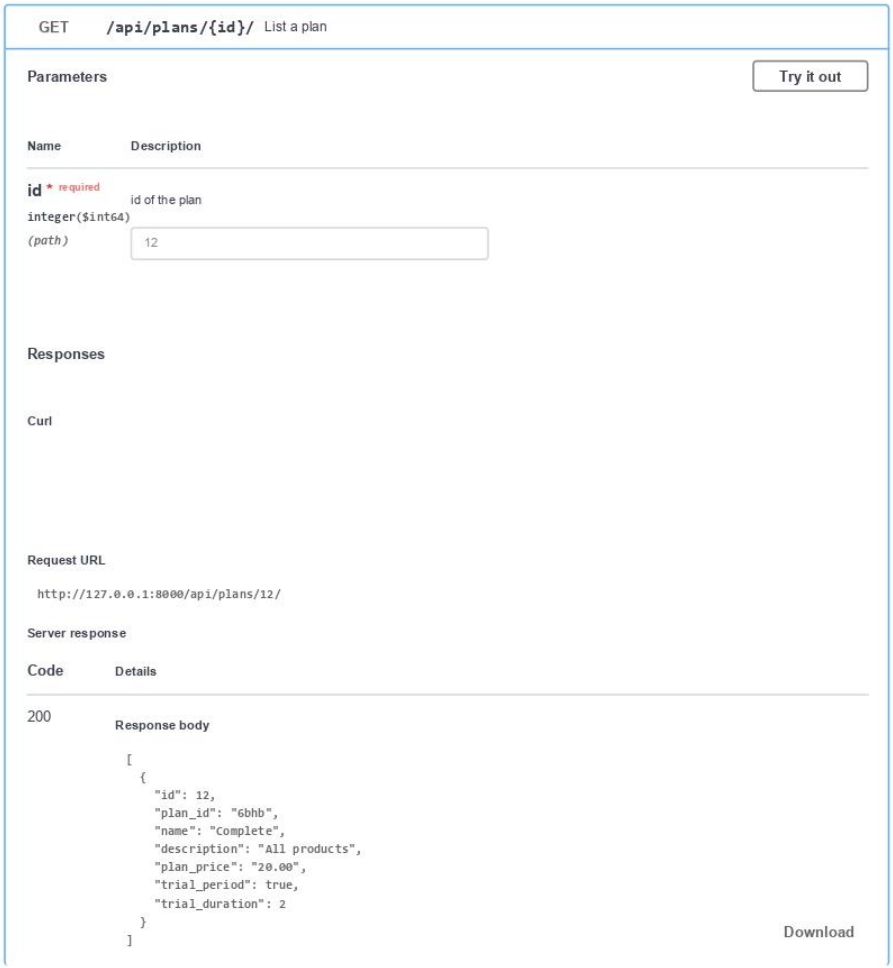

Download

Response headers

```
content-length: 395
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*



GET **/api/plans/{id}/** List a plan

Parameters Try it out

Name	Description
id * required integer($int64) *(path)*	id of the plan 12

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/plans/12/
```

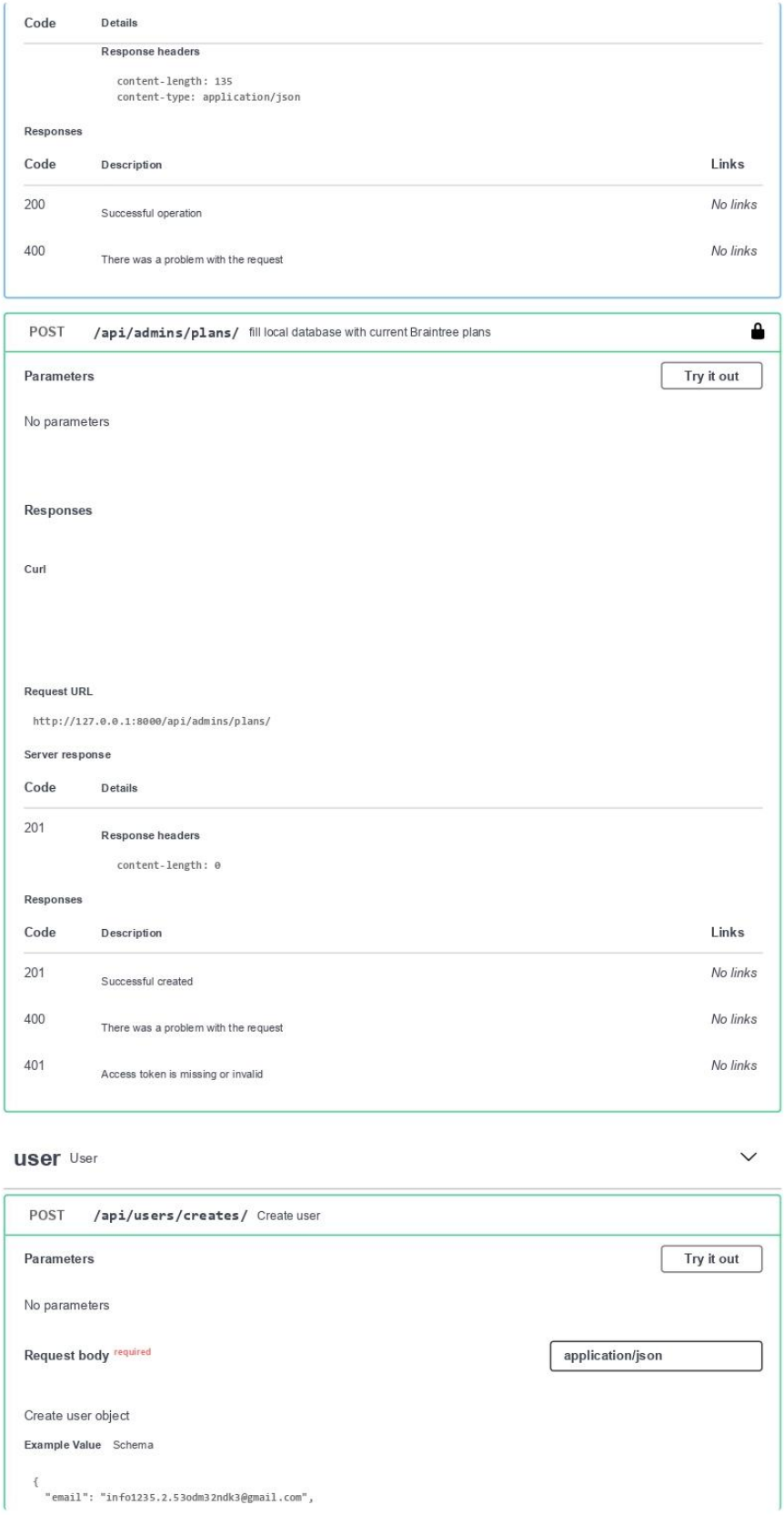
Server response

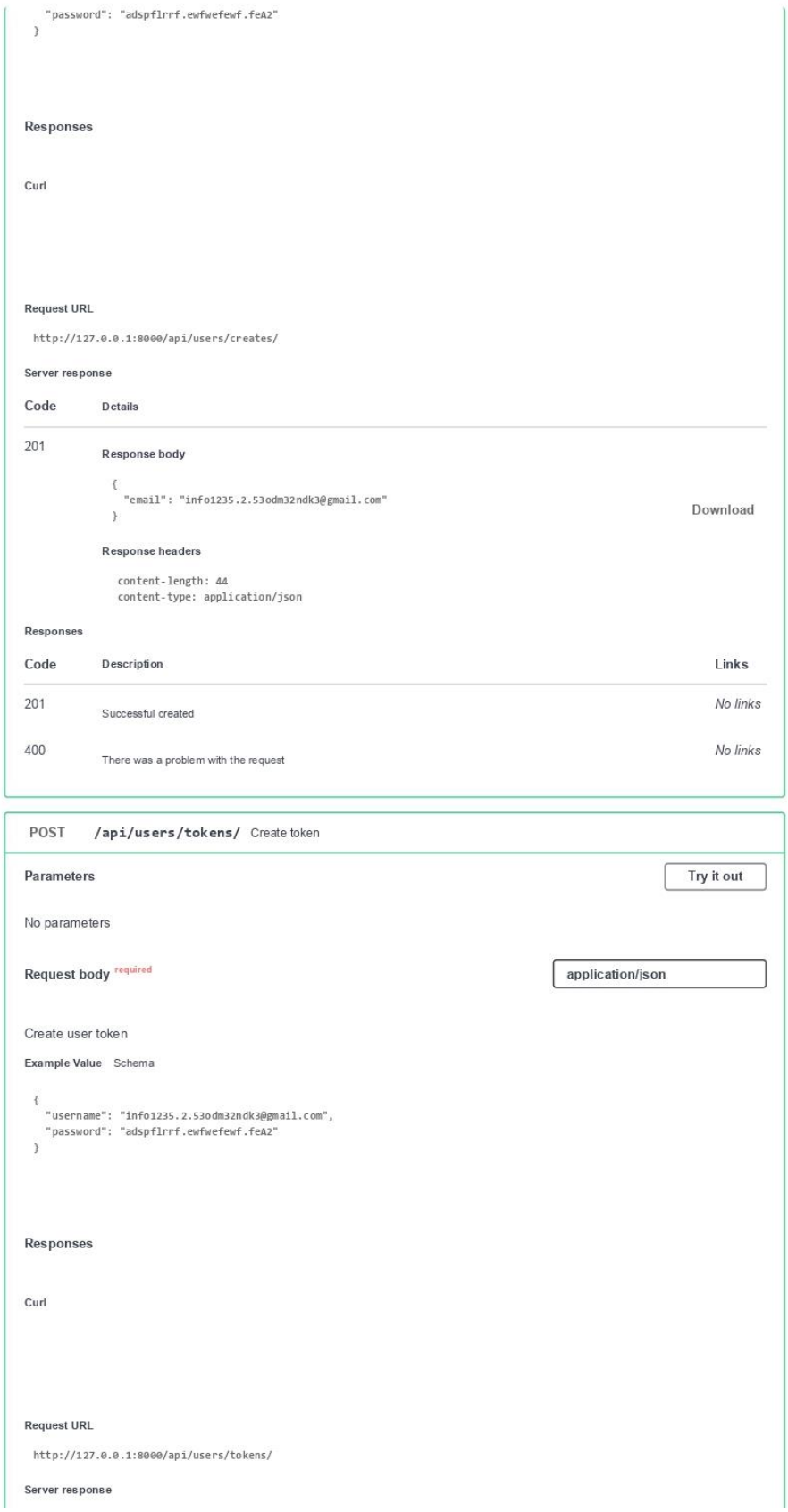
Code Details

200 Response body


```
[
  {
    "id": 12,
    "plan_id": "6bhb",
    "name": "Complete",
    "description": "All products",
    "plan_price": "20.00",
    "trial_period": true,
    "trial_duration": 2
  }
]
```


Download


Code
Details
Response headers
content-length: 135
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
POST
/api/admins/plans/
fill local database with current Braintree plans
Parameters
Try it out
No parameters
Responses
Curl
Request URL
http://127.0.0.1:8000/api/admins/plans/
Server response
Code
Details
201
Response headers
content-length: 0
Responses
Code
Description
Links
201
Successful created
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
user
User
POST
/api/users/creates/
Create user
Parameters
Try it out
No parameters
Request body
required
application/json
Create user object
Example Value
Schema
{
"email": "info1235.2.53odm32ndk3@gmail.com",



```
  "password": "adspflrrf.ewfwefewf.feA2"
}
```

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/creates/
```

Server response

Code	Details

201	**Response body**
	`{ "email": "info1235.2.53odm32ndk3@gmail.com" }`
	Download
	Response headers
	`content-length: 44`
	`content-type: application/json`

Responses

Code	Description	Links
201	Successful created	*No links*
400	There was a problem with the request	*No links*

POST **/api/users/tokens/** Create token

Parameters Try it out

No parameters

Request body required application/json

Create user token

Example Value Schema

```
{
  "username": "info1235.2.53odm32ndk3@gmail.com",
  "password": "adspflrrf.ewfwefewf.feA2"
}
```

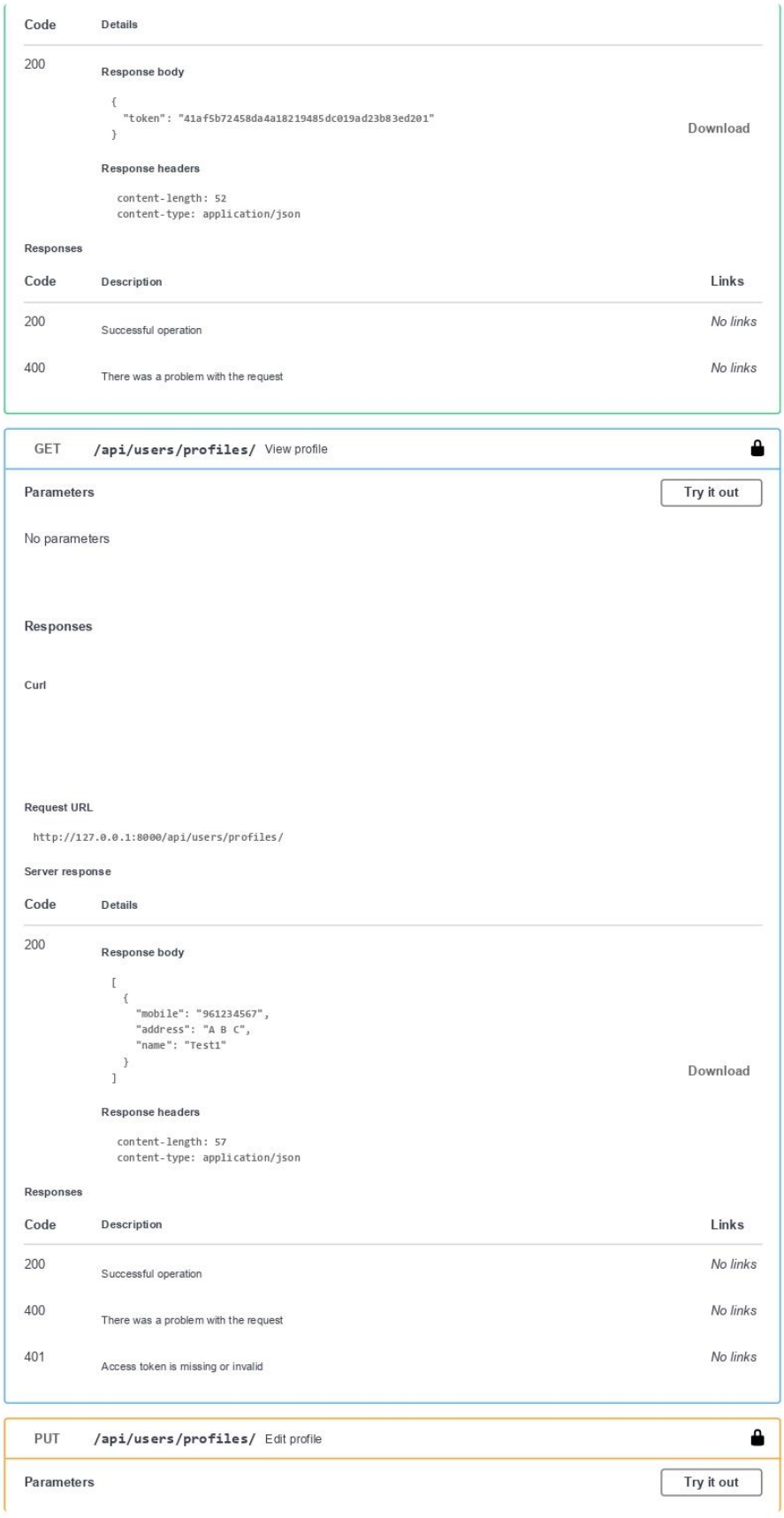
Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/tokens/
```

Server response



Code | Details

200

Response body

```
{
  "token": "41af5b72458da4a18219485dc019ad23b83ed201"
}
```

Download

Response headers

```
content-length: 52
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*

GET **/api/users/profiles/** View profile

Parameters | Try it out

No parameters

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/profiles/
```

Server response

Code | Details

200

Response body

```
[
  {
    "mobile": "961234567",
    "address": "A B C",
    "name": "Test1"
  }
]
```

Download

Response headers

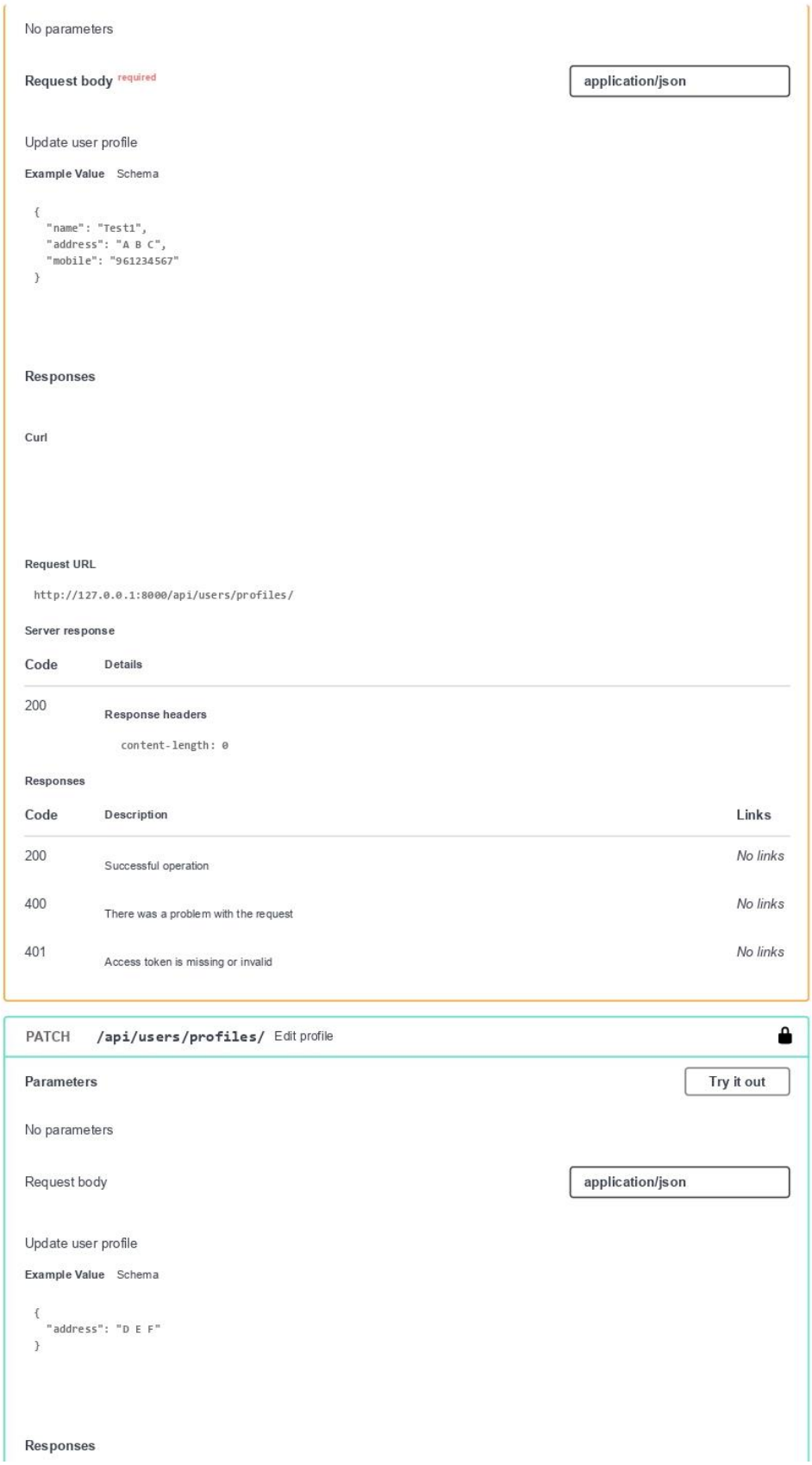
```
content-length: 57
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

PUT **/api/users/profiles/** Edit profile

Parameters | Try it out


No parameters
Request body required
application/json
Update user profile
Example Value Schema
{
"name": "Test1",
"address": "A B C",
"mobile": "961234567"
}
Responses
Curl
Request URL
http://127.0.0.1:8000/api/users/profiles/
Server response
Code Details
200
Response headers
content-length: 0
Responses
Code Description Links
200 Successful operation No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links
PATCH /api/users/profiles/ Edit profile
Parameters
Try it out
No parameters
Request body
application/json
Update user profile
Example Value Schema
{
"address": "D E F"
}
Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/profiles/
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*
409	Ther was a conflit with the request	*No links*

POST **`/api/users/passwords/recoveries/`** Solicitate password recovery

Parameters Try it out

No parameters

Request body required — application/json

Solicitate password recovery

Example Value Schema

```
{
  "username": "info1235.2.53odm32ndk3@gmail.com"
}
```

Responses

Curl

Request URL

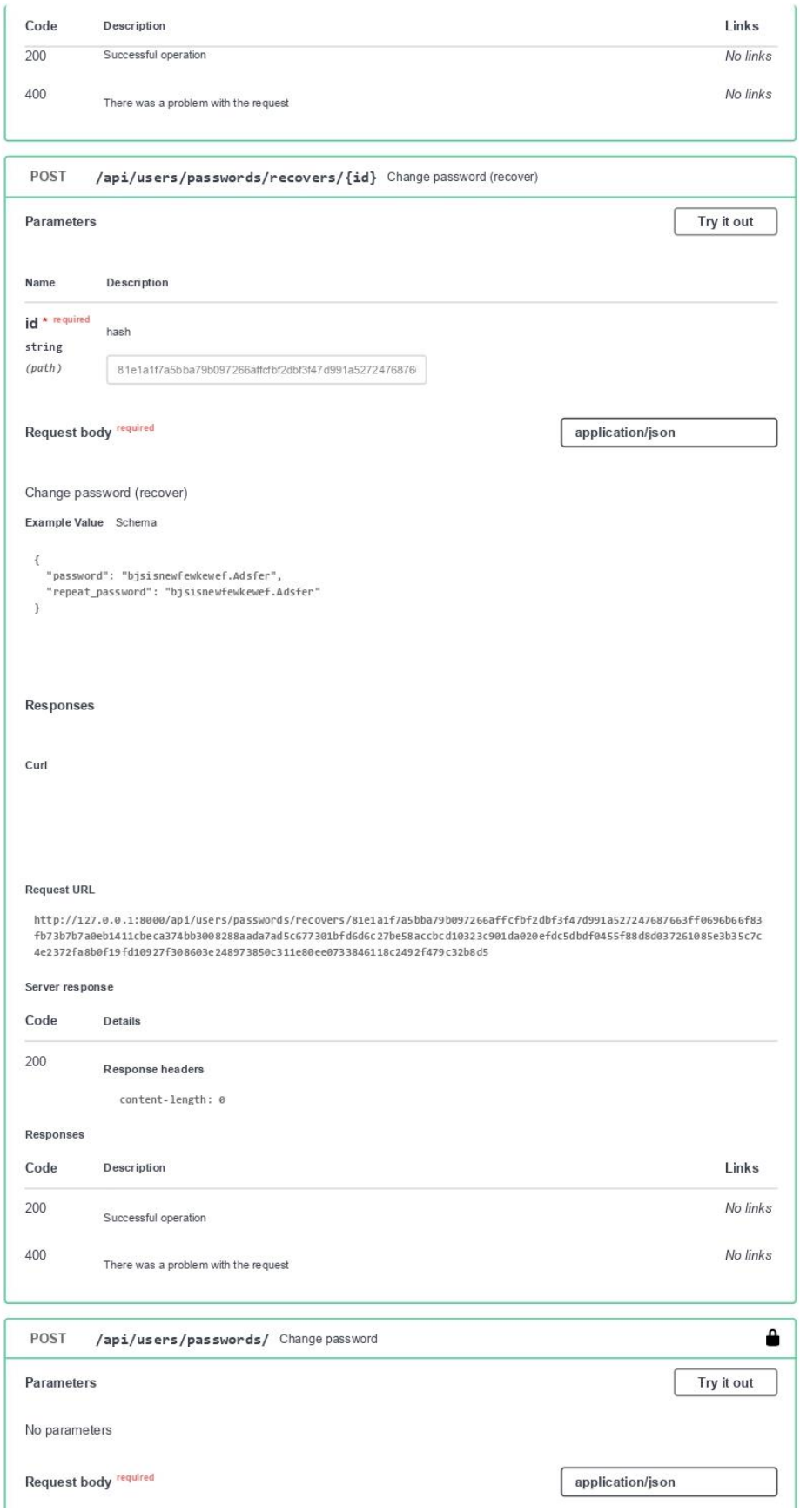
```
http://127.0.0.1:8000/api/users/passwords/recoveries/
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

Responses

Code	Description	Links



Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*

POST `/api/users/passwords/recovers/{id}` Change password (recover)

Parameters Try it out

Name	Description
id * required string *(path)*	hash 81e1a1f7a5bba79b097266affcfbf2dbf3f47d991a5272476876

Request body required — application/json

Change password (recover)

Example Value Schema

```
{
  "password": "bjsisnewfewkewef.Adsfer",
  "repeat_password": "bjsisnewfewkewef.Adsfer"
}
```

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/passwords/recovers/81e1a1f7a5bba79b097266affcfbf2dbf3f47d991a527247687663ff0696b66f83
fb73b7b7a0eb1411cbeca374bb3008288aada7ad5c677301bfd6d6c27be58accbcd10323c901da020efdc5dbdf0455f88d8d037261085e3b35c7c
4e2372fa8b0f19fd10927f308603e248973850c311e80ee0733846118c2492f479c32b8d5
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*

POST `/api/users/passwords/` Change password

Parameters Try it out

No parameters

Request body required — application/json

Change password

Example Value Schema

```
{
  "current_password": "bjsisnewfewkewef.Adsfer",
  "new_password": "nekslдfvewe.cewfceA",
  "repeate_password": "nekslдfvewe.cewfceA"
}
```

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/passwords/
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

POST **/api/users/usernames/** Change username

Parameters Try it out

No parameters

Request body required application/json

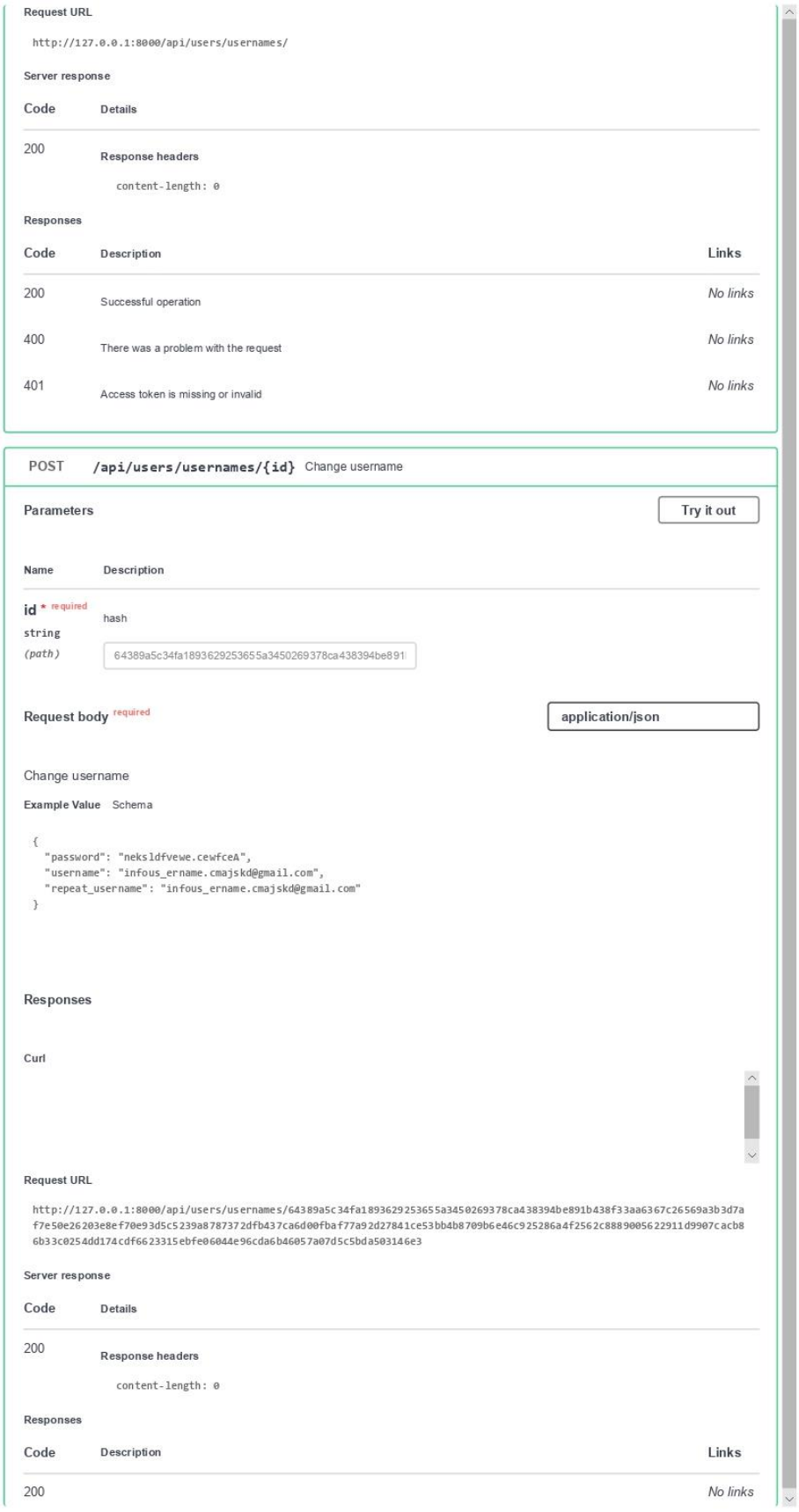
Change username

Example Value Schema

```
{
  "current_username": "info1235.2.53odm32ndk3@gmail.com",
  "password": "nekslдfvewe.cewfceA"
}
```

Responses

Curl



Request URL

http://127.0.0.1:8000/api/users/usernames/

Server response

Code	Details
200	Response headers content-length: 0

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

POST **/api/users/usernames/{id}** Change username

Parameters — Try it out

Name	Description
id * required string *(path)*	hash 64389a5c34fa1893629253655a3450269378ca438394be891

Request body required — application/json

Change username

Example Value | Schema

```
{
  "password": "neksldfvewe.cewfceA",
  "username": "infous_ername.cmajskd@gmail.com",
  "repeat_username": "infous_ername.cmajskd@gmail.com"
}
```

Responses

Curl

Request URL

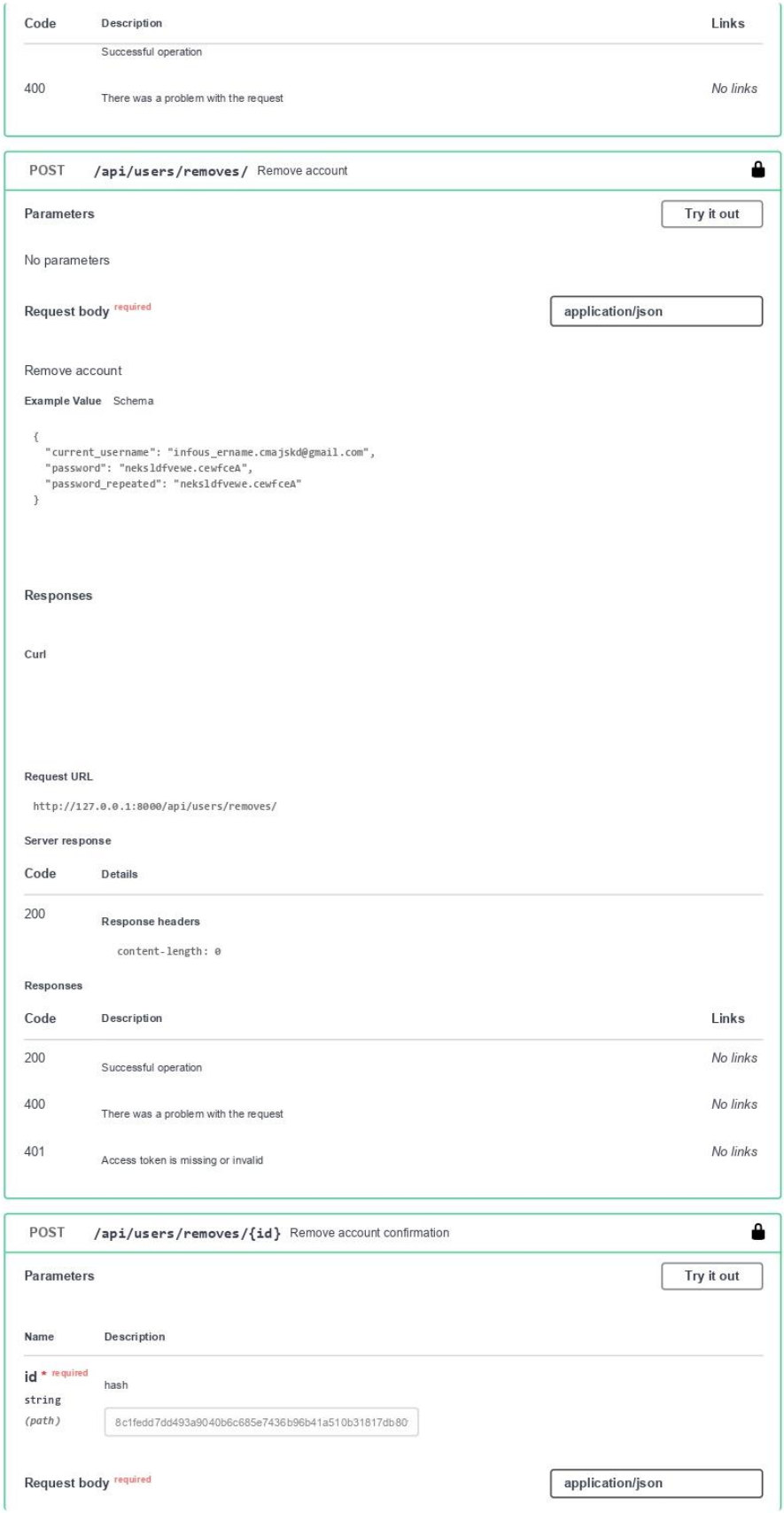
```
http://127.0.0.1:8000/api/users/usernames/64389a5c34fa1893629253655a3450269378ca438394be891b438f33aa6367c26569a3b3d7a
f7e50e26203e8ef70e93d5c5239a8787372dfb437ca6d00fbaf77a92d27841ce53bb4b8709b6e46c925286a4f2562c8889005622911d9907cacb8
6b33c0254dd174cdf6623315ebfe06044e96cda6b46057a07d5c5bda503146e3
```

Server response

Code	Details
200	Response headers content-length: 0

Responses

Code	Description	Links
200		*No links*



Code	Description	Links
	Successful operation	
400	There was a problem with the request	*No links*

POST `/api/users/removes/` Remove account

Parameters — Try it out

No parameters

Request body required — application/json

Remove account

Example Value Schema

```
{
  "current_username": "infous_ername.cmajskd@gmail.com",
  "password": "neksldfvewe.cewfceA",
  "password_repeated": "neksldfvewe.cewfceA"
}
```

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/users/removes/
```

Server response

Code	Details
200	**Response headers** `content-length: 0`

Responses

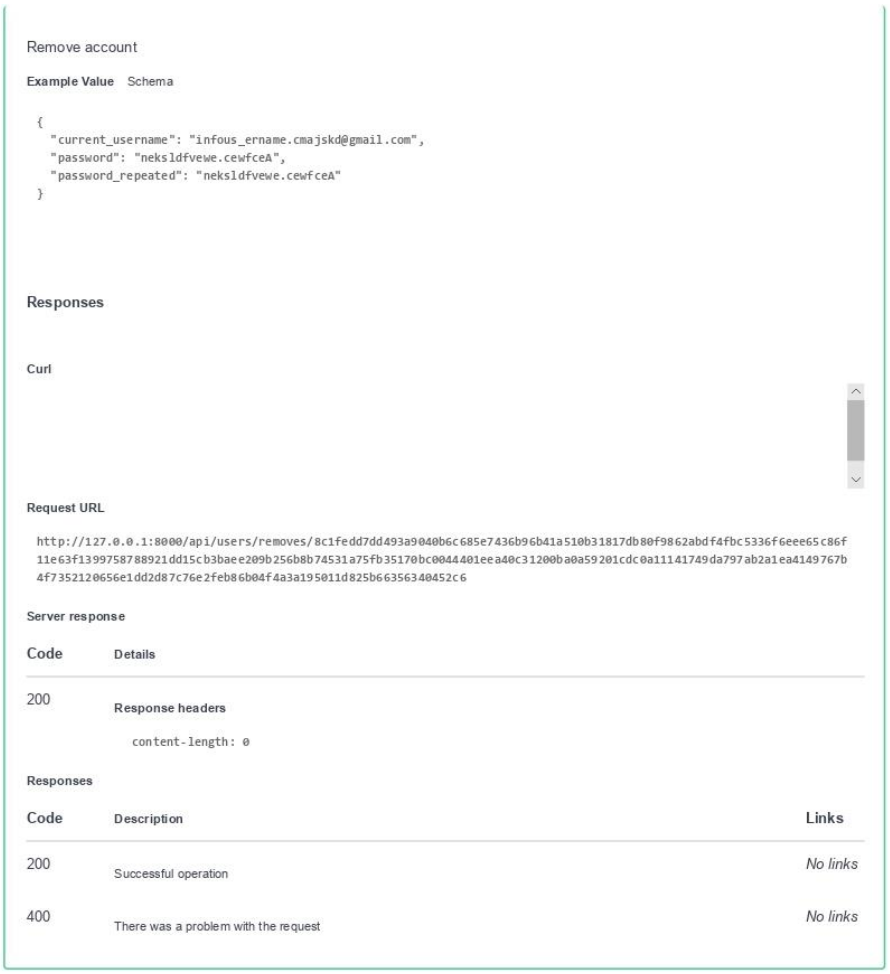
Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

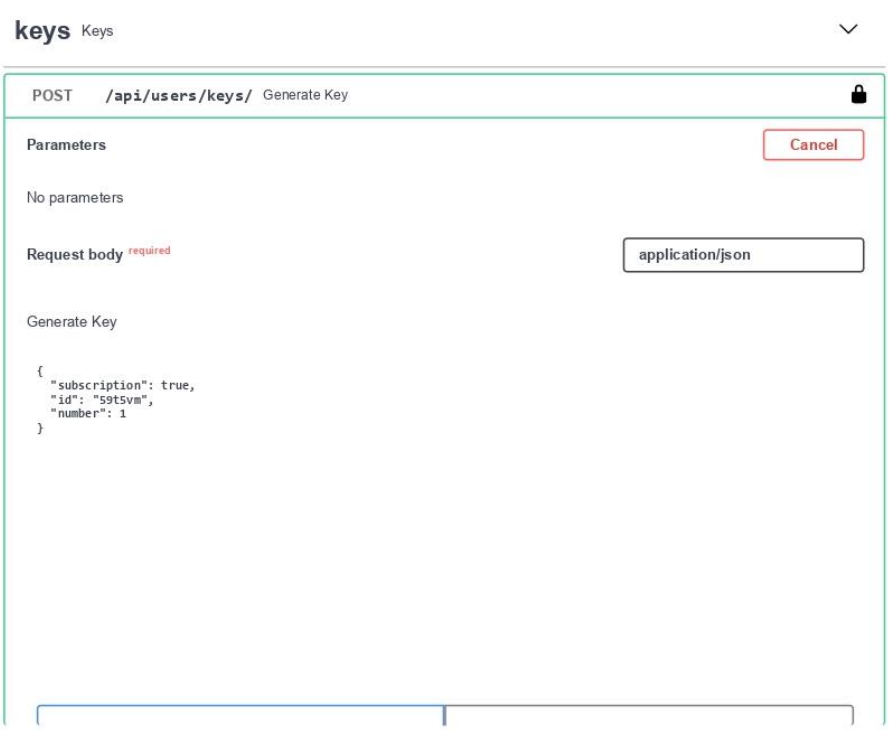
POST `/api/users/removes/{id}` Remove account confirmation

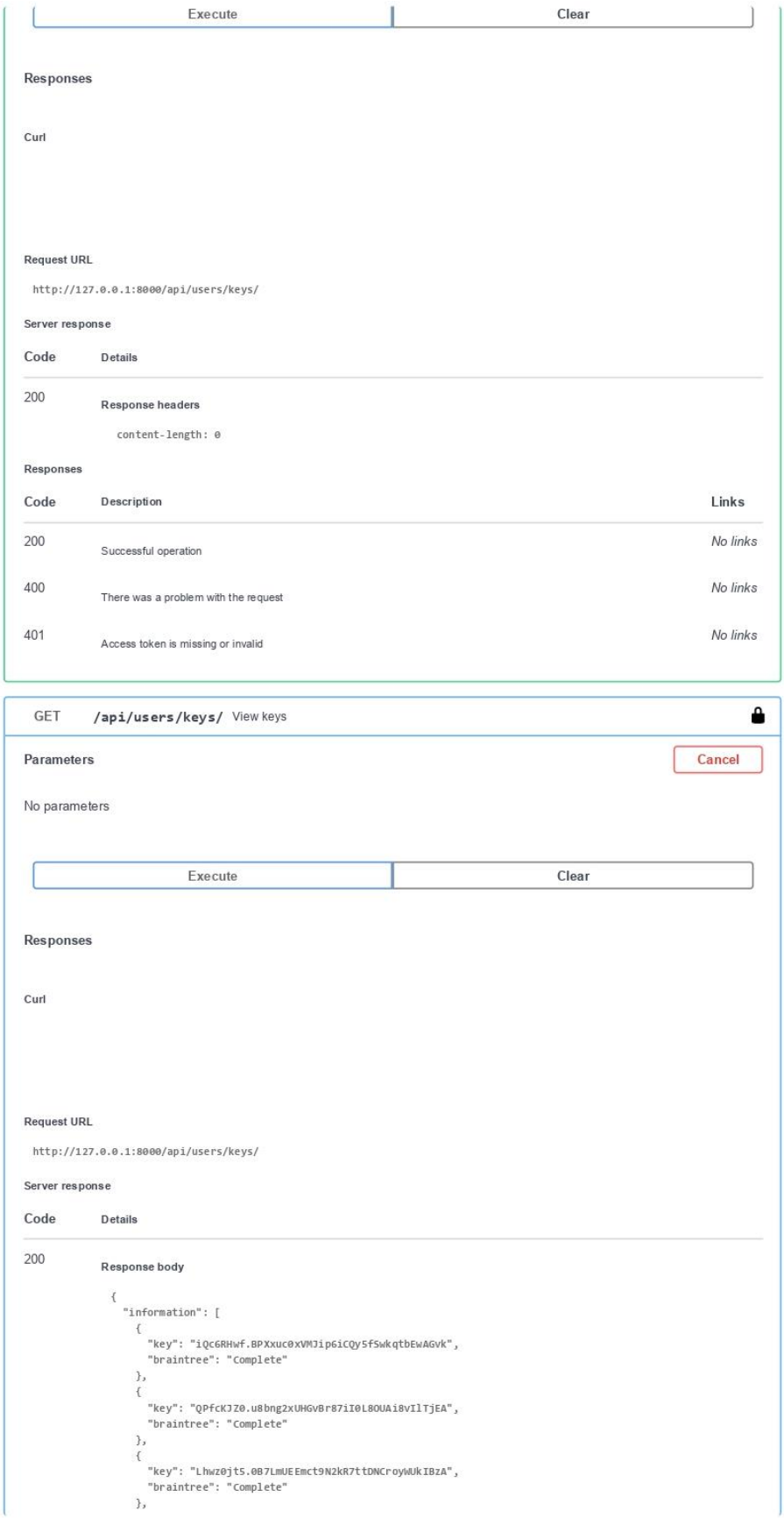
Parameters — Try it out

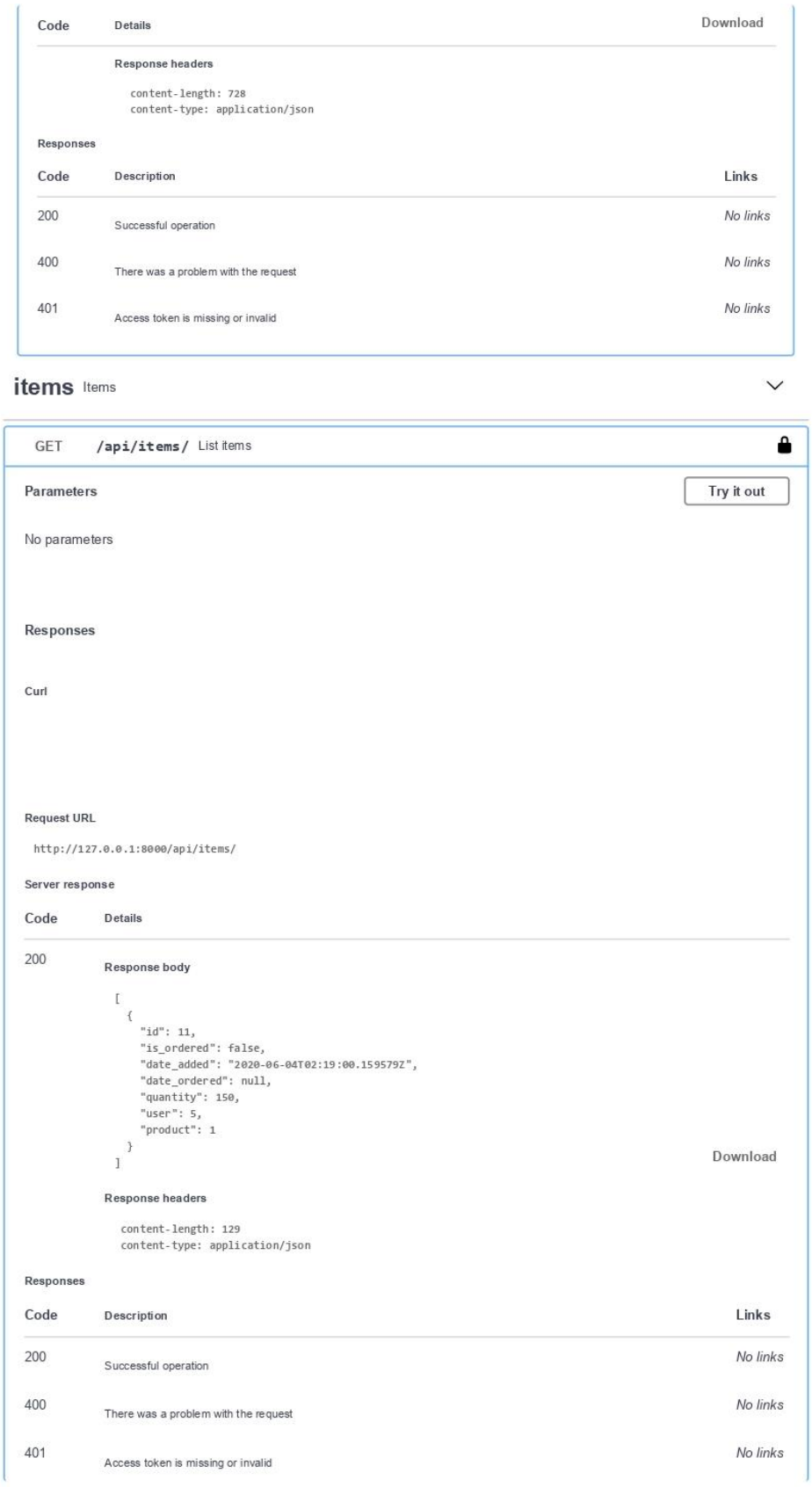
Name	Description
id * required string *(path)*	hash 8c1fedd7dd493a9040b6c685e7436b96b41a510b31817db80

Request body required — application/json


Remove account
Example Value Schema
{
"current_username": "infous_ername.cmajskd@gmail.com",
"password": "nekskdfvewe.cewfceA",
"password_repeated": "nekskdfvewe.cewfceA"
}
Responses
Curl
Request URL
http://127.0.0.1:8000/api/users/removes/8c1fedd7dd493a9040b6c685e7436b96b41a510b31817db80f9862abdf4fbc5336f6eee65c86f11e63f1399758788921dd15cb3baee209b256b8b74531a75fb35170bc0044401eea40c31200ba0a59201cdc0a11141749da797ab2a1ea4149767b4f7352120656e1dd2d87c76e2feb86b04f4a3a195011d825b663563404 52c6
Server response
Code Details
200
Response headers
content-length: 0
Responses
Code Description Links
200 Successful operation No links
400 There was a problem with the request No links


keys Keys
POST /api/users/keys/ Generate Key
Parameters
Cancel
No parameters
Request body required
application/json
Generate Key
{
"subscription": true,
"id": "59t5vm",
"number": 1
}


Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/users/keys/
Server response
Code
Details
200
Response headers
content-length: 0
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
GET
/api/users/keys/
View keys
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/users/keys/
Server response
Code
Details
200
Response body
{
"information": [
{
"key": "iQc6RHwf.BPXxuc0xVMJip6iCQy5fSwkqtbEwAGvk",
"braintree": "Complete"
},
{
"key": "QPfcKJZ0.u8bng2xUHGvBr87iI0L8OUAi8vIlTjEA",
"braintree": "Complete"
},
{
"key": "Lhwz0jt5.0B7LmUEEmct9N2kR7ttDNCroyWUkIBzA",
"braintree": "Complete"
},



Code | Details | Download

Response headers

```
content-length: 728
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

items Items

GET **/api/items/** List items

Parameters | Try it out

No parameters

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/items/
```

Server response

Code | Details

200

Response body

```
[
  {
    "id": 11,
    "is_ordered": false,
    "date_added": "2020-06-04T02:19:00.159579Z",
    "date_ordered": null,
    "quantity": 150,
    "user": 5,
    "product": 1
  }
]
```

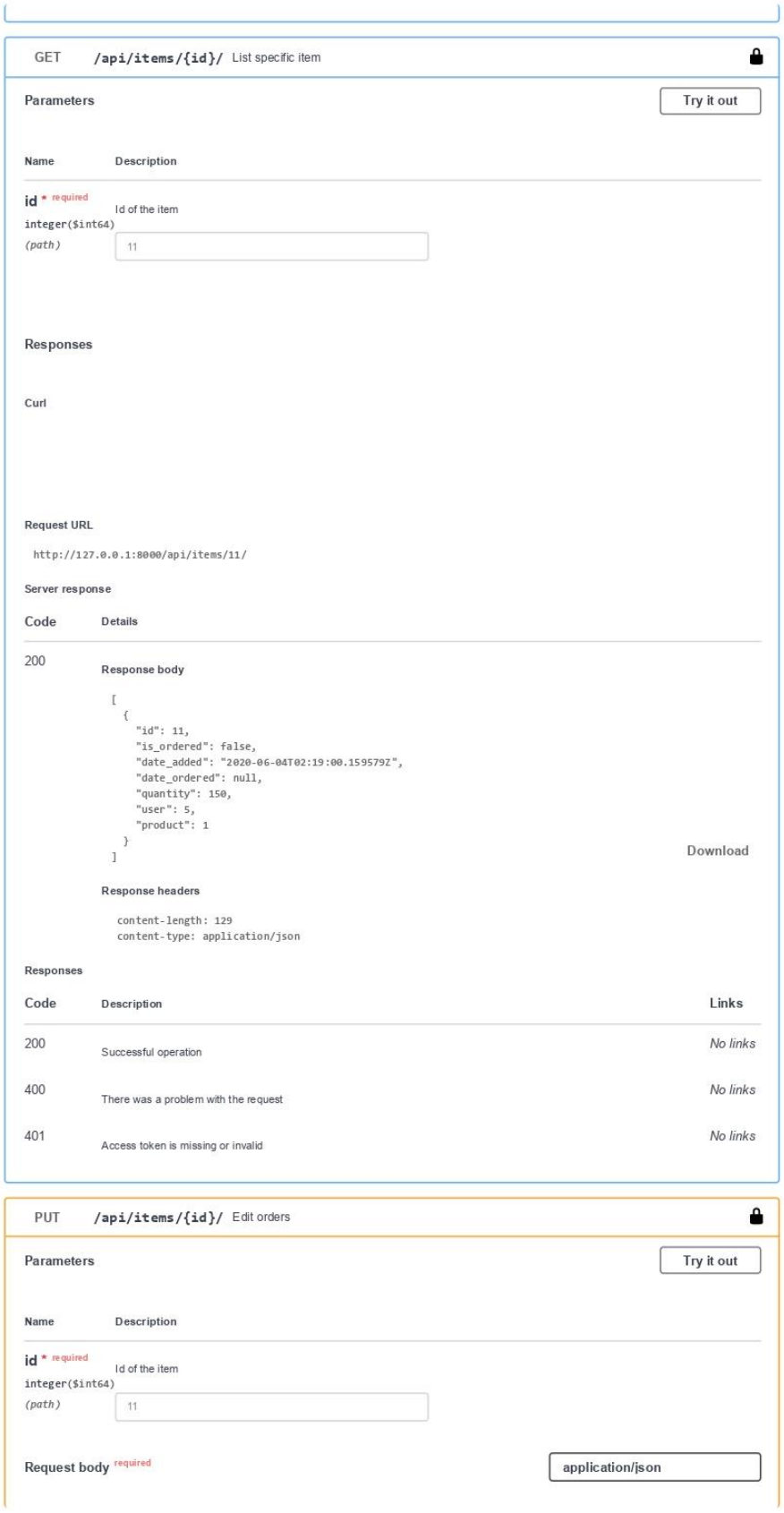
Download

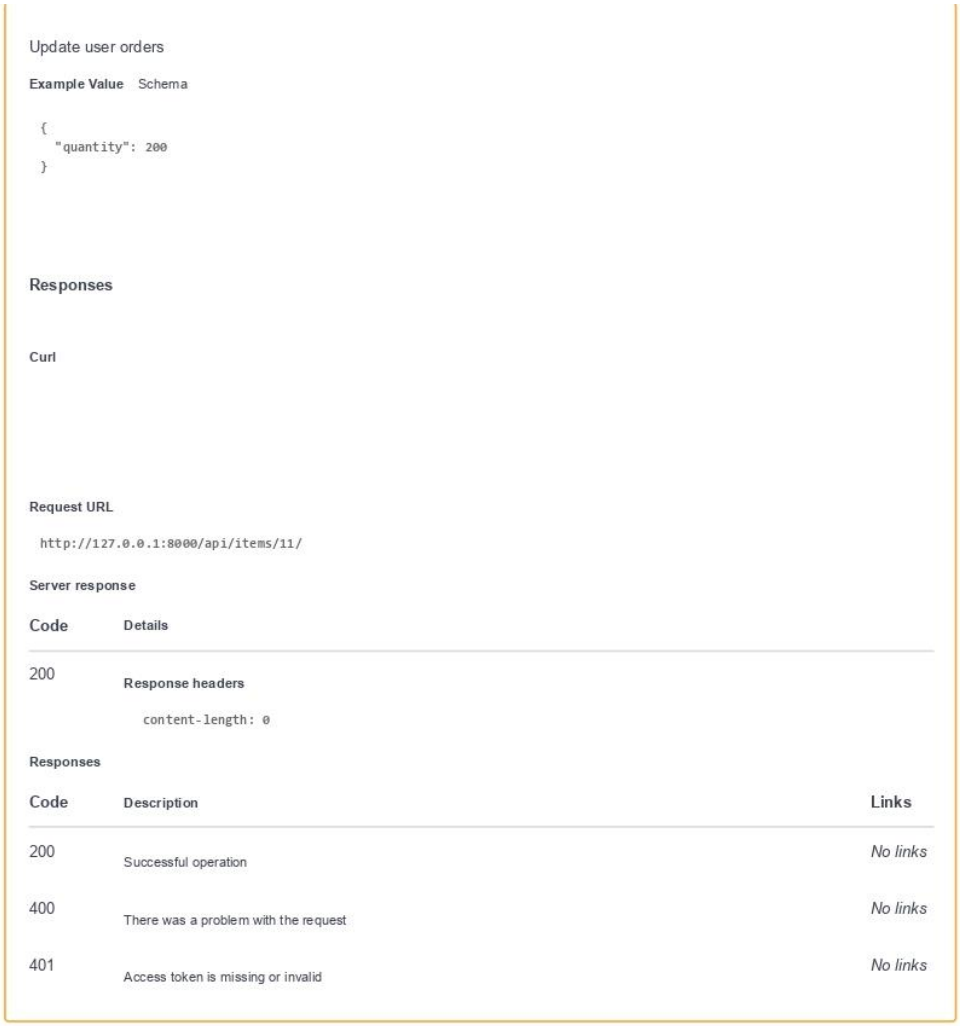
Response headers

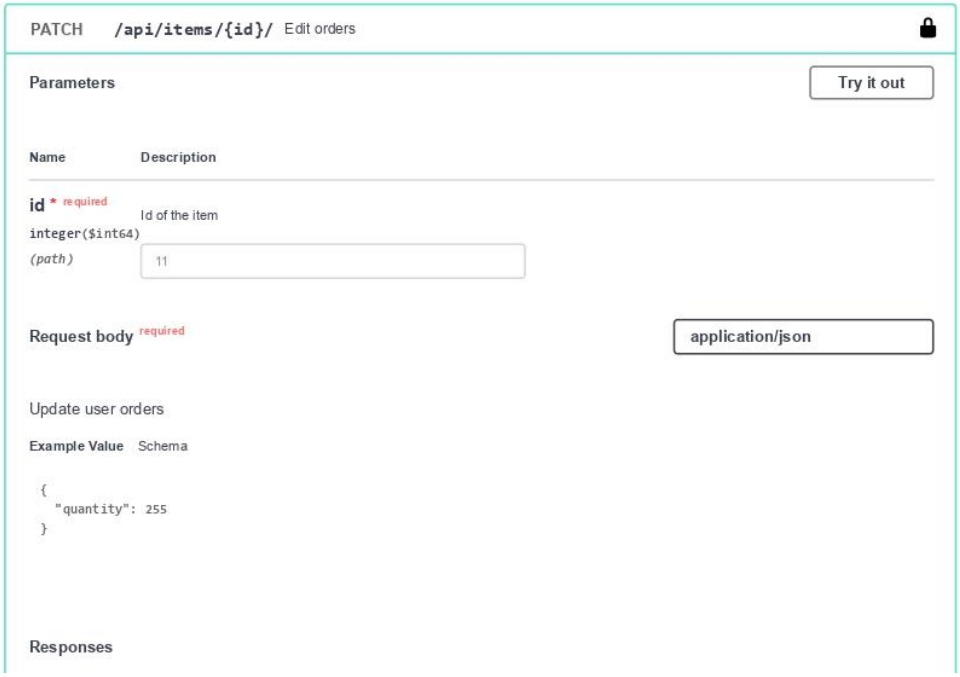
```
content-length: 129
content-type: application/json
```

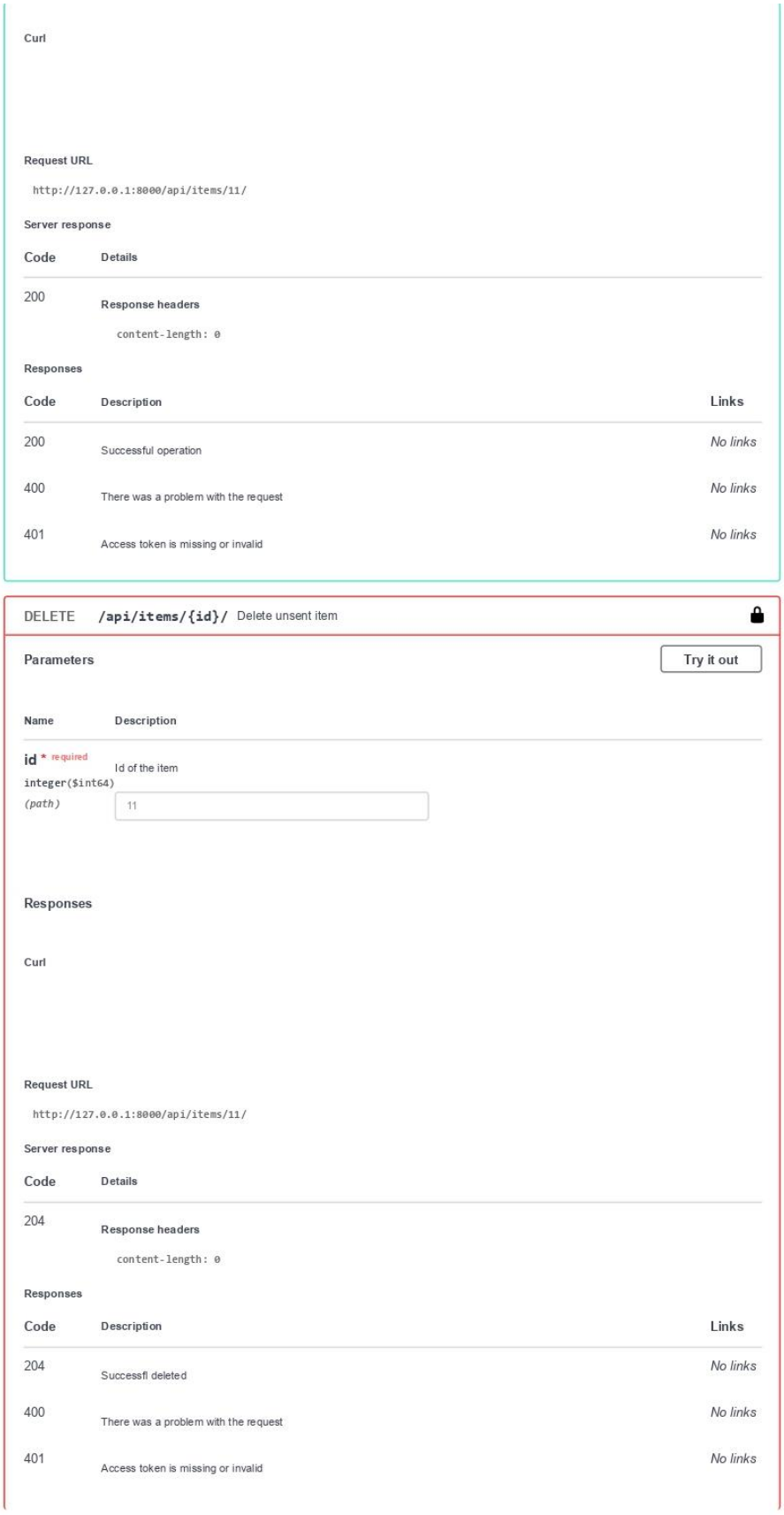
Responses

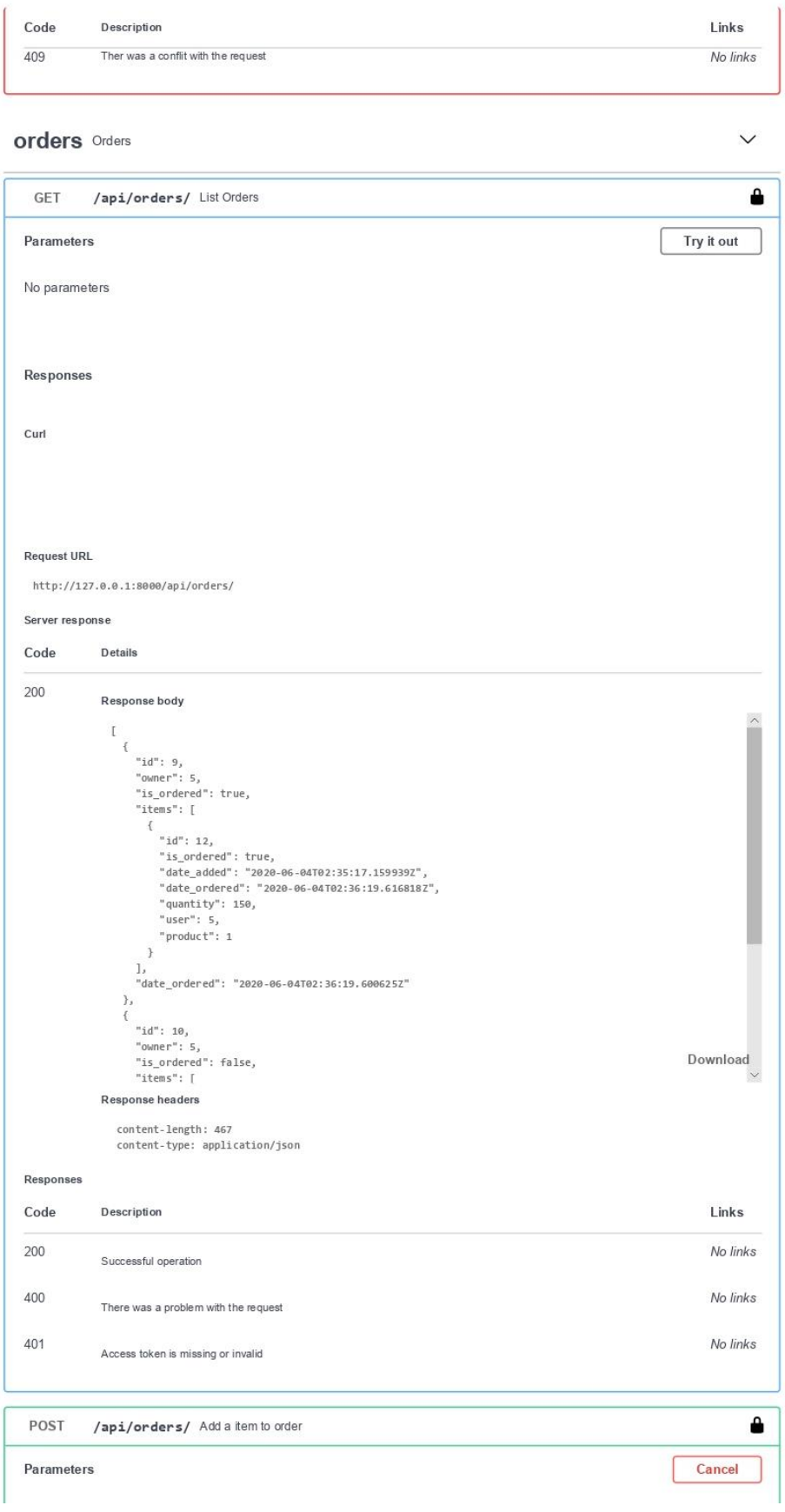
Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

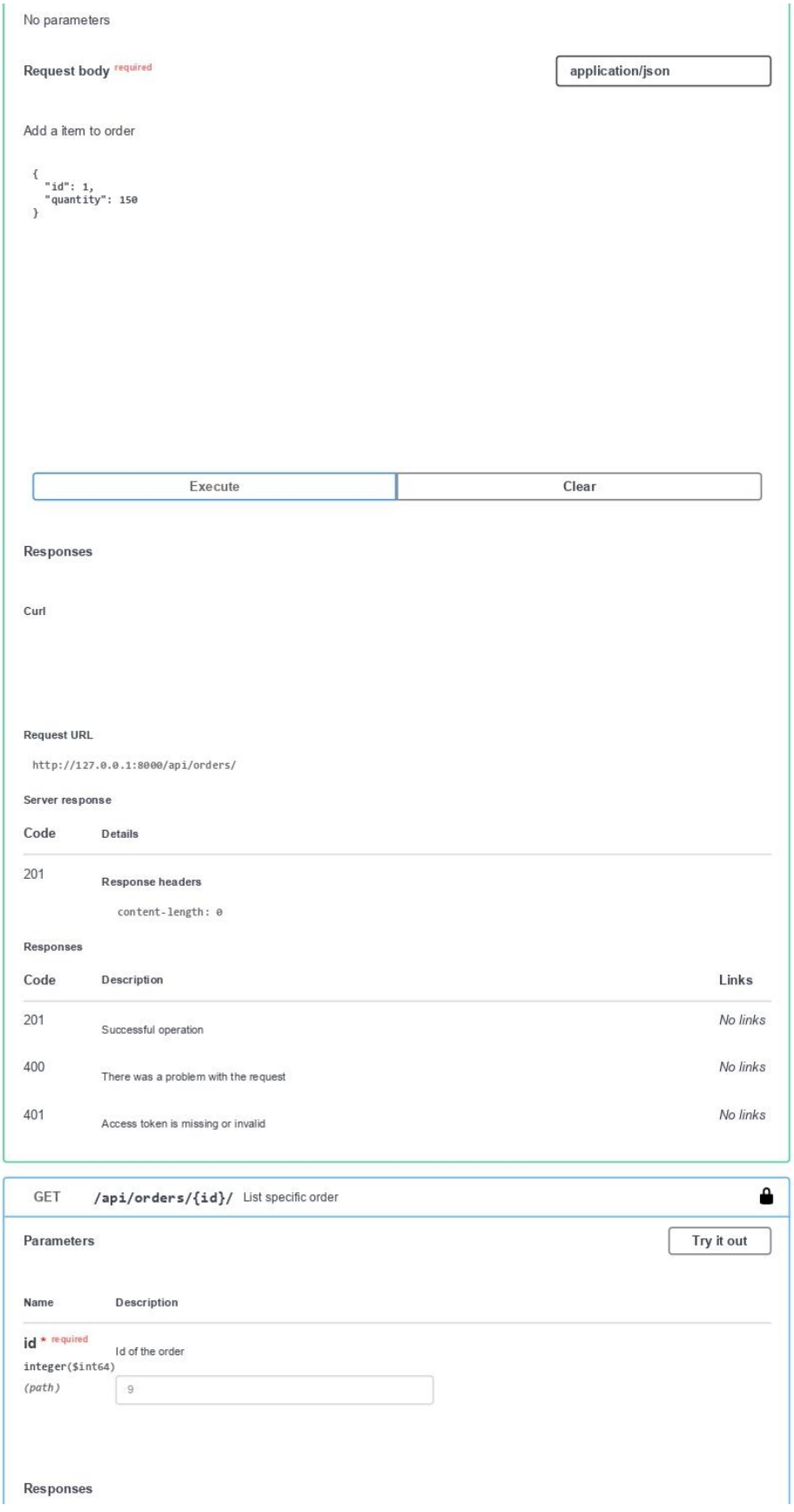

GET /api/items/{id}/ List specific item
Parameters
Try it out
Name
Description
id * required
integer($int64)
(path)
Id of the item
11
Responses
Curl
Request URL
http://127.0.0.1:8000/api/items/11/
Server response
Code
Details
200
Response body
[
{
"id": 11,
"is_ordered": false,
"date_added": "2020-06-04T02:19:00.159579Z",
"date_ordered": null,
"quantity": 150,
"user": 5,
"product": 1
}
]
Download
Response headers
content-length: 129
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
PUT /api/items/{id}/ Edit orders
Parameters
Try it out
Name
Description
id * required
integer($int64)
(path)
Id of the item
11
Request body required
application/json


Update user orders
Example Value Schema
{
"quantity": 200
}
Responses
Curl
Request URL
http://127.0.0.1:8000/api/items/11/
Server response
Code Details
200
Response headers
content-length: 0
Responses
Code Description Links
200 Successful operation No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links

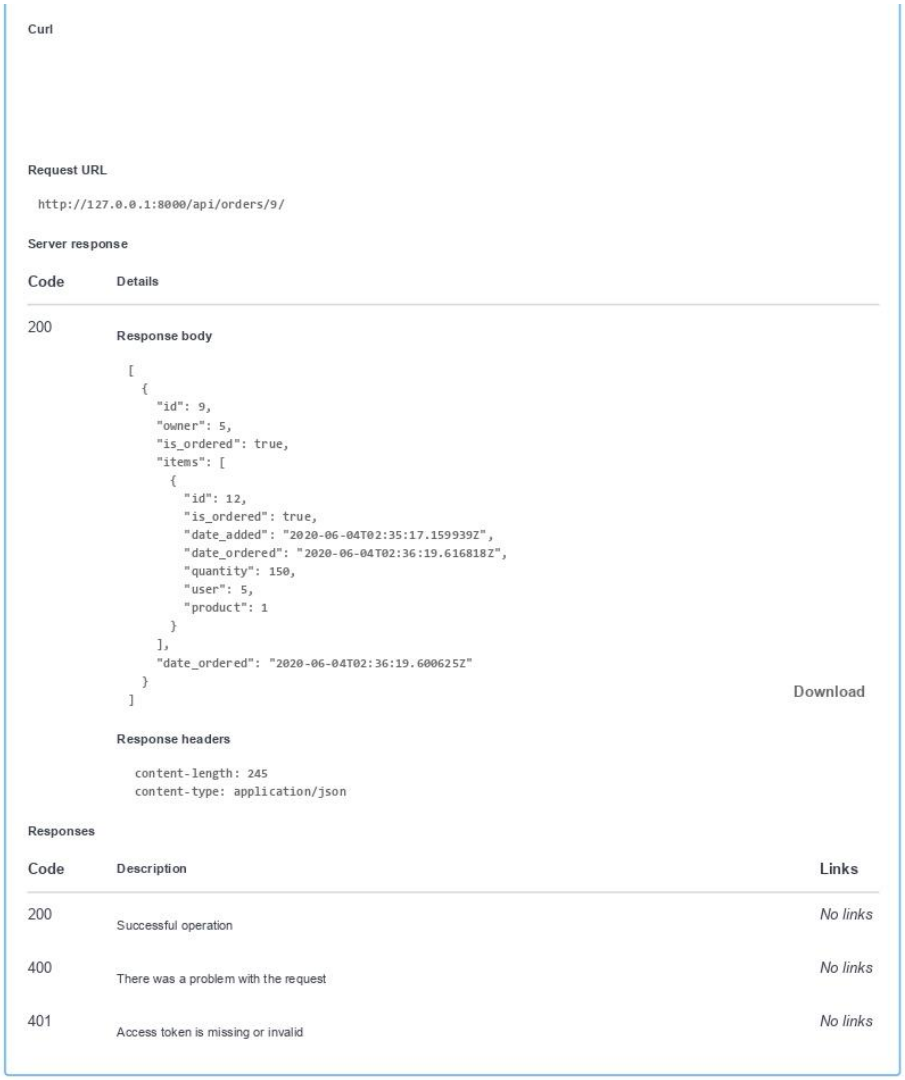

PATCH /api/items/{id}/ Edit orders
Parameters
Try it out
Name Description
id * required
integer($int64)
(path)
Id of the item
11
Request body required
application/json
Update user orders
Example Value Schema
{
"quantity": 255
}
Responses


Curl
Request URL
http://127.0.0.1:8000/api/items/11/
Server response
Code Details
200
Response headers
content-length: 0
Responses
Code Description Links
200 Successful operation No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links
DELETE /api/items/{id}/ Delete unsent item
Parameters
Try it out
Name Description
id * required
integer($int64)
(path)
Id of the item
11
Responses
Curl
Request URL
http://127.0.0.1:8000/api/items/11/
Server response
Code Details
204
Response headers
content-length: 0
Responses
Code Description Links
204 Successfl deleted No links
400 There was a problem with the request No links
401 Access token is missing or invalid No links


Code
Description
Links
409
Ther was a conflit with the request
No links
orders Orders
GET /api/orders/ List Orders
Parameters
Try it out
No parameters
Responses
Curl
Request URL
http://127.0.0.1:8000/api/orders/
Server response
Code
Details
200
Response body
[
{
"id": 9,
"owner": 5,
"is_ordered": true,
"items": [
{
"id": 12,
"is_ordered": true,
"date_added": "2020-06-04T02:35:17.159939Z",
"date_ordered": "2020-06-04T02:36:19.616818Z",
"quantity": 150,
"user": 5,
"product": 1
}
],
"date_ordered": "2020-06-04T02:36:19.600625Z"
},
{
"id": 10,
"owner": 5,
"is_ordered": false,
"items": [
Download
Response headers
content-length: 467
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
POST /api/orders/ Add a item to order
Parameters
Cancel


No parameters
Request body required
application/json
Add a item to order
{
"id": 1,
"quantity": 150
}
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/orders/
Server response
Code
Details
201
Response headers
content-length: 0
Responses
Code
Description
Links
201
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links

GET
/api/orders/{id}/
List specific order
Parameters
Try it out
Name
Description
id * required
integer($int64)
(path)
Id of the order
9
Responses



Curl

Request URL

```
http://127.0.0.1:8000/api/orders/9/
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 9,
    "owner": 5,
    "is_ordered": true,
    "items": [
      {
        "id": 12,
        "is_ordered": true,
        "date_added": "2020-06-04T02:35:17.159939Z",
        "date_ordered": "2020-06-04T02:36:19.616818Z",
        "quantity": 150,
        "user": 5,
        "product": 1
      }
    ],
    "date_ordered": "2020-06-04T02:36:19.600625Z"
  }
]
```

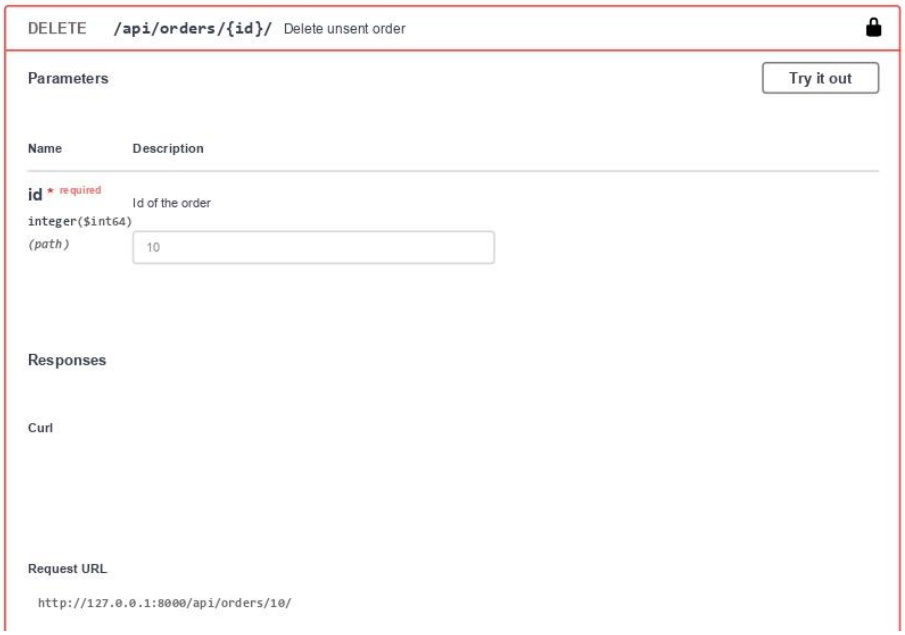
Download

Response headers

```
content-length: 245
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*



DELETE **/api/orders/{id}/** Delete unsent order

Parameters — Try it out

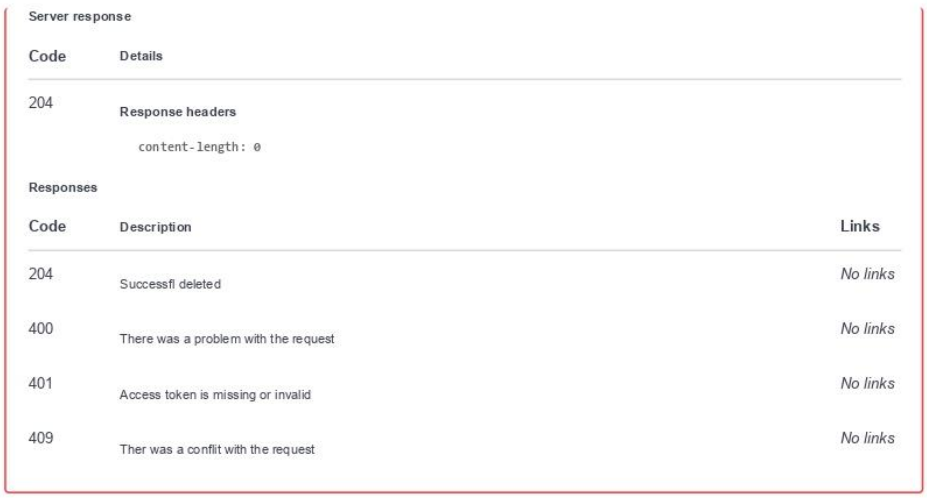
Name	Description
id * required integer($int64) *(path)*	Id of the order 10

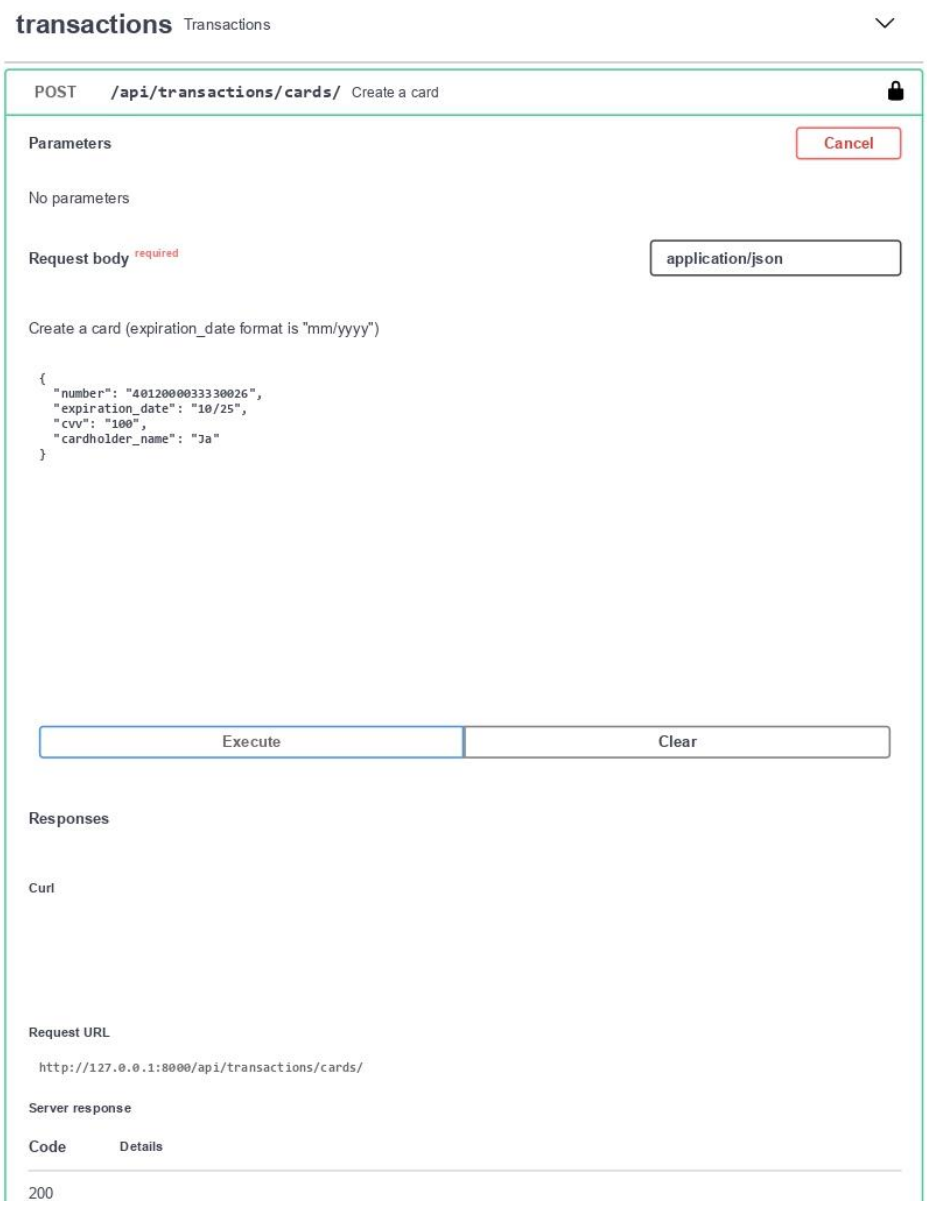
Responses

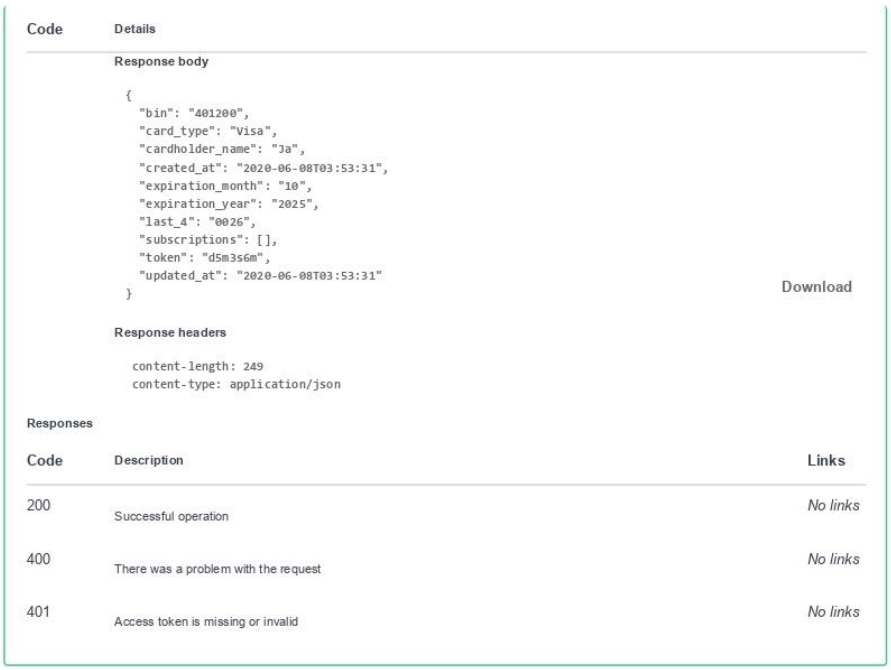
Curl

Request URL

```
http://127.0.0.1:8000/api/orders/10/
```


Server response
Code
Details
204
Response headers
content-length: 0
Responses
Code
Description
Links
204
Successfl deleted
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
409
Ther was a conflit with the request
No links


transactions Transactions
POST
/api/transactions/cards/
Create a card
Parameters
Cancel
No parameters
Request body
required
application/json
Create a card (expiration_date format is "mm/yyyy")
{
"number": "4012000033330026",
"expiration_date": "10/25",
"cvv": "100",
"cardholder_name": "Ja"
}
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/cards/
Server response
Code
Details
200



Code Details

Response body


```
{
  "bin": "401200",
  "card_type": "Visa",
  "cardholder_name": "Ja",
  "created_at": "2020-06-08T03:53:31",
  "expiration_month": "10",
  "expiration_year": "2025",
  "last_4": "0026",
  "subscriptions": [],
  "token": "d5m3s6m",
  "updated_at": "2020-06-08T03:53:31"
}
```

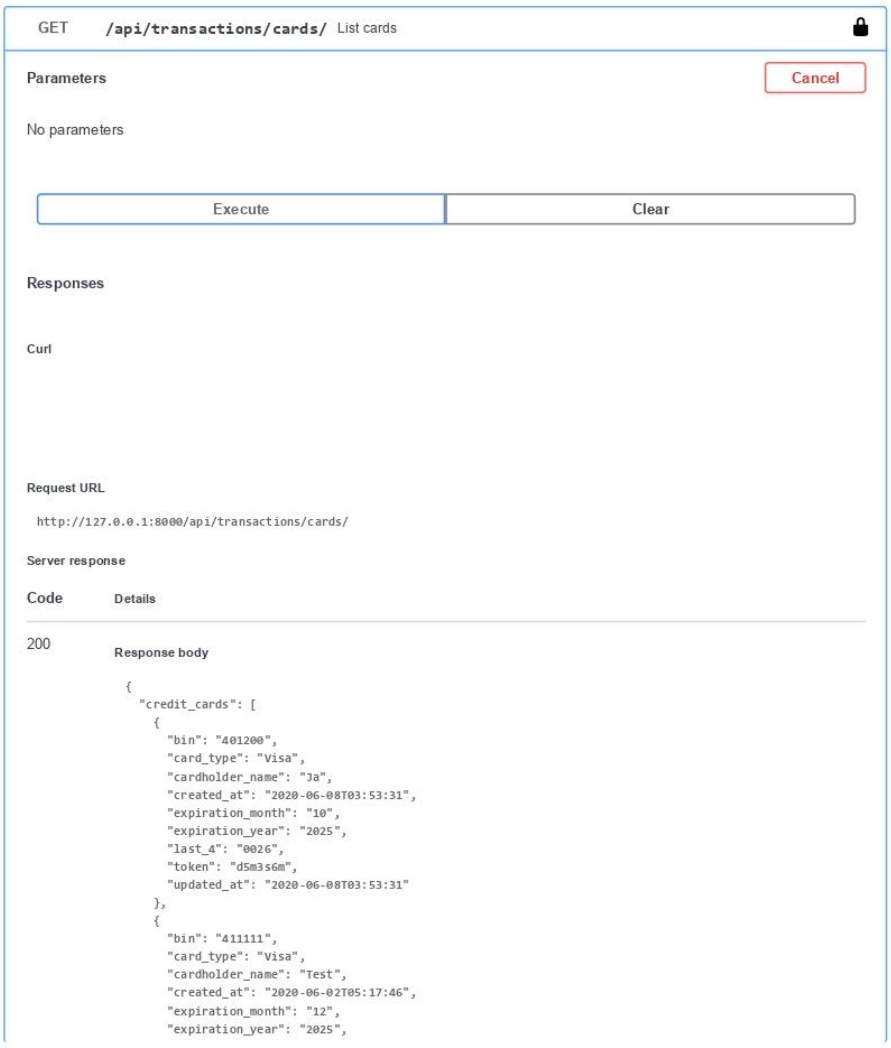

Download

Response headers

```
content-length: 249
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*



GET **/api/transactions/cards/** List cards

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/transactions/cards/
```

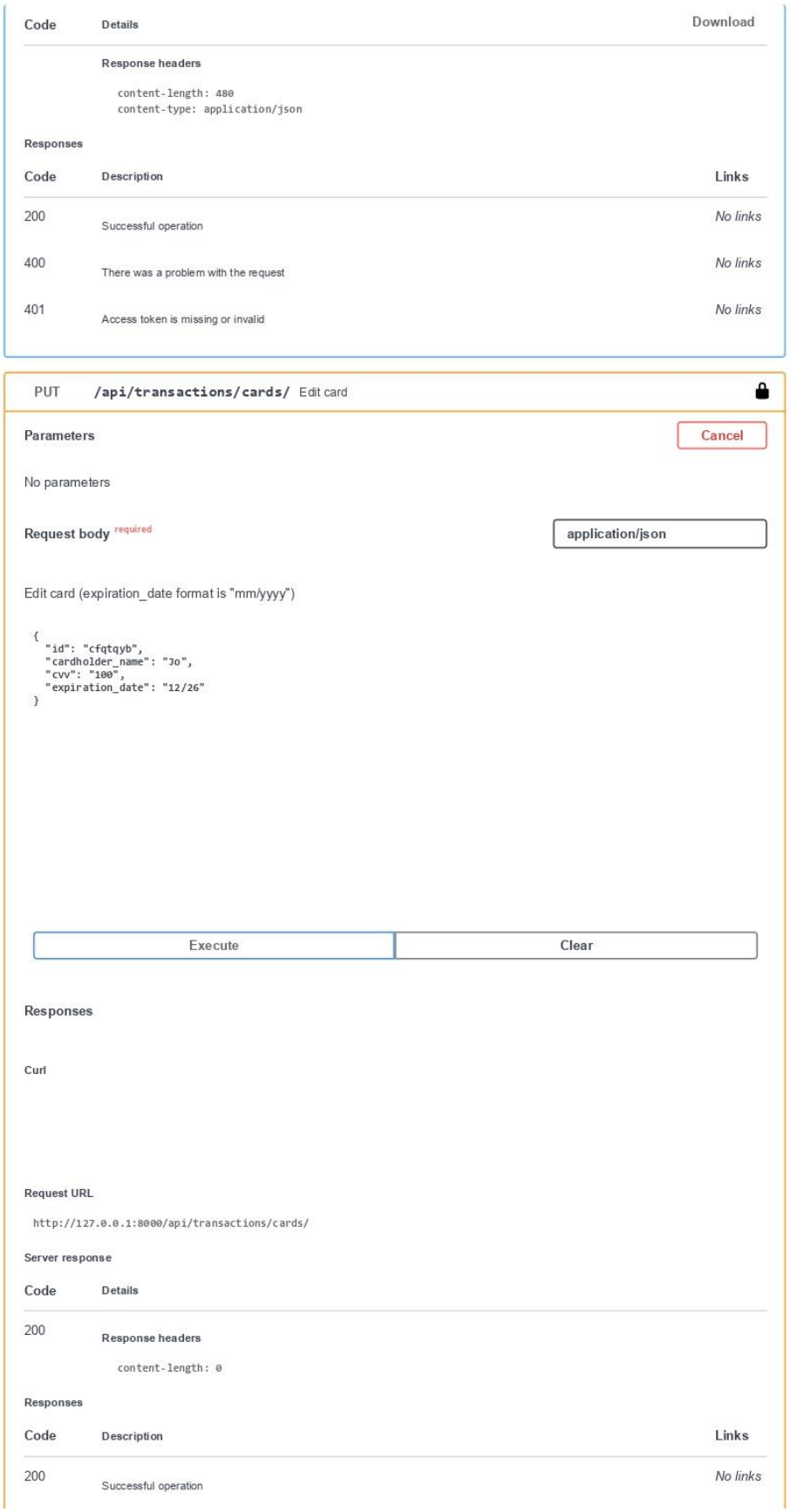
Server response

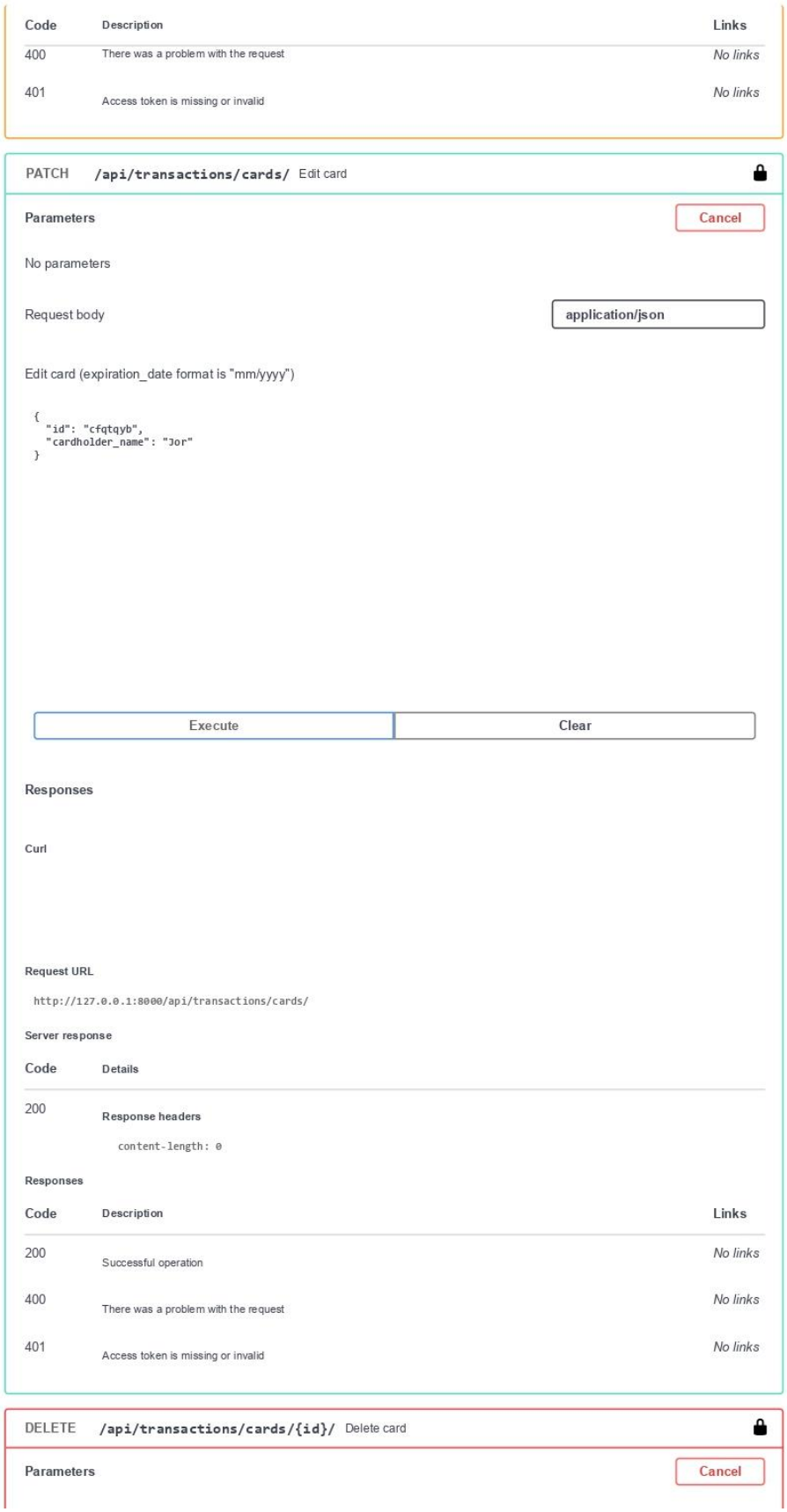
Code Details

200

Response body


```
{
  "credit_cards": [
    {
      "bin": "401200",
      "card_type": "Visa",
      "cardholder_name": "Ja",
      "created_at": "2020-06-08T03:53:31",
      "expiration_month": "10",
      "expiration_year": "2025",
      "last_4": "0026",
      "token": "d5m3s6m",
      "updated_at": "2020-06-08T03:53:31"
    },
    {
      "bin": "411111",
      "card_type": "Visa",
      "cardholder_name": "Test",
      "created_at": "2020-06-02T05:17:46",
      "expiration_month": "12",
      "expiration_year": "2025",
```


Code
Details
Download
Response headers
content-length: 480
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
PUT
/api/transactions/cards/
Edit card
Parameters
Cancel
No parameters
Request body
required
application/json
Edit card (expiration_date format is "mm/yyyy")
{
"id": "cfqtqyb",
"cardholder_name": "Jo",
"cvv": "100",
"expiration_date": "12/26"
}
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/cards/
Server response
Code
Details
200
Response headers
content-length: 0
Responses
Code
Description
Links
200
Successful operation
No links



Code	Description	Links
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

PATCH **/api/transactions/cards/** Edit card

Parameters Cancel

No parameters

Request body application/json

Edit card (expiration_date format is "mm/yyyy")

```
{
  "id": "cfqtqyb",
  "cardholder_name": "Jor"
}
```

Execute Clear

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/transactions/cards/
```

Server response

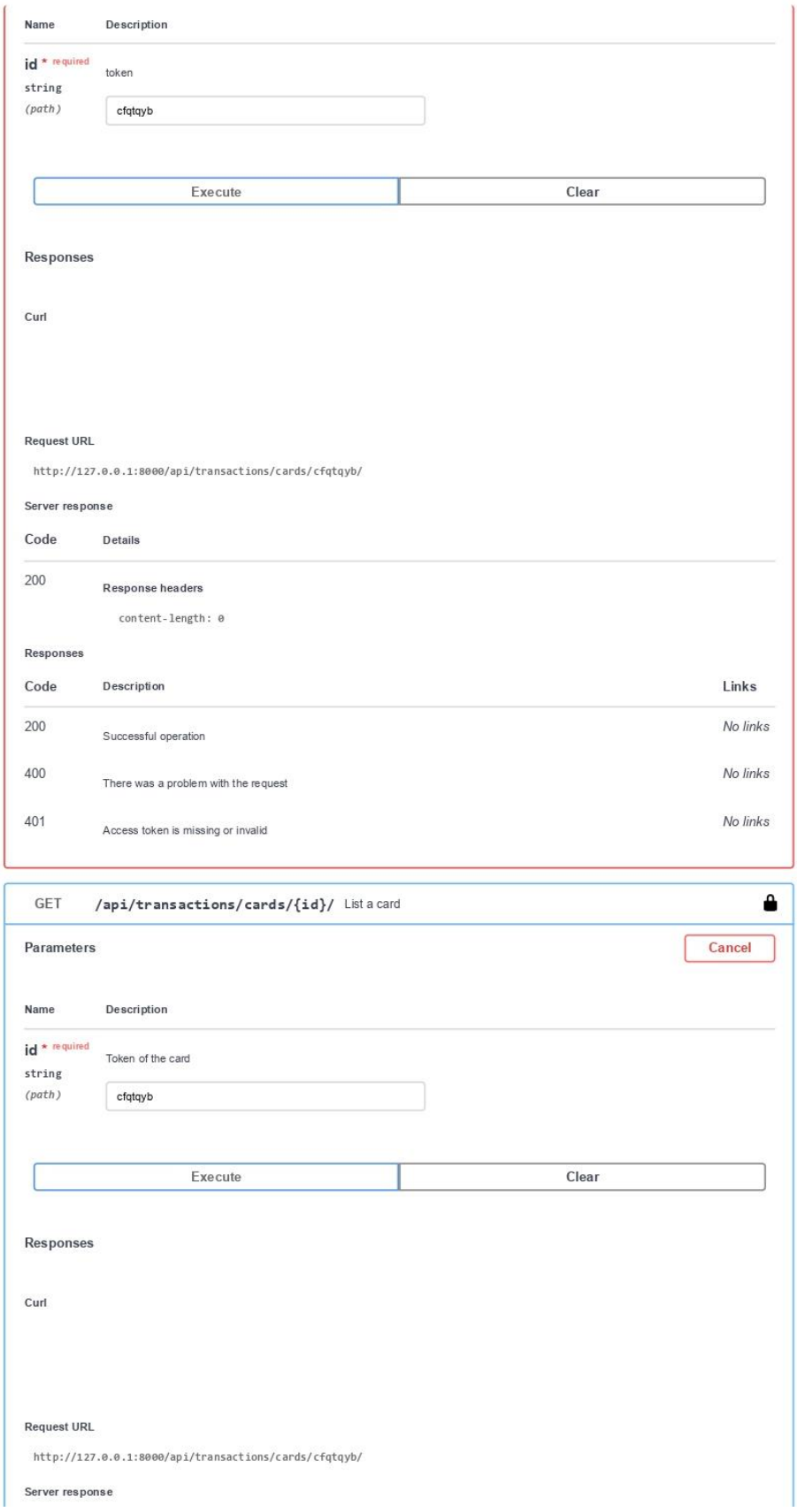
Code	Details
200	**Response headers** `content-length: 0`

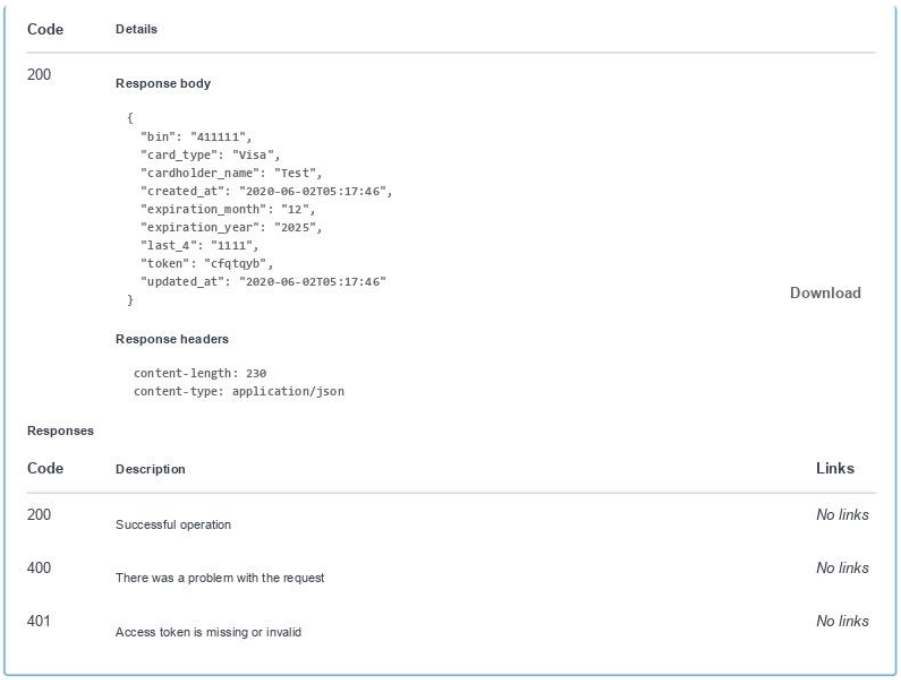
Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

DELETE **/api/transactions/cards/{id}/** Delete card

Parameters Cancel


Name
Description
id * required
string
(path)
token
cfqtqyb
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/cards/cfqtqyb/
Server response
Code
Details
200
Response headers
content-length: 0
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
GET
/api/transactions/cards/{id}/
List a card
Parameters
Cancel
Name
Description
id * required
string
(path)
Token of the card
cfqtqyb
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/cards/cfqtqyb/
Server response



Code | Details

200

Response body

```
{
  "bin": "411111",
  "card_type": "Visa",
  "cardholder_name": "Test",
  "created_at": "2020-06-02T05:17:46",
  "expiration_month": "12",
  "expiration_year": "2025",
  "last_4": "1111",
  "token": "cfqtqyb",
  "updated_at": "2020-06-02T05:17:46"
}
```

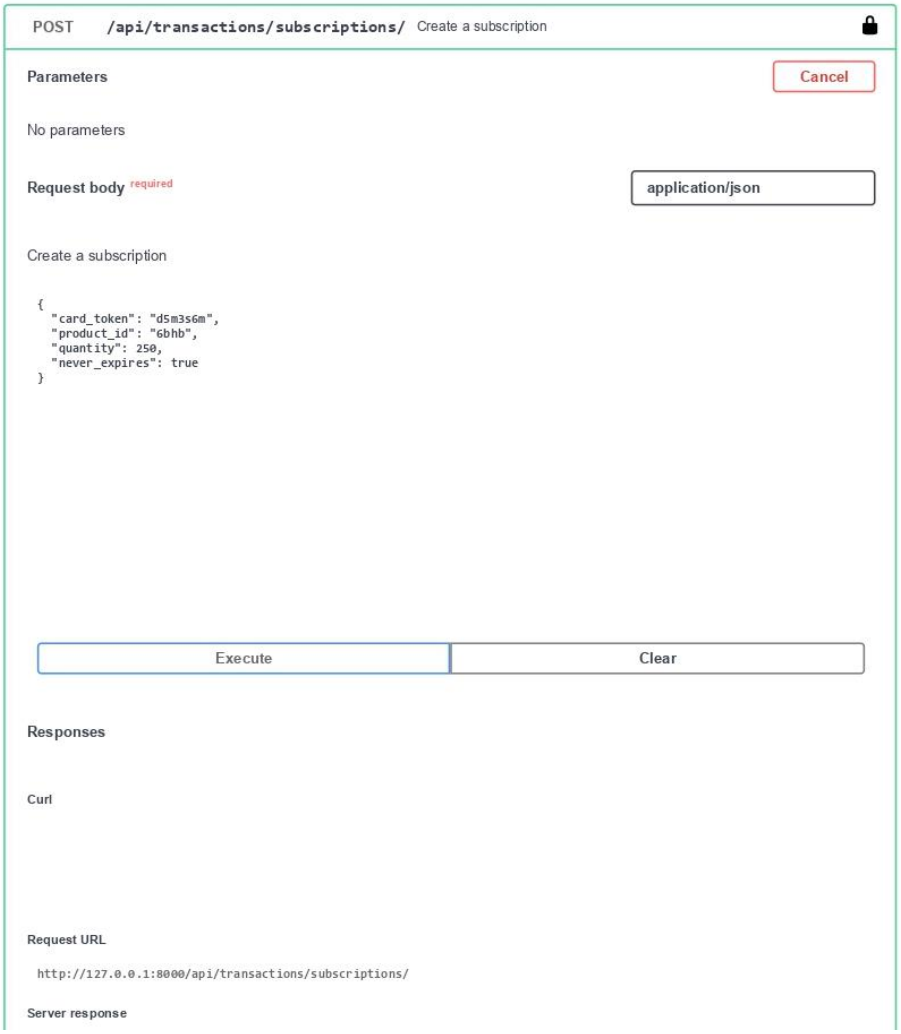
Download

Response headers

```
content-length: 230
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*



POST /api/transactions/subscriptions/ Create a subscription

Parameters | Cancel

No parameters

Request body required | application/json

Create a subscription

```
{
  "card_token": "d5m3s6m",
  "product_id": "6bhb",
  "quantity": 250,
  "never_expires": true
}
```

Execute | Clear

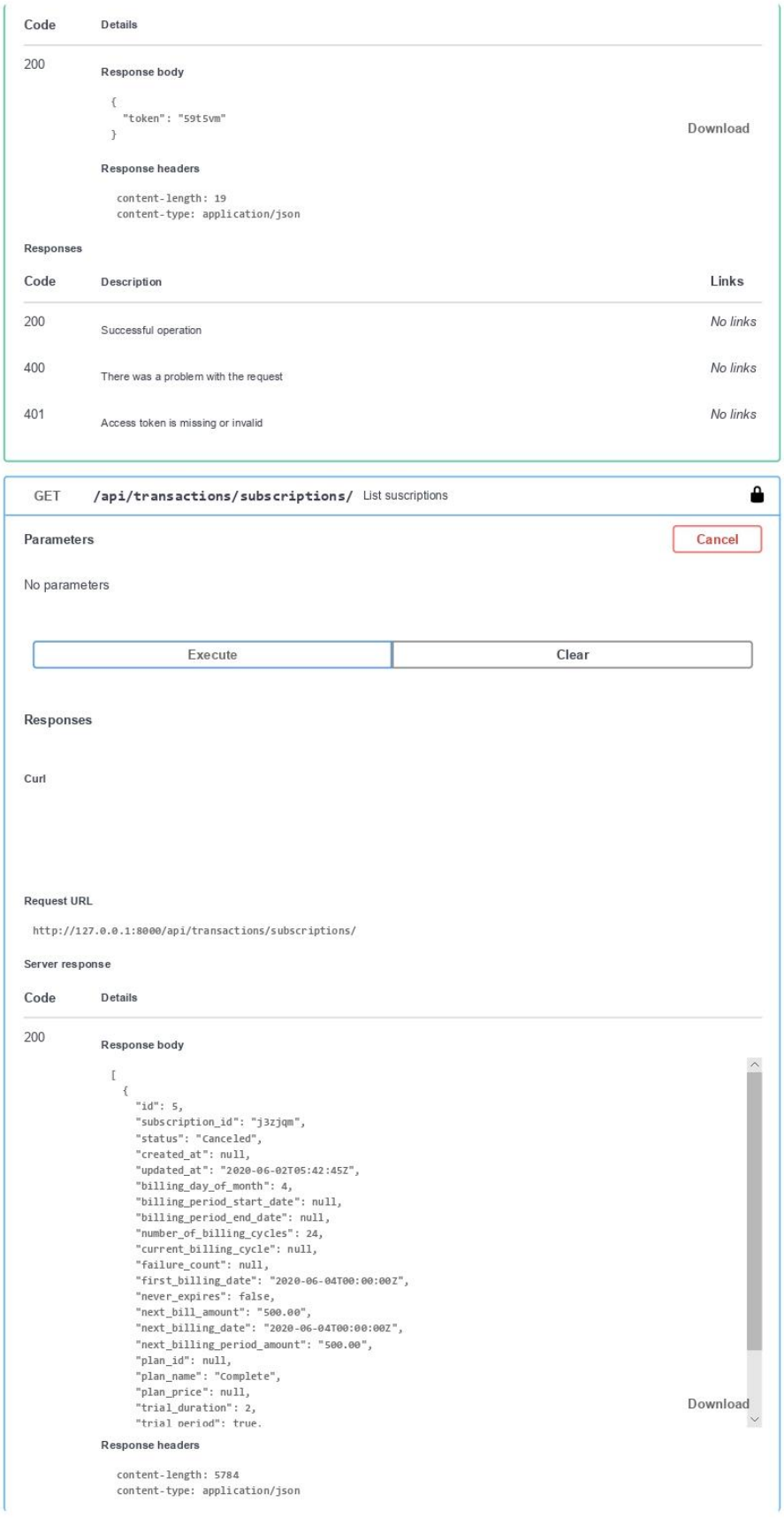
Responses

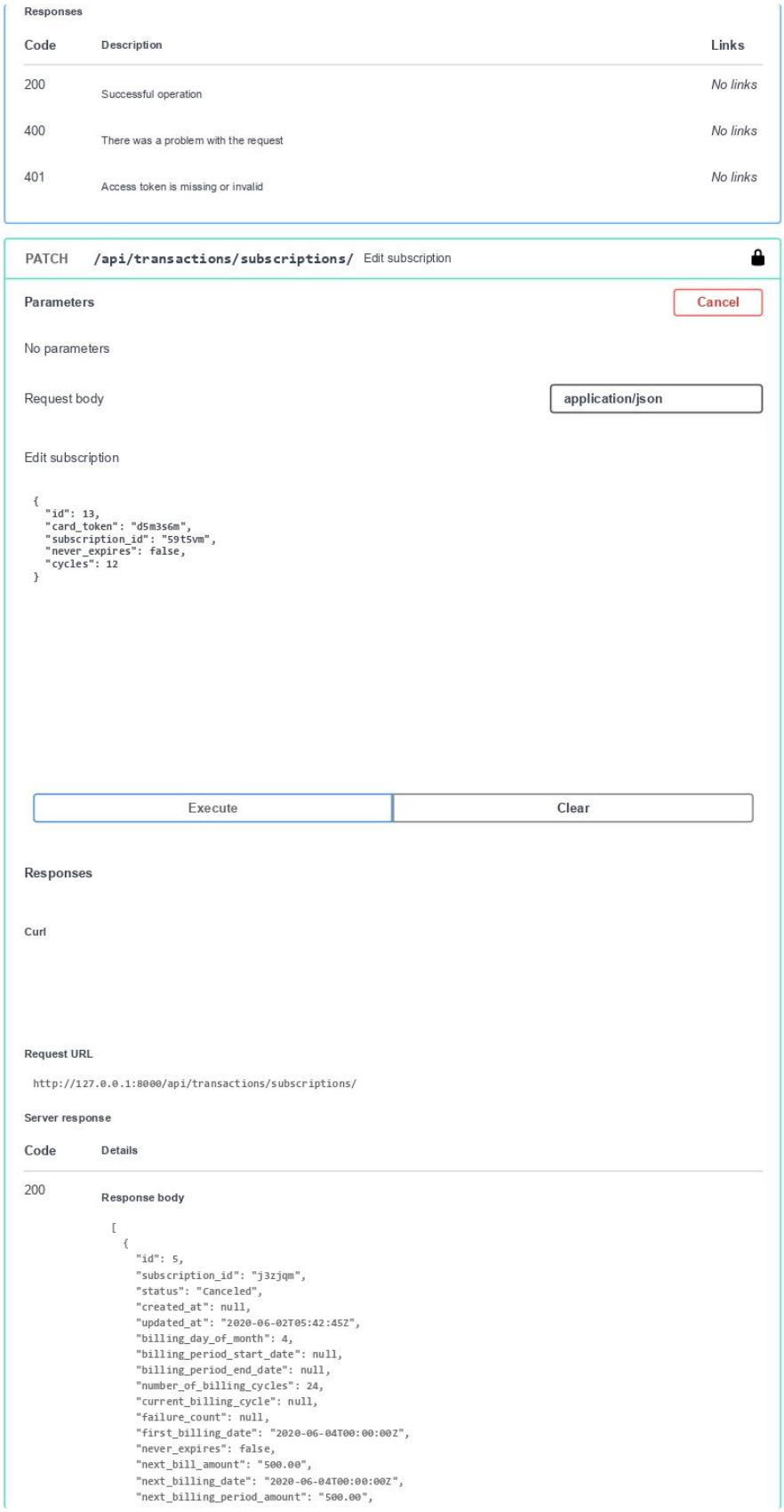
Curl

Request URL

```
http://127.0.0.1:8000/api/transactions/subscriptions/
```

Server response


Code
Details
200
Response body
{
"token": "59t5vm"
}
Download
Response headers
content-length: 19
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
GET
/api/transactions/subscriptions/
List suscriptions
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/subscriptions/
Server response
Code
Details
200
Response body
[
{
"id": 5,
"subscription_id": "j3zjqm",
"status": "Canceled",
"created_at": null,
"updated_at": "2020-06-02T05:42:45Z",
"billing_day_of_month": 4,
"billing_period_start_date": null,
"billing_period_end_date": null,
"number_of_billing_cycles": 24,
"current_billing_cycle": null,
"failure_count": null,
"first_billing_date": "2020-06-04T00:00:00Z",
"never_expires": false,
"next_bill_amount": "500.00",
"next_billing_date": "2020-06-04T00:00:00Z",
"next_billing_period_amount": "500.00",
"plan_id": null,
"plan_name": "Complete",
"plan_price": null,
"trial_duration": 2,
"trial_period": true,
Download
Response headers
content-length: 5784
content-type: application/json



Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

PATCH **/api/transactions/subscriptions/** Edit subscription

Parameters — Cancel

No parameters

Request body — application/json

Edit subscription

```
{
  "id": 13,
  "card_token": "d5m3s6m",
  "subscription_id": "59t5vm",
  "never_expires": false,
  "cycles": 12
}
```

Execute — Clear

Responses

Curl

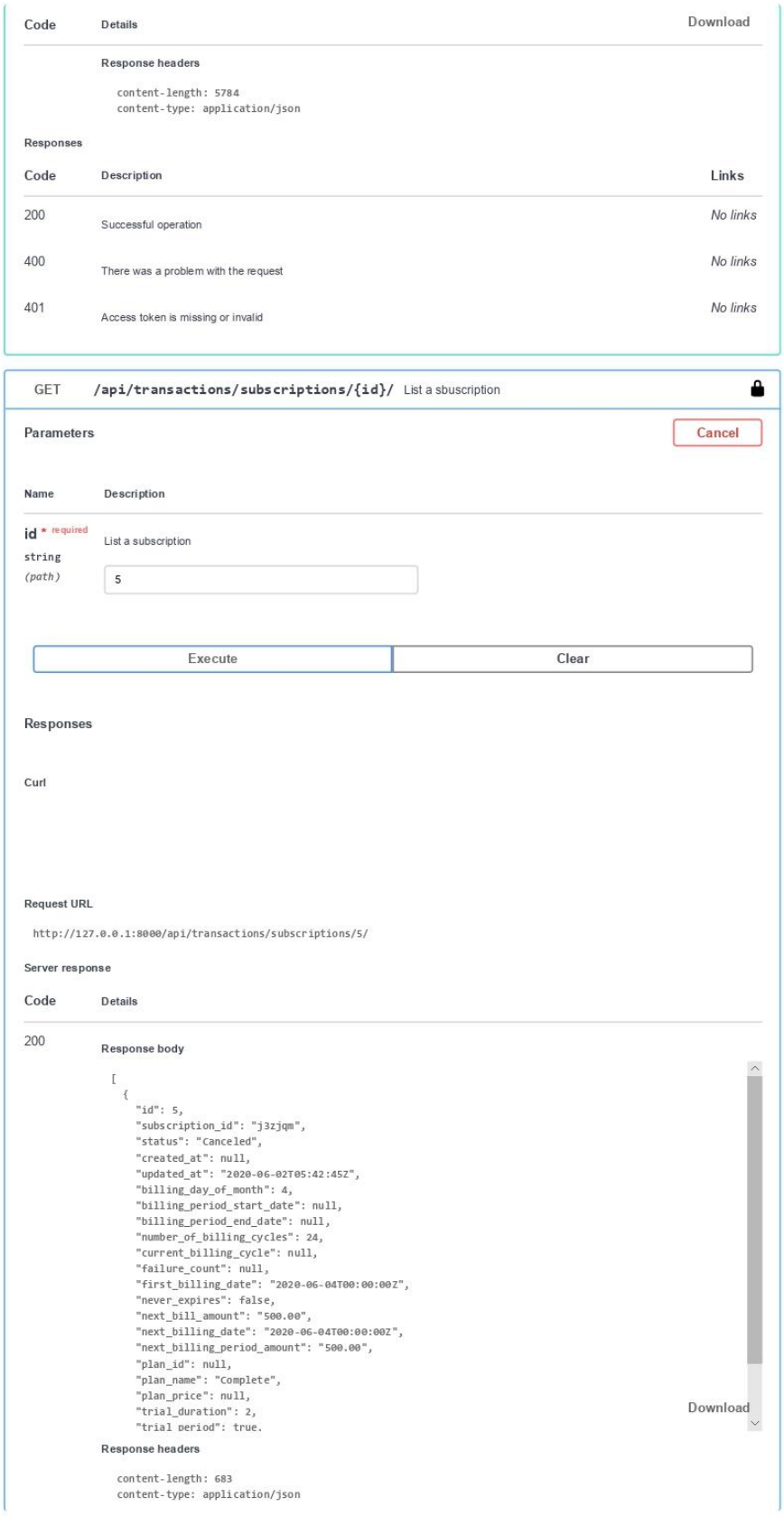
Request URL

```
http://127.0.0.1:8000/api/transactions/subscriptions/
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 5,
    "subscription_id": "j3zjqm",
    "status": "Canceled",
    "created_at": null,
    "updated_at": "2020-06-02T05:42:45Z",
    "billing_day_of_month": 4,
    "billing_period_start_date": null,
    "billing_period_end_date": null,
    "number_of_billing_cycles": 24,
    "current_billing_cycle": null,
    "failure_count": null,
    "first_billing_date": "2020-06-04T00:00:00Z",
    "never_expires": false,
    "next_bill_amount": "500.00",
    "next_billing_date": "2020-06-04T00:00:00Z",
    "next_billing_period_amount": "500.00",
```



Code Details Download

Response headers

```
content-length: 5784
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*

GET **/api/transactions/subscriptions/{id}/** List a sbuscription

Parameters Cancel

Name	Description
id * required string *(path)*	List a subscription 5

Execute Clear

Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/transactions/subscriptions/5/
```

Server response

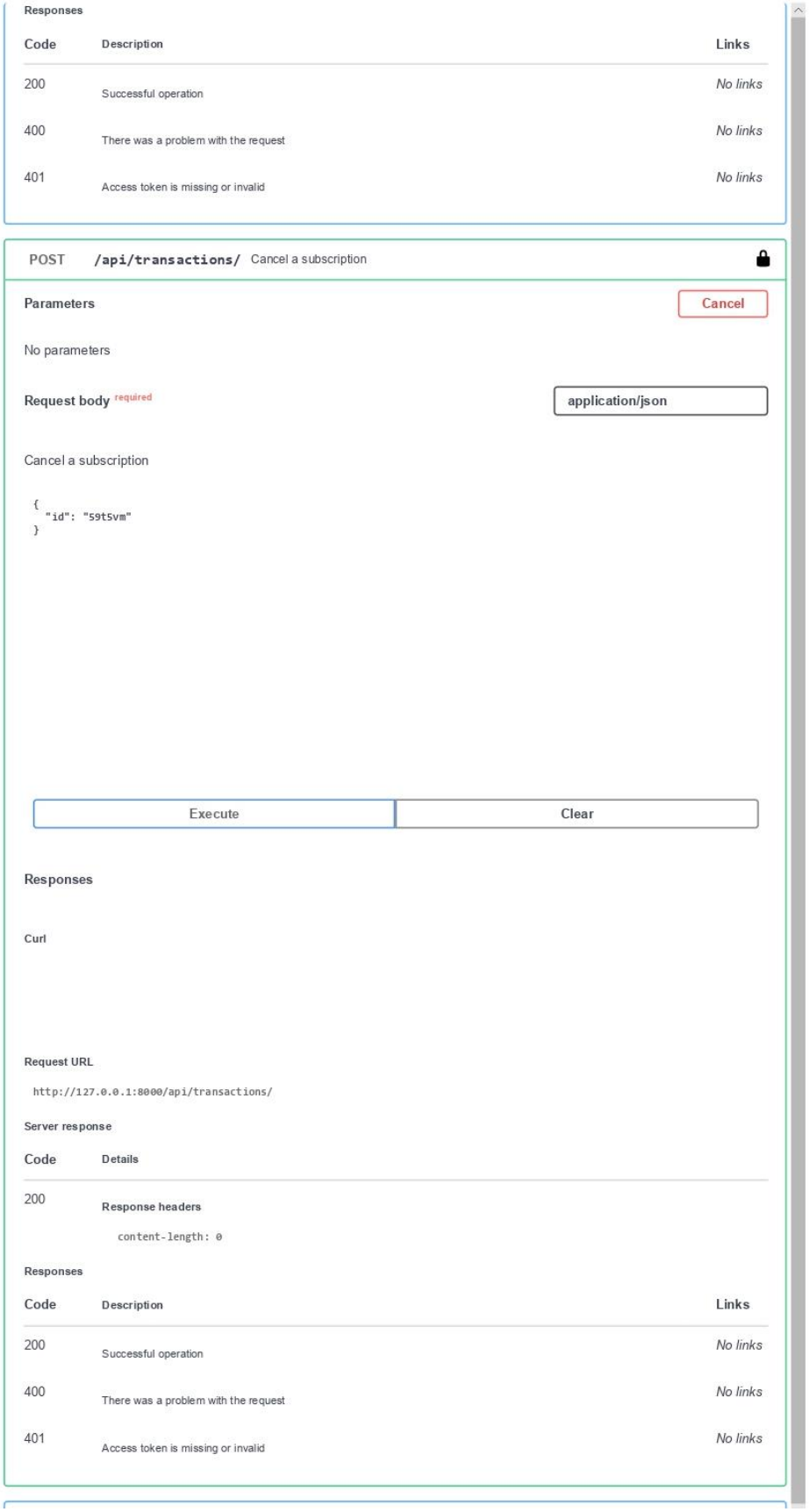
Code	Details
200	Response body

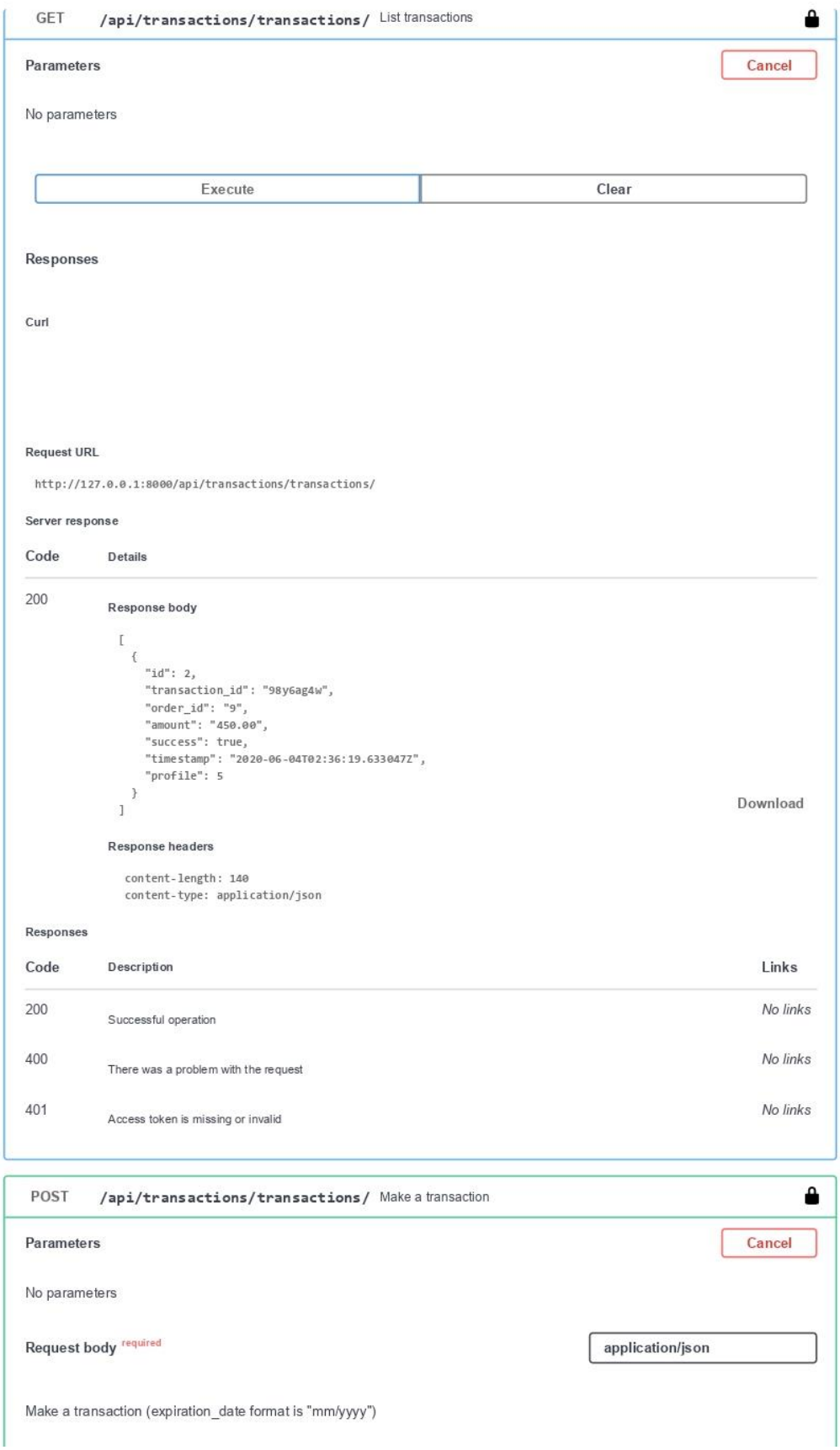

```
[
  {
    "id": 5,
    "subscription_id": "j3zjqm",
    "status": "Canceled",
    "created_at": null,
    "updated_at": "2020-06-02T05:42:45Z",
    "billing_day_of_month": 4,
    "billing_period_start_date": null,
    "billing_period_end_date": null,
    "number_of_billing_cycles": 24,
    "current_billing_cycle": null,
    "failure_count": null,
    "first_billing_date": "2020-06-04T00:00:00Z",
    "never_expires": false,
    "next_bill_amount": "500.00",
    "next_billing_date": "2020-06-04T00:00:00Z",
    "next_billing_period_amount": "500.00",
    "plan_id": null,
    "plan_name": "Complete",
    "plan_price": null,
    "trial_duration": 2,
    "trial_period": true,
```

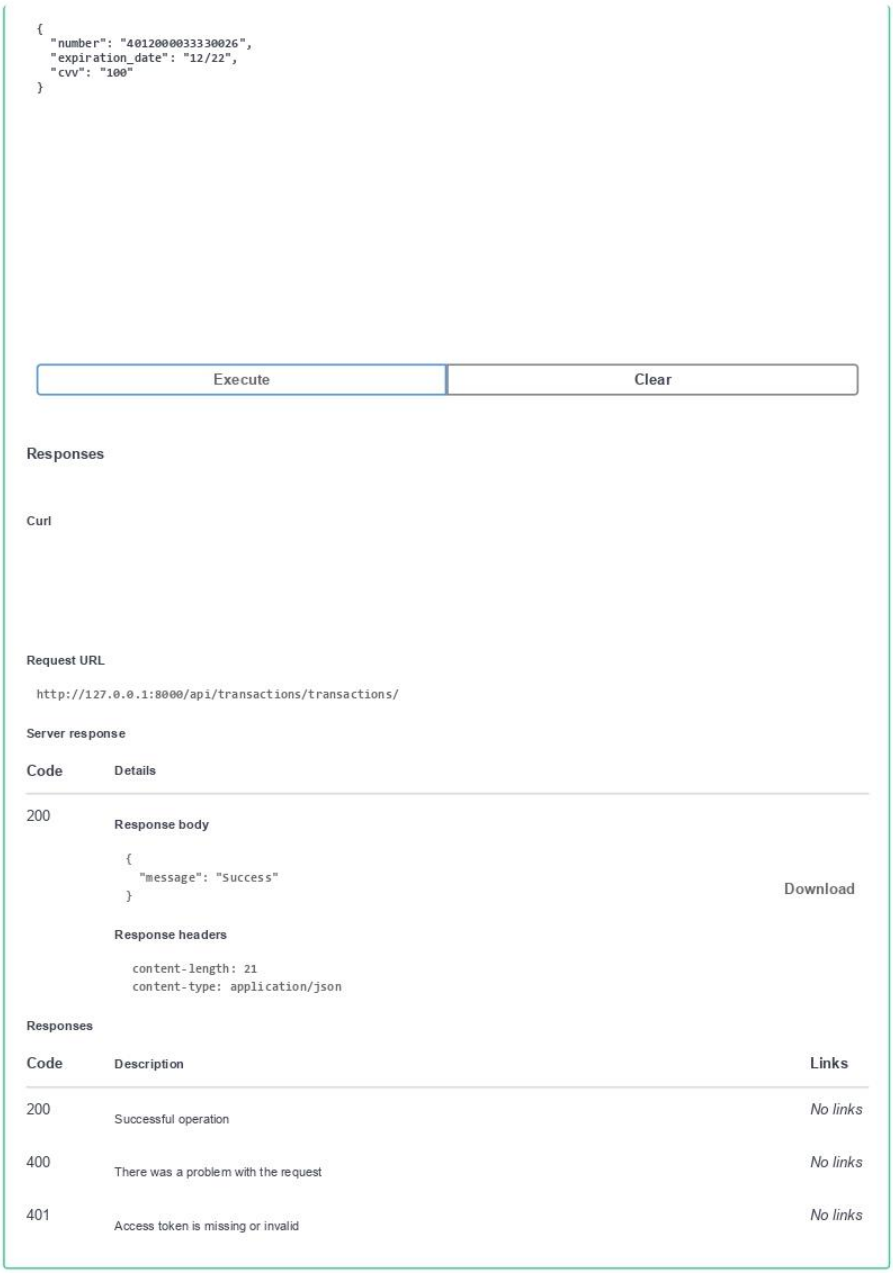

Download

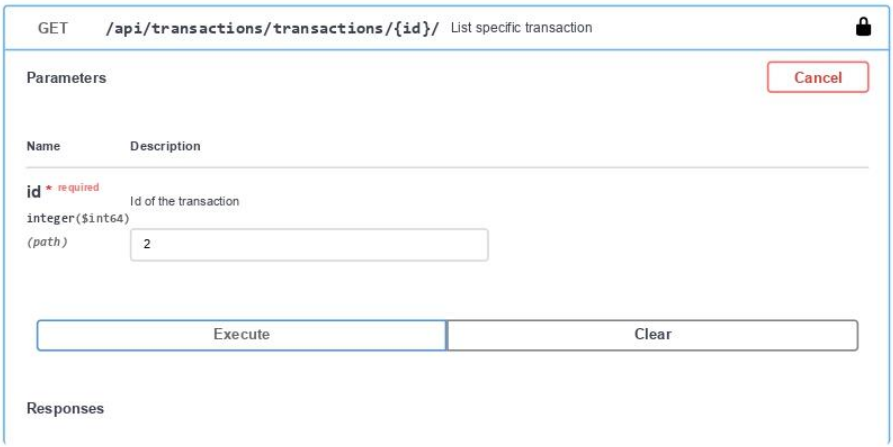
Response headers

```
content-length: 683
content-type: application/json
```


Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
POST
/api/transactions/
Cancel a subscription
Parameters
Cancel
No parameters
Request body
required
application/json
Cancel a subscription
{
"id": "59t5vm"
}
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/
Server response
Code
Details
200
Response headers
content-length: 0
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links


GET /api/transactions/transactions/ List transactions
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/transactions/
Server response
Code
Details
200
Response body
[
{
"id": 2,
"transaction_id": "98y6ag4w",
"order_id": "9",
"amount": "450.00",
"success": true,
"timestamp": "2020-06-04T02:36:19.633047Z",
"profile": 5
}
]
Download
Response headers
content-length: 140
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links
POST /api/transactions/transactions/ Make a transaction
Parameters
Cancel
No parameters
Request body required
application/json
Make a transaction (expiration_date format is "mm/yyyy")


{
"number": "4012000033330026",
"expiration_date": "12/22",
"cvv": "100"
}
Execute
Clear
Responses
Curl
Request URL
http://127.0.0.1:8000/api/transactions/transactions/
Server response
Code
Details
200
Response body
{
"message": "Success"
}
Download
Response headers
content-length: 21
content-type: application/json
Responses
Code
Description
Links
200
Successful operation
No links
400
There was a problem with the request
No links
401
Access token is missing or invalid
No links


GET
/api/transactions/transactions/{id}/
List specific transaction
Parameters
Cancel
Name
Description
id * required
integer($int64)
(path)
Id of the transaction
2
Execute
Clear
Responses

Curl

Request URL

```
http://127.0.0.1:8000/api/transactions/transactions/2/
```

Server response

Code	Details
200	**Response body** Download

```
[
  {
    "id": 2,
    "transaction_id": "98y6ag4w",
    "order_id": "9",
    "amount": "450.00",
    "success": true,
    "timestamp": "2020-06-04T02:36:19.633047Z",
    "profile": 5
  }
]
```

Response headers

```
content-length: 140
content-type: application/json
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	There was a problem with the request	*No links*
401	Access token is missing or invalid	*No links*