Abstract

This report presents the activities that I have performed during my MSc curricular internship at iTGROW. In this internship, I had the opportunity to join a Critical Software project, designated as *moduWeb Vision*, a web solution for the operation of HVAC systems (heating, ventilation and air conditioning) for the client company SAUTER, allowing the monitoring and control of buildings.

One of the components of this project is the automation of spaces. An automaton has recently been developed that deals with complex functions in order to allow precise control of temperature, lighting, shading, among others, with the internship being focused on the development of a web solution for management and configuration of these spaces and segments / divisions contained therein.

The development comprised a front-end part where an interface was created that allows the user to configure the segments and another back-end part where configuration information is propagated to the automatons.

This report presents the technology used in component development, its requirements, its architecture and its implementation. Regarding my participation in the validation of the solution, this report presents the testing process applied in the project, as well as the activities carried out in this area.

Keywords:

Building-automation, Java, JEE, jQuery UI

Resumo

Este relatório apresenta as atividades realizadas durante o meu estágio curricular de Mestrado na empresa iTGROW. Neste estágio tive a oportunidade de integrar um projeto da Critical Software denominado *moduWeb Vision*, uma solução web para visualização e operação de sistemas AVAC (aquecimento, ventilação e ar condicionado) para a empresa Sauter, permitindo a monitorização e controlo de edifícios.

Um dos componentes desse projeto é a automação de espaços, tendo sido desenvolvido recentemente um autómato que lida com funções complexas de maneira a permitir um controlo preciso da temperatura, iluminação, sombra, entre outros, tendo o estágio incidido no desenvolvimento de uma solução *web* para gestão e configuração desses espaços e segmentos/divisões neles contidos.

O desenvolvimento envolveu uma parte *front-end* em que foi criada uma interface que permite ao utilizador configurar os segmentos e uma outra parte *back-end* onde a informação relativa à configuração é propagada para os autómatos.

Este relatório apresenta a tecnologia utilizada no desenvolvimento do componente, os seus requisitos, a sua arquitetura e a respetiva implementação. Relativamente à minha participação na validação da solução, este relatório apresenta o processo de testes aplicado no projeto, bem como as atividades realizadas nessa área.

Palavras-chave:

Building-automation, Java, JEE, jQuery UI

Agradecimentos

A realização do meu estágio na empresa iTGROW representa o culminar do meu Mestrado em Informática e Sistemas, ramo de Desenvolvimento de Software, lecionado no Departamento de Engenharia Informática e de Sistemas (DEIS) do Instituto Superior de Engenharia de Coimbra (ISEC). O culminar desta etapa só foi possível graças ao contributo de diversas pessoas, às quais gostaria de deixar o meu agradecimento:

- Ao professor Francisco Pereira, meu orientador de estágio do ISEC, por toda a disponibilidade que demonstrou desde o início e por todos os conselhos que me foi transmitindo ao longo do estágio e da elaboração deste relatório.

- Ao Eng.º Nelson Vale, meu coordenador de estágio por parte da iTGROW, por todo o acompanhamento diário que me forneceu na empresa e por toda a experiência que me foi transmitindo ao longo do estágio.

- À minha família, pelo apoio prestado ao longo de todo o meu percurso académico, pois só assim foi possível superar muitas das dificuldades encontradas.

- Aos restantes professores do DEIS, que me transmitiram conhecimentos importantes não só para a realização deste estágio, mas para toda a minha formação académica.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

AVAC – Aquecimento, Ventilação e Ar Condicionado

BAC – Build Automation Control

CSS – Cascading Style Sheets

DEIS – Departamento de Engenharia Informática e de Sistemas

DOM – Document Object Model

HMTL – HyperText Markup Language

HTTP - HyperText Transfer Protocol

HTTPS - HyperText Transfer Protocol Secure

ISEC – Instituto Superior de Engenharia de Coimbra

JEE – Java Enterprise Edition

JS – JavaScript

JSON – JavaScript Object Notation

JSP – JavaServer Pages

SMS – Short Message Service

SVO – Structured View Object

XML – Extensible Markup Language

XSD – XML Schema Definition

1 Introdução

Este relatório foi desenvolvido no âmbito da unidade curricular de Estágio ou Projeto Industrial do Mestrado em Informática e Sistemas, ramo de Desenvolvimento de Software, lecionado no Departamento de Engenharia Informática e de Sistemas do Instituto Superior de Engenharia de Coimbra. O objetivo desta unidade curricular é integrar o aluno num projeto de *software* de elevada dimensão, sendo dada a oportunidade de escolha entre um projeto industrial ou um estágio numa empresa. Face a estas duas opções, decidi integrar a minha colaboração na iTGROW com o meu estágio curricular de mestrado. Deste modo, o presente relatório pretende sintetizar de uma forma clara e concisa todas as atividades desenvolvidas na iTGROW durante o meu estágio curricular.

O projeto que desenvolvi durante o estágio faz parte de um produto desenvolvido pela Critical Software para a empresa Suíça Sauter [1] (a Figura 1 apresenta o logotipo da empresa) intitulado *moduWeb Vision,* uma solução de visualização compacta baseada na web, e opcionalmente com ecrã *touch-screen* otimizado, que possibilita a operação de sistemas AVAC (aquecimento, ventilação e ar condicionado), permitindo a monitorização e controlo de edifícios a partir de qualquer lugar. Como parte da família de sistemas SAUTER EY-modulo, o *moduWeb Vision* da Sauter comunica via BACnet/IP e permite a integração de dispositivos de terceiros.

Figura 1 - Logotipo da empresa Sauter Group

O *moduWeb Vision* tem a capacidade de exibir componentes dos autómatos de um edifício em tempo real em forma de listas ou gráficos. A representação interativa e os diagramas de plantas dinâmicos fazem com que toda a informação esteja disponível num ápice. No caso de ocorrer alguma anomalia, o *moduWeb Vision* tem a capacidade de notificar os utilizadores instantaneamente por *email* ou SMS.

O *moduWeb Vision* é bastante simples de operar, seja através de um *browser*, de uma consola de um quadro elétrico, ou de uma forma móvel, através de um *tablet*, por exemplo. Onde quer que o utilizador esteja, pode ter um acesso conveniente ao centro de controlo da sua instalação.

Uma das áreas de especialização da Sauter é a automação de espaços, tendo desenvolvido recentemente um autómato (representado pela Figura 2) que lida com funções complexas de maneira a permitir um controlo preciso da climatização, iluminação e estores das diversas divisões que compõem um edifício, por forma a minimizar o consumo energético. Nesse sentido, o projeto que desenvolvi está relacionado com o desenvolvimento de uma solução para a gestão e configuração desses espaços/divisões, com o objetivo primário de conceder ao utilizador uma forma de gerir/configurar os vários segmentos que constituem cada espaço/divisão de uma forma flexível, sendo que essa configuração é posteriormente propagada para os autómatos.

O desenvolvimento deste projeto envolveu uma parte de *front-end* na qual desenvolvi uma interface que permite aos utilizadores configurar os segmentos e espaços/divisões, e uma parte de *back-end* onde a informação relativa à configuração é propagada para os autómatos. Uma vez que existiam limitações nos autómatos no que toca à escrita das configurações relativas aos espaços, foi necessário realizar alguma investigação no sentido de obter uma solução que permitisse contornar essas limitações.

Figura 2 - Autómato Sauter Ecos 500

No que diz respeito à solução desenvolvida durante este estágio para o projeto *moduWeb Vision*, esta é composta essencialmente por dois componentes: o *back-end* e o *front-end,* como já foi referido. Apesar destes componentes funcionarem de modo integrado, cada um possui uma finalidade específica dentro da solução desenvolvida, nomeadamente:

- *Back-end:* Atua como componente central da plataforma. Fornece toda a funcionalidade de comunicação com os autómatos e todos os objetos associados a estes, permitindo consultar estados e efetuar a leitura e escrita de valores, fornecendo diversos serviços e controladores essenciais à comunicação com o *front-end*. Neste caso, este componente suportou toda a parte de leitura e escrita de valores relativos aos segmentos, representados nos autómatos por um determinado tipo de objetos com diversas propriedades.
- *Front-end:* Fornece a interface com o cliente final da solução. Permite ao utilizador toda a interação com a solução desenvolvida, através da visualização das diversas configurações associadas aos segmentos, quer sobre a forma de listas, quer sobre uma forma gráfica. Permite também ao utilizador a configuração destes segmentos, através da execução interativa de diversas operações disponíveis inerentes ao desenvolvimento da solução.

Até à data de término do estágio curricular, a equipa do projeto *moduWeb Vision* era constituída por um *Project Manager*, um *Product Owner*, um *Scrum Master* e dois *Developers*. Inserido nesta equipa, desempenhei o papel de *Developer* e *Tester*. O meu estágio curricular foi realizado nas instalações da Critical Software, em Taveiro, e teve a duração de 11 meses e meio (5 meses a tempo parcial e 6 meses e meio a tempo inteiro), tendo início em 15 de Outubro de 2015 e fim em 30 de Setembro de 2016, tal como se encontra na proposta disponibilizada no anexo M. Após o término das tarefas que me foram delegadas, foi elaborado o presente relatório, cuja atividade teve lugar sensivelmente nos últimos dois meses do período do estágio.

Nesta introdução será ainda feita uma breve apresentação da entidade de acolhimento (a iTGROW). Uma vez que o projeto no qual me integrei é respeitante à empresa Critical Software, será também dada uma breve introdução a esta empresa. Serão ainda expostos os objetivos definidos para o meu estágio, bem como a respetiva calendarização. Por fim, será descrita a estrutura deste relatório.

1.1 Entidade de acolhimento

A iTGROW, cujo logotipo é apresentado na Figura 3, é um Agrupamento Complementar de Empresas (ACE) participado equitativamente pela empresa Critical Software e pelo Banco Português de Investimento (BPI). Está sediada nas instalações da Critical Software em Taveiro e tem como objetivo a qualificação informática de jovens licenciados, preparando-os para a vida profissional, mediante um programa de formação e treino de competências *on-the-job*. A iTGROW foi fundada em 2010 e conta atualmente com mais de 150 colaboradores.

Figura 3 - Logotipo da empresa iTGROW

A Critical Software, cujo logótipo é apresentado na Figura 4, é uma empresa de desenvolvimento de Software fundada em 1998, sediada em Taveiro (Coimbra). Possui subsidiárias no Porto, Lisboa, Estados Unidos da América, Brasil, Angola, Reino Unido, Moçambique e Singapura. Desenvolve soluções de *software* para mercados como o Espaço, Aeronáutica, Defesa, Banca, Telecomunicações, Energia, entre outros. A Critical Software destaca-se a nível mundial pela qualidade do *software* que desenvolve, a qual é garantida pelo seu *Quality Management System*. Atualmente, esta empresa conta com mais de 370 colaboradores distribuídos pelas suas diversas subsidiárias.

Figura 4 - Logotipo da empresa Critical Software

1.2 Objetivos e calendarização do estágio

A integração na equipa de desenvolvimento da Critical Software teve como objetivo a especificação, implementação e validação das funcionalidades do sistema.

Durante a realização do projeto fui integrado numa equipa de desenvolvimento em metodologia AGILE – SCRUM, sendo capaz de fazer a integração entre as soluções que desenvolvi e as aplicações existentes atualmente na Critical Software.

A calendarização do estágio pode ser consultada na proposta de estágio, disponível no anexo M.

1.3 Estrutura do relatório

Após a presente introdução, o capítulo 2 descreve a metodologia de desenvolvimento de *software* utilizada no projeto *moduWeb Vision*, a metodologia adotada para o meu estágio e as ferramentas de desenvolvimento utilizadas no projeto. O capítulo 3 apresenta a arquitetura da solução desenvolvida, descrevendo inicialmente uma visão geral do projeto *moduWeb Vision* e posteriormente uma visão mais detalhada da arquitetura. O capítulo 4 apresenta os requisitos definidos para a solução a desenvolver, expressos sob a forma de *User Stories.* No capítulo 5 são descritos os vários detalhes que estiveram envolvidos na implementação da solução. O capítulo 6 descreve o processo de testes aplicados à solução desenvolvida. Por fim, o capítulo 7 apresenta as conclusões finais retiradas da realização do meu estágio.

2 Metodologia

Nesta secção é descrita a metodologia de desenvolvimento de *software* adotada no desenvolvimento da solução de automação de espaços do projeto *moduWeb Vision*, bem como o respetivo *tailoring* aplicado, isto é, os ajustes feitos a esta metodologia de acordo com as características específicas do projeto. Será ainda apresentada a metodologia adotada para o estágio e as ferramentas utilizadas durante o desenvolvimento do projeto.

2.1 Metodologia de desenvolvimento

A metodologia de desenvolvimento adotada para o projeto *moduWeb Vision* foi o SCRUM. A razão para a escolha desta metodologia deveu-se principalmente ao facto de esta permitir entregas rápidas e iterativas ao cliente, um requisito exigido por este.

O SCRUM é uma metodologia de desenvolvimento de *software* ágil criada nos anos 1990 por Ken Schwaber e Alex Armstrong [2]. Esta metodologia é amplamente utilizada nos dias de hoje, inclusivamente por grandes empresas de desenvolvimento de *software*, como por exemplo a Google, a Microsoft, a Amazon, a Nokia e a IBM. Muitos especialistas definem o SCRUM não como uma metodologia, mas sim como uma *framework* ou conjunto de *guidelines* para a gestão de projetos de *software* e conceção de produtos de *software*.

Baseada nos princípios ágeis, a metodologia SCRUM apela à intercomunicação entre os membros da equipa por oposição à documentação excessiva. Nesta metodologia não existe um líder de equipa que define o que cada membro faz, mas sim uma equipa a funcionar como um todo e a tomar decisões em grupo. Para dar suporte a esta equipa, o SCRUM define dois cargos cruciais: O *Scrum Master* e o *Product Owner*. O *Scrum Master* pode ser visto como um facilitador que ajuda os membros da equipa a seguir o SCRUM da maneira mais eficaz possível. O *Product Owner* estabelece a ponte de ligação entre a equipa e o cliente, definindo o produto à equipa e ajudando esta a desenvolvê-lo de acordo com as espectativas e necessidades reais do cliente. Este é também responsável por manter o *Product Backlog* atualizado, o qual contém o conjunto total de *User*

Stories a serem implementadas durante o projeto. Uma *User Story* consiste numa pequena descrição de uma funcionalidade, descrita sob a perspetiva do utilizador.

O desenvolvimento de *software* em SCRUM é realizado de forma iterativa, onde a cada iteração é dado o nome de *Sprint* (ver Figura 5). As Sprints devem ter duração fixa e nunca devem ser superiores a um mês. A primeira atividade realizada em cada *Sprint* é a *Sprint Planning Meeting*. Durante esta reunião, o *Product Owner* é responsável por apresentar à equipa as *User Stories* do *Product Backlog* com maior prioridade e discutir com esta quais as que devem entrar no *Sprint Backlog*. O *Sprint Backlog* contém por sua vez a lista de *User Stories* a implementar durante a *Sprint* que se está a iniciar. Durante o decorrer da *Sprint*, o SCRUM define que todos os dias uma reunião de curta duração (15 minutos) deve ser realizada por todos os elementos da equipa (exceto pelo *Product Owner*), sendo gerida pelo *Scrum Master*. Esta reunião serve para sincronizar a equipa, onde cada elemento deve dizer o que realizou no dia anterior, o que vai realizar no dia que se segue e quais têm sido os principais impedimentos. No final da *Sprint* a última atividade a realizar é a *Sprint Review Meeting*. Esta reunião serve para demonstrar o trabalho realizado durante a *Sprint*, nomeadamente o incremento de funcionalidade adicionada ao produto. Esta reunião tem como principal objetivo recolher feedback do *Product Owner* e de outros *Stakeholders* que tenham sido convidados para a reunião. Este feedback pode por sua vez resultar na alteração de funcionalidades implementadas, sendo que estas alterações serão refletidas na adição ou remoção de *User Stories* no *Product Backlog*. Outra atividade também realizada no final de cada *Sprint* é a *Sprint Retrospective Meeting*. Nesta reunião devem participar todos os elementos da equipa, tendo como objetivo a reflexão sobre o decorrer da *Sprint* que acabou de terminar e a tentativa de encontrar oportunidades para melhorar o funcionamento da próxima *Sprint*.

Uma descrição mais pormenorizada do SCRUM pode ser encontrada no *website* oficial desta metodologia [2].

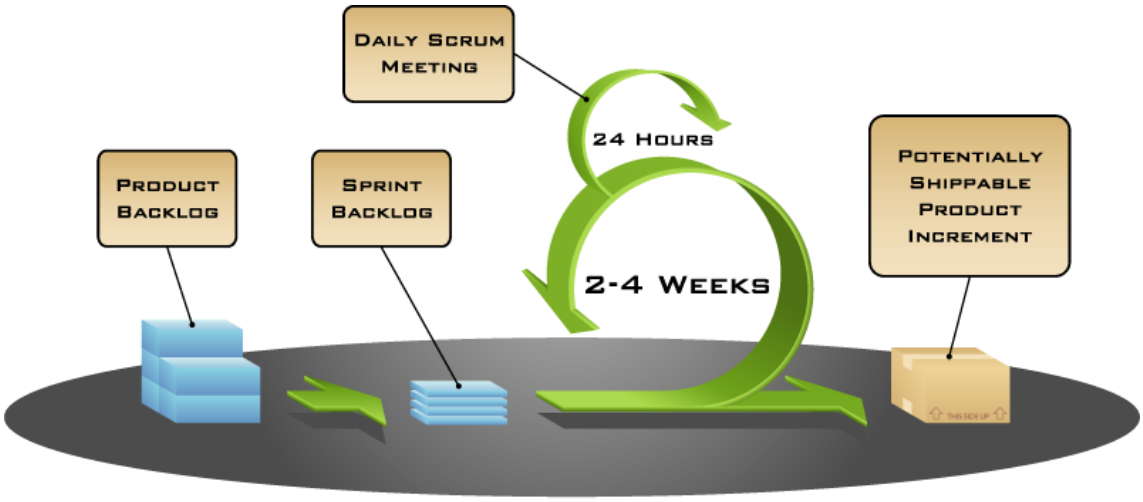


Figura 5 - Esquema de funcionamento do SCRUM (retirado de [3])

No que toca à adoção do SCRUM no desenvolvimento do projeto *moduWeb Vision*, para além das vantagens já referidas à adaptação desta metodologia ao projeto, foram ainda evidenciadas as seguintes vantagens:

- O cliente demonstrou desde o início grande disponibilidade em participar ativamente no desenvolvimento da solução, podendo assim avaliar continuamente o seu progresso.
- A participação constante do cliente no projeto possibilitou também uma validação contínua dos requisitos, bem como a deteção de defeitos nestes ainda antes de serem implementados.
- A sincronização diária entre os membros da equipa faz com que estes tenham conhecimento constante do estado do projeto e que haja uma transferência contínua de conhecimento.
- Dada a natureza versátil desta metodologia, é possível uma melhor minimização e mitigação dos riscos levantados para o projeto.

No que toca ao *tailoring* aplicado à metodologia SCRUM, para que esta se adeque melhor às características do projeto, foram definidos os seguintes pontos:

- A reunião *Daily Scrum Meeting* é realizada às 10h00 da manhã, numa sala predefinida, nas instalações da Critical Software em Taveiro.
- A reunião de *Sprint Planning Meeting* é realizada no primeiro dia de cada *Sprint*, sendo dividida em duas partes. Na primeira parte (realizada de manhã), o *Product Owner* faz a

apresentação à equipa das *User Stories* do *Product Backlog* que passarão para o *Sprint Backlog* da respetiva *Sprint*. Na segunda parte (realizada à tarde), a equipa efetua as estimativas das *User Stories* definidas no *Sprint Backlog*.

- A reunião de *Sprint Review Meeting* é realizada no último dia de cada *Sprint*, durante a parte da manhã. A reunião de *Sprint Retrospective Meeting* é realizada no mesmo dia, durante a tarde.
- Caso reste algum tempo em alguma das três reuniões mencionadas nos dois pontos anteriores, este deve ser aproveitado para a correção de defeitos (ou sugestões de melhoria).
- Cada *User Story* possui um critério de aceitação. Este critério de aceitação tem como objetivo tornar a *User Story* testável e será utilizado pelos *Testers* e pelo *Product Owner* para considerar a *User Story* como fechada ou não.
- Cada *User Story* deve ter sempre que possível um ou mais *mockups* (protótipos) associados. Estes *mockups* representam já os ecrãs finais da interface da solução e funcionam como complemento à descrição da *User Story*.
- Cada *User Story* deve ser dividida em sub-tarefas. As estimativas de esforço devem ser realizadas ao nível da sub-tarefa e não ao nível da *User Story*.
- Foi definida uma *Wiki* (repositório de informação) para o projeto onde cada membro da equipa deverá documentar, de forma sintética, o seu conhecimento adquirido ao longo do projeto (principalmente o *know-how* adquirido sobre as tecnologias utilizadas).
- No fim de cada *Sprint*, durante a *Sprint Review Meeting*, cada membro da equipa deve mencionar três pontos positivos e três pontos negativos acerca da *Sprint* que acabou de se realizar.

Toda a metodologia inicialmente definida para o projeto foi constantemente refinada ao longo do decorrer do projeto, tendo a maior parte destes refinamentos sido originados na sequência das *Sprint Retrospective Meetings*. Estes ajustes e melhorias possibilitaram que a equipa trabalhasse de um modo mais otimizado ao longo das *Sprints*, nunca comprometendo o sincronismo e transferência de conhecimento entre os elementos da equipa.

2.2 Metodologia do estágio

Uma vez que sou colaborador da iTGROW desde 2014, toda a minha contextualização e integração com o ambiente da Critical Software foi feita nas primeiras semanas em que incorporei a empresa. Durante este período recebi formação não só das práticas de desenvolvimento de *software* da Critical Software, mas também das ferramentas genéricas utilizadas na empresa, como o *WISE*, o *JIRA*, entre outras.

Uma vez que a metodologia adotada para o projeto foi o SCRUM, o meu estágio envolveu reuniões diárias, as designadas *Daily Scrum Meetings*. Estas reuniões foram bastante importantes, no sentido de contribuírem bastante para a minha coordenação com a restante equipa, não só durante a fase de desenvolvimento como na fase de validação da solução. Além destas reuniões inerentes ao projeto, tive algumas reuniões com o professor orientador por parte do ISEC, dando-lhe conhecimento do meu progresso e outro tipo de informações importantes ao longo do estágio.

2.3 Ferramentas utilizadas

Esta secção pretende apresentar as ferramentas utilizadas durante o desenvolvimento do projeto de estágio. As tabelas 1, 2, 3 e 4 descrevem as ferramentas utilizadas a nível de: desenvolvimento, controlo de versões, suporte e gestão do projeto, e uso interno, respetivamente. Cada uma das tabelas apresenta o nome da ferramenta em causa, uma descrição, a sua natureza (se são comerciais, *freeware*, *open source* ou de uso interno na Critical Software) e o logotipo correspondente.

Tabela 1 - Ferramentas de desenvolvimento utilizadas no projeto

Nome	Descrição	Natureza	Logotipo
Eclipse IDE	*Integrated Development Environment (IDE)* multi-linguagem desenvolvido em *Java*, mantido pela Eclipse Foundation. Utilizado por todos os *developers* do projeto *moduWeb Vision*.	*Open Source*	

Tabela 1 - Ferramentas de desenvolvimento utilizadas no projeto (continuação)

Nome	Descrição	Natureza	Logotipo
Java Enterprise Edition (JEE)	Pilha de tecnologias *Java* utilizada na construção da solução.	*Freeware*	
Gradle	Ferramenta utilizada na automação de *builds* e gestão de dependências que trabalha com *scripts* escritos na linguagem *Groovy*.	*Open Source*	
SAUTER Case Suite	Conjunto de programas de *software* da empresa SAUTER utilizados para lidar com projetos de automação de edifícios. No contexto deste estágio, foram principalmente utilizados na criação de *dynamic pictures* (abordadas no próximo capítulo).	Comercial	

Tabela 2 - Ferramentas de controlo de versões utilizadas no projeto

Nome	Descrição	Natureza	Logotipo
Git	Sistema de controlo de versões desenvolvido por Lunis Torvalds e utilizado no repositório do projeto.	*Open Source*	
SourceTree	Cliente *Git* e Mercurial para Windows e Mac que fornece uma interface gráfica que facilitou a gestão do repositório *Git* do projeto.	*Freeware*	
Subversion (SVN)	Ferramenta de controlo de versões desenvolvida pela Apache, utilizada no repositório da *wiki* do projeto.	*Open Source*	
Tortoise SVN	Cliente de *Subversion* para Windows, integrado com a interface do *Windows Explorer*, que facilitou a gestão do repositório da *wiki* do projeto.	*Open Source*	

Tabela 3 - Ferramentas de suporte e gestão utilizadas no projeto

Nome	Descrição	Natureza	Logotipo
JIRA	Ferramenta de *Issue Tracking*, *Bug Tracking* e gestão de projetos desenvolvida pela Atlassian e utilizada pela Critical Software para registar todos os *Issues* que iam surgindo durante o desenvolvimento.	Comercial	
Spira Team	Ferramenta de *Issue Tracking*, *Bug Tracking* e gestão de projetos desenvolvida pela Inflectra e utilizada pela Sauter para registo de *Issues* que iam surgindo ao longo do projeto.	Comercial	
Microsoft OneNote	Programa desenvolvido pela Microsoft utilizado para a *wiki* do projeto *moduWeb Vision*.	Comercial	

Tabela 4 - Ferramentas internas utilizadas no projeto

Nome	Descrição	Natureza	Logotipo
Wise	Ferramenta interna da Critical Software utilizada para a gestão de toda a sua informação (projetos, colaboradores, qualidade, tarefas, etc.). Foi utilizada durante o projeto para registar o esforço diário despendido.	Uso Interno	

3 Arquitetura

Este capítulo apresenta a arquitetura geral do projeto *moduWeb Vision*, tendo como foco principal a solução desenvolvida referente à automação de espaços. Será exposta em primeiro lugar uma visão geral do sistema de maneira a contextualizar o leitor e posteriormente uma visão mais detalhada referente aos módulos aplicáveis no contexto deste projeto.

3.1 Visão geral

Como se encontra referido no primeiro capítulo deste relatório, *o moduWeb Vision* é um produto que permite, através de um simples *browser*, efetuar a monitorização e controlo em tempo real de todos os sistemas de gestão de um edifício (sejam eles sistemas de aquecimento, ventilação, ar condicionado, iluminação, etc.), possibilitando ao utilizador tomar decisões rápidas e informadas, de maneira a otimizar o consumo energético e manter níveis de eficiência e conforto ideais nos edifícios. Nesse sentido, o *moduWeb Vision* surge como uma solução web otimizada para os *browsers* mais atuais e também para dispositivos *touch*.

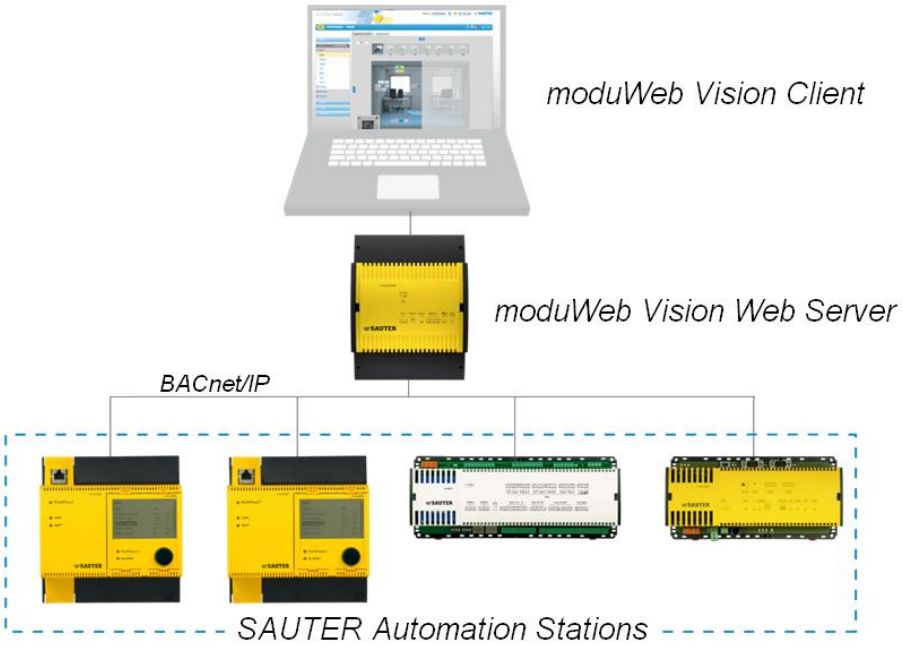


Figura 6 - Interligação do moduWeb Vision com outros componentes

Enquanto solução *web*, este produto segue um modelo cliente-servidor em que o servidor se encontra implementado num dispositivo compacto desenvolvido pela SAUTER [4], representado na Figura 6 com a designação *moduWeb Vision Web Server*. Este servidor tem a capacidade de comunicar com diversos autómatos de um edifício, através de protocolos *BACnet/IP* [5], projetados para permitir a comunicação entre este tipo de sistemas. Já no cliente, representado na Figura 6 com a designação *moduWeb Vision Client*, é possibilitada a visualização, monitorização e controlo desses sistemas através de qualquer *browser* compatível com *HTML5*, estando também otimizado para dispositivos *touch*.

O *moduWeb Vision*, enquanto produto possui diversas funcionalidades, entre as quais se destacam:

- Acesso controlado com *login* através de preenchimento de *username* e *password*, suportando diferentes tipos de utilizadores (administrador, especialista, convidado), em que cada tipo de utilizador possui acesso a diferentes níveis de informação.
- Visualização e operação de autómatos e dos diferentes tipos de objetos a eles associados (denominados por *datapoints*) e suas propriedades, sob a forma de listas estruturadas e diversos componentes visuais. Estes *datapoints* representam objetos que podem ser essencialmente de três tipos:
    - Analógicos: por exemplo um sensor de temperatura que pode apresentar uma vasta gama de valores;
    - Binários: por exemplo um interruptor de uma lâmpada que tem apenas dois estados: ligado/desligado;
    - Multi-estados: por exemplo estores elétricos que podem apresentar um número determinado de estados: abrir, fechar e parar.
- Monitorização dos autómatos associados, possuindo um sistema de geração de alarmes que é acionado quando se verifica alguma anomalia;
- Notificação de utilizadores via *SMS/Email*;
- Criação e visualização de gráficos que registam os valores de determinadas propriedades dos *datapoints* dos autómatos, em tempo real.
- Criação e visualização de agendas que permitem definir valores objetivo para um determinado *datapoint*, num determinado dia do ano/mês/semana.

No *moduWeb Vision*, os *datapoints* dos autómatos podem ser associados a componentes visuais dinâmicos que pretendem de alguma forma representar elementos geralmente presentes em sistemas AVAC (aquecimento, ventilação e ar condicionado). Nesse sentido, um componente visual pode ser algo tão simples como uma representação gráfica de um interruptor de uma lâmpada, um potenciómetro, um manómetro, um medidor de nível, entre outros. Esses componentes visuais são definidos em ficheiros XML (gerados por ferramentas internas do cliente SAUTER [6]) e exibidos no *moduWeb Vision* em páginas dinâmicas denominadas por *dynamic pictures*. A Figura 7 ilustra um exemplo real de uma *dynamic picture* (contornada a laranja) que representa a planta de uma sala de aulas de uma escola onde estão configurados diversos *datapoints* que permitem controlar e visualizar diversos componentes, nomeadamente: interruptores, ar condicionado, estores elétricos e temperatura exterior. Na mesma figura, podemos ver contornada a verde, uma estrutura hierárquica contendo diversos nós. Esta estrutura é denominada por *SVO Tree*, sendo que a sigla *SVO* significa *Structured View Object.* Na prática, esta estrutura permite que diferentes *dynamic pictures* possam ser organizadas de forma hierárquica, facilitando a diferenciação entre edifícios, andares, salas, entre outros.

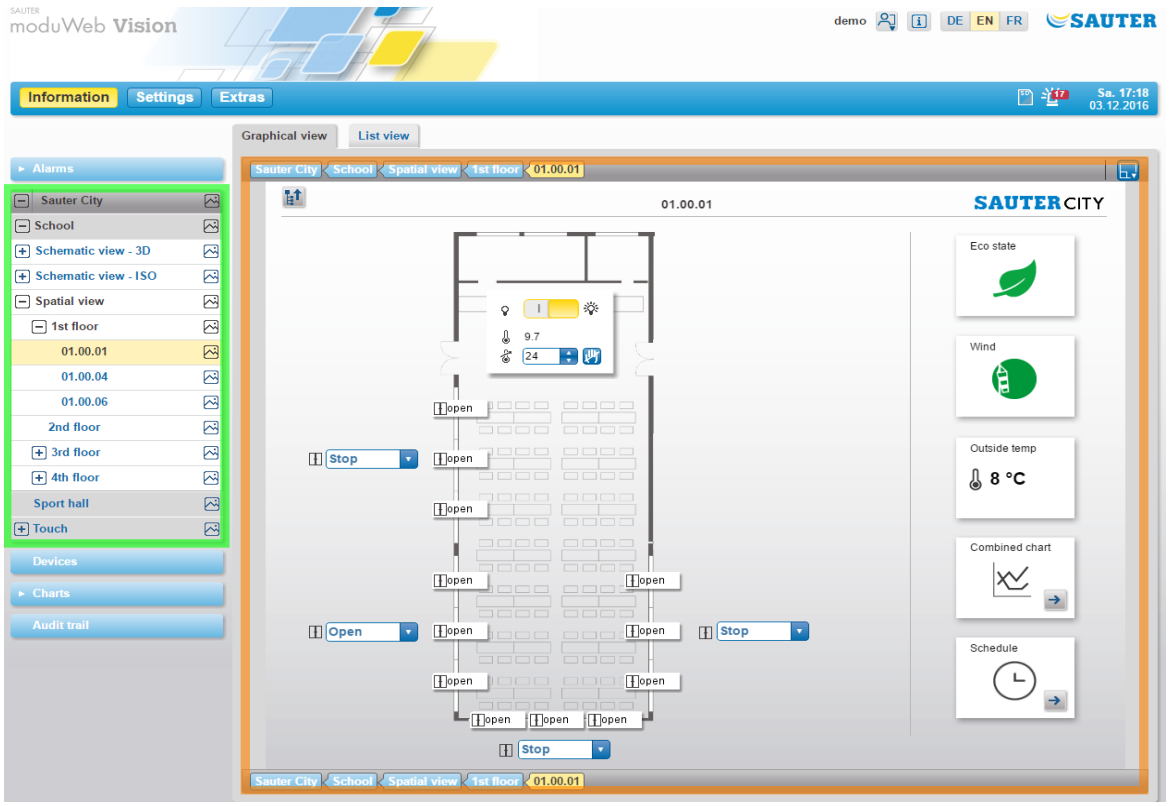


Figura 7 - Exemplo de uma dynamic picture no moduWebVision

Adicionalmente, os *datapoints* associados a uma determinada *dynamic picture* podem ser exibidos sob a forma de listas, representadas por tabelas, como pode ser observado na Figura 8, que podem possibilitar a edição de propriedades desses *datapoints.*

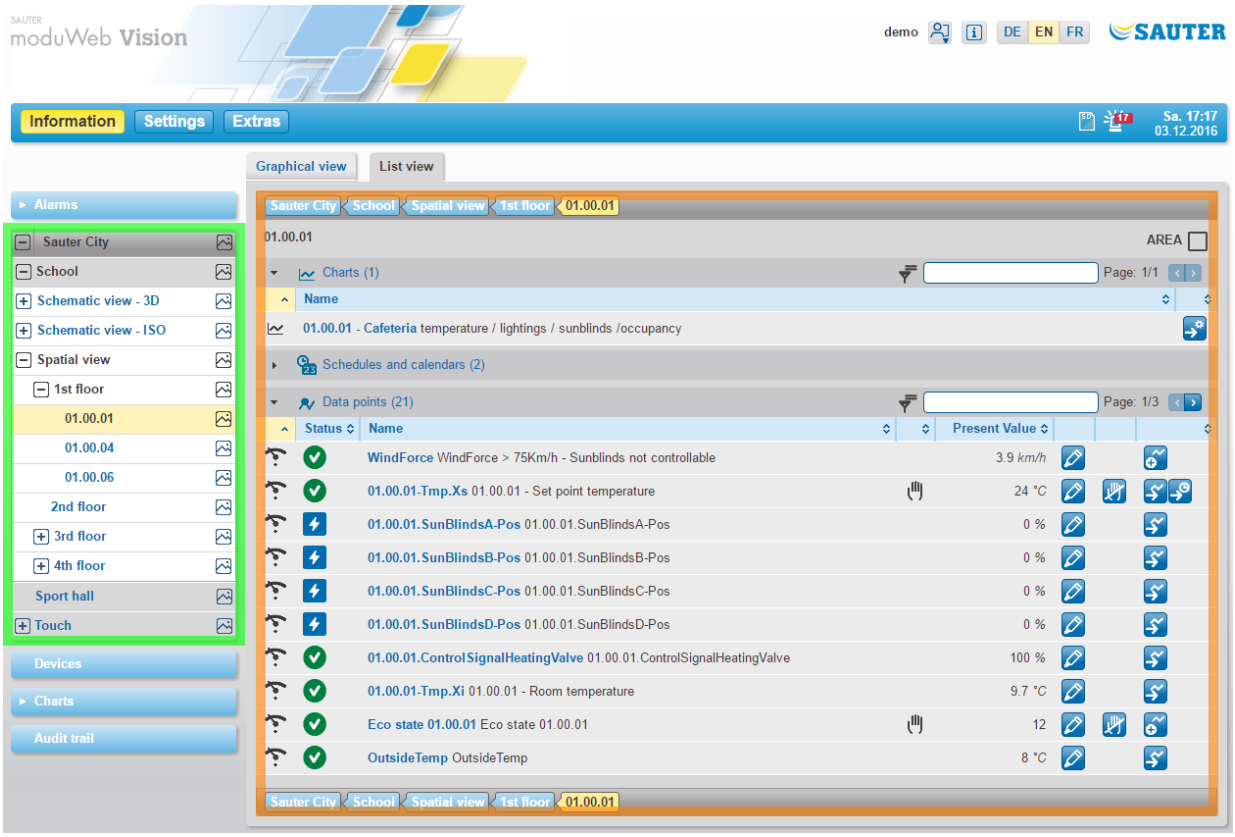


Figura 8 - Exemplo de uma lista de datapoints no moduWebVision

O projeto desenvolvido enquadra-se nesta funcionalidade previamente descrita relativa à "Visualização e operação de autómatos e dos diferentes tipos de objetos a eles associados (denominados por *datapoints*) e suas propriedades". Neste caso específico, o que está em causa é a visualização e operação de objetos de um autómato concebido recentemente pela SAUTER para lidar com funcionalidades específicas de automação de espaços, designadas internamente por *Room Management* [7]. De uma forma geral, este autómato disponibiliza objetos que pretendem representar os segmentos/divisões de um edifício. Esses objetos são designados internamente por *Segments* (segmentos) e possuem diversas propriedades que os permitem associar-se entre si por forma a criar grupos, designados por *Room Groups*. O facto de estes segmentos poderem ser associados entre si pode trazer inúmeras vantagens a nível da automação de espaços. Ao ter disponível um mecanismo que permita associar as configurações (seja de ar-condicionado, estores,

luminosidade, entre outros) a estas divisões e simultaneamente permita configurar diferentes agrupamentos entre essas divisões, passa a ser possível criar configurações comuns a várias divisões e deixa de ser necessário reprogramar todas as configurações de cada vez que uma divisão é alterada, o que traz bastantes benefícios.

A visualização e operação deste tipo de objetos (*Segments*) foi implementada no decorrer deste projeto e ocorre essencialmente de duas formas:

- Através da representação visual dos segmentos nas *dynamic pictures* (como se encontra ilustrado na Figura 9), sob a forma de polígonos (que representam o espaço físico associado a uma divisão) que possuem diferentes cores e estão posicionados de forma específica (adicionalmente podem ser adicionados outros componentes já existentes, como por exemplo imagens de plantas de uma divisão, de maneira a facilitar a visualização dos segmentos sobre as divisões dessa planta).

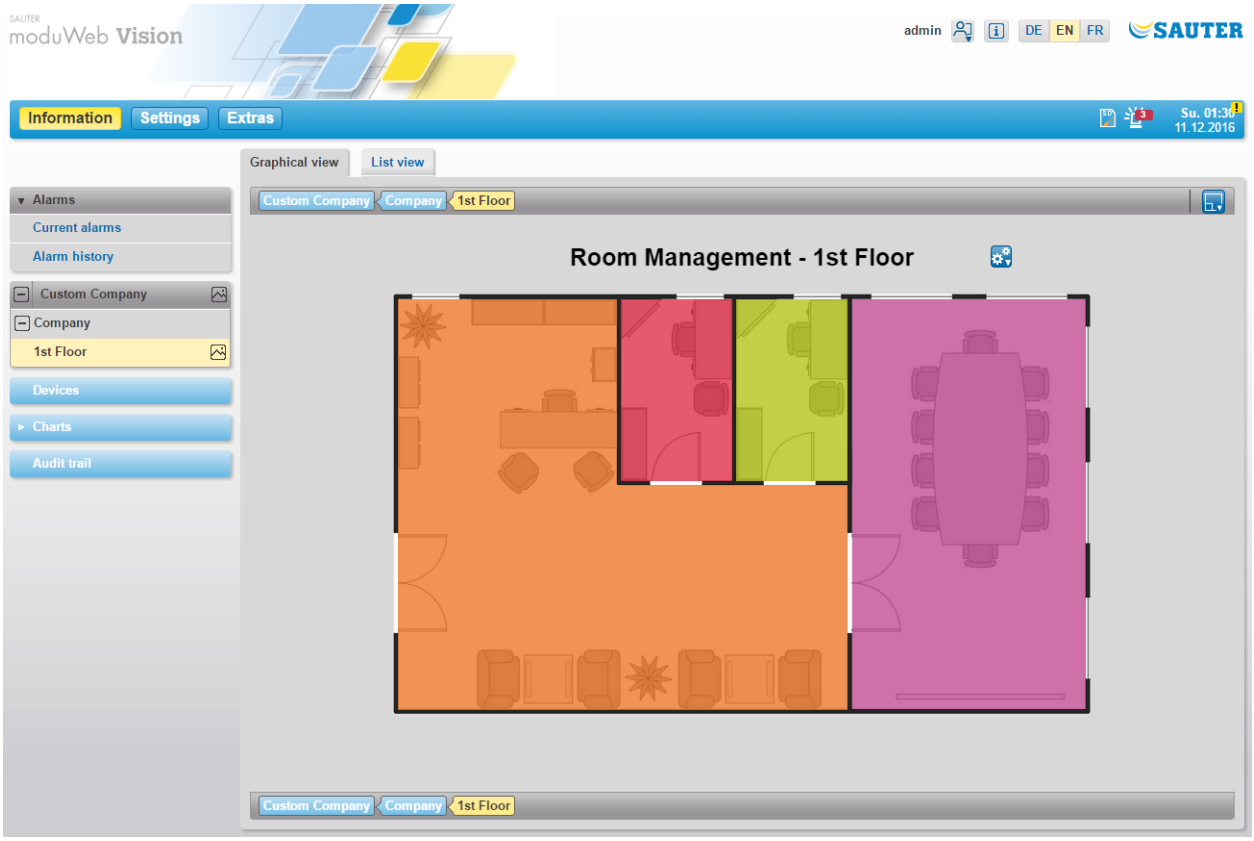


Figura 9 - Representação visual de segmentos numa dynamic picture

- Através da implementação de um mecanismo que permite proceder à configuração dos diversos segmentos de um edifício, isto é, um configurador que permite criar combinações/associações entre diferentes segmentos de uma *dynamic picture* (como se encontra ilustrado na Figura 10), propagando depois essas configurações de volta para os autómatos.

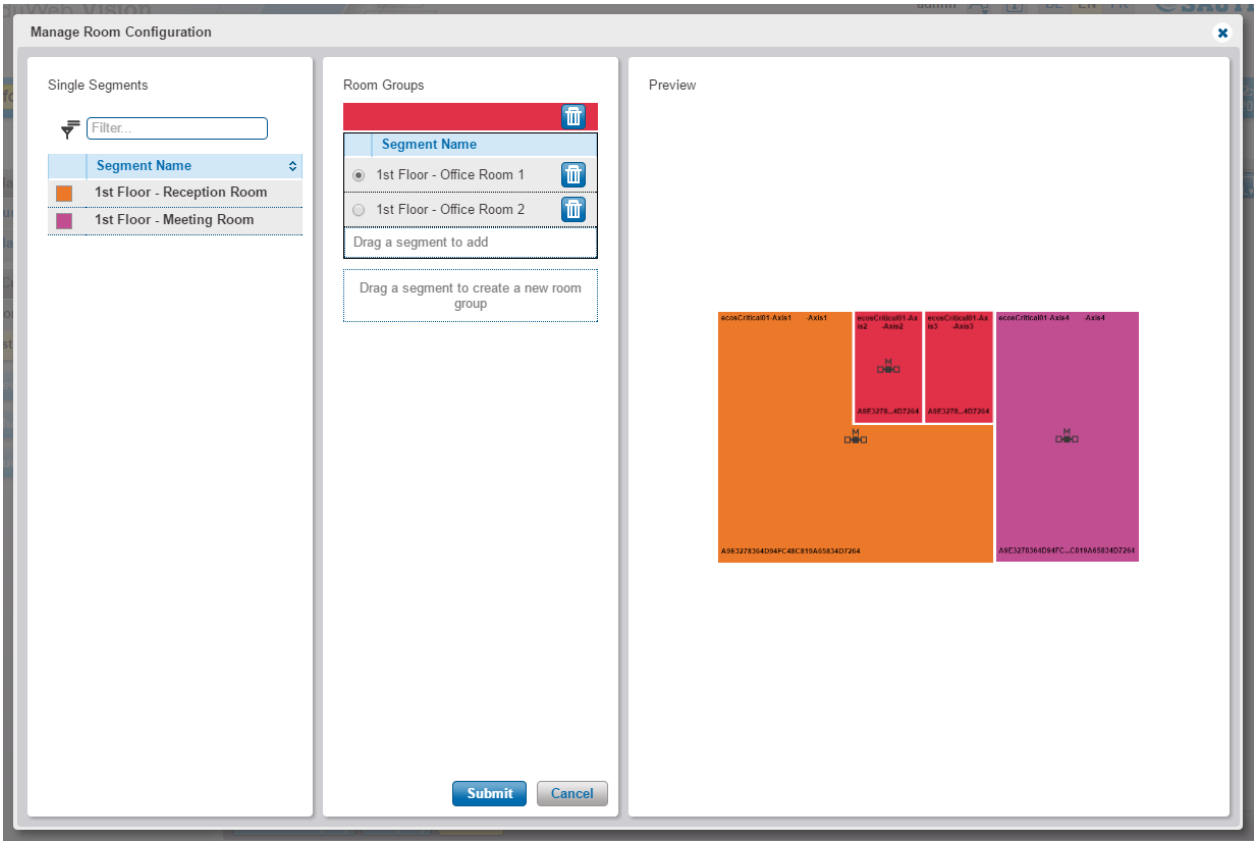


Figura 10 - Representação visual do configurador de segmentos

Como foi referido no início deste capítulo, toda esta descrição geral do projeto *moduWeb Vision* e da solução desenvolvida tem como objetivo contextualizar o leitor antes de entrar em detalhes relativamente à arquitetura propriamente dita.

Seguidamente será então apresentada de uma forma mais detalhada, a arquitetura por detrás da solução desenvolvida no âmbito deste projeto.

3.2 Visão detalhada

Como já foi referido neste capítulo, o projeto *moduWeb Vision* segue uma arquitetura cliente-servidor. As diversas funcionalidades disponibilizadas pelo *moduWeb Vision* são implementadas no servidor de uma forma modular. Nesse sentido, existem diferentes módulos específicos para cada tipo de funcionalidade.

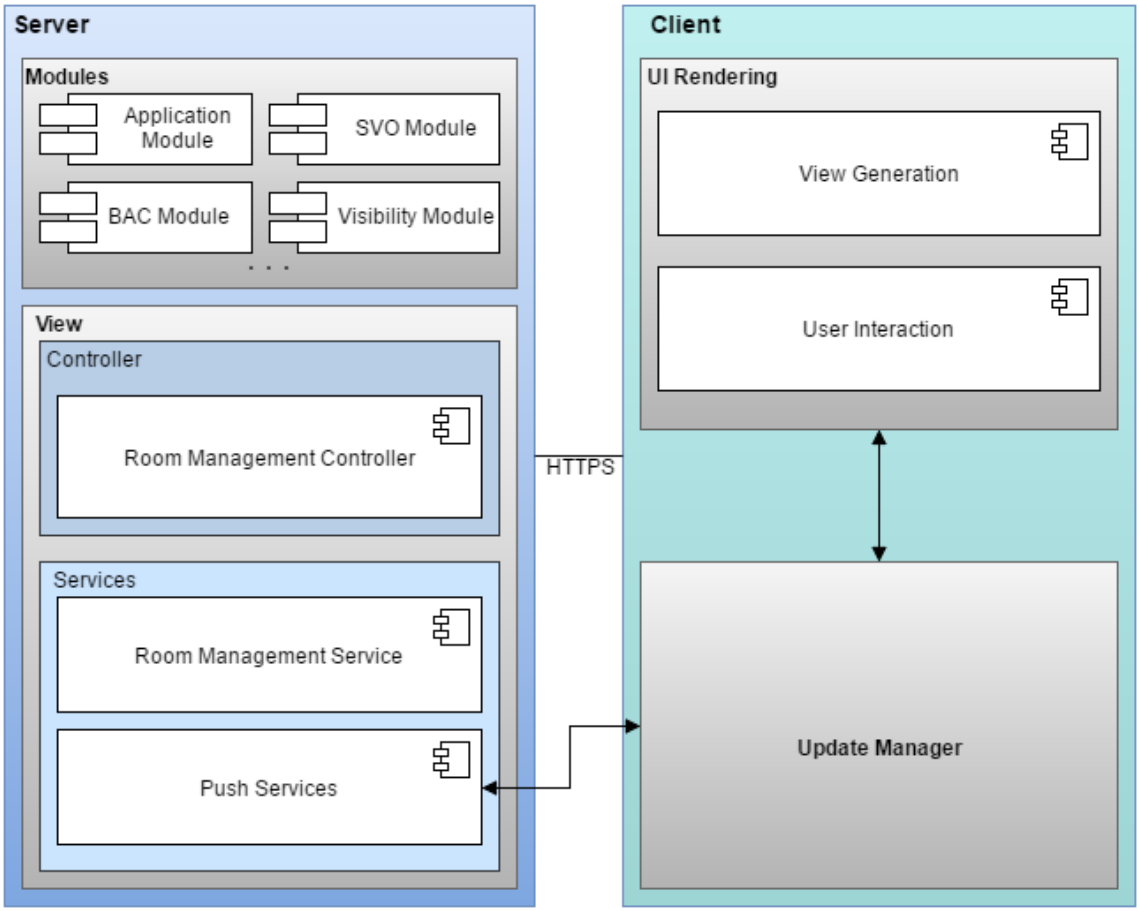


Figura 11 - Diagrama de componentes da solução desenvolvida

3.2.1 Principais módulos

A manipulação e operação dos objetos dos autómatos especializados na automação de espaços, solução desenvolvida neste estágio, está diretamente associada a quatro módulos existentes no servidor *moduWeb Vision*, nomeadamente: *Application, SVO, BAC e Visibility,* representados na Figura 11.

O módulo *Application* é um módulo onde se encontram implementados todos os componentes que estão diretamente relacionados com a aplicação *web*, como por exemplo *Servlets*, *Beans*, controladores, páginas *JSP*, ficheiros *CSS*, ficheiros *Javascript*, entre outros.

O módulo *BAC (build automation control)* é responsável por toda a comunicação do servidor *moduWeb Vision* com os autómatos a ele associados, permitindo entre outras coisas, efetuar operações de leitura e escrita sobre objetos do tipo *BACObject* (objetos que representam os *datapoints* dos autómatos no servidor).

O módulo *SVO (structured view object)* é essencialmente responsável pelo carregamento da *SVOTree* (já referida anteriormente neste capítulo) com as diversas *dynamic pictures* a partir de um ficheiro XML designado por *svo.xml*.

O módulo *Visibility* é um módulo que, entre outras coisas, dá acesso a uma classe denominada *VisibilityData* que contém direitos de visualização específicos de cada utilizador em relação à *SVOTree*, ou seja, determinados tipos de utilizadores podem ter acessos condicionados a certos nós da *SVOTree*.

3.2.2 Controladores e serviços *room management*

Na Figura 11 pode também observar-se um controlador e um serviço designados por *Room Management Controller* e *Room Management Service*, respetivamente. Estes dois componentes foram inteiramente implementados no âmbito deste projeto. O controlador tem como função tratar dos pedidos por parte do cliente (*browser*) relacionados com a configuração dos segmentos de uma determinada *dynamic picture*, nomeadamente pedidos para obtenção dos dados necessários relacionados com a configuração dos segmentos (aquando da abertura do configurador), e pedidos para submeter uma nova configuração dos segmentos (aquando da submissão da configuração). Já o serviço implementado tem como função principal auxiliar o controlador na obtenção de propriedades específicas relacionadas com a configuração dos segmentos.

A Figura 12 apresenta o diagrama de sequência que ilustra o processo de obtenção dos dados de uma configuração de segmentos, que ocorre quando o utilizador abre o configurador.

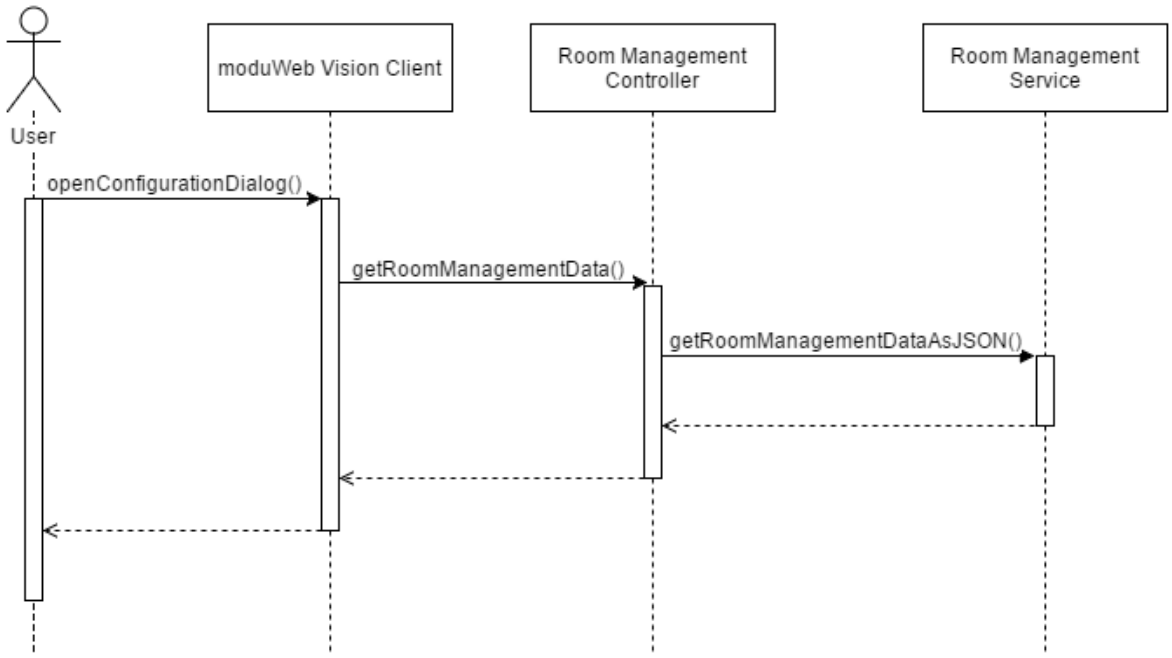


Figura 12 - Diagrama de sequência da obtenção de dados de uma configuração de segmentos

A Figura 13 apresenta o diagrama de sequência que ilustra o processo de submissão dos dados de uma nova configuração de segmentos.

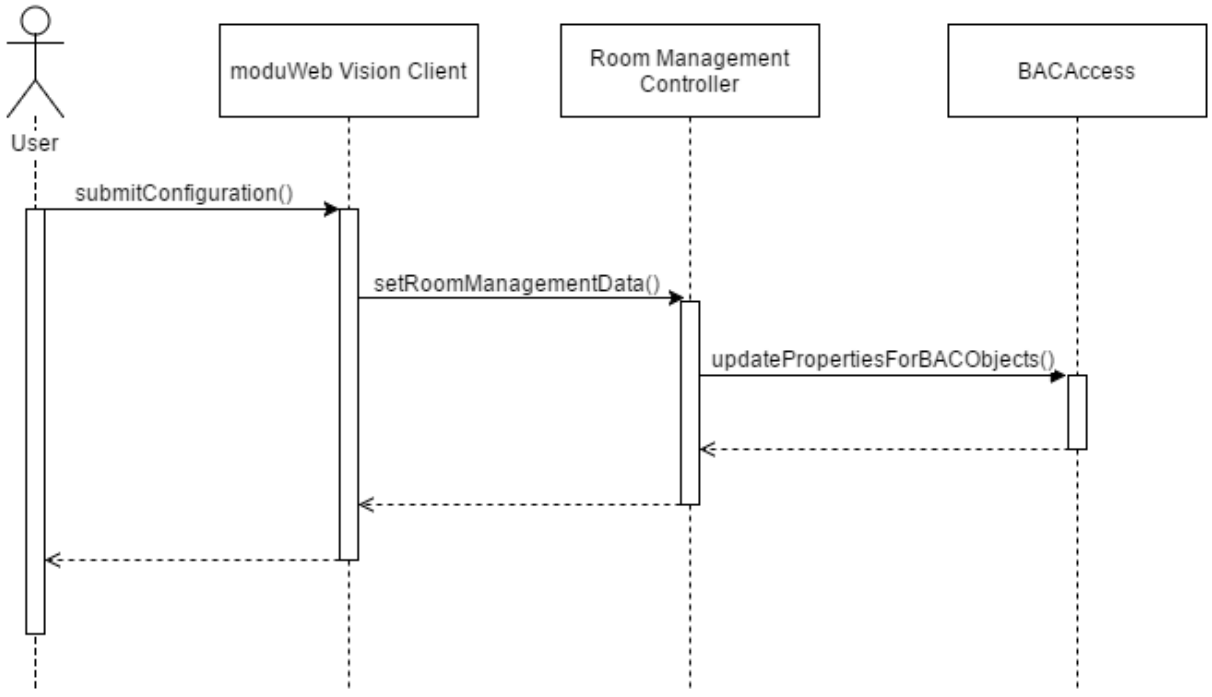


Figura 13 - Diagrama de sequência da submissão de dados de uma configuração de segmentos

3.2.3 Renderização do *layout*

As *dynamic pictures* exibidas através do *moduWeb Vision* comportam uma série de componentes visuais que necessitam de ser renderizados nas páginas web. Nesse sentido, o *HTML5* fornece diversas possibilidades para a visualização de conteúdo. Tendo em conta essas possibilidades, as *dynamic pictures* do *moduWeb Vision* baseiam-se essencialmente nos seguintes elementos:

- *Canvas*: Elemento *HTML5* que permite o desenho de múltiplas formas gráficas;
- *DOM*: Elementos simples do tipo *Document Object Module* que podem ser estilizados e posicionados via *CSS*.

Uma vez que o formato das *dynamic pictures* é baseado em definições *HTML/CSS*, a utilização de uma abordagem *DOM* permite poupar bastante tempo. Além disso, a manipulação de *inputs* (posicionamento do rato, clique, deteção de *focus*, etc.) está disponível nativamente para elementos *DOM*, o que constitui também uma vantagem. A dinamização da interface baseada em eventos requer um esforço minimizado de implementação, uma vez que o *browser* fornece a maioria dos eventos necessários.

Por outro lado, uma solução baseada em *Canvas* é ideal para conteúdos altamente dinâmicos como por exemplo videojogos, uma vez que a interface pode ser criada com base no *frame rate* do *browser* e não em eventos. Deste modo, elementos baseados em *Canvas* têm uma performance mais notória em relação a soluções baseadas no *DOM*, se os elementos forem complexos e excessivamente estilizados. Contudo, existem algumas desvantagens associadas. A manipulação de eventos e o *input* de texto tornam-se bastante complexos numa solução puramente baseada em *Canvas*. Além disso, a manutenção de uma *dynamic picture* inteiramente baseada em *Canvas* não seria trivial.

Tendo em conta todos os aspetos mencionados em cima, verifica-se que tanto a abordagem *DOM* como a abordagem *Canvas* possui vantagens e desvantagens. Nesse sentido, de maneira a atingir um resultado ideal, as *dynamic pictures* do *moduWeb Vision* são implementadas utilizando uma combinação destas duas abordagens, tirando partido das vantagens de cada uma. Deste modo, pode afirmar-se que a geração de componentes visuais e a interação com o utilizador a nível das *dynamic pictures* (representadas a Figura 11 pelos componentes *View Generation* e *User Interaction*, respetivamente) são baseadas na combinação dessas duas abordagens.

Para manipulação do *DOM* relativo às *dynamic pictures*, são utilizadas as *frameworks jQuery e jQuery UI*. A *framework jQuery UI* fornece uma *Widget Factory* que permite criar componentes gráficos de uma forma simples e robusta baseada num modelo de herança. Para evitar problemas de baixa performance, existem componentes individuais que são criados utilizando elementos *Canvas*, construídos e mantidos pela respetiva *widget*.

A Figura 14 ilustra de uma forma simplificada o modelo de herança dos componentes da *dynamic picture* do projeto *moduWeb Vision.*

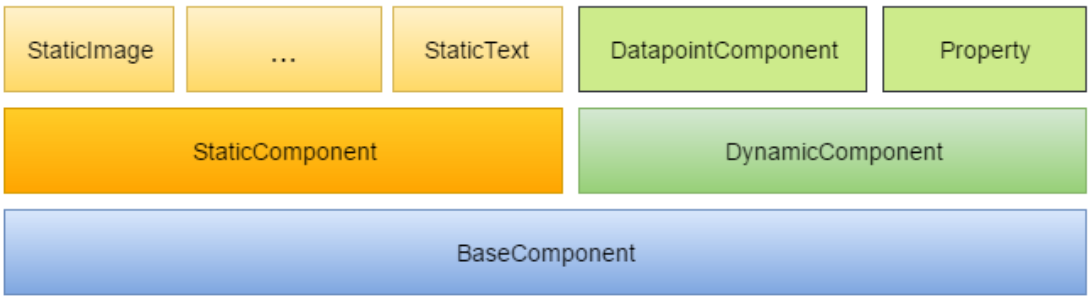


Figura 14 - Modelo de herança dos componentes no moduWeb Vision

3.2.4 Atualização dinâmica de propriedades dos componentes

Como já foi referido neste capítulo, uma *dynamic picture* consiste numa série de componentes visuais que pretendem de alguma forma representar elementos que estão geralmente associados a sistemas AVAC (aquecimento, ventilação e ar condicionado).

Estes componentes podem ser divididos em duas categorias principais: componentes dinâmicos e componentes estáticos (como pode ser observado na Figura 14). Cada componente renderizado no *front-end* necessita de um conjunto de dados, que podem ser subdivididos nas seguintes categorias: dados de configuração, parâmetros iniciais e dados *BACnet* dinâmicos. Os componentes estáticos são componentes cujos dados se mantêm inalterados entre cada pedido ao servidor (podem representar algo tão simples como um texto ou uma imagem). Os componentes dinâmicos são componentes que possuem dados *BACnet* dinâmicos, isto é, dados que representam propriedades de *datapoints* de um autómato cujo valor pode ser alterado a qualquer momento (por exemplo o valor de um sensor de luminosidade). Por forma a exibir os valores destes componentes dinâmicos sempre atualizados nas *dynamic pictures*, recorre-se à propagação de um padrão *Observer* por todo o sistema. Esta abordagem permite que as alterações nos autómatos sejam propagadas para a

interface do utilizador de uma forma rápida. Para atingir esta finalidade, o *moduWeb Vision* recorre a um servidor denominado *CometD* [8] que utiliza uma tecnologia *push* [9] (ilustrado na Figura 11 pelo componente *Push Services*), cuja principal característica é a utilização de uma ligação persistente *HTTP* por forma a permitir transmitir dados para o cliente sem que exista um pedido explícito por parte deste. Nesse sentido, pode dizer-se que o *CometD* permite o *pushing* de dados em vários canais através de *long-polling.* Através de *long-polling*, o cliente inquire o servidor, requisitando novos dados. O servidor mantém o pedido em aberto até que novos dados estejam disponíveis. Uma vez disponíveis, o servidor responde enviando esses dados. Quando o cliente recebe os dados, efetua automaticamente outro pedido, e a operação é repetida.

Cada cliente pode registar-se em um ou mais canais, recebendo todas as atualizações nele inseridas pelo servidor, tal como ilustra a Figura 15. Isto permite executar um *pushing* centralizado de dados *BACnet* (dados que representam propriedades de *datapoints* de um autómato) do servidor *moduWeb Vision* para todos os clientes que estejam num determinado momento a visualizar uma *dynamic picture.*

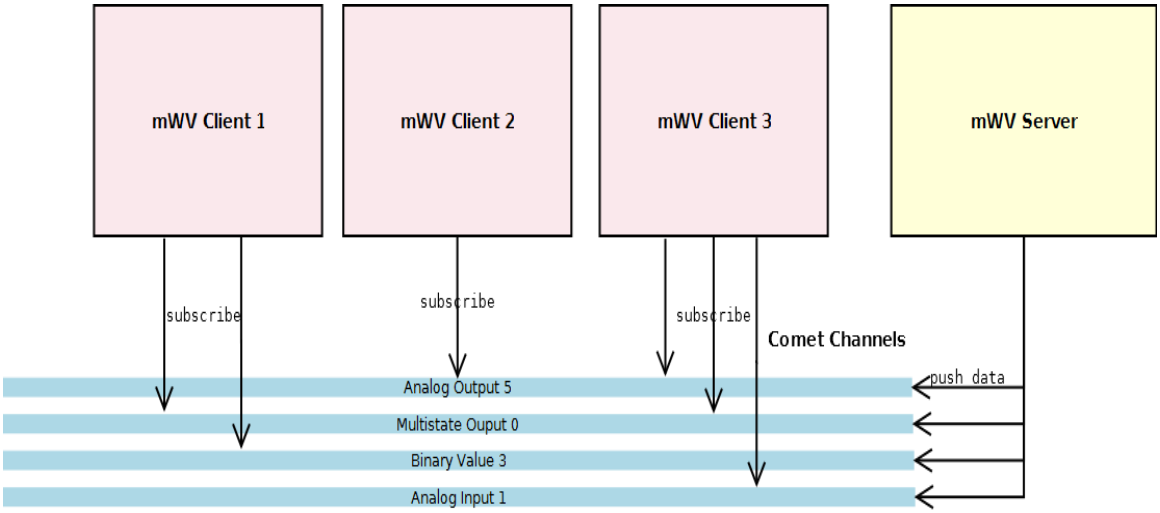


Figura 15 - Subscrição de canais CometD

Cada componente dinâmico exibe uma propriedade principal (geralmente essa propriedade é designada por *Present Value* e representa o valor atual do *datapoint*), podendo ser exibidas opcionalmente outras propriedades referentes ao mesmo *datapoint*. Consequentemente é criado um canal *CometD* para cada objeto *BACnet* visualizado.

A arquitetura geral da atualização dinâmica de dados *BACnet* através do *CometD* pode ser dividida em três fases: fase de registo, fase de observação e fase de cancelamento. Os procedimentos que acontecem durante estas fases encontram-se ilustrados na Figura 16 e vão ser descritos seguidamente.

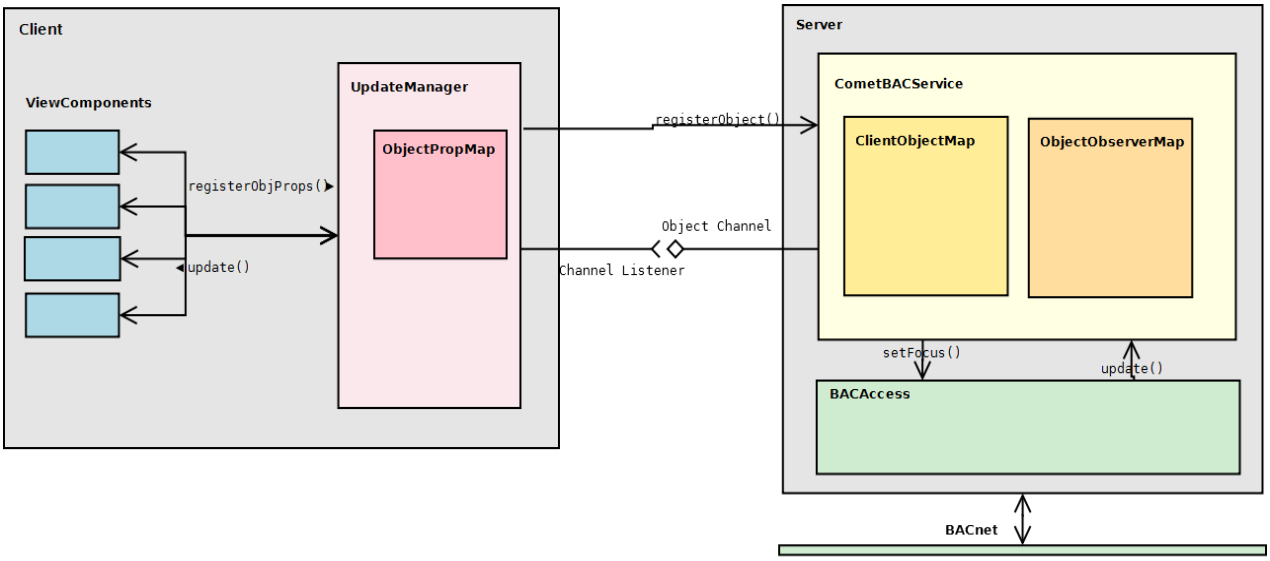


Figura 16 - Prodecimentos entre cliente e servidor associados ao CometD

3.2.4.1 Fase de registo

A fase de registo é a primeira fase pela qual cada cliente passa aquando da exibição de uma nova *dynamic picture* contendo componentes dinâmicos.

1. Cada componente dinâmico regista num *plugin jQuery* designado por *UpdateManager* (representado na Figura 11 pelo componente Update Manager) todas as propriedades do objeto do qual necessita de exibir os seus dados;
2. O *UpdateManager* agrupa todas as propriedades numa lista e mapeia-as nos componentes que estão interessados nos seus valores.
3. Depois de todos os componentes terem sido inicializados, o *UpdateManager* regista um *listener* para todos os canais dos objetos nos quais os componentes estão interessados.
4. Posteriormente, o *UpdateManager* conecta-se ao servidor *CometD* e regista-se para todas as propriedades requeridas.
5. O servidor recebe o pedido, guarda as propriedades num objeto *ClientObjectMap* e mapeia-as para o *ID* do cliente.

6. O servidor cria um objeto *ObjectObserverMap* para todos os objetos cujo cliente se registou.
7. Por último, o servidor converte todas as propriedades de cada objeto registado para propriedades conformadas com *BACnet* e regista-as no *BACAccess*, uma interface para acesso ao módulo de controlo dos autómatos.

3.2.4.2 Fase de observação

A fase de observação é a fase em que são esperados os envios de valores atualizados das propriedades registadas nos canais de observação.

1. O cliente está em modo de observação, à espera que sejam enviados valores para um dos canais de observação.
2. O cliente envia uma mensagem de *keep alive* periodicamente sinalizando que ainda está interessado nas propriedades registadas.
3. Um dos objetos *ObjectObserverMap* recebe uma atualização via *BACnet*.
4. O *ObjectObserverMap* analisa todas as propriedades atualizadas e verifica se existe algum cliente registado interessado nelas.
5. Se existir pelo menos um cliente interessado numa propriedade, o *observer* renderiza-a num determinado formato especificado pelo cliente.
6. O *ObjectObserverMap* envia depois a propriedade renderizada para o *ObjectChannel*.
7. No lado cliente, o *UpdateManager* recebe a mensagem de atualização no *ObjectChannel*.
8. O *UpdateManager* consulta a lista de todos os componentes interessados nas propriedades atualizadas.
9. Posteriormente, o *UpdateManager* chama o método de *callback* para todos os componentes interessados.
10. Os componentes recebem os *callbacks* e renderizam os componentes com os novos valores.

3.2.4.3 Fase de cancelamento

O cancelamento de uma conexão segue-se sempre à fase de observação e pode ser iniciado de forma ativa pelo cliente quando este muda de *dynamic picture* ou ao fechar o *browser*, ou de forma passiva, se o servidor deixar de receber a mensagem de *keep alive* por um período de tempo definido.

1. O servidor inicia o cancelamento da conexão de um cliente registado.
2. O servidor remove o *focus* do ID do cliente do *BACAccess*.
3. O *ClientObject* é removido de todos os *ObjectObserverMap*.
4. Se o *ClientObject* for o único interessado no *BACObject*, o *ObjectObserverMap* é removido.
5. O *ClientObject* é removido do *ClientObjectMap*.

No *front-end*, um *script jQuery* fornecido pelo *CometD* é utilizado para a interação direta com o servidor. Anexado ao *script CometD* está o *UpdateManager* do *moduWeb Vision*, que gere todas as propriedades dos objetos requisitadas pelos componentes, regista-as no servidor e distribui as atualizações recebidas dos canais de volta aos respetivos componentes.

Todos os componentes que exibem dados *BACnet* derivam de uma *widget jQuery UI* base denominada *DynamicComponent* implementada num ficheiro denominado *jquery-ui.component.dynamic.js*. Esta *widget* fornece métodos para registar propriedades para um *datapoint*. Uma vez registado, o *DynamicComponent* irá automaticamente chamar o método de *callback* no caso de ser desencadeada alguma atualização pelo *UpdateManager*. Deste modo, o *binding* completo de dados remotos é implementado apenas por esta *widget*. Como resultado, todas as restantes *widgets* implementadas que derivam desta não necessitam de se preocupar com a atualização das propriedades dos *datapoints* nos componentes visuais e podem concentrar-se no seu propósito principal: exibir determinado conteúdo e reagir ao *input* do utilizador.

Após a apresentação da arquitetura da solução desenvolvida, onde foram descritos vários aspetos relativos ao servidor, ao cliente e à interação entre estes, no próximo capítulo, referente aos requisitos, serão apresentados mais detalhes no que concerne à interação do utilizador com a solução desenvolvida.

4 Requisitos

Neste capítulo são apresentados os requisitos do projeto *moduWeb Vision* que dizem respeito ao módulo desenvolvido durante o estágio, relacionado com a automação de espaços. Dada a natureza da metodologia adotada para o projeto (SCRUM), estes requisitos foram expressos sob a forma de *User Stories* [2]. Coube ao *Product Owner* do projeto escrever e distribuir estas *User Stories* pelas *Sprints* do projeto, fazendo a devida priorização de acordo com as necessidades do cliente. A escrita destas *User Stories* foi elaborada a partir do seguinte *template*:

"Como <utilizador> pretendo <executar determinada ação> para que <possa atingir determinado objetivo>."

A utilização deste *template* permitiu a uniformização da escrita das *User Stories* e facilitou a sua interpretação por parte dos elementos da equipa. As três variáveis do *template* são respetivamente:

- *<utilizador>* – Menciona o utilizador que vai usufruir da funcionalidade, sendo referido o seu papel na utilização do sistema (role);
- *<executar determinada ação>* – Expressa a funcionalidade pretendida para o sistema;
- *<possa atingir determinado objetivo>* – Expressa o motivo que leva o utilizador a usufruir da funcionalidade, ou o valor de negócio que esta funcionalidade traz para o sistema.

As *User Stories* definidas são apresentadas nas tabelas 5, 6, 7, 8, 9, 10 e 11. Estas *User Stories* abrangem, de uma forma geral, as funcionalidades desenvolvidas neste projeto e podem ser divididas em duas funcionalidades principais:

- Visualização de componentes *room management* nas *dynamic pictures* dos nós SVO.
- Configuração dos segmentos presentes nas *dynamic pictures.*

Seguidamente serão apresentadas as *User Stories* definidas para cada uma destas funcionalidades em forma de tabelas que contêm: um identificador do número da *Sprint* em que a *User Story* foi desenvolvida, um identificador do número da *User Story* e a descrição da *User Story.*

4.1 Visualização de componentes *room management* nas *dynamic pictures*

A implementação de componentes visuais referentes aos segmentos e outros componentes *room management* nas *dynamic pictures* ocorreu nas primeiras três *Sprints* e incluiu:

- Implementação de componentes visuais que representam os segmentos nas *dynamic pictures* e de mecanismos de associação aos autómatos correspondentes.
- Implementação de componentes visuais respeitantes a um botão de configuração que permite a exibição/ocultação de segmentos numa *dynamic picture* e a exibição do configurador de segmentos.
- Implementação de um mecanismo que permite condicionar a exibição de outros componentes visuais nas *dynamic pictures* que não sejam segmentos, de acordo com propriedades intrínsecas aos segmentos (explicado com mais detalhe no próximo capítulo referente à implementação da solução).

A Tabela 5 apresenta então as *User Stories* definidas que se referem à implementação de componentes *room management*.

Tabela 5 - User Stories referentes à implementação de componentes Room Management

Sprint	***#***	***User Story***
1	1	Como utilizador, pretendo ter acesso a uma interface que me permita visualizar uma representação das divisões (denominadas por segmentos) que constituem um determinado espaço físico de um edifício, sob a forma de polígonos de dimensões configuráveis.
1	2	Como utilizador, pretendo que cada segmento tenha associado a si um objeto do tipo *group-axis*, pertencente aos autómatos ecos 5 desenvolvidos pela *Sauter*, particularmente concebidos para lidar com os requisitos da automação de espaços.

Tabela 5 - User Stories referentes à implementação de componentes Room Management (continuação)

Sprint	***#***	***User Story***
1	3	Como utilizador, pretendo que cada segmento representado na interface por um polígono, possua uma cor de fundo associada, diferente para cada um, de maneira a facilitar a diferenciação dos demais segmentos.
2	4	Como utilizador, pretendo que a interface possibilite a existência de um botão que permita despoletar a exibição e ocultação dos segmentos (em anexo B), para que possa visualizar com mais clareza eventuais elementos subjacentes como por exemplo plantas de divisões.
2	5	Como utilizador, pretendo que o resultado do clique no botão de exibição/ocultação de segmentos seja guardado entre pedidos para facilitar o controlo do utilizador.
2	6	Como utilizador, pretendo que a interface possibilite a existência de um botão (em anexo B) que permita despoletar a exibição de uma *dialog*, que terá como função a configuração dos segmentos presentes no nó SVO.
3	7	Como utilizador, pretendo que outros componentes da *dynamic picture* que não sejam segmentos e estejam presentes nessa *dynamic picture* possam ser exibidos segundo determinadas condições associadas a estes segmentos. Assim sendo, deve ser possível mostrar determinado componente apenas quando: • Um segmento é singular *(Single Segment)* e não pertence a nenhum grupo *(Room Group)*; • Determinados segmentos pertencem a um mesmo *grupo*; • Determinados segmentos pertencem exclusivamente a um mesmo grupo; • Determinados segmentos não pertencem a um mesmo grupo; • Um segmento pertence a um grupo e é o segmento principal desse grupo (*Master Segment*).

4.2 Configuração dos segmentos presentes nas *dynamic pictures*

A configuração dos segmentos foi possibilitada a partir da implementação de um configurador que permitiu a criação de combinações/associações entre diferentes segmentos de uma *dynamic picture.* A implementação desta funcionalidade ocorreu durante as últimas sete *Sprints*, podendo as *User Stories* relativas a esta funcionalidade ser divididas nas seguintes categorias:

- Organização gráfica de secções no configurador de segmentos;
- Funcionalidades da secção de *Single Segments*;
- Funcionalidades da secção de *Room Groups*;
- Funcionalidades da secção de *Preview;*
- Funcionalidades da secção *PlanId;*
- Funcionalidades gerais do configurador de segmentos.

Seguidamente serão então apresentadas as *User Stories* especificadas para cada uma das categorias acima mencionadas.

4.2.1 Organização gráfica de secções no configurador de segmentos

A Tabela 6 apresenta as *User Stories* referentes à organização gráfica de secções no configurador de segmentos.

Tabela 6 - User stories referentes à organização gráfica de secções no configurador

Sprint	***#***	***User Story***
4	8	Como utilizador, pretendo que a *dialog* para a configuração dos segmentos exiba uma tabela que permita visualizar os segmentos que não fazem parte de nenhum grupo (designados por *Single Segments*), indicando o seu nome e a respetiva cor (em anexo C).
4	9	Como utilizador, pretendo que a *dialog* para a configuração dos segmentos possua uma secção que permita a criação de grupos de segmentos (denominados por *Room Groups*) através de *drag & drop* a partir da tabela de *Single Segments* para a referida secção (em anexo D).

Tabela 6 - User stories referentes à organização gráfica de secções no configurador (continuação)

Sprint	***#***	***User Story***
4	10	Como utilizador, pretendo que a *dialog* de configuração dos segmentos exiba uma secção de *preview* (em anexo E), que permita mostrar o resultado temporário das alterações que o utilizador executa sobre os segmentos a nível gráfico.
4	11	Como utilizador, pretendo que, no caso de existirem segmentos no nó SVO que possuam diferentes *PlanId's* (propriedade específica destes objetos nos autómatos), seja exibido na *dialog* de configuração uma secção que permita a filtragem dos segmentos exibidos na *dialog* de configuração segundo um *PlanId* comum (em anexo F).
4	12	Como utilizador, pretendo que os *Room Groups* sejam representados na forma de tabelas em que o cabeçalho deve possuir como cor de fundo a cor do segmento principal do *Room Group* (*Master Segment*) e as entradas das tabelas representam o nome dos segmentos que constituem o *Room Group*, designados por *Slave Segments* (em anexo D).

4.2.2 Funcionalidades da secção de *Single Segments*

A Tabela 7 apresenta as *User Stories* referentes às funcionalidades da secção de *Single Segments*.

Tabela 7 - User Stories referentes às funcionalidades da secção de Single Segments

Sprint	***#***	***User Story***
5	13	Como utilizador, pretendo que a tabela que apresenta os S*ingle Segments* possibilite a filtragem pelo nome do segmento (em anexo C).
5	14	Como utilizador, pretendo que na *dialog* de configuração, a ação de *drag & drop* de um segmento não configurado para um *Room Group* novo ou já existente exiba uma mensagem de aviso (em anexo H).

4.2.3 Funcionalidades da secção de *Room Groups*

A Tabela 8 apresenta as *User Stories* referentes às funcionalidades da secção de *Room Groups*.

Tabela 8 - User stories referentes às funcionalidades da secção de Room Groups

Sprint	***#***	***User Story***
5	15	Como utilizador, pretendo ter a possibilidade de alterar o *Master Segment* de cada *Room Group* sendo que a cor do cabeçalho do *Room Group* deve ser atualizada de acordo com o segmento selecionado.
5	16	Como utilizador, pretendo ter a possibilidade de eliminar *Room Groups* e segmentos de *Room Groups*, sendo que estes segmentos devem voltar para a tabela de *Single Segments* após esta ação.
5	17	Como utilizador, pretendo que, caso elimine o último segmento de um *Room Group*, esse *Room Group* seja também eliminado, uma vez que ficará vazio.
6	18	Como utilizador, pretendo que, caso durante a configuração de segmentos, exista um *Room Group* contendo apenas um segmento, esse *Room Group* seja eliminado e o segmento nele contido volte para a tabela de *Single Segments*, quando voltar a abrir a *dialog* novamente.
6	19	Como utilizador, pretendo que, no caso de tentar persistir uma configuração em que existam *Room Groups* com apenas um segmento, seja apresentada uma mensagem que informe o utilizador de que o segmento em causa será movido para a lista de *Single Segments*, e que o *Room Group* irá ser consequentemente removido.
6	20	Como utilizador, pretendo que, aquando da eliminação de um *Master Segment* de um *Room Group*, o próximo segmento desse *Room Group* seja selecionado como *Master Segment* desse *Room Group*.
6	21	Como utilizador, pretendo que, ao mover um *Master Segment* para um *Room Group* já existente, este se torne automaticamente um *Slave Segment* desse *Room Group*.

Tabela 8 - User stories referentes às funcionalidades da secção de Room Groups (continuação)

Sprint	***#***	***User Story***
6	22	Como utilizador, pretendo ter a possibilidade de mover segmentos entre diferentes *Room Groups* existentes, através de *drag & drop.*
7	23	Como utilizador, pretendo ter a possibilidade de mover segmentos de *Room Groups* já existentes para novos *Room Groups*, através de *drag & drop*.
7	24	Como utilizador, pretendo ter a possibilidade de reorganizar os *Room Groups* verticalmente na *dialog* de configuração, através de *drag & drop.* Após recarregar a *dialog* de configuração, a ordem mostrada deverá ser a original.
7	25	Como utilizador, pretendo que, ao mover todos os segmentos de um determinado *Room Group* para outro existente ou para um novo *Room Group*, o *Room Group* de origem seja eliminado.
7	26	Como utilizador, pretendo que na *dialog* de configuração, a remoção de um segmento que não esteja presente no nó SVO exiba uma mensagem de aviso ao utilizador (em anexo G).

4.2.4 Funcionalidades da secção de *Preview*

A Tabela 9 apresenta as *User Stories* referentes às funcionalidades da secção de *Preview*.

Tabela 9 - User Stories referentes às funcionalidades da secção de Preview

Sprint	***#***	***User Story***
7	27	Como utilizador, pretendo que a secção de pré-visualização da *dialog* de configuração seja atualizada a cada ação executada pelo utilizador.

4.2.5 Funcionalidades da secção *PlanId*

A Tabela 10 apresenta as *User Stories* referentes às funcionalidades da secção *PlanId*.

Tabela 10 - User Stories referentes às funcionalidades da secção PlanId

Sprint	***#***	***User Story***
8	28	Como utilizador, pretendo que, caso todos os segmentos da planta visualizada possuam o mesmo *PlanId*, não seja mostrada a secção que permite a filtragem deste parâmetro na *dialog* de configuração.
8	29	Como utilizador, pretendo que, no caso de existirem segmentos de outro *PlanId* configurados no *layout*, seja mostrada uma mensagem de aviso antes da exibição *dialog* de configuração (em anexo I).
8	30	Como utilizador, pretendo que, no caso de existirem segmentos de outro *PlanId* configurados no *layout*, estes sejam mostrados na secção de pré-visualização com uma cor cinza.
8	31	Como utilizador, pretendo que, no caso de existirem segmentos de outro *PlanId* configurados no *layout*, estes sejam removidos dos *Room Groups*, sendo que no caso de os segmentos "perderem" o seu *Master Segment*, devem tornar-se *Single Segments*.

4.2.6 Funcionalidades gerais

A Tabela 11 apresenta as *User Stories* referentes às funcionalidades gerais do configurador.

Tabela 11 - User stories referentes às funcionalidades gerais do configurador

Sprint	***#***	***User Story***
8	32	Como utilizador, pretendo que todas as alterações efetuadas na *dialog* de configuração sejam guardadas em cache e não sejam propagadas para os autómatos até o utilizador submeter a configuração.

Tabela 11 - User stories referentes às funcionalidades gerais do configurador (continuação)

9	33	Como utilizador, pretendo que a visibilidade dos segmentos nas diferentes seções da *dialog* de configuração seja implementada de acordo a matriz de visibilidade (em anexo K).
10	34	Como utilizador, pretendo que os segmentos possam exibir determinados ícones (em anexo J) que representem estados especiais na *dialog* de configuração. Os ícones correspondem aos seguintes estados: • *O* segmento não está configurado no sistema: O segmento não está acessível através da interface *BACAccess*. • O *Master Segment* não está no sistema: O *Master Segment* do objeto considerado não está no sistema mas o objeto em si está. • O segmento está *offline*: O dispositivo/autómato ao qual o segmento pertence não está acessível, apesar de o segmento estar referenciado no nó SVO. Não é possível ler ou escrever propriedades deste objeto. • O *Master Segment* está *offline*: O *Master Segment* do objeto considerado está offline mas o objeto em si não. • O segmento está inconsistente: Significa que há incoerências em algumas propriedades do objeto.

4.3 Algumas considerações

Para a maioria das *User Stories* especificadas, foi produzido um protótipo com a interface final (excetuando os casos em que não existiam interfaces aplicáveis). Estes protótipos foram desenvolvidos por *designers* do projeto, neste caso pertencentes ao cliente Sauter e serviram não só para garantir a utilização das melhores práticas de usabilidade, mas também para detalhar a funcionalidade associada à *User Story* (e.g. ações disponíveis, mensagens de aviso, mensagens de erro, etc.). Na Figura 17 exemplifica-se um destes protótipos, neste caso definido para a *User Story* nº 7.

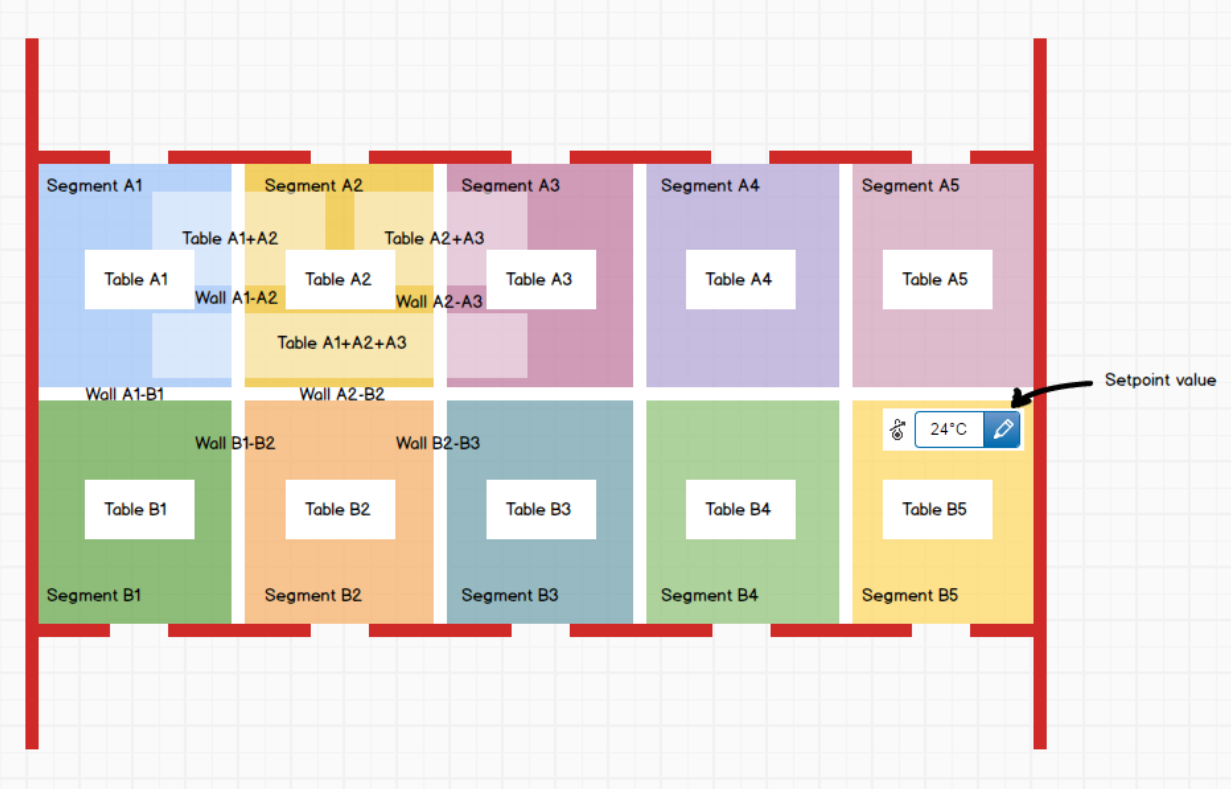


Figura 17 - Protótipo da User Story nº7

5 Implementação

Este capítulo descreve a implementação da solução de automação de espaços desenvolvida no âmbito deste projeto.

O projeto *moduWeb Vision* foi desenvolvido na linguagem de programação *Java* [10], com recurso à ferramenta *Gradle* [11]. Esta ferramenta permitiu a automatização da compilação dos diversos módulos que compõem o projeto. Utiliza uma linguagem de domínio específico baseada em Groovy [12] para declarar a configuração do projeto.

5.1 Implementação do *back-end*

A implementação do *back-end* aconteceu em primeiro lugar e foi dividida em duas fases, nomeadamente:

- Implementação de novos elementos estruturais;
- Implementação de controladores e serviços.

Seguidamente, serão apresentados detalhes de implementação destas duas fases.

5.1.1 Implementação de novos elementos estruturais

Como se encontra descrito no capítulo referente à arquitetura, a estrutura hierárquica SVO e os componentes nela apresentados através das *dynamic pictures* são especificados num ficheiro XML designado por *svo.xml.* A forma como estes componentes são especificados segue determinadas diretivas definidas num ficheiro XSD denominado *svo.xsd*. Desta forma, para conseguir representar os novos elementos implementados no âmbito deste estágio, foi necessário começar por atualizar esse XSD no sentido de acomodar esses novos elementos.

Os novos elementos criados estão relacionados com a automação de espaços e neste caso em particular com a gestão das divisões/segmentos de um edifício. Nesse sentido, foi adicionado um elemento ao XSD com a designação "*Room_Management*", ilustrado na Figura 18, que engloba os seguintes elementos:

- Um identificador do elemento (*Id*), em forma de uma *string*.
- Uma *flag* (*Show_Segments*) que permite definir se os segmentos devem ser exibidos ou ocultados na *dynamic picture*.
- Um botão (*Config_Button*), exibido numa determinada posição (configurável) da *dynamic picture*, e que permite ao utilizador abrir o configurador dos segmentos.
- Uma sequência de segmentos (*Segment*), cuja especificação se encontra ilustrada na Figura 19. Cada segmento é representado por um identificador (*Id)*, uma referência para o *id* do objeto do autómato a que este está associado (*SVO_BAC_Object_Ref*), uma posição específica na *dynamic picture* (*Position*), e um polígono (*Polygon*) composto por três ou mais coordenadas *xy*.
- Uma sequência de elementos do tipo *GroupAxis_Conditioned_Container*, cuja especificação se encontra ilustrada na Figura 20. Estes elementos permitem cumprir uma funcionalidade específica, descrita na *User Story* 7 do capítulo dos requisitos, que de uma forma geral pretende que outros componentes (*Component*) das *dynamic pictures* que não sejam segmentos, sejam exibidos segundo determinadas condições relacionadas com segmentos específicos. Estas condições estão representadas na Figura 20 pelos elementos *Master, Slave, Alone, Together, Together_Exclusive, Not_Together*. Cada um desses elementos pode possuir referências para segmentos (*Group_Axis_Object*) e permite, por exemplo, que um determinado componente (por exemplo uma imagem), referenciado no elemento *Component*, não seja exibido se um determinado segmento, referenciado no elemento *Group_Axis_Object* for *Master Segment*. Na secção 5.2.1 serão apresentados casos práticos da aplicação desta funcionalidade.

```
<xs:complexType name="Room_Management">
    <xs:sequence>
        <xs:element name="Segment" type="Segment" minOccurs="0" maxOccurs="56" />
        <xs:element name="Config_Button" minOccurs="1" maxOccurs="1">
            <xs:complexType>
                <xs:attributeGroup ref="Position" />
            </xs:complexType>
        </xs:element>
        <xs:element name="GroupAxis_Conditioned_Container"
            type="GroupAxis_Conditioned_Container" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="Id" type="Simple_String" use="required" />
    <xs:attribute name="Show_Segments" type="xs:boolean" use="optional" default="false" />
</xs:complexType>
```

Figura 18 - Elemento Room_Management no ficheiro svo.xsd

```
<xs:complexType name="Segment">
    <xs:attribute name="Id" type="Simple_String" use="required" />
    <xs:attributeGroup ref="SVO_BAC_Object_Ref" />
    <xs:attributeGroup ref="Position" />
    <xs:sequence>
        <xs:element name="Polygon" type="Polygon" />
    </xs:sequence>
</xs:complexType>
```

Figura 19 - Elemento Segment no ficheiro svo.xsd

```
<xs:complexType name="GroupAxis_Conditioned_Container">
    <xs:sequence>
        <xs:element name="Component" type="Component" minOccurs="1"
            maxOccurs="unbounded" />
        <xs:choice maxOccurs="1" minOccurs="1">
            <xs:element name="Master">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Group_Axis_Object" type="SVO_BAC_Object_Ref" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Slave">...
            </xs:element>
            <xs:element name="Alone">...
            </xs:element>
            <xs:element name="Together">...
            </xs:element>
            <xs:element name="Together_Exclusive">...
            </xs:element>
            <xs:element name="Not_Together">...
            </xs:element>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Id" type="Simple_String" use="required" />
    <xs:attributeGroup ref="Position" />
</xs:complexType>
```

Figura 20 - Elemento GroupAxis_Conditioned_Container no ficheiro svo.xsd

Após a adição dos novos componentes nos ficheiros XSD e XML do módulo SVO, para que o servidor fosse capaz de processar estes dados, foi necessário implementar dois processos:

- Desserialização dos dados, isto é, a leitura dos dados XML provenientes do ficheiro *svo.xml* referentes aos elementos *Room_Management* e a instanciação de objetos Java para representar a informação lida a partir do XML.
- Serialização dos dados, isto é, a leitura dos dados a partir de objetos Java e a posterior escrita desses dados novamente no ficheiro XML.

O processo de desserialização dos dados é necessário para que o servidor leia o ficheiro *svo.xml* e a partir daí mapeie essa informação em objetos Java, que sejam utilizáveis pelo servidor. Este processo ocorre sempre que o servidor é iniciado.

O processo de serialização dos dados é necessário para que os valores dos objetos Java possam ser novamente mapeados e escritos no ficheiro XML. Este processo ocorre quando um utilizador efetua alterações ao SVO.

Após o processamento dos dados relativos ao *Room Management* a partir do ficheiro *svo.xml*, estes são convertidos num formato *JSON* e enviados para o lado cliente quando o servidor recebe um pedido de visualização de uma *dynamic picture* que contenha elementos *Room Management*. Posteriormente, esses dados em formato *JSON* são utilizados como dados de configuração pelas diversas *widgets jQuery UI* criadas, descritas na secção 5.2.1.

A funcionalidade referente à visualização dos novos componentes *Room Management* nas *dynamic pictures* não exigiu a implementação de novos controladores, uma vez que já existia implementado um controlador relacionado com a visualização de componentes nas *dynamic pictures*, sendo apenas necessário adaptá-lo aos novos componentes implementados. No entanto, para a funcionalidade referente ao configurador de segmentos, por se tratar de uma funcionalidade nova, específica deste tipo de componentes e por possuir várias especificidades associadas, optou-se pela implementação de um controlador, descrito na próxima secção, juntamente com a descrição de um serviço também implementado no âmbito da funcionalidade do configurador de segmentos.

5.1.2 Implementação de controladores e serviços

Depois de implementados os novos elementos estruturais no módulo SVO, seguiu-se a implementação dos controladores e serviços, neste caso relativos à funcionalidade do configurador de segmentos, implementados no módulo *Application.*

Sempre que um utilizador executa um pedido de uma página no *moduWeb Vision* a partir de um *browser*, este pedido é processado por uma *servlet*. De uma forma simples, uma *servlet* pode ser definida como um objeto *Java* que recebe requisições/pedidos e produz uma resposta que pode ser por exemplo uma página *HTML*. No caso do projeto *moduWeb Vision*, existe uma *servlet* principal, denominada *MasterControllerServlet* que processa a generalidade dos pedidos reencaminhando-os para diferentes controladores, de acordo com a funcionalidade em causa. Um controlador, por

sua vez, é uma classe *Java* que possui diversos métodos que pretendem dar resposta a estes pedidos. Nesse sentido, a solução desenvolvida neste projeto envolveu a criação de um novo controlador, designado por *RoomManagementConfigController*, para dar resposta a pedidos provindos do *client-side* aquando da manipulação do configurador de segmentos. Este controlador foi inteiramente implementado no desenvolvimento da solução e lida essencialmente com:

- Pedidos para obtenção de dados relativos aos segmentos/divisões presentes numa determinada *dynamic picture*, utilizados posteriormente no configurador. Estes dados são descritos na secção 5.3.1 deste capítulo e dizem respeito a propriedades que os segmentos possuem enquanto objetos de um autómato e que permitem identificar as diferentes associações entre si. São obtidos a partir de uma interface fornecida pelo módulo BAC do projeto (descrito no capítulo referente à arquitetura), responsável por toda a comunicação do servidor *moduWeb Vision* com os autómatos a ele associados.
- Pedidos para a submissão de uma nova configuração de segmentos. Estas configurações chegam ao controlador num formato *JSON* em que são descritas referências para os objetos que representam os segmentos nos autómatos, propriedades desses objetos, e o valor a definir nessas propriedades de acordo com a configuração submetida pelo utilizador. Mais uma vez, recorre-se à interface fornecida pelo módulo BAC, neste caso para efetuar operações de escrita sobre as propriedades dos objetos dos autómatos que representam os segmentos.

Adicionalmente, foi implementada uma classe designada por *RoomManagementService* que funciona como um prestador de serviços ao controlador implementado. Esta classe possui métodos que permitem "assistir" o controlador. Os principais incluem:

- Métodos para obtenção de dados relativos ao *Room Management* de uma determinada *dynamic picture* em formato *JSON* a partir de objetos Java, utilizados posteriormente na *widget* do configurador, implementada no lado cliente (abordada na secção 5.2.2) para preencher as diversas secções que compõem o configurador com a informação dos diversos segmentos*;*
- Métodos para obtenção de diversas propriedades referentes aos Segmentos a partir de referências (id's) que representam estes objetos enquanto *datapoints* de um determinado autómato.

5.2 Implementação do *front-end*

A implementação do *front-end* ocorreu após a implementação de toda a estrutura necessária no lado servidor e foi dividida em 2 fases:

- Implementação de componentes *room management* nas *dynamic pictures* dos nós SVO.
- Implementação do configurador de segmentos.

Tanto na fase de implementação de componentes *room management* nas *dynamic pictures* dos nós *SVO* como na fase de implementação do configurador de segmentos, estiveram presentes duas ferramentas essenciais:

- *jQuery* [13], uma biblioteca *JavaScript cross-browser* desenvolvida para simplificar *scripts client-side* que interagem com *HTML*.
- *jQuery UI* [14], uma biblioteca implementada a partir do *jQuery* que permite implementar diversos componentes visuais em páginas *web*, além de fornecer outras funcionalidades básicas que potenciam a construção de páginas web interativas. Neste caso, importa referir que todos os componentes visuais inerentes ao *room management* foram implementados através de uma funcionalidade disponibilizada pelo *jQuery UI*, denominada por *Widget Factory* [15]. No contexto do *jQuery UI*, uma *widget* é um componente visual que permite exibir numa página web um conjunto de dados e que tipicamente contém uma combinação de HTML, CSS e JavaScript.

5.2.1 Implementação de componentes *room management* nas *dynamic pictures*

A implementação de componentes visuais nas *dynamic pictures* segue uma determinada estrutura. Como se encontra descrito no capítulo referente à arquitetura, essa estrutura pode ser divida em dois tipos:

- Componentes estáticos;
- Componentes dinâmicos.

Estes componentes, independentemente da categoria em que se encontram, são implementados através da *Widget Factory* do *jQuery UI*, e nesse sentido cada componente representa uma *widget* diferente.

Por forma a implementar os novos componentes associados à funcionalidade do *room management*, foi seguida a mesma lógica de implementação dos componentes já existentes.

Desse modo, foi então necessário implementar novas *widgets* que pudessem representar os novos componentes necessários.

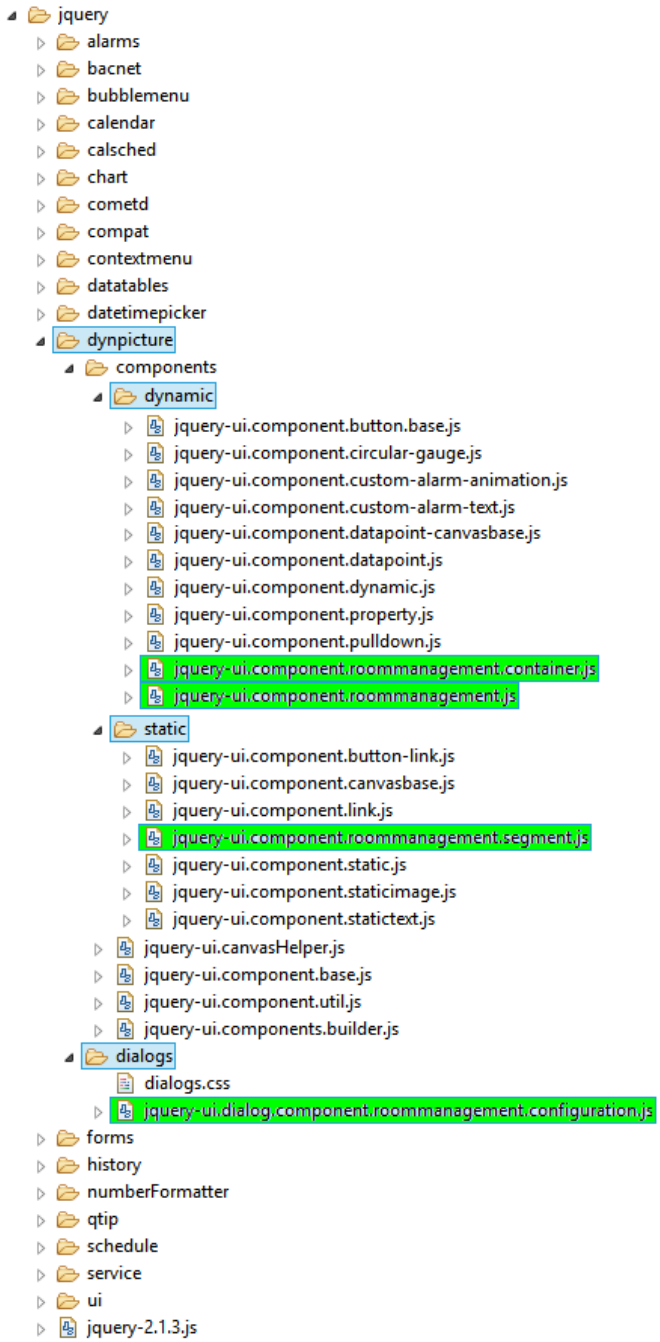


Figura 21 - Novas widgets jQuery UI implementadas

A Figura 21 ilustra com uma cor de fundo verde, os novos componentes (*widgets*) criados no desenvolvimento da solução.

A primeira fase de implementação do *front-end*, denominada "Implementação de componentes *room management* nas *dynamic pictures* dos nós *SVO*", está diretamente relacionada com a implementação de três das *widgets* representadas na Figura 21, referentes aos ficheiros: *jquery-ui.component.roommanagement.container.js, jquery-ui.component.roommanagement.js* e *jquery-ui.component.roommanagement.segment.js.*

A *widget* correspondente ao ficheiro *jquery-ui.component.roommanagement.container.js* foi implementada para cumprir uma *User Story* específica do cliente. Essa *User Story* encontra-se descrita no capítulo dos Requisitos com o número 7 e de forma geral descreve uma funcionalidade que tem como objetivo possibilitar que outros componentes da *dynamic picture* que não sejam segmentos e estejam presentes na *dynamic picture* possam ser exibidos segundo determinadas condições/propriedades associadas a esses segmentos. Este componente é dinâmico, uma vez que as propriedades dos segmentos dos autómatos podem ser alteradas a qualquer momento.

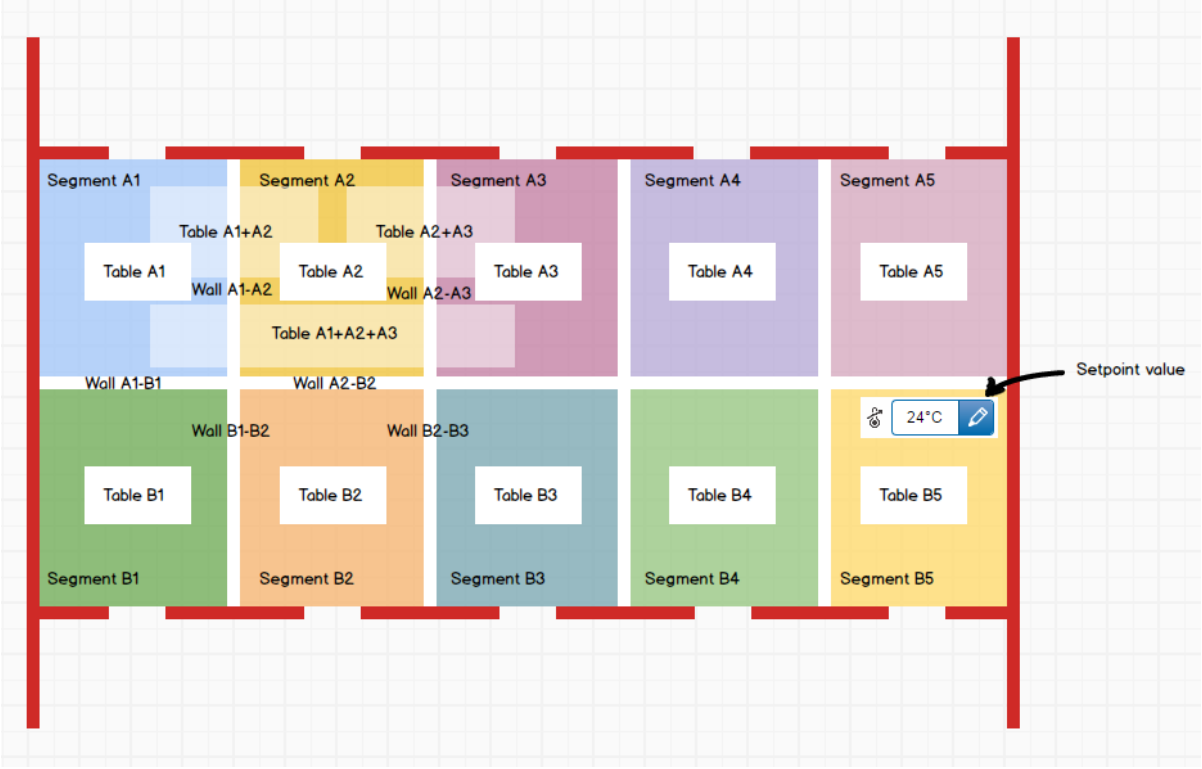


Figura 22 - Exemplo de aplicação da widget jquery-ui.component.roommanagement.container.js

A Figura 22 ilustra um exemplo real de aplicação desta *widget*. Nessa figura, podemos ver a representação de uma planta de um andar de um edifício numa *dynamic picture*. Representadas a vermelho estão paredes fixas enquanto que as paredes móveis estão representadas por componentes com uma cor branca. Na imagem, para além de um conjunto de segmentos representados com diferentes cores, pode ser observado outro tipo de componentes que representam mesas.

Através da implementação da referida *widget*, foi possível condicionar a exibição de determinados componentes com base em propriedades dos segmentos. Pegando no exemplo ilustrado na Figura 22, através desta funcionalidade é possível, por exemplo:

- Exibir a mesa denominada "Table A1+A2" sempre que os Segmentos A1 e A2 fizerem parte do mesmo grupo;
- Ocultar a parede "Wall A1-B1" sempre que os Segmentos A1 e B1 fizerem parte do mesmo grupo;
- Exibir a mesa "Table A1" apenas se o Segmento A1 não pertencer a nenhum grupo.

As *widgets* correspondentes aos ficheiros *jquery-ui.component.roommanagement.js* e *jquery-ui.component.roommanagement.segment.js* estão diretamente relacionadas. A *widget* implementada no ficheiro *jquery-ui.component.roommanagement.segment.js* foi implementada com o intuito de criar um componente que pudesse representar a forma poligonal de um segmento e as adicionais especificidades associadas a este. Nesse sentido, os segmentos são então representados por esta *widget* na *dynamic picture* e possuem diversas propriedades, das quais se destacam:

- Nome: Representa o nome que o segmento possui enquanto objeto de um autómato.
- PlanId: Representa uma propriedade deste tipo de objetos cuja utilidade será explicada mais à frente;
- Ícone: Cada segmento possui um determinado estado, sendo que todos os estados "anormais" possuem um ícone específico que é representado no polígono do segmento.
- Cor: Cada segmento possui uma cor, que pode variar consoante o segmento se encontre num determinado grupo de segmentos ou sozinho.

Apesar da *widget* que representa os segmentos possuir propriedades que podem ser alteradas ao longo do tempo, esta foi criada como uma *widget* estática, uma vez que as alterações às suas propriedades são geridas pela *widget* implementada no ficheiro *jquery-ui.component.roommanagement.js*, descrita a seguir.

A *widget* implementada no ficheiro *jquery-ui.component.roommanagement.js* foi criada para cumprir os requisitos que dizem respeito aos componentes *Room Management* nas *dynamic pictures* dos nós *SVO*. Nesse sentido, esta *widget* é responsável por criar e atualizar os componentes dos segmentos (descritos em cima). Além disso, esta é também responsável por criar um componente que representa um botão que permite ao utilizador ação de exibir/ocultar os segmentos na *dynamic picture* e ainda abrir uma *dialog* onde se encontra implementado o configurador dos segmentos.

A Figura 23 apresenta um exemplo real de utilização desta *widget*, onde se podem observar diversos segmentos e o botão referido anteriormente.

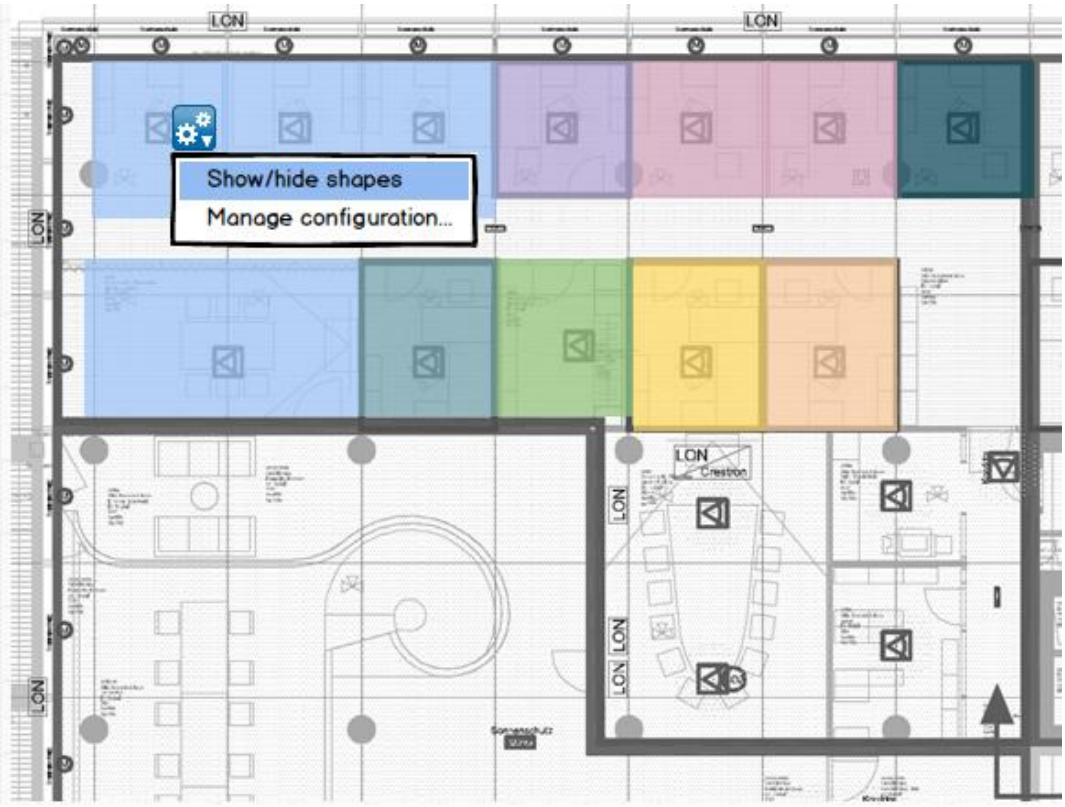


Figura 23 - Exemplo de aplicação da widget jquery-ui.component.roommanagement.js

5.2.2 Implementação do configurador de segmentos

A segunda fase de implementação do *front-end* envolveu a "Implementação do configurador de segmentos". A implementação deste configurador no *front-end* foi também baseada na criação de uma *widget jQuery UI*, implementada no ficheiro *jquery-ui.dialog.component.roommanagement.configuration.js*, sob a forma de uma *dialog* com várias secções com diferentes funções. O objetivo principal deste componente foi, tal como o nome indica, o de possibilitar ao utilizador criar diferentes configurações entre os segmentos de uma determinada *dynamic picture.* As diferentes secções implementadas neste configurador podem ser visualizadas na Figura 24 e descrevem de forma resumida:

- Uma lista de segmentos em forma de tabela com possibilidade de filtragem, em que cada segmento nela contido não se encontra associado a nenhum grupo e nesse sentido é denominado por *Single Segment*, representado pelo nome do segmento e por um pequeno quadrado que possui a cor que o identifica.
- Uma secção que apresenta ao utilizador os diferentes grupos de segmentos que estão associados entre si formando grupos, denominados por *Room Groups*. Cada grupo de segmentos possui um segmento que é responsável pelo grupo, designado *Master Segment*, sendo que todos os segmentos nele contidos obtêm a cor desse segmento. Nesta secção são possibilitadas as seguintes ações: Remoção de um grupo de segmentos, remoção de um segmento de um grupo, alteração do *Master Segment* de um grupo através da seleção de um *radio button*, *drag & drop* de múltiplos segmentos da lista de *Single Segments* para um grupo existente ou para um novo grupo, *drag & drop* de múltiplos segmentos de um grupo para outro grupo já existente ou para um novo grupo. Esta secção possui ainda dois botões que permitem ao utilizador submeter a configuração efetuada ou cancelar a operação.
- Uma secção de pré-visualização das alterações efetuadas que é atualizada de cada vez que o utilizador executa uma ação sobre os diferentes grupos de segmentos permitindo ao utilizador ter uma ideia visual do resultado final das configurações efetuadas.
- Uma secção que permite filtrar a exibição de segmentos no configurador segundo uma propriedade inerente a estes objetos denominada por *PlanId*.

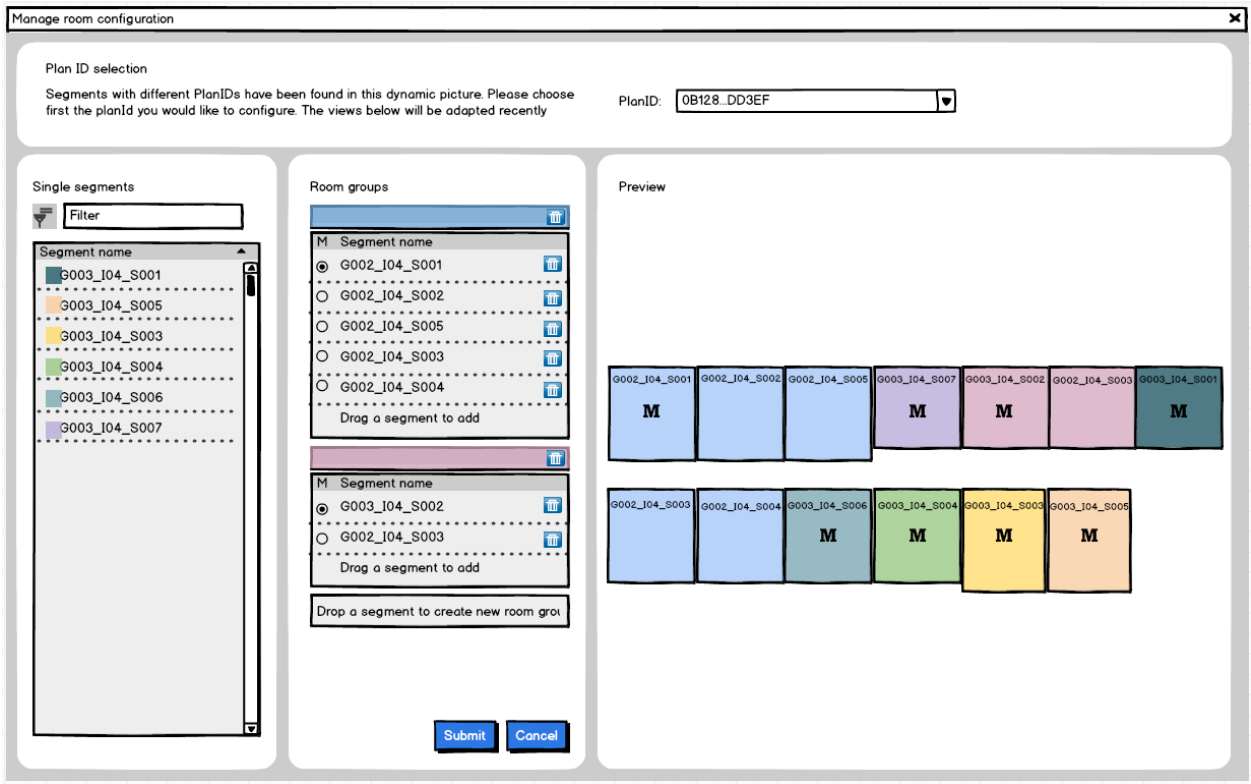


Figura 24 - Configurador de Segmentos

Seguidamente serão apresentados alguns detalhes de implementação que estiveram envolvidos no desenvolvimento de toda a solução.

5.3 Detalhes de implementação

5.3.1 Propriedades dos autómatos

Como se encontra referido no capítulo dedicado à descrição da arquitetura da solução, um segmento é representado internamente por um determinado tipo de objeto *BACnet* de um autómato específico, desenvolvido pelo cliente SAUTER para lidar com funcionalidades de automação de espaços. No sentido de implementar a solução de visualização e configuração dos segmentos de uma determinada *dynamic picture,* foi necessário estudar inicialmente estes objetos de maneira a compreender os seus princípios de funcionamento, as suas propriedades e de que forma estas poderiam ser utilizadas para permitir o desenvolvimento da solução. Entre as diferentes propriedades inerentes a este tipo de objetos, foram utilizadas as seguintes:

- *object_name*: propriedade utilizada para representar o nome do segmento no autómato;
- *axis_id:* representa um identificador que é único para cada segmento;
- *master_id:* no caso do segmento em causa estar inserido num determinado grupo de segmentos, esta propriedade representa o identificador do segmento que é responsável pelo grupo (*Master Segment*). Se o segmento em causa não pertencer a nenhum grupo ou pertencer mas como *Master Segment* desse grupo, o valor desta propriedade é igual ao da propriedade *axis_id.*
- *group_axis_binding:* esta propriedade representa um *array* com um número determinado de posições que são preenchidas com valores de propriedades *axis_id.* No caso do segmento em questão pertencer a um grupo e ser *Master Segment* desse grupo, este *array* irá conter os identificadores dos restantes segmentos que compõem o grupo (designados por *Slave Segments*). No caso do segmento em questão estar inserido num grupo e ser *Slave Segment* desse grupo, neste *array* será preenchida apenas uma posição que indica o identificador do *Master Segment* desse grupo.
- *plan_id:* esta propriedade representa o identificador de uma planta onde o segmento pode estar associado.

Das cinco propriedades descritas acima, foi necessário efetuar operações de leitura sobre todas elas e operações de escrita apenas sobre a propriedade *master_id.* As operações de leitura foram necessárias para enviar os dados destas propriedades dos segmentos a partir dos autómatos para a

widget do configurador, de maneira a preencher as diversas secções que o compõem. Já as operações de escrita recaíram apenas sobre a propriedade *master_id*, uma vez que o *axis_id* dos segmentos não é alterável e a propriedade *group_axis_binding* é gerida automaticamente pelos autómatos, com base nas alterações efetuadas à propriedade *master_id* dos segmentos.

5.3.2 Atualização dinâmica de componentes nas *dynamic pictures*

Tal como é descrito no capítulo referente à arquitetura, a atualização dinâmica dos componentes exibidos nas *dynamic pictures* é possibilitada através de um servidor *CometD* que utiliza uma tecnologia *push.*

Na fase de implementação de componentes *room management* nas *dynamic pictures* dos nós SVO, foi necessário fazer uso desta tecnologia já implementada no projeto de maneira a permitir a atualização automática das propriedades dos segmentos. Para atingir essa finalidade, foi implementado na *widget* do ficheiro *jquery-ui.component.roommanagement.js* um método para registo das propriedades inerentes aos segmentos no *UpdateManager* e um método de *callback* que é chamado quando as propriedades são atualizadas. Após receber as atualizações das propriedades pelo método de *callback*, foi apenas necessário aplicar esses novos valores nos segmentos, voltando a redesenhá-los nas *dynamic pictures.*

5.3.3 Desafios encontrados

No decorrer deste projeto, sucederam-se alguns desafios durante a fase de desenvolvimento. Alguns deles foram facilmente solucionados, outros exigiram um maior investimento e investigação. Um desses desafios surgiu durante o desenvolvimento do configurador de segmentos, e está relacionado com uma propriedade dos objetos *BACnet* que representam os segmentos nos autómatos, nomeadamente a propriedade *group_axis_binding*, já referida neste capítulo.

Como foi referido, a propriedade *group_axis_binding* representa um *array* com um número determinado de posições que são preenchidas com valores de propriedades *axis_id*. No caso do segmento em questão pertencer a um grupo e ser *Master Segment* desse grupo, este *array* contém os identificadores dos restantes segmentos que compõem o grupo. No caso do segmento em

questão estar inserido num grupo e ser *Slave Segment* desse grupo, neste *array* será preenchida apenas uma posição que indica o identificador do *Master Segment* desse grupo.

Durante o desenvolvimento verificou-se que esse *array*, por razões técnicas inerentes ao autómato, apenas possuía oito posições disponíveis, o que quer dizer que só seria possível criar grupos de segmentos com oito *Slave Segments* no máximo, o que seria uma limitação grave. Uma vez que estava fora de questão alterar esta propriedade diretamente nos autómatos de maneira aumentar o tamanho do *array group_axis_binding,* foi necessário investigar um pouco no sentido de desenvolver uma solução alternativa para resolver este problema.

Após alguma investigação, a solução encontrada passou pela utilização de uma hierarquia. Na prática, isto significa que um *Slave Segment* pode ser utilizado como um Sub-Master para outros *Slave Segments.* Esta hierarquia não teve influência na funcionalidade, serviu apenas para concentração/otimização do volume de transferências dentro de um grupo de segmentos. Para um utilizador, um grupo de segmentos pode conter até 56 *Slave Segments*.

O procedimento inerente a esta solução será descrito seguidamente.

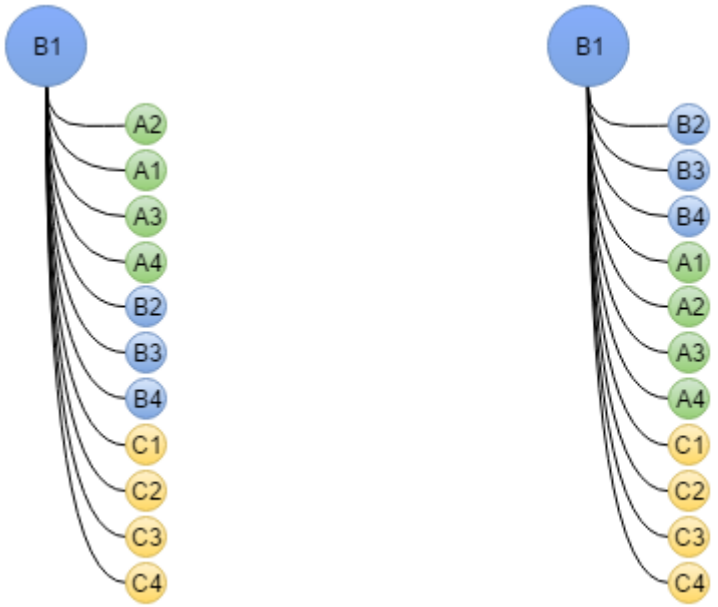
Se existirem oito ou menos *Slave Segments* definidos para um *Master Segment*, segue-se o procedimento normal e nesse sentido todos os *Slave Segments* são registados na propriedade *group_axis_binding* de forma normal.

Se existirem mais do que oito *Slave Segments* definidos para um mesmo *Master Segment*, é aplicado o seguinte algoritmo:

- **Passo 1 – Ordenação**:
  - Os *Slave Segments* de um *Master Segment* são ordenados pelos seguintes critérios:
    1. Identificador (*id*) do autómato;
    2. Valor da propriedade *axis_id* do *Slave Segment* em causa;
  - Exceções para o critério 1:
    - *Slave Segments* que pertençam ao mesmo autómato que o *Master Segment* serão inseridos no topo da lista.
  - Exemplo:
    - Neste exemplo, os segmentos são representados por uma combinação de uma letra e um algarismo, onde a letra descreve o autómato e o algarismo representa

a propriedade *axis_id*. O elemento B1 representa o *Master Segment* e todos os outros elementos são *Slave Segments* desse *Master Segment:*

- Estrutura inicial*:* Estrutura ordenada:



- **Passo 2 – Criação da hierarquia**
  - No segundo passo do algoritmo, a lista previamente ordenada é utilizada para criar uma estrutura em árvore. Por essa razão, é importante que no final:
    - Nenhum *Master Segment* (ou *sub-master*) possua mais do que oito *Slave Segments;*
  - Para atingir este resultado, é seguido o seguinte conjunto de regras:

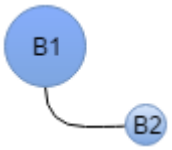
    Regra 1 - Na primeira vez, é adicionado o *Master Segment* no topo da lista. Os restantes segmentos serão adicionados em baixo deste seguindo determinadas regras. Todos os elementos são considerados numa abordagem *top-down*;

    Regra 2 - Se já existir um elemento do mesmo autómato, o segmento a inserir como *Slave Segment* será inserido como um "filho" do primeiro elemento encontrado.

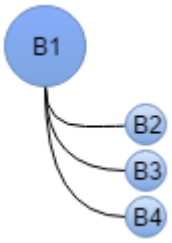
    Regra 3 - Caso contrário, isto significa que não foi encontrado nenhum segmento do mesmo autómato e o segmento a inserir será inserido como "filho" do primeiro segmento encontrado.

- Aplicando este passo ao exemplo anterior, o resultado seria o seguinte:
  - Inicialmente:

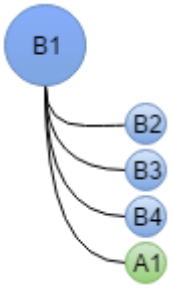
  - Após inserir B2 (pela regra 2):



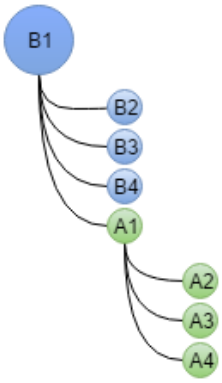
  - Após inserir B3 e B4 (pela regra 2):



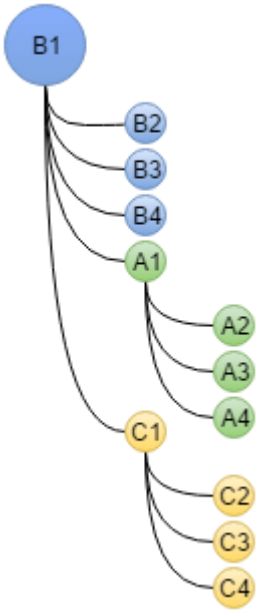
  - Após inserir A1 (pela regra 3):



  - Após inserir A2, A3 e A4 (pela regra 2):



- Após inserir C1, C2, C3 e C4 (pelas regras 3 e 2):



Essencialmente, pode dizer-se que este algoritmo solucionou o problema da limitação causada pela propriedade *group_axis_binding* através de uma heurística simples, ao introduzir o conceito de *sub-master* associado a uma estrutura em árvore. A nível de implementação, isto implicou, para além da implementação do algoritmo aquando da submissão de uma configuração de segmentos, a necessidade de, ao obter o *Master Segment* de um determinado segmento, ter o cuidado de verificar se este representava um *sub-master* ou se era efetivamente um *Master Segment.*

6 Testes

Este capítulo apresenta o processo de testes aplicados à solução desenvolvida no projeto *moduWeb Vision* no âmbito deste estágio.

A Critical Software possui um departamento de Qualidade dedicado, o qual delega um conjunto de pessoas responsáveis pela verificação e validação nos projetos da empresa, designadas por *Software Product Assurance Engineers (SPAEs)*. Contudo, não foi alocado nenhum *SPAE* ao projeto (por opção do cliente). Nesse sentido, foram seguidas as diretivas do *SCRUM* que defendem que cada membro da equipa é responsável por testar todos os módulos que desenvolve, estando um módulo finalizado apenas quando se encontra completamente testado. Além deste teste no final do desenvolvimento dos módulos, foram aplicados neste projeto diversos tipos de testes, nomeadamente: testes de aceitação, testes de sistema, testes exploratórios e testes de compatibilidade. Estes testes serão discutidos com mais pormenor seguidamente.

Relativamente ao processo de testes de aceitação adotado no projeto, este funcionou do seguinte modo: para cada *Sprint*, ao ser estimado o esforço necessário para implementar as *User Stories* na *Sprint Planning Meeting*, eram também incluídas as estimativas de esforço para a especificação e execução de testes. Durante o decorrer da *Sprint*, os testes eram especificados e executados, sendo os problemas/defeitos encontrados reportados na ferramenta *JIRA* e consequentemente resolvidos. Uma vez que a equipa não possuía um *SPAE*, estes testes eram executados por uma pessoa do projeto diferente daquela que implementou a *User Story*. Apesar dos testes serem apenas conhecidos durante o decorrer das *Sprints*, o processo de testes aplicado no projeto foi relativamente eficaz e permitiu que todas as *User Stories* fossem aceites nas *Sprint Review Meetings.* Por forma a não sobrecarregar o processo de validação no final das *Sprints*, foi definido um dia limite antes do fim de cada *Sprint* para que todas as *User Stories* ficassem prontas para testes. Esta solução permitiu assegurar o tempo necessário para testar as *User Stories* corretamente. Os casos de teste encontram-se especificados no anexo L.

Foram também executados alguns testes de sistema e testes exploratórios durante o decorrer do projeto. Os testes de sistema têm como principal objetivo o teste do sistema como um todo e a

verificação de que os requisitos da solução foram corretamente implementados. Já os testes exploratórios são testes informais realizados no sistema que não possuem uma especificação previamente definida nem os respetivos resultados esperados. A execução deste tipo de testes permitiu o levantamento de problemas não identificados pelos testes de sistema e de aceitação. Exemplo disso é o problema referido no capítulo relativo à implementação da solução, a respeito limitação da propriedade *group_axis_binding* nos objetos dos autómatos, que apenas permitia a criação de grupos de segmentos com oito elementos. Este problema não foi detetado pelos testes de aceitação nem pelos testes de sistema. Foi apenas detetado aquando da realização de testes exploratórios, na tentativa de criar um grupo com mais de oito segmentos a partir do configurador implementado. Após a deteção do problema, este foi reportado na ferramenta *JIRA* e posteriormente resolvido através da solução referida no capítulo relativo à implementação.

Além dos testes de sistema e exploratórios, foram adicionalmente executados testes de compatibilidade entre *browsers*. Quando se desenvolve um projeto para a *web*, é importante assegurar a sua compatibilidade entre diferentes *browsers*, implementando o mesmo design para diferentes versões e proprietários. No caso deste projeto, o próprio cliente indicou que a solução desenvolvida deveria ser suportada nos seguintes *browsers:* Internet Explorer (a partir da versão 10), Google Chrome (a partir da versão 47), Firefox (a partir da versão 38), Opera (a partir da versão 34) e Safari (a partir da versão 8). Toda a solução foi testada com sucesso nos diferentes *browsers* requeridos.

A demonstração final da solução desenvolvida teve lugar no dia 7 de Outubro de 2016, tendo todos os testes de aceitação sido executados na presença do cliente, bem como alguns cenários de teste específicos que este sugeriu. Esta demonstração foi conduzida por mim e foi considerada um sucesso pelo cliente. Com o término desta demonstração, ficou finalizado este projeto e solução foi colocada em produção.

7 Conclusões

Este relatório apresentou as atividades realizadas no decorrer do meu estágio curricular de Mestrado na empresa iTGROW. Durante o estágio tive a oportunidade de integrar um projeto da Critical Software – *moduWeb Vision* o qual consistiu no desenvolvimento de uma solução de automação de espaços para a empresa cliente SAUTER. Neste projeto foram-me confiadas as tarefas de desenvolvimento e validação, mais concretamente o desenvolvimento do componente relacionado com a automação de espaços e a validação de toda a solução desenvolvida.

A minha participação no projeto *moduWeb Vision* permitiu-me fazer parte de um projeto de desenvolvimento de *software* de elevada dimensão e adquirir experiência profissional relevante na área de engenharia de software, incluindo ambas as vertentes de desenvolvimento e validação. Relativamente aos conhecimentos técnicos adquiridos (*hard skills*), foi-me confiado o desenvolvimento de um componente de grande importância para a solução de automação de espaços do projeto *moduWeb Vision*. Este desafio permitiu que adquirisse conhecimentos em diversas tecnologias, que os aplicasse no desenvolvimento do componente pretendido e que ficasse com um *know-how* importante para futuros projetos da empresa. Para além das capacidades técnicas, este estágio permitiu melhorar as minhas capacidades de comunicação e de resolução autónoma de problemas (*soft skills*).

No decorrer do estágio foram também surgindo algumas dificuldades. Por exemplo, durante o desenvolvimento do configurador de segmentos, por vezes demorei mais tempo a executar tarefas do que o que tinha sido estimado, devido a alguma inexperiência com algumas tecnologias. Contudo, este problema acabou por ser ultrapassado com a realização de estimativas mais altas, ajustadas, conseguindo realizar as tarefas de uma forma menos pressionada e consequentemente com uma qualidade superior.

Apesar de todas as dificuldades que foram surgindo ao longo do estágio, fui sempre capaz de ultrapassá-las, tendo tido sempre ajuda dos meus colegas de equipa.

Para finalizar este relatório, quero salientar que a realização deste estágio contribuiu fortemente para a minha evolução profissional e pessoal. Toda a experiência que adquiri será uma mais-valia

no futuro, pois permitirá que participe em projetos de igual ou maior complexidade e que desempenhe papéis com igual ou superior responsabilidade. Quero também expressar a minha gratidão por ter atingido os objetivos inicialmente estabelecidos, facto que não seria possível sem o apoio prestado pelo meu orientador de estágio do ISEC e pelo meu coordenador de estágio da empresa.

Ao chegar ao fim deste capítulo de conclusões, termino o meu relatório de estágio curricular, realizado no âmbito da unidade curricular de Estágio ou Projeto Industrial do Mestrado em Informática e Sistemas, ramo de Desenvolvimento de Software, lecionado no Departamento de Engenharia Informática e de Sistemas do ISEC.

Referências

[1] SAUTER Group, Website oficial da SAUTER, [Online], http://www.sauter-controls.com/, acedido em Outubro de 2016.

[2] Scrum.org, Website oficial do SCRUM, [Online], http://www.scrum.org, acedido Outubro de 2016.

[3] Mountain Goat Software, [Online], https://www.mountaingoatsoftware.com/agile/scrum/overview, acedido em Outubro de 2016.

[4] SAUTER *moduWeb Vision*, [Online], http://www.sauter-controls.com/en/products-sauter/product-details/pdm/ey-ws-500-web-server-for-moduweb-vision-and-moduweb500-bacnet-networks.html, acedido em Novembro de 2016.

[5] Protocolo BACnet/IP, [Online], http://www.bacnet.org/Tutorial/BACnetIP/, acedido em Novembro de 2016.

[6] SAUTER Case Suite, [Online], http://www.sauter-controls.com/en/products-sauter/product-details/pdm/gzs-100-150-case-suite.html, acedido em Novembro de 2016.

[7] SAUTER Room Automation, [Online], http://www.sauter-controls.com/uploads/tx_cabagpdm/593475.pdf, acedido em Novembro de 2016.

[8] CometD, [Online], https://cometd.org/, acedido em Novembro de 2016.

[9] Tecnologia Push, [Online], https://www.pushtechnology.com/, acedido em Novembro de 2016.

[10] Oracle, Java Language and Virtual Machine Specifications, [Online], http://docs.oracle.com/javase/specs, acedido em Outubro de 2016.

[11] Gradle Build Tool, [Online], https://gradle.org/, acedido em Novembro de 2016.

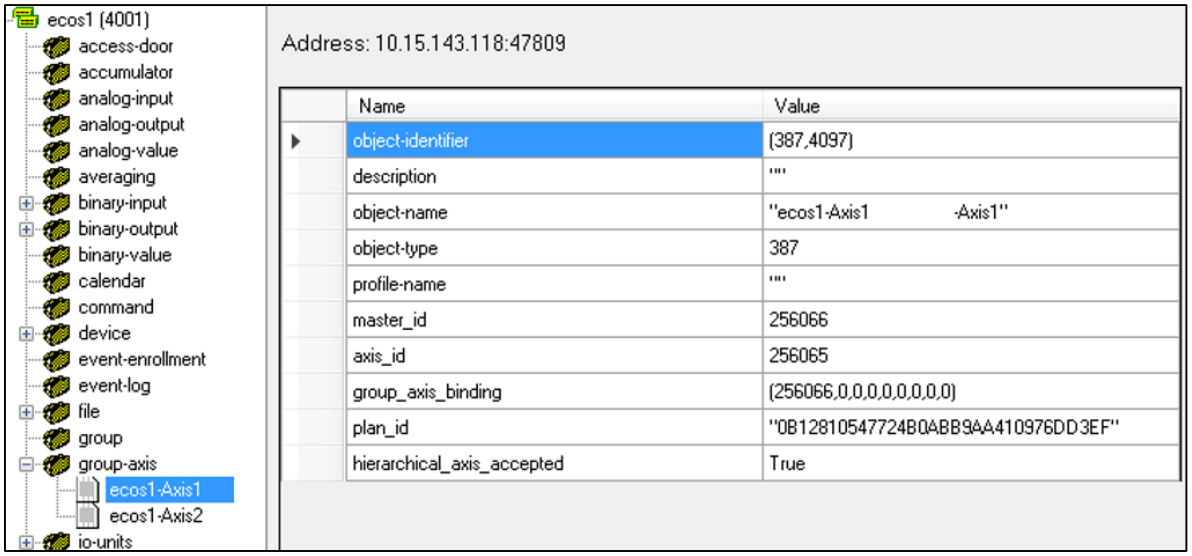
[12] Gradle Build Language Reference, [Online], https://docs.gradle.org/current/dsl/, acedido em Novembro de 2016.

[13] jQuery, uma biblioteca de JavaScript, [Online], https://jquery.com/, acedido em Novembro de 2016.

[14] jQuery UI, [Online], https://jqueryui.com/, acedido em Novembro de 2016.

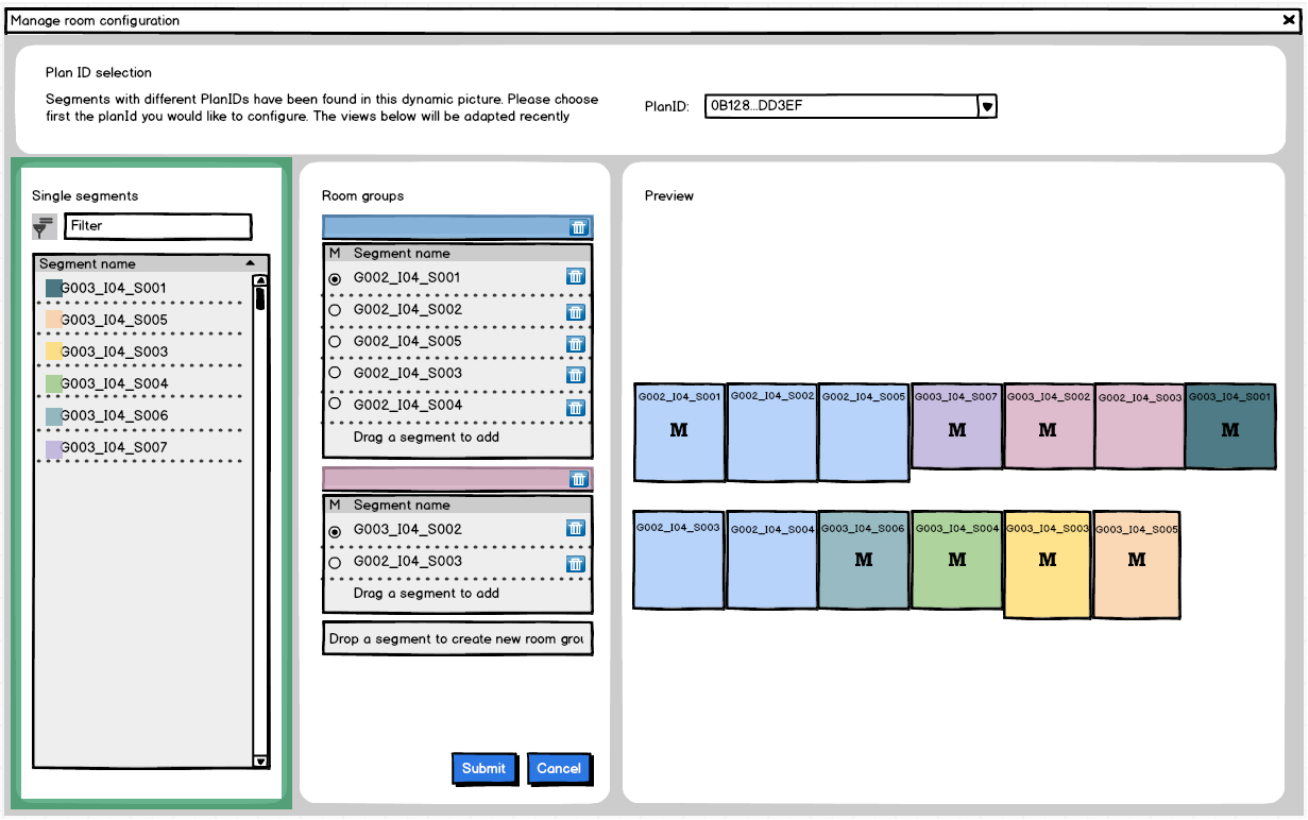
[15] jQuery UI Widget Factory, [Online], https://jqueryui.com/widget/, acedido em Novembro de 2016

ANEXO A – Lista de atributos de um objeto *group-axis*

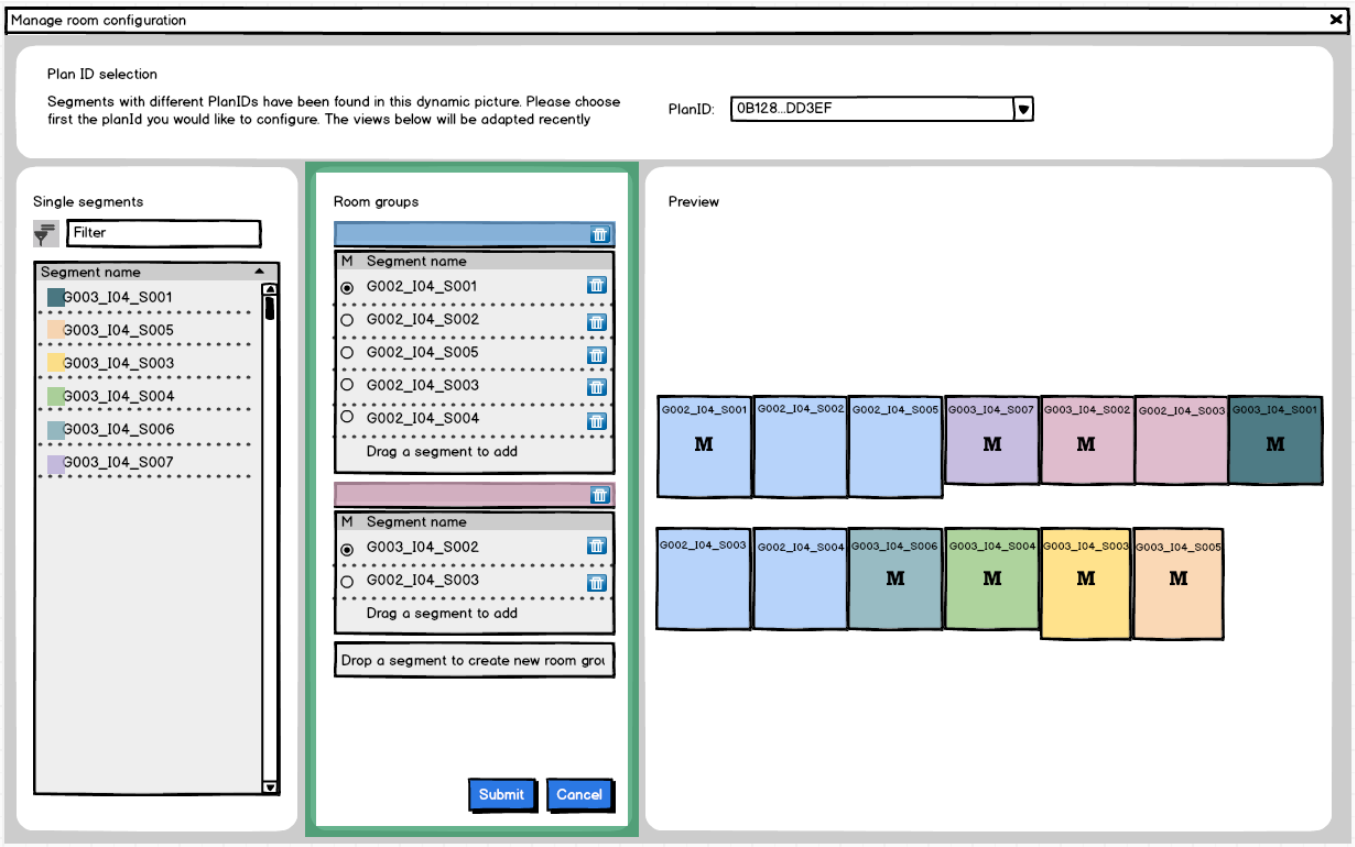


ANEXO B – *Mockup* do botão de multiselecção para exibição/ocultação de segmentos na *dialog* de configuração

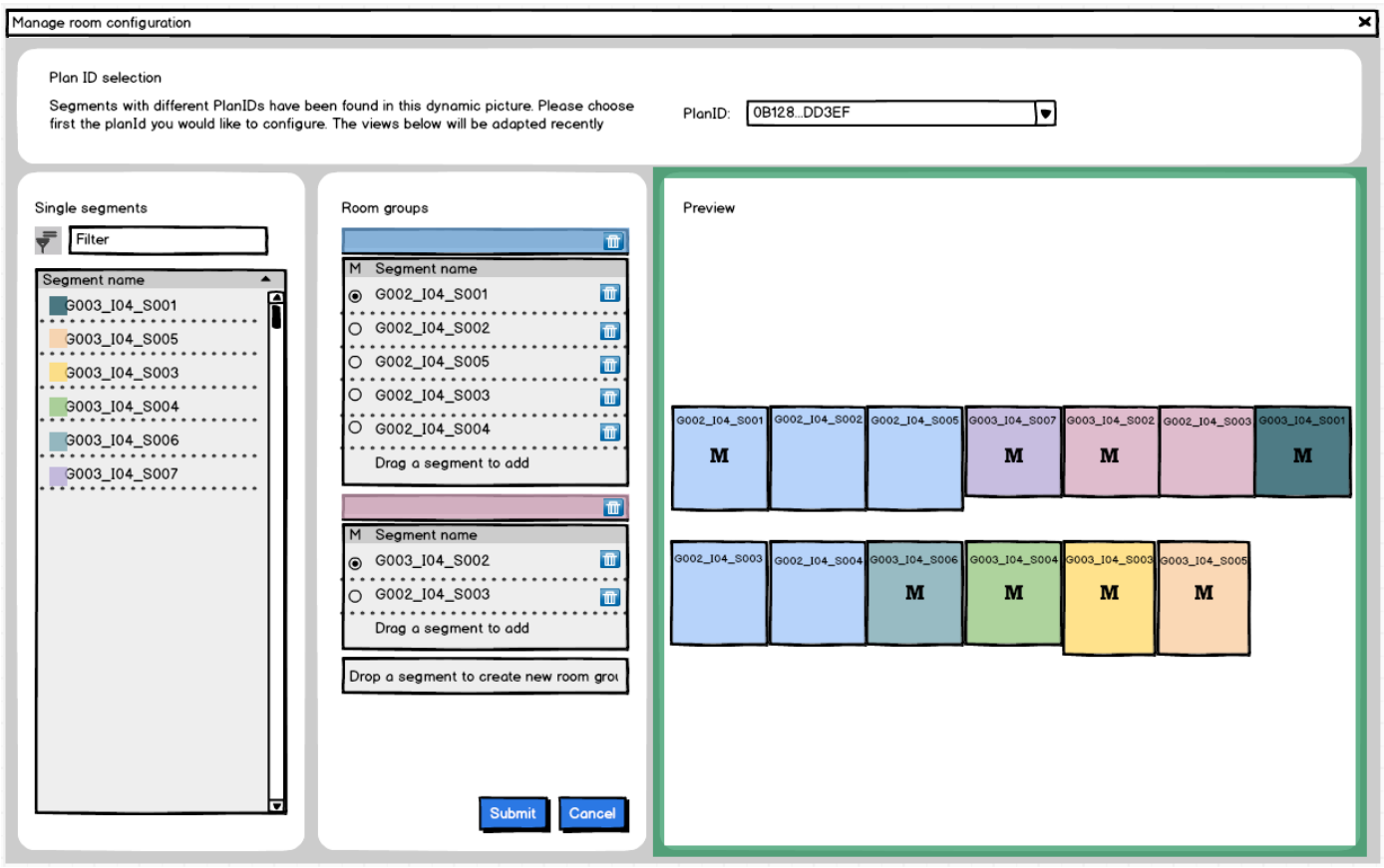
ANEXO C – *Dialog* de configuração com área de *Single Segments* em destaque



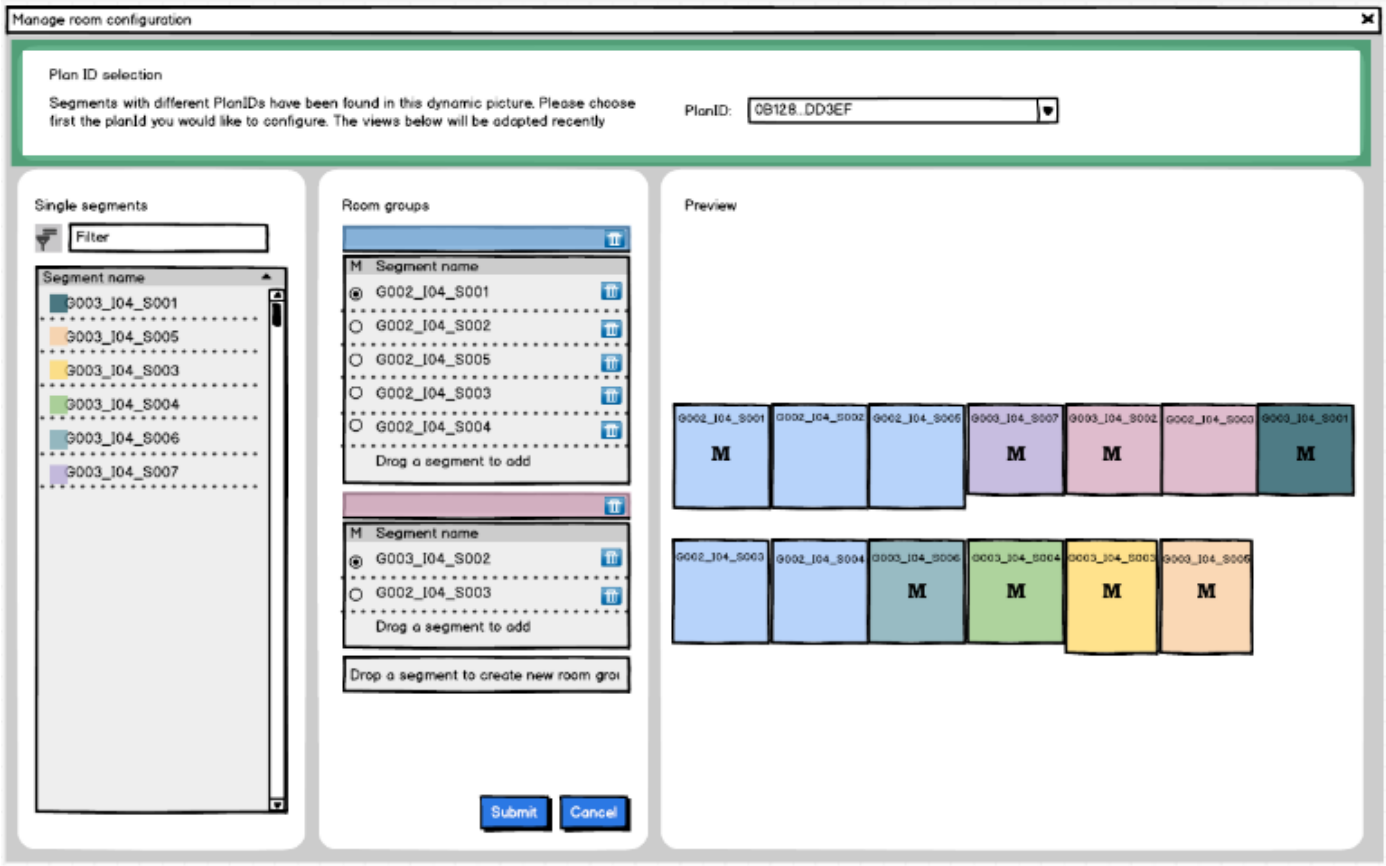
ANEXO D – *Dialog* de configuração com área de *Room Groups* em destaque



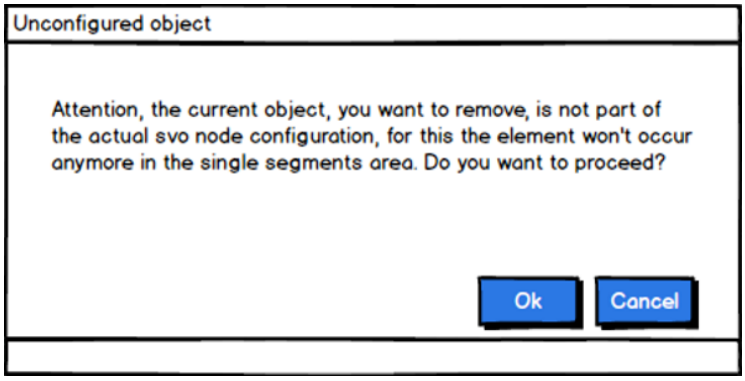
ANEXO E – *Dialog* de configuração com área de *Preview* em destaque



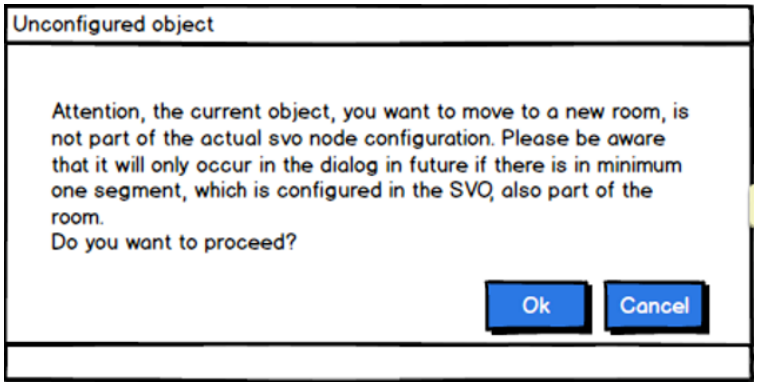
ANEXO F – *Dialog* de configuração com área de *PlanId* em destaque



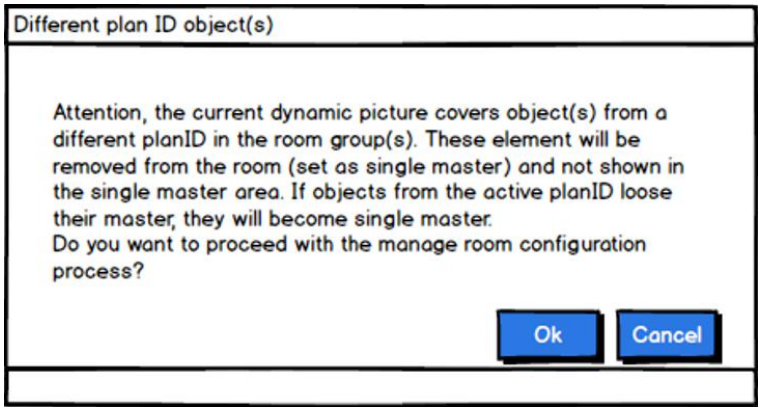
ANEXO G – Mensagem de aviso de remoção de segmento não configurado



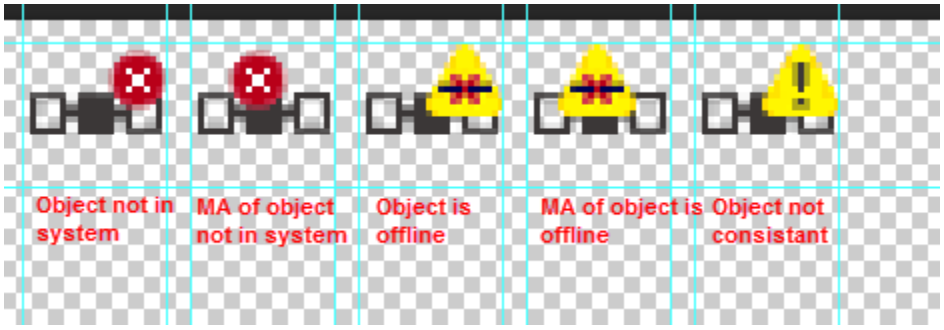
ANEXO H – Mensagem de aviso de adição de segmento não configurado a um *Room Group*



ANEXO I – Mensagem de aviso de existência de diferentes *PlanId's*



ANEXO J – Ícones de representação de estado dos segmentos



ANEXO K – Matriz de visibilidade dos segmentos

	Configuração do Sistema			Valor atual			Visibilidade na *dialog* de Configuração		
	Segmeno está no Sistema	Segmento está na Lista Subordinada	Segmento está no layout	Single Segment	Master Segment	Slave Segment	Tabela de Segmentos Singulares	Grupos de Segmentos	Secção de Pré-visualização
Caso de Uso 1A	**Sim**	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Visível**	**Oculto**	**Visível**
Caso de Uso 1B	**Sim**	**Sim**	**Sim**	**Não**	**Sim**	**Não**	**Oculto**	**Visível**	**Visível**
Caso de Uso 1C	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Sim**	**Oculto**	**Visível**	**Visível**
Caso de Uso 2A	**Sim**	**Sim**	**Não**	**Sim**	**Não**	**Não**	**Visível**	**Oculto**	**Oculto**
Caso de Uso 2B	**Sim**	**Sim**	**Não**	**Não**	**Sim**	**Não**	**Oculto**	**Visível**	**Oculto**
Caso de Uso 2C	**Sim**	**Sim**	**Não**	**Não**	**Não**	**Sim**	**Oculto**	**Visível**	**Oculto**
Caso de Uso 3A	**Sim**	**Não**	**Não**	**Sim**	**Não**	**Não**	**Oculto**	**Oculto**	**Oculto**
Caso de Uso 3B	**Sim**	**Não**	**Não**	**Não**	**Sim**	**Não**	**Oculto**	**Visível**	**Oculto**
Caso de Uso 3C	**Sim**	**Não**	**Não**	**Não**	**Não**	**Sim**	**Oculto**	**Visível**	**Oculto**
Caso de Uso 4A	**Não**	**Não**	**Não**	**Não**	**Sim**	**Não**	**Oculto**	**Visível**	**Oculto**
Caso de Uso 4B	**Não**	**Não**	**Não**	**Não**	**Não**	**Sim**	**Oculto**	**Visível**	**Oculto**

ANEXO L – Casos de teste

Nome:	TC-01: Representação de segmentos nas *dynamic pictures.*
User Story:	1
Propósito:	Exibição de uma interface que represente as divisões de um determinado espaço físico de um edifício sob a forma de polígonos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml.
Output:	Exibir os segmentos configurados sob a forma de polígonos.

Nome:	TC-02: Representação de cores nos segmentos.
User Story:	3
Propósito:	Exibição de cores de fundo nos segmentos de uma determinada *dynamic picture.*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml.
Output:	Exibir os segmentos configurados sob a forma de polígonos com uma cor definida para cada segmento.

Nome:	TC-03: Botão de configuração.
User Story:	4
Propósito:	Exibição de um botão que permita a exibição/ocultação dos segmentos de uma *dynamic picture*.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição/ocultação de segmentos.
Output:	Exibição e ocultação dos segmentos representados na *dynamic picture*.

Nome:	TC-04: Manutenção do estado do botão de exibir/ocultar segmentos entre pedidos.
User Story:	5
Propósito:	Manter o resultado do clique no botão de exibição/ocultação de segmentos entre pedidos
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de ocultação/exibição de segmentos; 3. Refrescar a página.
Output:	Após refrescar a página, os segmentos devem continuar visíveis ou ocultos, consoante a ação do utilizador sobre o botão antes do refrescamento da página.

Nome:	TC-05: Botão para exibição da *dialog* de configuração de segmentos.
User Story:	6
Propósito:	Exibir uma *dialog* para configuração dos segmentos após o clique num botão.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração.
Output:	Após o clique no botão, deve ser exibida uma *dialog*.

Nome:	TC-06: Tabela de *Single Segments* na *dialog* de configuração de segmentos.
User Story:	8
Propósito:	Exibir uma tabela que contenha os *Single Segments* presentes numa determinada *dynamic picture.*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração
Output:	Após o clique no botão, deve ser exibida na *dialog* de configuração uma tabela que contém os *Single Segments* da *dynamic picture* em causa.

Nome:	TC-07: Filtragem na tabela de *Single Segments.*
User Story:	13
Propósito:	Exibir um campo na tabela de *Single Segments* que permita a filtragem dos segmentos pelo seu nome.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Preencher o campo de filtragem da tabela.
Output:	Os segmentos exibidos na tabela devem ser apenas aqueles que correspondem ao filtro aplicado.

Nome:	TC-08: Tabela de *Room Groups* na *dialog* de configuração de segmentos.
User Story:	9
Propósito:	Exibir uma tabela que contenha os *Room Groups* presentes numa determinada *dynamic picture* e permita a criação de novos *Room Groups* através de *drag & drop* da tabela de *Single Segments.*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Efetuar um *drag & drop* de um segmento da tabela de *Single Segments* para a zona dos *Room Groups.*
Output:	Devem ser exibidos *Room Groups* existentes na tabela de *Room Groups* e, após a ação de *drag & drop*, os segmentos em causa devem ser movidos da tabela de *Single Segments* para a tabela de *Room Groups.*

Nome:	TC-09: *Styling* dos *Room Groups*.
User Story:	12
Propósito:	Aplicar um determinado *styling* aos *Room Groups* da *dialog* de configuração de segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração.
Output:	Os *Room Groups* devem estar representados na forma de tabelas em que o cabeçalho deve possuir como cor de fundo a cor do segmento principal do *Room Group* (*Master Segment*) e as entradas das tabelas representam o nome dos segmentos que constituem o *Room Group*,

Nome:	TC-10: Alteração do *Master Segment* de um *Room Group*.
User Story:	15
Propósito:	Ter a possibilidade de alterar o *Master Segment* de cada *Room Group* da *dialog* de configuração de segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Clicar no *radio button* associado a um segmento de um *Room Group*
Output:	O Segmento ao qual se encontra associado o *radio button* que o utilizador selecionou deve ser tornado o *Master Segment* desse *Room Group* e o cabeçalho do *Room Group* deve possuir como cor de fundo, a cor associada ao novo *Master Segment*.

Nome:	TC-11: Remoção de segmentos de *Room Groups* e remoção de *Room Groups*
User Story:	16
Propósito:	Ter a possibilidade de eliminar *Room Groups* e segmentos de *Room Groups* na *dialog* de configuração de segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Clicar no botão de remoção de um segmento contido num *Room Group;* 4. Clicar no botão de remoção de um *Room Group.*
Output:	O Segmento no qual o utilizador executou a ação de remoção deve passar para a tabela de *Single Segments* e o *Room Group* que o utilizador removeu deve ser apagado e todos os segmentos nele contidos devem passar para a tabela de *Single Segments.*

Nome:	TC-12: Remoção do último segmento de um *Room Group.*
User Story:	17
Propósito:	Eliminação de *Room Group* aquando da remoção do último segmento nele contido.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Clicar no botão de remoção de um segmento contido num *Room Group* que possua apenas um único segmento.
Output:	O *Room Group* deve ser removido da secção de *Room Groups* e o único segmento nele contido deve passar para a tabela de *Single Segments.*

Nome:	TC-13: Remoção de *Room Groups* com apenas um segmento.
User Story:	18
Propósito:	Eliminação de *Room Group* com um único segmento, após a reabertura da *dialog* de configuração.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Criar um *Room Group* com apenas um segmento; 4. Submeter a configuração; 5. Clicar no botão de exibição da *dialog* de configuração.
Output:	Os *Room Groups* contendo apenas um segmento devem ser removidos, passando os segmentos neles contidos para a tabela de *Single Segments.*

Nome:	TC-15: Remoção de um *Master Segment* de um *Room Group.*
User Story:	20
Propósito:	Selecionar um novo *Master Segment* de um *Room Group,* quando o anterior é removido do *Room Group* em questão.
Autor:	David Rego
Input:	1. Aceder a uma dynamic picture com segmentos configurados no svo.xml 2. Clicar no botão de exibição da dialog de configuração 3. Criar um *Room Group* com mais de um segmento. 4. Remover o *Master Segment* do *Room Group.*
Output:	O segmento que sucede o *Master Segment* inicial do *Room Group* deve ser selecionado como o novo *Master Segment* desse *Room Group.*

Nome:	TC-16: Movimentação de um *Master Segment* para um *Room Group* existente.
User Story:	21
Propósito:	Inserir o *Master Segment* no *Room Group* de destino como um *Slave Segment*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Criar um *Room Group* com pelo menos um segmento; 4. Mover um *Master Segment* para o novo *Room Group* criado.
Output:	O segmento deve ser adicionado ao *Room Group* de destino como um *Slave Segment.*

Nome:	TC-17: Movimentação um segmento de um *Room Group* para outro já existente.
User Story:	22
Propósito:	Permitir a movimentação de segmentos a partir de *Room Groups* para outros já existentes.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Criar dois *Room Group* com pelo menos um segmento; 4. Mover segmento de um dos *Room Groups* para o outro *Room Group*, através de *drag & drop.*
Output:	O segmento deve ser adicionado ao *Room Group* de destino como um *Slave Segment.*

Nome:	TC-18: Movimentação um segmento de um *Room Group* para um novo *Room Group*.
User Story:	23
Propósito:	Permitir a movimentação de segmentos a partir de *Room Groups* para novos *Room Groups.*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml 2. Clicar no botão de exibição da *dialog* de configuração 3. Criar um *Room Group* com pelo menos um segmento 4. Mover segmento do *Room Group* para um novo *Room Group*, através de *drag & drop.*
Output:	O segmento deve ser adicionado ao novo *Room Group* como M*aster Segment*

Nome:	TC-19: Reorganização vertical de *Room Groups.*
User Story:	24
Propósito:	Permitir a reorganização vertical de *Room Groups*.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Criar dois *Room Group* com pelo menos um segmento; 4. Mover os *Room Groups* verticalmente para novas posições, através de *drag & drop.*
Output:	Os Room Groups devem ser reorganizados verticalmente, de acordo com a ação de *drag & drop* do utilizador.

Nome:	TC-20: Movimentação de todos os segmentos de um *Room Group* para outro *Room Group.*
User Story:	25
Propósito:	Remoção de *Room Groups* aquando da movimentação de todos os segmentos nele contidos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml 2. Clicar no botão de exibição da *dialog* de configuração 3. Criar um *Room Group* com pelo menos um segmento 4. Mover todos os segmentos do *Room Group* criado para um novo *Room Group*, através de *drag & drop.*
Output:	O *Room Group* de origem deve ser eliminado

Nome:	TC-21: Manter alterações efetuadas em cache até submissão da configuração.
User Story:	32
Propósito:	Manutenção das alterações efetuadas na *dialog* de configuração em cache até submissão da configuração.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Efetuar alterações à configuração; 4. Fechar a *dialog* de configuração; 5. Clicar no botão de exibição da *dialog* de configuração.
Output:	As configurações iniciais dos segmentos devem ser mantidas.

Nome:	TC-22: Exibição de mensagem após remoção de um segmento que não esteja presente no nó SVO.
User Story:	26
Propósito:	Exibição de uma mensagem de aviso ao utilizador aquando da remoção de um segmento de um *Room Group* que não esteja presente no nó SVO.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Remover um segmento de um *Room Group* que não esteja presente no nó SVO.
Output:	Deve ser apresentada ao utilizador uma mensagem de aviso.

Nome:	TC-23: Exibição de mensagem após movimentação de um segmento não configurado para um *Room Group*
User Story:	14
Propósito:	Exibição de uma mensagem de aviso ao utilizador aquando da ação de *drag & drop* de um segmento não configurado para um *Room Group* novo ou já existente.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Mover um segmento não configurado da tabela de *Single Segments* para um *Room Group.*
Output:	Deve ser apresentada ao utilizador uma mensagem de aviso.

Nome:	TC-24: Secção de pré-visualização da *dialog* de configuração de segmentos
User Story:	10
Propósito:	Exibição de uma secção de pré-visualização na *dialog* de configuração dos segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml 2. Clicar no botão de exibição da *dialog* de configuração
Output:	Deve ser exibida uma secção de pré-visualização, que permita mostrar o resultado temporário das alterações que o utilizador executa sobre os segmentos a nível gráfico.

Nome:	TC-25: Atualização da secção de pré-visualização após cada alteração.
User Story:	27
Propósito:	Atualização da secção de pré-visualização na *dialog* de configuração dos segmentos após cada ação do utilizador sobre os segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml; 2. Clicar no botão de exibição da *dialog* de configuração; 3. Efetuar alterações sobre os *Single Segments* ou *Room Groups.*
Output:	As alterações efetuadas devem refletir-se na secção de pré-visualização.

Nome:	TC-26: Secção de filtragem de *PlanId's.*
User Story:	11, 28
Propósito:	Exibição de uma secção que permita a filtragem de segmentos pela propriedade *PlanId.*
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml com diferentes *PlanId's;* 2. Clicar no botão de exibição da *dialog* de configuração
Output:	Deve ser exibida uma secção na *dialog* de configuração que permita a filtragem de segmentos por *PlanId.*

Nome:	TC-27: Mensagem de aviso de segmentos com diferentes *PlanId's.*
User Story:	29
Propósito:	Exibição de uma mensagem de aviso na *dialog* de configuração no caso de existirem segmentos de outro *PlanID* configurados no *layout*.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml com diferentes *PlanId's;* 2. Clicar no botão de exibição da *dialog* de configuração.
Output:	Deve ser exibida uma mensagem de aviso.

Nome:	TC-28: Exibição de segmentos com diferentes *PlanId's.*
User Story:	30
Propósito:	Exibição de segmentos na *dialog* de configuração com uma cor de fundo cinza caso estes não façam parte do *PlanId* selecionado.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml com diferentes *PlanId's;* 2. Clicar no botão de exibição da *dialog* de configuração; 3. Selecionar um *PlanId.*
Output:	Os segmentos que não pertencem ao *PlanId* selecionado devem ser representados com uma cor de fundo cinza.

Nome:	TC-29: Exibição de segmentos de *PlanId's* não selecionados em *Room Groups.*
User Story:	31
Propósito:	Remoção de segmentos de *Room Groups* diferentes do *PlanId* selecionado.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml com diferentes *PlanId's* 2. Clicar no botão de exibição da *dialog* de configuração 3. Selecionar um *PlanId* que possua *Room Groups* com segmentos de outro *PlanId*
Output:	Os segmentos que não pertencem ao *PlanId* selecionado devem ser removidos dos *Room Groups*, sendo que no caso de os segmentos "perderem" o seu *Master Segment*, devem tornar-se *Single Segments*.

Nome:	TC-30: Visibilidade dos segmentos na *dialog* de configuração.
User Story:	33
Propósito:	Condicionar a visibilidade dos segmentos da *dialog* de configuração.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml com diferentes *PlanId's;* 2. Clicar no botão de exibição da *dialog* de configuração.
Output:	Verificar que a visibilidade dos segmentos se encontra de acordo com a matriz de visibilidade (definida no anexo K).

Nome:	TC-31: Representação de estados dos segmentos na *dynamic picture* e na *dialog* de configuração.
User Story:	34
Propósito:	Representação de estados especiais dos segmentos através da exibição de ícones nos segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo.xml que possuam os diferentes estados: segmento não configurado, *Master Segment* não configurado, segmento offline, *Master Segment* offline, segmento inconsistente; 2. Clicar no botão de exibição da *dialog* de configuração.
Output:	Os segmentos exibidos nas *dynamic pictures* e na *dialog* de configuração devem refletir o seu estado através da exibição de ícones específicos, definidos no anexo J.

Nome:	TC-32: Exibição condicionada de componentes nas *dynamic pictures.*
User Story:	7
Propósito:	Exibição de outros componentes da *dynamic picture* que não sejam segmentos e estejam presentes na vista gráfica segundo determinadas condições associadas a estes segmentos.
Autor:	David Rego
Input:	1. Aceder a uma *dynamic picture* com segmentos configurados no svo,xml e outro tipo de componentes cuja exibição está condicionada a propriedades desses segmentos.
Output:	Os componentes exibidos nas *dynamic pictures* que não sejam segmentos devem ser exibidos de acordo com determinadas propriedades inerentes aos segmentos aos quais estes se encontram associados, de acordo com a especificação da User Story 7.

ANEXO M – Proposta de Estágio

PROPOSTA DE ESTÁGIO

Ano Letivo de 2015/2016

Mestrado em Informática e Sistemas - Desenvolvimento de Software

TEMA

Automação de Espaços - SAUTER

SUMÁRIO

A *Critical Software S.A.* possui atualmente um conjunto de competências que coloca ao dispor dos seus clientes na resolução de problemas de negócio. Uma dessas competências é na área de Gestão Energética de Edifícios Inteligentes, uma área em forte expansão a nível mundial e do qual se espera que coloque a *Critical Software* na linha da frente no fornecimento deste tipo de soluções.

Um dos produtos em que a Critical está a trabalhar é o moduWeb Vision da empresa suíça Sauter, uma solução web para visualização e operação de sistemas HVAC, permitindo a monitorização e controlo de edifícios.

Uma das áreas de especialização da Sauter é a automação de espaços, tendo desenvolvido recentemente um autómato que lida com funções complexas de maneira a permitir um controlo preciso da temperatura, iluminação, sombra, etc., estando o estágio proposto relacionado com o desenvolvimento de uma solução para gestão e configuração desses espaços.

Âmbito

O trabalho a realizar está relacionado com o desenvolvimento de uma solução web para gestão da configuração desses espaços, com o objetivo do utilizador gerir/configurar os vários segmentos que constituem cada espaço/divisão de uma forma flexível, sendo depois essa configuração propagada para os autómatos.

O desenvolvimento envolverá uma parte *front-end* em que o utilizador irá desenvolver uma interface que permita ao utilizador configurar os segmentos e espaços/divisões e uma outra parte *back-end* onde a informação relativa à configuração será propagada para os autómatos.

Uma vez que poderão existir limitações nos autómatos ao nível de performance no que toca à escrita das configurações relativas aos espaços, o estagiário terá de realizar alguma investigação no sentido de obter uma solução que permita contornar essas limitações.

Objetivos

O estagiário será integrado na equipa de desenvolvimento da *Critical Software* e terá como objetivo a prototipagem, especificação, implementação e validação das funcionalidades do sistema.

Pretende-se que o estagiário especifique as soluções a desenvolver, incluindo a análise das aplicações e serviços existentes, efetue a prototipagem necessária e efetue a validação do que foi implementado.

O estagiário vai estar integrado numa equipa de desenvolvimento em metodologia AGILE - SCRUM, sendo capaz de fazer a integração entre as soluções que vai desenvolver e as aplicações existentes atualmente na *Critical Software*.

Programa de trabalhos

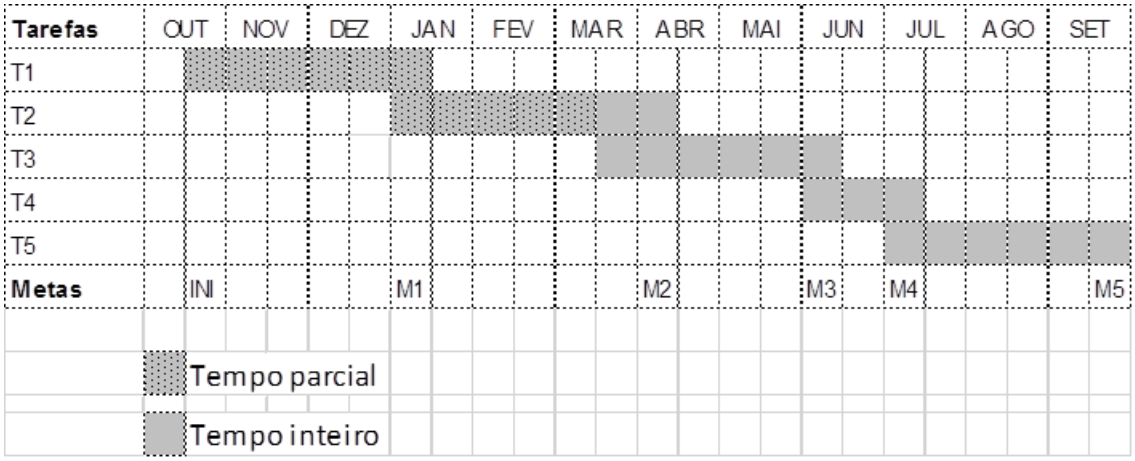
O estágio consistirá nas seguintes atividades e respetivas tarefas:

- ***T1*** *– Análise dos sistemas e processos existentes e definição do âmbito do sistema a desenvolver*;
- ***T2*** *– Análise de requisitos e especificação das funcionalidades a desenvolver acompanhada da prototipagem que se revelar necessária. Esta especificação e análise terão o apoio de elementos da equipa da Critical Sofware. Durante esta fase o estagiário deverá produzir um relatório técnico preliminar sobre o seu trabalho.*
- ***T3*** *- Codificação dos módulos especificados. A codificação será acompanhada de atividades de verificação (e.g. inspeções de código) a serem realizadas em conjunto com outros elementos da Critical Software, bem como de testes unitários realizados pelo estagiário.*
- ***T4*** *- Atendimento e suporte ao cliente final, participando nas atividades de apoio à aceitação e exploração do sistema.*
- ***T5*** *– Produção de um relatório de estágio*

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:



INI		Início dos trabalhos
M1	(INI + 12 Semanas)	Tarefa T1 terminada
M2	(INI + 24 Semanas)	Tarefa T2 terminada
M3	(INI + 32 Semanas)	Tarefa T3 terminada
M4	(INI + 36 Semanas)	Tarefa T4 terminada
M5	(INI + 46 Semanas)	Tarefa T5 terminada

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 relativo ao Estado da Arte terminado.

R1.2: Documento de Visão e Âmbito terminado.

M2:

R2.1: Documento de Especificação de Requisitos de Software terminado.

R2.2: Prototipagem terminada.

M3:

R3.1: Desenvolvimento terminado.

R3.2: Relatório de testes terminado.

R3.3: Manual de utilização terminado.

M4:

R4.1: Aceitação por parte do cliente.

M5:

R5.1: Relatório de estágio terminado.

Local de Trabalho

O estágio será executado nas instalações da empresa Critical Software em Coimbra.

Metodologia

Toda a documentação e comunicação neste projeto é feita em Inglês.

A metodologia de trabalho será AGILE, mais concretamente SCRUM, utilizando sprints com uma duração de 3 semanas.

Orientação

O aluno ficará inserido num projeto da CRITICAL, integrando a equipa iTGROW com acesso a todo o programa formativo, de acompanhamento e de avaliação de desempenho inerente a este programa.

ISEC:

Nome: Francisco Pereira (xico@isec.pt)

Orientador nomeado pela iTGROW enquanto entidade de acolhimento:

Nome: (nelson-f-vale@criticalsoftware.com)

Cargo: Engenheiro Sénior na Critical Software

Caracterização e Remuneração

- Data de início: 15-10-2015
- Data de fim: 30-09-2016 (esta data poderá se antecipada caso os trabalhos a desenvolver demorem menos tempo que o inicialmente planeado)
- Horário: 9h às 18h (podendo ser ajustado diretamente entre o estagiário e projeto, de acordo com os interesses de ambos)
- Tipo de regalias oferecidas: O formando terá uma remuneração equiparável a um engenheiro júnior / Trainee, incluindo subsídio de deslocação.
- Tipo de formação oferecida aos estagiários: Será integrado em todas as ações de formação aplicáveis em curso na empresa, sejam elas metodológicas, tecnológicas, linguísticas ou orientadas às competências comportamentais.

Sobre a iTGROW A.C.E e Critical Software S.A.

A iTGROW é uma academia de formação que funciona como porta de entrada de elementos juniores para a CRITICAL Software e o BPI. A missão da iTGROW passa por desenvolver talento em jovens que tenham decidido seguir uma carreira informática, através da sua exposição às melhores práticas e envolvimento em projetos exigentes, no domínio da engenharia de software. Para efetivar esta missão, quando chegam à iTGROW, os colaboradores são integrados em equipas de desenvolvimento de software da CRITICAL ou do BPI, frequentando simultaneamente um programa de formação profissional. Os projetos a que ficam alocados são reais, exigentes e ambiciosos, e configuram um excelente contexto para aprendizagem e evolução. Deste modo, os colaboradores podem fazer parte de equipas que trabalha em mercados tão distintos como Banca, Seguros, Governo, Saúde, Telecomunicações, Segurança, Espaço, Defesa, Transportes, etc.

Ao optar por esta proposta de estágio, o aluno terá oportunidade de se integrar numa equipa de engenharia da CRITICAL Software na área da Gestão Energética e de conhecer todas as práticas de desenvolvimento que fazem da CRITICAL uma referência na área de IT, ao mesmo tempo que frequenta um programa de formação profissional promovido pela iTGROW.

Os programas de tutoria da ITGROW baseiam-se na partilha de experiência de um engenheiro sénior com vários anos de experiência com os novos elementos que chegam à equipa, pelo que existe uma grande proximidade entre os elementos mais experientes dos projetos de engenharia e os mais juniores, que acabam de sair da universidade. O bom relacionamento com os colegas da equipa, a disponibilidade que todos mostram para ajudar, e a acessibilidade dos elementos mais seniores são, muitas vezes, destacados pelos colaboradores da iTGROW como pontos fortes da organização.