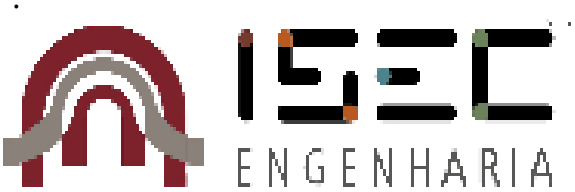


Departamento

de Engenharia Eletrotécnica

Plataforma Inteligente para Controlo de Acessos

Relatório de Projeto apresentado para a obtenção do grau de Mestre em Engenharia Eletrotécnica

Autor

Keven Alex Oliveira Lopes

Orientadores

Professora Fernanda Coutinho (Prof. Adjunto, DEE/ISEC)

Professor Luis Marques (Prof. Adjunto, DEE/ISEC)

Instituto Superior de Engenharia de Coimbra

Coimbra, Maio 2016

AGRADECIMENTOS

Agradeço a todos aqueles que me ajudaram a chegar onde cheguei. Sem esse apoio, nunca teria chegado aqui. Embora de formas diferentes, todos me ajudaram a concluir com sucesso mais esta etapa da minha vida.

Começando pelos meus orientadores - Professora Fernanda Coutinho e Professor Luís Marques, que durante todo este tempo sempre se mostraram disponíveis para me tirarem dúvidas e disponibilizarem-me os materiais necessários. Agradeço-lhes muito as sugestões para melhorar o trabalho, mas, sobretudo, o rigor e a exigência que sempre impuseram, levando-me sempre a melhorar em todos os aspetos e a superar-me a mim mesmo, conseguindo ultrapassar todos os obstáculos que encontrei ao longo deste percurso.

Agradeço ao Instituto Superior de Engenharia de Coimbra, em particular ao Departamento de Engenharia Eletrotécnica, onde desenvolvi este trabalho.

Agradeço a todos os meus amigos e colegas que sempre me influenciaram e ajudaram neste percurso académico - o Hélio, o Miguel e, de uma forma geral, o grupo dos meus colegas "*Maltas la d'boxe*" que muito me apoiaram nesta reta final. Agradeço àqueles que, nas fases mais difíceis, me apoiaram moralmente ajudando-me a ter força para prosseguir o trabalho.

Por fim, o meu agradecimento mais importante vai para a minha família - os meus irmãos, minha irmã, primos, primas, tios e tias. Quero dedicar um agradecimento muito especial aos meus pais, pelo enorme apoio e compreensão que sempre tiveram comigo não só ao longo destes meus anos académicos, mas em todos os vinte e três anos da minha vida, pois o que mais prezo é deixá-los orgulhosos.

Keven Alex Oliveira Lopes

RESUMO

Nos dias de hoje, a segurança e comodidade oferecidas pelos equipamentos são requisitos essenciais e decisivos no ato de compra. Com este trabalho, pretendeu-se criar uma solução para um controlador de acesso a uma garagem, que ofereça elevada segurança e comodidade para o utilizador, comparativamente a outras soluções existentes no mercado. O controlador mais comum é baseado num dispositivo de comando, com botões que permitem ao utilizador desencadear a abertura/fecho do portão. Esta solução apresenta algumas desvantagens em termos de segurança, dado que qualquer indivíduo com acesso ao dispositivo pode abrir a garagem. As desvantagens em termos de conveniência são significativas: o utilizador necessita de localizar e utilizar o comando sempre que precisa de entrar na garagem, o que desvia atenção da condução e pode mesmo obrigá-lo a parar o veículo, por outro lado, em caso extravio do comando, o utilizador vê-se impossibilitado de aceder à garagem. Por fim, não é possível estabelecer uma política de controlo de acesso avançada, dado que o acesso é dependente único e exclusivamente da posse física do comando.

O desenvolvimento deste trabalho apresentou vários desafios e abrangeu diferentes áreas de conhecimento, nomeadamente o desenvolvimento de protótipos de *hardware*, a sua programação e interligação através de redes de comunicação com e sem fios, a exploração das características do barramento *CAN* e investigar como seria possível obter em tempo real o código *VIN* de um veículo.

A possibilidade de controlar o acesso através de dispositivos tipo *smartphone* e de gerir os acessos de forma segura através de uma plataforma disponível *online* (*website*) foram também objetivos deste projeto e foram atingidos.

A solução desenvolvida, designada por *Gates Technology*, é representada por um logotipo que tem uma imagem de um *chip* (representa o microcontrolador e todos os módulos de eletrónica utilizados) com um símbolo de *wireless* (representa a conectividade à distância e acesso à *internet).*

Palavras-chave *– Garagens; Arduino; ESP8266; FTDI 232; Controller Area Network; NodeMCU; Controlo de Acessos; Base de Dados; Número de Identificação do Veículo.*

ABSTRACT

These days, equipment security and convenience are essential requirements, both being decisive factors in purchase decisions. This work aims to create a garage access solution, which offers higher security and user convenience than other solutions available on the market. Garage access systems are commonly based on a wireless control device with buttons that allow the user to open or close the garage gate. This approach has some disadvantages in terms of security, since anyone with physical access to the device can open the garage. Disadvantages in terms of convenience are also significant: the user needs to physically locate and operate the command when entering or leaving the garage. This distracts from driving efforts and may even force the user to immobilize the vehicle in front of the garage gate. If the command is misplace, lost, or stolen, the user may find himself unable to access the garage. Finally, it is not possible to establish an advanced access control policy, since access is only and exclusively dependent on the physical possession of the command.

Developing this work presents many challenges and requires expertise in different areas of knowledge, including the development of hardware prototypes, its programming and interconnection through wired and wireless communication networks, the exploitation of CAN bus characteristics to obtain, in real time, the VIN code of the vehicle.

Other goals of the project, that were also achieved, include the possibility of controlling access via Smartphone devices and also managing securely the access through an online platform (website).

This solution, called Gates Technology, is represented by a logo that has an image of a microchip (illustrating the microcontroller and all electronic modules used) with a wireless symbol (representing connectivity distance and *Internet* access).

Keywords *– Garages; Arduino; ESP8266; FTDI 232; Controller Area Network; NodeMCU; Access Control; Data Base; Vehicle Number Identification.*

“You see things; you say, 'Why?'
But I dream things that never were; and I say 'Why not? “
(George Bernard Shaw)

Gates Technology
“Just a ‘Smart’ different point of view”
(Keven Lopes)

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE SCRIPTS

ACRÓNIMOS

ACAP – Associação Automóvel de Portugal
AP – Access Point
Apps – Aplicações para *Smartphones*
Bps – bits per second
CAN – Controller Area Network
CMS – Content Management System
CTS – Clear to Send
DTR – Data Terminal Ready
DIRD – Department of Infrastructure and Regional Development
ER – Entidade Relacionamento
GPIO – General Purpose Input/Output
GSM – Global System for Mobile Communications
ID – Identificação
IDE – Integrated Development Environment
IP – Internet Protocol
ISEC – Instituto Superior de Engenharia de Coimbra
OBDII – On-board diagnostics
PWM – Pulse Width Modulation
SOF – Start of Frame
SSID – Service Set Identifier
TCP – Transmission Control Protocol
Tq – Time-quantum
VIN – Vehicle Identification Number
CSMA/CR – Carrier Sense Multiple Access / Collision Resolution

1. INTRODUÇÃO

Neste capítulo é dada uma visão geral sobre o tema deste trabalho, nomeadamente a motivação (Secção 1.1), o enquadramento (Secção 1.2) e quais os principais objetivos a alcançarem (Secção 1.3). Por último, a Secção 1.4 apresenta a estrutura do relatório.

1.1- Motivação

Nos dias de hoje deparamo-nos com uma sociedade cada vez mais dependente e interessada na tecnologia. Este interesse crescente motiva o desenvolvimento de produtos (ou o melhoramento de produtos já existentes) dotando-os das vantagens que se pode extrair da tecnologia.

De uma forma geral, as pessoas vivem o seu dia-a-dia a um ritmo acelerado, bombardeadas por múltiplas solicitações e as mais variadas preocupações nas vertentes profissionais, familiares ou pessoais. Por isso, têm mais com que se preocupar do que perderem o seu tempo à procura do comando para abrir a porta da garagem, quando no fim de um dia de trabalho regressam a casa, extenuados, ou, quando de manhã saem de casa apressados para deixarem os filhos na escola ou atrasados para irem trabalhar. Tendo isto em mente, surgiram-me as seguintes questões:

- E se eu tivesse uma garagem "inteligente" que não me obrigasse a ter um comando ou chave para lá entrar/sair?
- E se o comando (ou a chave) de abertura da garagem pudesse ser virtual? E ser desencadeada a partir de um *smartphone* ou *Tablet*?
- E se o sistema de identificação do meu carro pelo controlador da garagem fosse inteligente e automático, sem eu ter de fazer nada, apenas aproximar-me com o meu carro da porta da garagem?

A resposta a estas questões foi a motivação principal que esteve por detrás do desenvolvimento deste trabalho.

1.2- Enquadramento

Nesta secção aborda-se o crescimento do mercado das telecomunicações, em particular dos telemóveis e da indústria automóvel.
Segundo estudos realizados pela Emarketer[1], prevê-se que em 2016 cerca de um quarto da população mundial esteja a utilizar *smartphones*. Prevê-se que esse número cresça para mais de 2 bilhões de utilizadores, isto é, cerca de um terço da população mundial. E prevê-se também que o número de *smartphones* ultrapasse o de "*feature* phones" (os celulares mais simples com acesso à *internet*). De acordo com o jornal "*Telegraph*"

[1] www.emarketer.com

(Anexo 5), muito deste crescimento deve-se à China onde já existem mais de 500 milhões de utilizadores de *smartphones*. Este número deverá subir para mais de 700 milhões em 2018. Segundo o barómetro de Telecomunicações da *Marktest*, a nível nacional cerca de 60% dos portugueses utilizam *smartphones.* E, desde dezembro de 2012, a utilização deste tipo de aparelhos aumentou em 83%. Há estudos que revelam que atualmente há um carro por cada 7 pessoas, à escala mundial. Automóveis, comerciais ligeiros e veículos pesados já somam mais de um bilião em todo o mundo. Esse número exclui tratores, máquinas de obras, motocicletas e outros veículos de uso industrial.
No ano de 2015, o mercado automóvel em Portugal voltou a acelerar. A Associação Automóvel de Portugal (ACAP) prevê que o ano termine com as vendas de carros novos a chegarem perto dos 210 mil veículos de passageiros. Entre janeiro e junho, o mercado de veículos ligeiros cresceu 31%, para 114.938 unidades, face a igual período de 2014. Em termos acumulados, acrescidos de veículos pesados, no primeiro semestre do ano foram vendidos em Portugal 116.813 veículos automóveis, ou seja, mais 31,2% do que no ano anterior. Todos os dados anteriormente referidos, podem ser consultados no Anexo 5.

Face ao exposto, concluímos que tanto a nível nacional como também mundial, existe um grande mercado para se fazerem estudos e se desenvolverem novos produtos que conjuguem tecnologia e automóveis, tendo em vista facilitar e dar mais qualidade de vida ao cidadão comum. Deste modo, acreditamos que o produto desenvolvido neste projeto tem potencial para um dia ser comercializado com sucesso.

1.3- Objetivos

O objetivo principal deste trabalho consiste no desenvolvimento de uma plataforma inteligente de acesso a uma garagem com as seguintes funcionalidades:

- Acesso à garagem através do ***VIN* (*Vehicle Identification Number*)**
  O dispositivo permite a comunicação *wi-fi* entre o carro e a garagem. Quando este se aproxima da garagem, o identificador *VIN* do carro (obtido através do barramento *CAN*) é solicitado pelo controlador da garagem. Se for validado, então a porta abrirá automaticamente. De realçar que este procedimento é automático, sem ser necessário ao utilizador usar um comando de acesso ou sair do carro para abrir a porta.

- Acesso à garagem através da **aplicação *Android***
  O acesso ao controlador da garagem pode também ser feito através de uma aplicação *Android* que comunica com o controlador instalado na garagem.

- Acesso à **plataforma *online (website)***
  Existe também uma plataforma *online*, alojada numa *cloud*, à qual o utilizador pode aceder, fazendo o *login* na sua conta, alterar ou atualizar os seus dados, inserir amigos ou convidados (com autorização temporária) para acederem à garagem, entre outros. O administrador tem acesso a todos os registos,

nomeadamente para os editar, apagar ou adicionar novos utilizadores. Pode também consultar o histórico dos acessos (identificador, dia e hora).

- **Segurança**
  Garantir sempre que a informação transmitida entre o utilizador e o controlador da garagem seja encriptada.

- **Tolerância a falhas**
  O controlador da garagem tem localmente armazenado uma cópia dos códigos de acesso para assim prevenir eventuais situações em que controlador não consiga aceder à *cloud,* evitando assim que o utilizador fique impossibilitado de utilizar a garagem.

1.4- Estrutura do Relatório Projeto

O presente relatório encontra-se estruturalmente dividido em oito capítulos, listagem das referências bibliográficas e cinco anexos, cujos conteúdos são os que se seguem:

- **Capítulo 1**: Introdução – é o capítulo atual, onde se faz um breve enquadramento do tema do trabalho e definem-se os objetivos do mesmo.
- **Capítulo 2:** Estado da Arte – apresenta o estado da arte relativamente às soluções comerciais existentes no mercado de controlo de acesso a garagens. É feita uma análise comparativa e explica-se em que aspetos a nossa solução se distingue das existentes.
- **Capítulo 3:** Tecnologias Utilizadas – onde é apresentada algum do trabalho de pesquisa mais relevante para o trabalho a ser desenvolvido, apresenta um breve resumo dos diferentes módulos que serão utilizados e também explica, de uma forma simples, diferentes tópicos para uma melhor compreensão do trabalho.
- **Capítulo 4:** Base de Dados – neste capítulo apresenta-se a organização, estruturação e criação da base de dados. Explica como foi criada a base de dados que será utilizada pela *Gates Technology*, nomeadamente as ligações entre as tabelas.
- **Capítulo 5:** Plataforma *Online*/*Website* – este é o capítulo onde se descreve a criação da plataforma *online* para o controlo da garagem, registos de novos clientes, ou seja, controlo total sobre a base de dados. Apresentam-se todos os aspetos mais importantes sobre o *website*[2].
- **Capítulo 6:** Aplicação *Android* – este é a secção que se explica o processo de criação da aplicação *Android* que será utilizado pelo *Gates Technology*. Refere-se o funcionamento, a constituição e também os diferentes mapas de navegação.
- **Capítulo 7:** Dispositivo *Gates Technology* – neste capítulo é feito uma descrição da constituição do *setup* experimental, referindo-se os diferentes módulos de

[2] http://www.gatestechnology.eu

comunicação utilizados, apresentando-se também os testes experimentais feitos e os seus resultados.

- **Capítulo 8:** Conclusões e Trabalho Futuro – neste capítulo é apresentado um breve resumo do trabalho, os conhecimentos adquiridos durante a sua realização, todos os obstáculos ultrapassados terminando com a indicação de algumas perspetivas de trabalho futuro.
- **Anexos:** onde estão todos os anexos relativos ao projeto, nomeadamente: Anexo da Base de Dados, Anexo da Plataforma *Online/Website*, Anexo da Aplicação *Android*, Anexo do Dispositivo *Gates Technology* e Anexo com os sítios da *internet* consultados, onde especificamos todos os *links* que podem ser consultados para uma melhor compreensão do projeto e dos diferentes materiais utilizados na criação do mesmo.

2. ESTADO DA ARTE

Neste capítulo é feita uma apresentação e análise comparativa dos principais produtos que existem atualmente no mercado de acesso a garagens (Secção 2.1). Na Secção 2.2 é explicado de que forma a nossa solução se diferencia das já existentes.

2.1- Soluções Comerciais

Após algum trabalho de pesquisa, decidimos eleger os seguintes cinco produtos comerciais para análise, por serem aqueles que mais presenças têm no mercado:

a) Comandos Tradicionais
b) GogoGate
c) GateControle
d) Aplicação móvel - controle o seu portão.

Da Secção 2.1.1 à Secção 2.1.4 vão ser apresentados resumidamente cada um destes produtos.

2.1-1. Comando Tradicional

Os comandos tradicionais dominam o mercado do acesso aos portões das garagens. A Empresa Hörmann (Figura 2-1) lidera este mercado na Europa. Estes produtos comunicam através do comando da BiSecur (Anexo 5 [w55]). Vêm equipados com um interruptor e iluminação. Podem-se encontrar diferentes comandos da BiSecur, nomeadamente: o emissor HS1BS, HS2BS e HS4BS, onde o dígito numérico representa o número de teclas que tem cada comando. Existem também os modelos HSD2ABS, o HSD 2CBS, o HSP4BS, o teclado codificado por radiofrequência FCT10BS e o leitor de impressão digital por radiofrequência FFL12BS. Os preços de alguns destes modelos estão representados na tabela seguinte (à data de abril de 2015):

Figura 2-1: Logotipo da Hörmann.

Identificador do Comando	**Preço**
Comando Tradicional	25€
HS1BS	40€
HS4BS	65€
FCT10BS	111€
FCT10BS	115€
FFL12BS	149€

Tabela 2-1: Preços dos diferentes tipos de comandos.

2.1-2. GogoGate

O produto GogoGate (Anexo 5 [w2]) é um sistema de controlo sem fios que utiliza uma aplicação *Android* e pode funcionar a partir de um *smartphone, tablet* ou computador.

Para além de necessitar de um destes três dispositivos, também é necessário instalar o *kit* de instalação na garagem (Figura 2-2) que inclui instalar o sensor por cima do portão da garagem.

Figura 2-2: *kit* de instalação do GogoGate.

A aplicação está disponível tanto para *Android* como também para sistemas *IOS*. O *kit* de instalação do GogoGate, encontra-se disponível em diversas plataformas de vendas *online*, como por exemplo na Amazon, Ebay e outros *websites* a um preço que varia entre 150 € a 200 €.

2.1-3. GateControle

O GateControle (Anexo 5) é um produto desenvolvido para o controlo dos portões de garagem. Funciona com base na comunicação *GSM* (Sistema Global para Comunicação) - uma tecnologia móvel e o padrão mais utilizado nos telemóveis de todo o mundo. O GateControle, quando adicionado ao automatismo presente no portão da garagem, permite efetuar o controle do mesmo a partir do telemóvel sem qualquer custo. Para controlar o portão, basta efetuar uma chamada telefónica para o número do aparelho instalado na garagem. Tem a possibilidade de adicionar até 1000 utilizadores. O *kit* de instalação vem com os seguintes equipamentos: Gate Controle (Figura 2-3), Antena *GSM*, cabo *USB*, adaptador *AC/DC*, *CD* com o *Software* e manual de utilizador. Este dispositivo está disponível apenas do *website* "*www.ebay.com*" com um custo de aproximadamente 130 €.

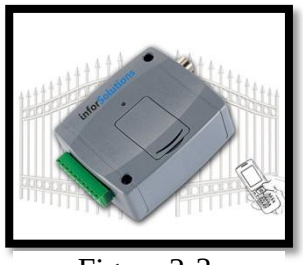


Figura 2-3: Ilustração do GateControle.

2.1-4. Aplicação "Controle seu Portão"

A aplicação designada por "*Controle seu Portão*" foi desenvolvida pela HPS Tecnologia Ltda., do Brasil. Os comentários que se encontram nas redes sociais sobre este produto não são muito positivos, tendo uma classificação de 2.2 numa escala de zero a cinco. Embora seja publicitado que a aplicação é gratuita, os utilizadores pagam quando adquirem a aplicação!

Figura 2-4: Imagem apresentada no Google Play para a aplicação

Para se utilizar este sistema, é preciso adquirir um módulo para ser instalado no automatismo já existente na garagem. O módulo pode ser para versão *Bluetooth* ou *Ethernet*. Pode ser utilizado em conjunto com o controlo tradicional ou sem ele. Cada módulo tem a capacidade de gravar um número elevado de utilizadores, sendo que a abertura e o fecho do portão da garagem são feitos através da *internet*.

2.1-5. Discussão

A Tabela *2-2* apresenta uma análise comparativa entre os vários produtos apresentados anteriormente.

	Wireless	GSM	Online	Offline	Seguro	Configuração	Aplicacão	Website	Sem Comando	Preço
Comando Tradicional	X			X	X	X				Só Comando 25 €
GogoGate	X			X	X	X	X			150 €
GateControle	X	X		X	X	X				130 €
Controle seu Portão	X		X			X	X		X	Grátis

Tabela 2-2: Tabela comparativa dos produtos existentes no mercado.

- ***Wireless*** - Todos os 4 produtos partilham o facto de terem comunicação *wireless*.
- ***GSM*** - No caso do GateControle, o comando de acesso à garagem utiliza comunicação por *GSM* (chamada telefónica para um número associado ao dispositivo da garagem). Esta é considerada uma desvantagem, pois obriga a que o utilizador tenha saldo no telefone.
- ***Online*** - Somente está presente para a aplicação móvel "Controle seu Portão". A aplicação só funciona se tiver conexão à *Internet*. Caso esteja *offline*, já não é possível enviar os comandos para o portão da garagem.
- ***Offline*** – Excetuando a aplicação "Controle seu Portão", todos os restantes produtos não utilizam a *Internet* para fazer as comunicações.
- **Segurança** - A aplicação móvel "Controle seu Portão" não tem muita segurança quando comparada com a dos restantes dos produtos o que se justifica por ainda estar numa fase inicial de lançamento.
- **Configuração** - O funcionamento de qualquer um desses produtos obriga a que seja feita uma configuração prévia do mesmo. Tais configurações podem representar uma desvantagem na medida em que obriga a que as pessoas tenham alguns conhecimentos técnicos.
- **Aplicação** – Somente o "Controle seu Portão" e o "GogoGate" vêm acompanhados de uma aplicação móvel para as plataformas *Android* e *IOS* (*Iphones*), que representa uma vantagem.
- ***Website*** - Nenhum destes produtos possibilita que o controlo seja feito utilizando uma plataforma *online*.
- **Comando** – Refere-se aos produtos que funcionam sem a necessidade de um comando. Somente a aplicação *Android* não exige comando.

2.2- Solução Proposta e Comparação com as Existentes

Depois de enumerar as vantagens e as desvantagens dos produtos existentes, isso ajudou-nos a identificar melhor quais as características da solução que pretendíamos desenvolver. À nossa solução demos o nome de ***Gates Technology***.

O nosso objetivo foi o de conseguir uma solução que tire partido das tecnologias existentes e que prescinda da necessidade de telecomando.

Nas duas figuras seguintes apresentam-se os diagramas que representam o ***Gates Technology*** e o respetivo funcionamento.

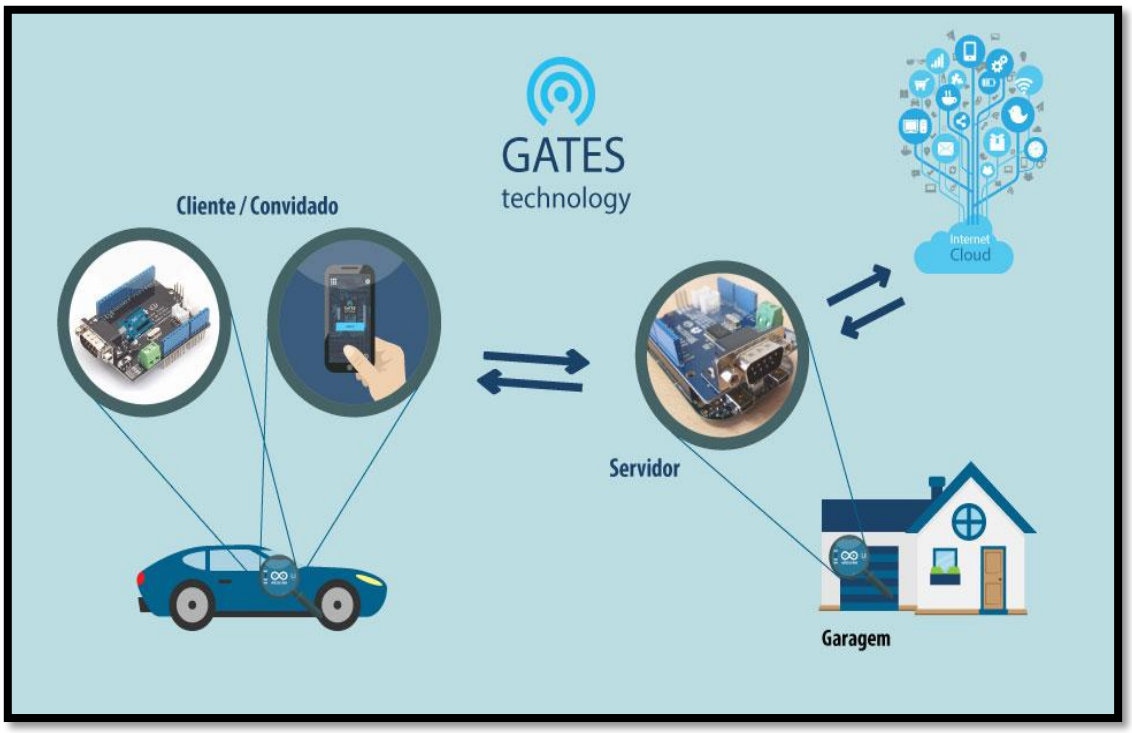


Figura 2-5: ***Gates Technology*** – visão geral.

A Figura *2-5* permite dar uma visão geral da forma como é feito o acesso do carro à garagem. Há um dispositivo instalado no carro do cliente o qual comunica com o servidor da garagem, por *wireless*, podendo ir até uma distância de aproximadamente 200 metros. O servidor (da garagem) tem a função de verificar se o carro está ou não registado na sua base de dados. A base de dados está guardada na *cloud* (*online*) como representado no diagrama. Se o número de identificação enviado pelo carro for validado pelo servidor, a porta deverá abrir.

A Figura *2-5* apresenta ainda o mecanismo de acesso à garagem por um convidado, isto é, por um "utilizador" com autorização temporária. Para este caso, é utilizada a aplicação para o *smartphone* desenvolvida para a plataforma *Android.* A aplicação vai comunicar diretamente com o servidor, depois de efetuar o início da sessão com os dados fornecidos pelo "cliente responsável" pelo convidado. A aplicação *Android* é uma ferramenta extra de acesso que pode também ser utilizada pelos clientes habituais.

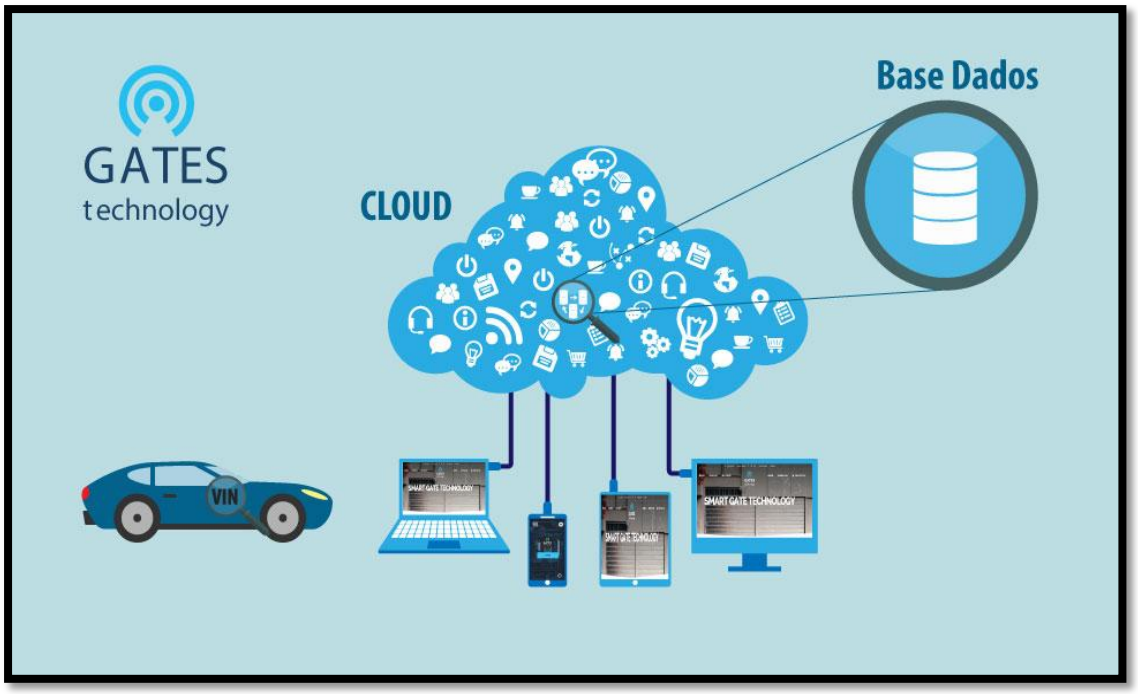


Figura 2-6: *Gates Technology* – Registo de informação.

A Figura *2-6* mostra o processo manual de registo de um cliente, nomeadamente do registo do *VIN*. Este processo pode ser feito na plataforma *online (website)* por qualquer dispositivo com conexão à *internet*.

Comparação do *Gates Technology* com os outros produtos já existentes:

- O acesso à garagem é possível apenas com a aproximação do carro à garagem. Esta característica não é fornecida por nenhum outro produto presente no mercado.
- Através da aplicação, é possível ao gestor da garagem controlar e saber todos os eventos da garagem, nomeadamente, ver quem saiu/entrou assim como saber a informação associada aos carros registados na base de dados. Existe ainda a possibilidade de o administrador receber uma mensagem *SMS* sempre que a garagem for aberta/fechada utilizando a aplicação. A aplicação fornece ainda um espaço de lazer em que se pode jogar ouvir músicas ou até ver notícias. De realçar que nenhuma das soluções existentes oferece todas estas possibilidades.
- O *website* desenvolvido pode ser utilizado pelo cliente e/ou administrador para controlar diretamente a garagem, adicionar convidados, registar veículos ou até mesmo consultar os eventos realizados pelos convidados. Quando comparado com outros produtos, podemos ver que nenhum disponibiliza estas funcionalidades através do *website*, pois esses outros produtos não disponibilizam uma base de dados para o servidor da garagem.

- Não é possível comparar, sob a perspetiva de preço, a nossa solução com as existentes no mercado visto a nossa não ser (ainda) comercializada.

A Tabela *2-3* apresenta uma análise comparativa similar à apresentada anteriormente, mas desta vez incluímos as características da nossa solução *Gates Technology*.

	Wireless	GSM	Online	Offline	Seguro	Configuração	Aplicacão	Website	Sem Comando	Preço
Comando Tradicional	X			X	X	X				Só comando 25 €
GogoGate	X			X	X	X	X			150 €
GateControle	X	X		X	X	X				130 €
Controle seu Portão	X		X			X	X		X	Grátis
Gates Technology	X		X	X	X		X	X	X	N/A

Tabela 2-3: Tabela comparativa incluindo o *Gates Technology*.

Como plasmado na Tabela *2-3,* as características da solução que propomos permite, por um lado facilitar a utilização da garagem e, por outro, tira partido das vantagens oferecidas pela tecnologia. Em súmula, o *Gates Technology*:

- ***Wireless*** - suporta comunicação *wireless*.
- ***GSM*** - não utiliza comunicação *GSM*.
- ***Online/Offline*** - é possível aceder à garagem com ou sem conexão á *internet*.
- **Segurança** – a nossa preocupação com a segurança está presente na base de dados, na plataforma *online*, na aplicação *Android* e principalmente no dispositivo instalado no carro do cliente que comunica diretamente com a garagem.
- **Configuração** – não precisa de nenhuma configuração prévia do produto, no entanto tudo é feito no momento da instalação do serviço.
- **Aplicação** – tem uma aplicação *Android* com diversas opções de navegação. Permite abrir ou fechar o portão da garagem e consultar informação diretamente na base de dados.
- ***Website*** – é possível aceder ao site como cliente ou como administrador. Permite controlar diretamente a garagem, consultar os eventos da garagem, adicionar ou eliminar registo de carros, inserir convidados, entre outras funcionalidades.
- ***Comando –*** não utiliza um comando físico.

3. TECNOLOGIAS UTILIZADAS

Descreve-se agora de uma forma geral todos os diferentes conteúdos utilizados na criação do *Gates Technology*, dá uma visão geral sobre a rede *CAN*, dos *softwares* utilizados e também fala-se um pouco sobre os diferentes dispositivos utilizados em conjunto para obter no final do trabalho o resultado esperado.

3.1- Protocolo *CAN*

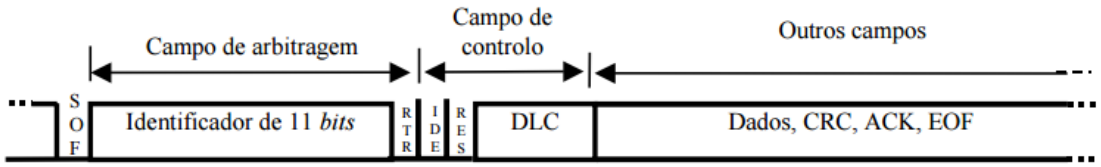
3.1-1. Conceito Geral

Apresentado em 1986 por Robert Bosch GmbH, o protocolo *CAN* (*Controller Area Network*) é um protocolo de comunicação série muito popular na implementação de sistemas de tempo real distribuídos, especialmente na indústria automóvel e na automação industrial. A sua popularidade surge de um dos seus pontos fortes, a flexibilidade, que permite a sua utilização quer em sistemas que exigem largura de banda considerável, quer em sistemas em que as baixas latências são um elemento fulcral. Outro ponto forte é a simplicidade do barramento e das ligações físicas, que a juntar ao seu baixo custo de implementação, o tornam um protocolo muito atraente do ponto de vista custo-benefício (Doecke et al, 2015), (Fan et al, 2014), (Huebner et al, 2006). Atualmente, na indústria automóvel, o protocolo *CAN* é utilizado para interligar sensores, atuadores, módulos de comando, entre outros. Anteriormente estes cablagens mais complexas, podendo agora tirar partido da simplicidade deste tipo de comunicação, e podendo atingir velocidades até 1 Mbit/s, na especificação atual (2.0). Normas europeias exigem a sua implementação nos automóveis fabricados a partir de 2008. A especificação 2.0 deste protocolo divide-se em 2 partes: 2.0A e 2.0B. A Tabela 3-1, indica uma pequena comparação feita para as diferentes versões da rede *CAN*. Comparação essa baseada no padrão (*Standard*), na taxa máxima de comunicação e no identificador da *frame* (Guimarães Barbosa, 2003).

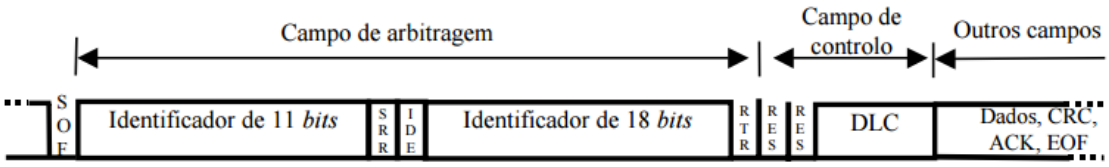
Nomenclatura	**Norma**	**Velocidade máxima de Transmissão**	**Identificador**
CAN Baixa-velocidade	*ISO* 11519	125 Kbit/s	11 *Bits*
Versão 2.0A	*ISO* 11898:1992	1 Mbps	11 *Bits*
Versão 2.0B	*ISO* 11898:1995	1 Mbps	29 *Bits*

Tabela 3-1: Tabela comparativa para as diferentes versões da rede *CAN*.

O formato das *frames*/tramas de dados em *CAN* encontra-se representado na Figura *3-1* O primeiro *bit* é designado por *SOF* (*start of frame*), e destina-se a sincronizar a transmissão (início do processo de arbitragem). Seguem-se os *bits* do identificador – 11 *bits* no formato *Standard* (*CAN* versão 2.0A) e 29 *bits* para versão *Extended* (*CAN* versão 2.0B). Os três *bits* seguintes são *bits* de controlo (*RTR*, *SRR* e *IDE*), que ainda fazem parte do processo de arbitragem no acesso ao barramento/*bus*. É o *bit IDE* que define quando se trata de uma *frame Standard* ou *Extended* e o *bit RTR* indica se é uma *frame* de Dados ou Remota. Na Figura *3-1* –a) podemos ver o formato *Extended*, onde o *bit IDE* é colocado a 1, e a seguir a este encontram-se os 18 *bits* restantes do identificador, (Durbin & Norbeck, 2002).



(a) Formato duma *frame* de dados segundo a versão 2.0B (Formato *standard*).



(b) Formato duma *frame* de dados segundo a versão 2.0B (Formato estendido).

Figura 3-1: Formato dos diferentes tipos de *frames*, Fonte: (Anexo 5[3] [w9]).

Transferência de Mensagens

Conforme referido anteriormente, as *frames* do protocolo *CAN* possuem dois formatos que diferem no tamanho do identificador:

- *Frames Standard*, cujo identificador possui 11 *bits*;
- *Frames Extended*, cujo identificador possui 29 *bits*.

A transmissão e receção de informação, numa comunicação presente num sistema *CAN* são efetuadas e controladas através de quatro tipos diferentes de *frames*:

- *Frame* de Dados (*Data frame*) – veicula informação no campo de dados, podendo ser transmitidos entre 0 e 8 bytes por *frame*;
- *Frame* Remota (*Remote frame*) – pedido de envio de uma *frame* de Dados com ID especificado;
- *Frame* de Erro (*Error frame*) – sinalização de erros;
- *Frame* de Sobrecarga (*Overload frame*) – solicita tempo de espera antes do envio da próxima *frame*.

[3] ANEXO 5 – Sítios da Internet Relevantes

A Figura *3-2* representa a constituição de uma *frame* de Dados (*CAN* 2.0A). Nesta o *Message Identifier* de 11 *bits* é o identificador único da mensagem. O *payload (Data Field)* contém a informação enviada em cada mensagem, sendo no máximo de 8 *bytes*.

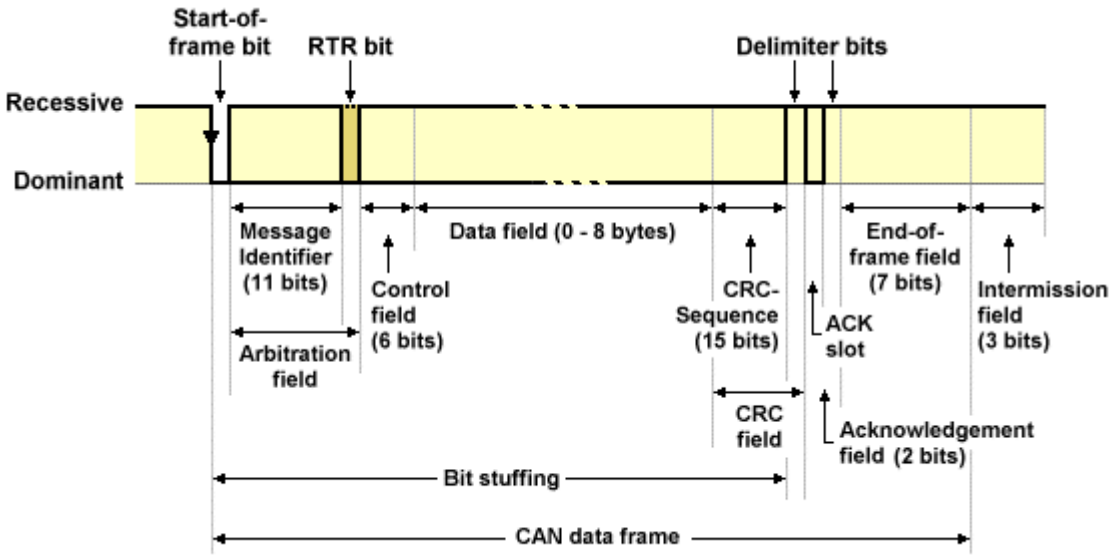


Figura 3-2: Campos de uma mensagem da rede *CAN bus*, Fonte: (Anexo 5[4] [w9]).

Timing* e tempo de *bit: Na rede *CAN*, cada nó utiliza um oscilador local com uma frequência pré-programada, gerado por um oscilador. *Time-quantum* (*Tq*) é o nome adotado para definir, cada conjunto de n períodos dessa mesma frequência, em que o tempo de transmissão de cada *bit (bit-time*) é nesse caso definido como múltiplo de *Tq*. O *bit-time* subdivide-se em quatro segmentos distintos, em que cada um deles tem a duração múltipla de *Tq*. A composição desses segmentos é tida como: segmento de sincronização com duração de 1 *Tq*, um segmento de propagação com 1 a 8 *Tq* e por último dois segmentos de fase com 1 a 8 *Tq* cada (Figura *3-3*), (Venda, NA).

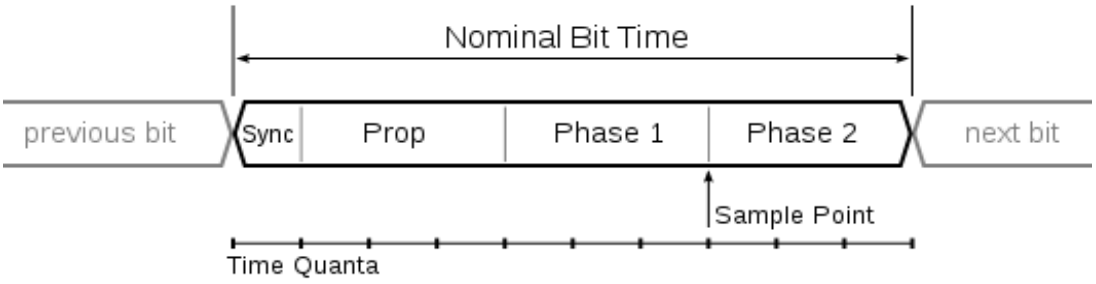


Figura 3-3: Esquema do *timing* de um *bit*, Fonte: (Anexo 5 [w11,12]).

[4] ANEXO 5 – Sítios da Internet Relevantes

Acesso ao meio: o método de acesso ao meio é determinístico, sendo utilizada a técnica designado por *CSMA/CR (CSMA/Collision Resolution*), onde cada *bit* possui o nível dominante (0 lógico) ou recessivo (1 lógico). O processo de arbitragem é ganho pelo nó que estiver a tentar transmitir a mensagem com *ID* mais baixo. Devido ao mecanismo de arbitragem utilizado, a relação comprimento máximo do *bus*/barramento função da velocidade de transmissão (ou vice-versa) é limitada. Na Tabela *3-2* pode-se verificar que uma rede *CAN* pode estender-se por alguns Km, à custa de uma velocidade de transmissão baixa e que para a velocidade máxima de 1 *Mbps* o comprimento do *bus* está limitado a 40 metros.

Bit rate **(Kbps)**	**Tempo de *bit*** **(µs)**	**Comprimento máximo do *bus*** **(m)**
20	50	2500
50	20	1000
100	10	500
250	4	200
500	2	100
1000	1	40

Tabela 3-2: Relação *bit-rate* / comprimento máximo do *bus*.

Bus: o meio físico de transmissão é um par entrançado, sendo as linhas designadas por CAN_H e CAN_L, onde o sinal elétrico lido pelo *transceiver* é o sinal diferencial entre estes. Para minimizar as reflexões o *bus* deve ser terminado pela impedância característica da linha, tal como se representa na Figura *3-4*.

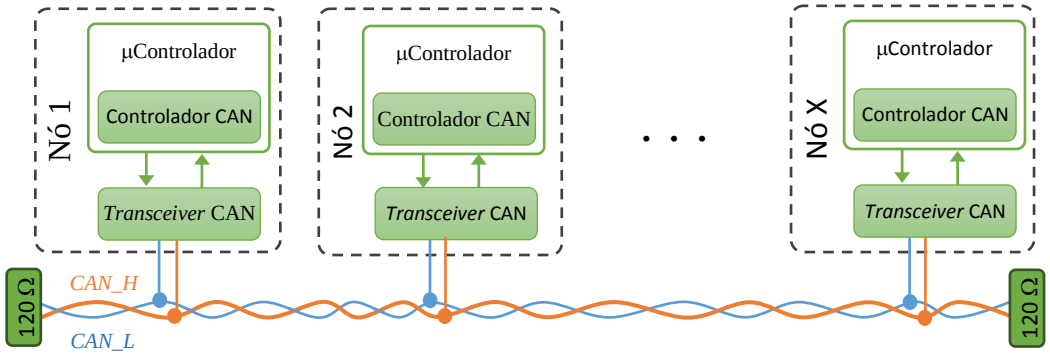


Figura 3-4: Rede *CAN*.

3.1-2. Conector *OBD* II

No carro a saída designada *OBD* II é o único ponto de acesso para a comunicação com a rede *CAN*, pelo que se torna necessário descrevê-la. A camada de aplicação da comunicação via interface *OBD* II é definida na norma ISO15031 que especifica os aspetos comunicação entre o veículo e o equipamento externo de diagnóstico, em termos de interpretação de dados e tempos de resposta. Esta norma especifica também termos e definições que são utilizadas na comunicação – baseada em troca de mensagens – via *OBD* II, (Woody et al, 2007).

Durante vários anos, os conectores utilizados nos veículos para comunicação *OBD* assumiram várias formas, algumas específicas para cada fabricante. No entanto, com toda a normalização que se tem verificado nos últimos anos, o conector passou a ser também especificado. Esta norma define, portanto que o conector utilizado num veículo compatível com a norma *OBD* II deverá ser conector o SAE J1962. Cada protocolo de comunicação legislado utiliza pinos específicos no conector SAE J1962 (Morais Pires, 2005), (Nunes Baldissera, 2011).

A versão do conector presente nos veículos é a versão fêmea representada na Figura *3-6* à esquerda. Nos equipamentos externos de diagnóstico é utilizado o conector macho representado na Figura *3-6* à direita.

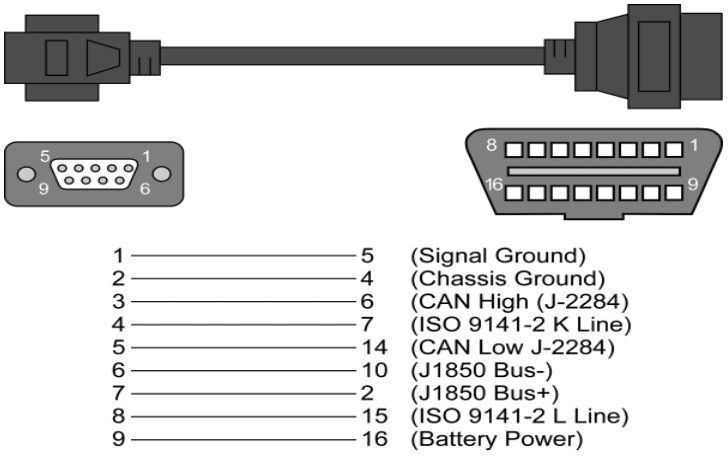


Figura 3-5: Conector SAE J1962 legendado.

Figura 3-6: Conector SAE J1962 fêmea (Esquerda), Conector SAE J1962 macho (Direita).

3.1-3. Número de Identificação do Veículo

"O *VIN* - *Vehicle Identification Number* ou em português NIV (Número de Identificação do Veículo) é um identificador único e universal de registo de um veículo automóvel produzido. A combinação de letras e números torna cada veículo único, visando promover um registo individual que servirá para diversos fins, entre eles codificação de dados do bem, identificação deste em circunstâncias diversas, como acidente, furto e transações envolvendo compra e venda.".

A Figura *3-7* indica diferentes lugares onde uma pessoa pode localizar o *VIN*. No interior na frente é o lugar mais comum para encontrar o número de identificação do carro. Também pode ser encontrado por baixo do capô em frente ao motor ou então perto da roda da frente do lado do condutor, no para-lamas. Há ainda locais adicionais, como por exemplo ao pé da mala por cima da roda, junto à roda de traseira ou então por dentro da porta do lado do condutor, (Fan et al, 2014), sítios da *internet* relevantes no Anexo 5 [78].

Figura 3-7: Diferentes lugares onde pode encontrar o *VIN* e exemplo de uma placa de um veículo[5].

Definição do *VIN*

Existem quatro formatos *standard* utilizados para especificar o *VIN*, que se encontram definidos em:

- FMVSS 115;
- ISO Standard 3779 (usado na Europa e em muitos outros países do mundo);
- SAE J853 (muito semelhante ao standard da ISO);
- ADR 61/2 (usado na Austrália, com referências às normas ISO 3779 e 3780).

Por exemplo, os campos do *VIN* definidos no *standard* ISO3779 encontram-se representados na Tabela *3-3*, sítios consultados em Anexo 5.

[5] http://www.dmv.org/vehicle-history/find-vin.php

Standard	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ISO 3779	World manufacturer identifier			VDS						VIS							
European Union[7] more than 500 vehicles/year	World manufacturer identifier			Indication of "*the general characteristics of the vehicle*"						Indication which provide "*clear identification of a particular vehicle*"							
European Union[7] fewer than 500 vehicles/year	World manufacturer identifier		9	Indication of "*the general characteristics of the vehicle*"						Indication which provide "*clear identification of a particular vehicle*"							
North America more than 500 vehicles/year	World manufacturer identifier			Vehicle Attributes					Check Digit	Model Year	Plant Code	Sequential Number					
North America fewer than 500 vehicles/year	World manufacturer identifier		9	Vehicle Attributes					Check Digit	Model Year	Plant Code	Manufacturer Identifier			Sequential Number		

Tabela 3-3: Campos do *VIN*, de acordo com norma ISO 3779, Fonte: Anexo 5 [9,10,11].

Cálculo do dígito de verificação (*Check digit*)

A Tabela *3-4* define a codificação letra/número de modo a que cada letra corresponda a um número (entre 1 e 9) que é utilizado para cálculo do dígito de verificação do *VIN*. A Tabela *3-5* atribui um "peso" a cada posição, que será utilizada como auxiliar no cálculo do *Check digit*.

A: 1	**B**: 2	**C**: 3	**D**: 4	**E**: 5	**F**: 6	**G**: 7	**H**: 8	N/A
J: 1	**K**: 2	**L**: 3	**M**: 4	**N**: 5	N/A	**P**: 7	N/A	**R**: 9
	S: 2	**T**: 3	**U**: 4	**V**: 5	**W**: 6	**X**: 7	**Y**: 8	**Z**: 9

Tabela 3-4: *Transliteration key*, valores para o *VIN Decoding*, Fonte: (Anexo 5 [9, 10]).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Weight	8	7	6	5	4	3	2	10	0	9	8	7	6	5	4	3	2

Tabela 3-5: Relacionar a posição com um "peso" para o cálculo.

VIN	1	M	8	G	D	M	9	A		K	P	0	4	2	7	8	8
Value	1	4	8	7	4	4	9	1	0	2	7	0	4	2	7	8	8
Weight	8	7	6	5	4	3	2	10	0	9	8	7	6	5	4	3	2
Products	8	28	48	35	16	12	18	10	0	18	56	0	24	10	28	24	16

Tabela 3-6: Exemplo de *VIN*, Fonte: (Anexo 5).

Vamos considerar o exemplo do *VIN* "**1M8GDM9A_KP042788**" (Tabela *3-6*) onde o caracter *underscore* corresponde ao dígito de verificação (a calcular). Este é obtido da seguinte forma:

- Primeiro faz-se a conversão das letras nos seus respetivos números usando a Tabela *3-4*;
- O "peso" (*weight*) é obtido da Tabela *3-5*;

- A coluna do produto é obtida através da multiplicação do "*Value*" com o "*Weight*";
- Os valores na linha "*products*" são todos somados, obtendo-se uma soma igual a 351;
- Obtendo o resto da divisão deste número por 11 encontramos o valor do *check digit*, neste caso 10, ao que corresponde a letra X na posição 9 do *VIN*;

  O *VIN* é então "**1M8GDM9AXKP042788".**

A rede *CAN* é constituída por 10 diferentes modos de operação. O *VIN* é obtido usando o *Mode 09* de operação, onde é a parte que contem diversas informações sobre o veículo, respetivamente o *VIN*. Os modos encontram-se listados na Tabela *3-7*.

Rede CAN	Descrição
Mode 01(hex)	Show current data
Mode 02(hex)	Show freeze frame data
Mode 03(hex)	Show stored Diagnostic Trouble Codes
Mode 04(hex)	Clear Diagnostic Trouble Codes and stored values
Mode 05(hex)	Test results, oxygen sensor monitoring (non CAN only)
Mode 06(hex)	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
Mode 07(hex)	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
Mode 08(hex)	Control operation of on-board component/system
Mode 09(hex)	Request vehicle information
Mode 0A(hex)	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

Tabela 3-7: Constituição da Rede *CAN*, Fonte: (Anexo 5 [11]).

3.2- Arduino

O *Arduino* (Figura *3-8 Arduino UNO*) foi projetado com a finalidade de ser de fácil entendimento, de fácil programação, e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambiente Linux, Mac OS e Windows. Além disso, algo muito importante também deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *Open-Source*, desenvolvendo e divulgando gratuitamente os seus projetos. O equipamento tem uma composição física de sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas. Ele é baseado numa placa do tipo microcontrolador, com acessos de Entrada/Saída (*I/O*), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar à das linguagens C e C++. O *Arduino* (Baichtal, John, 2014) utiliza o microcontrolador *Atmega*. Um microcontrolador (também denominado *MCU*) é um computador num *chip*, que contém um microprocessador, memória e periféricos de entrada/saída. Incluído no interior de algum outro dispositivo, que, neste caso, é o *Arduino*, para que possa controlar suas funções ou ações, (Baichtal, 2014), (Devarakonda et al, 2015), (Buchanan, 1997).

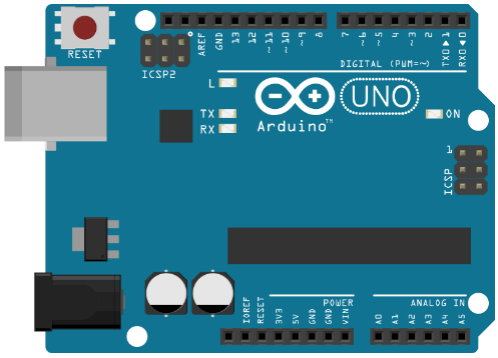


Figura 3-8: Representação do *Arduino*[6].

A Figura *3-9* apresenta um diagrama de blocos de uma cadeia de processamento utilizando o *Arduino*.

Figura 3-9: Diagrama de blocos de uma cadeia de processamento utilizando o *Arduino*.

Características Básicas

A Tabela 3-8 apresenta as características básicas do *Arduino*.

Microcontrolador	ATmega328 / ATmega168
Pinos de entrada analógica	6
Pinos de I/O digitais	14 (dos quais 6 podem ser saídas PWM)
Tensão operacional	$5\ V$
Tensão de alimentação (recomendada)	$7 - 12\ V$
Tensão de alimentação (limites)	$6 - 20\ V$
Corrente contínua por pino de I/O	$40\ mA$
Corrente contínua para o pino 3.3 V	$50\ mA$
Memória flash	32 KB (2KB usados para o *bootloader*) / $16\ KB$
SRAM	$2\ KB$
EEPROM	$1\ KB$
Freqüência de clock	$16\ MHz$

Tabela 3-8: Características básicas do *Arduino UNO*.

A *Figura 3-10* representa uma simples legenda dos pinos do *Arduino*. Neste caso em específico o *Arduino UNO* que foi utilizado no projeto.

[6] https://www.arduino.cc/

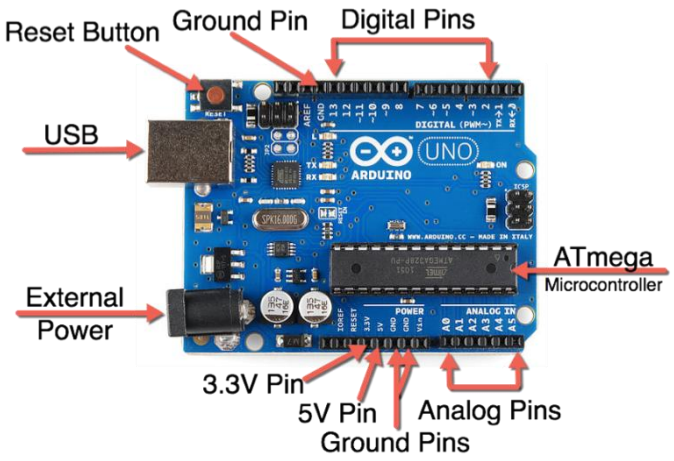


Figura 3-10: Legenda das portas do *Arduino*, Fonte: (Anexo 5 [26]).

Para fazer a programação do *Arduino*, é preciso ter disponível o *IDE* (*Integrated Development Environment*). Esse *software* está disponível no *website* oficial do *Arduino* "*https://www.Arduino.cc/* ". Pode-se fazer o *download* gratuito do "*installer*" ou do "*zip file*" (não precisa ser instalado). A Figura *3-11* representa o ambiente de trabalho do *software* utilizado para a programação do *Arduino*. Ao conectar o *Arduino* com o computador, teremos que selecionar a porta *COM*. Depois disso e de já ter o código desenvolvido, podemos verificar os erros e de seguida carregar para o *Arduino*. A Figura *3-12* representa todo esse procedimento, sítios da *internet* relevantes em Anexo 5.

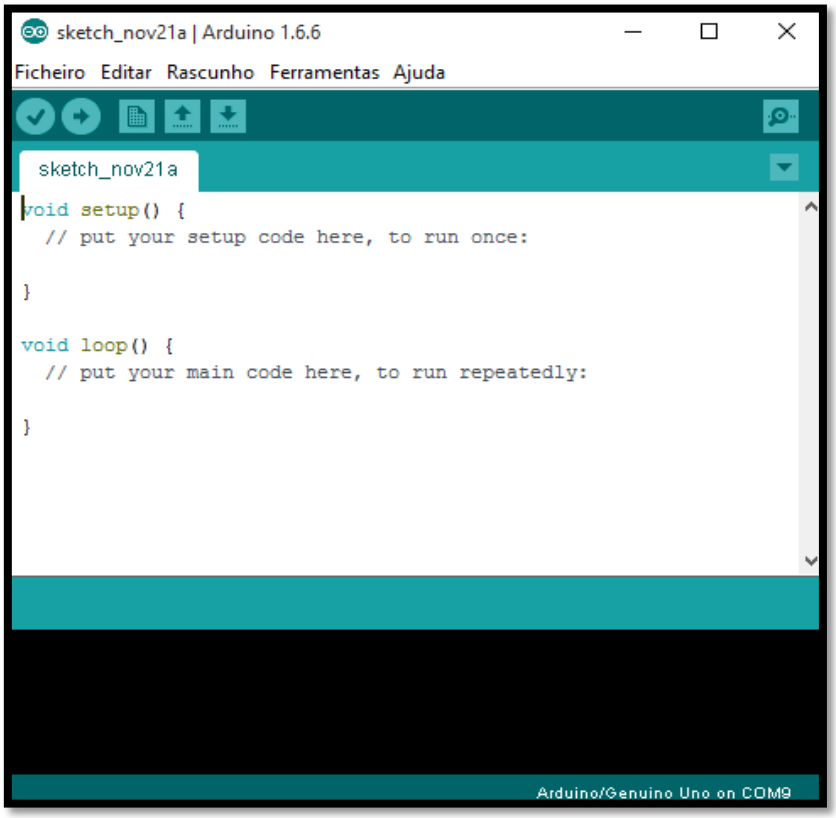


Figura 3-11: Ambiente trabalho do *Arduino IDE*.

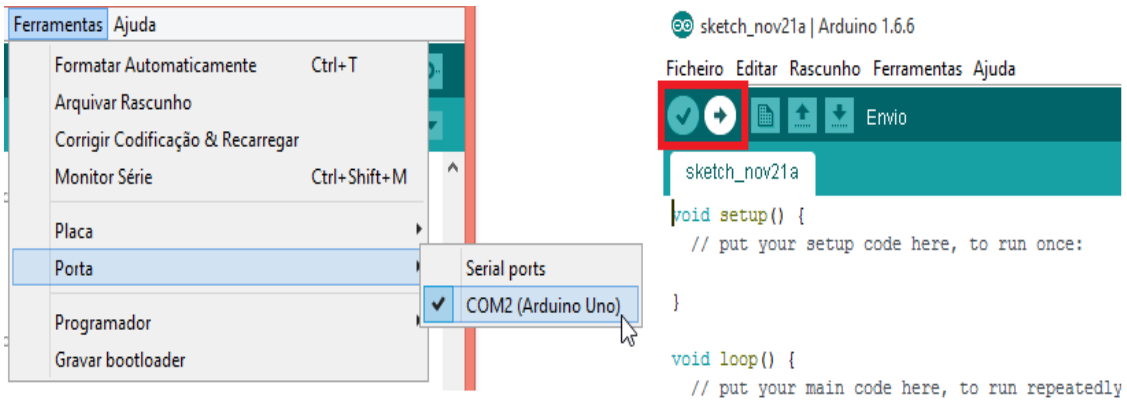


Figura 3-12: Procedimento de carregar um programa para o *Arduino*.

3.3- Módulo ESP8266

O módulo de comunicação ESP8266 (Figura *3-14*), pelas suas características e simplicidade na utilização, chamou a atenção de muitos programadores em meados do ano de 2014. O módulo permite ligar-se a redes wifi e também fazer simples comunicações *TCP/IP*. O módulo ESP8266 pode funcionar como um interface para comunicar com uma rede wifi existente, sendo necessário atribuir os *SSID* (nome da rede *wireless*) e a palavra passe para iniciar a comunicação (por exemplo quando se quer que o módulo ESP8266 comunique com o router da sua casa). Pode, ainda, funcionar como um *Access Point*; neste caso vamos atribuir o nome da rede *Wireless* e uma palavra passe, que pode ser acedido por qualquer outro dispositivo (podemos ver o módulo na lista das redes sem fios, podendo então ligar o computador ao módulo). Tem ainda um modo especial que é o *STATIONAP*, que é a combinação dos outros dois modos. Permite criar uma rede wifi e ainda ligar-se a um router, (Espressif Systems IOT Team, 2015), sítios relevantes consultados em Anexo 5 [32,44,49].

Figura 3-13: Representação do módulo ESP8266.

Características

- 802.11 b / g / n
- Wi-Fi Direct (P2P), soft-AP
- Pilha de protocolos *TCP / IP*
- Processadores de *MAC* e integrados

- Regulador *on-chip* com baixo *dropout* para fornecimento de energia aos periféricos integrados
- Potência de saída em +20dBm no modo 802.11b
- Sensor de temperatura
- Suporta a diversidade de antena
- Energia de consumo em modo *sleep* menor que 10uA
- Processador de baixo consumo de 32 *bits*: pode funcionar como um processador de aplicação
- 2.0 SDIO, SPI, UART
- STBC, 1x1 MIMO, MIMO 2x1
- A-MPDU, A-MSDU agregação e a 0,4 *within wake*
- 2ms, para conectar e transferir pacotes de dados
- Consumo de energia em *Standby* de menos de 1.0mW (DTIM3)

Aplicações diversas para o Módulo ESP8266

- *Smart Power plug*.
- Automação residencial.
- Rede de Malha.
- Controle Sem Fio Industrial.
- Monitor de Bebe.
- Redes de Sensores.
- Eletrônicos vestíveis.
- Localização sem fio de Dispositivos.
- *ID Tag* de Segurança.
- Sinais de Sistema de Posicionamento Sem Fio.

A *Figura 3-14* identifica todas as entradas e saídas do ESP8266, onde o pino RX da porta série recebe dados de comunicação com outros dispositivos, tendo em atenção o funcionamento com níveis de 3,3 V. O pino TX da porta série é usado para enviar dados do ESP8266 para outros dispositivos. O pino VCC é utilizado para a alimentação do módulo, nesse caso 3,3 V. A GPIO 0 é um pino também muito importante, pois pode ser utilizada como uma saída de controlo do módulo quando configurado como *output*. No entanto, quando for necessário fazer o *flash* do *firmware* do módulo, terá que estar ligado à massa. RESET tem que estar ligada aos 3,3 V quando se pretende fazer um *reset* ao módulo. Pino CH_PD tem sempre que estar ligada aos 3,3 V para ativar o *wifi*. A GPIO 2, também é utilizada para o controlo, geralmente utilizada como um output. Por fim, temos o GND que representa a massa, e como está indicado, tem que estar ligado à massa da alimentação do módulo.

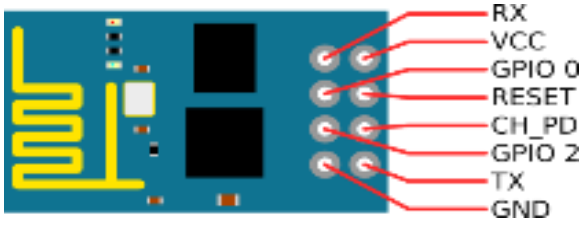


Figura 3-14: Legenda das portas do módulo ESP8266.

3.4- FTDI 232

O conversor FTDI232 é um conversor *USB*/Série (*UART*), com diferentes características avançadas. Este tipo de conversão é necessário dado que o microcontrolador típico não possui porta *USB*, sendo então muito utilizada atualmente para a programação de diferentes módulos, por isso, foi utilizada no desenvolvimento do projeto para a programação do módulo de comunicação ESP8266. A Figura *3-15* representa o FTDI232 utilizado.

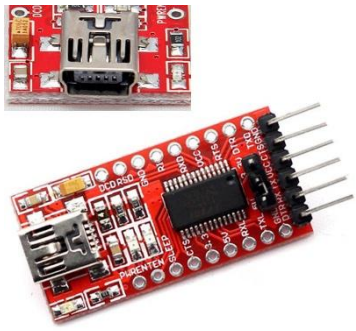


Figura 3-15: Representação do FTDI 232.

Características

- Todos os Protocolos *USB* disponíveis no *Chip*. Sem *USB firmware* requerido.
- Totalmente integrado 1024 *bit* EEPROM de armazenamento e configuração CBUS I/O.
- Totalmente integrado com resistências de terminação *USB*.
- Taxas de transferência de dados de transmissão 300 a 3 *Mbaud* (RS422, RS485, RS232) nos níveis TTL.
- Características únicas *USB* FTDI *Chip-ID* ™.
- Circuits integrate, *power-on-reset*.
- Opção inversão de sinal *UART*.

A Figura *3-16* apresenta uma legenda detalhada do FTDI 232 utilizado no projeto. Este funciona a 5 V ou a 3,3 V, dependendo da escolha do utilizador. O GND, VCC, TX e RX, foram, neste caso, os pinos necessários para o nosso caso em específico, em que se utiliza o GND ligado a massa, o VCC pode funcionar a 5 V ou a 3,3 V para alimentar os diferentes módulos. No nosso caso pelo facto de o módulo ESP8266 funcionar a 3,3 V, foi essa a tensão utilizada. Para transmitir dados para o módulo liga-se a porta Tx do FTDI à porta Rx do ESP8266. Caso estejamos a utilizar o FTDI a 5 V, é preciso converter níveis de tensão de 5 para 3,3V. No nosso caso não foi preciso, pelo facto de utilizar já o FTDI a 3,3V. A porta Rx do FTDI vai ligar à porta Tx do ESP8266.

O pino CTS corresponde ao sinal de controlo "*Clear to Send Control Input / Handshake Signal*" e o DTR (*output*) a "*Data Terminal Ready Control Output / Handshake Signal* ".

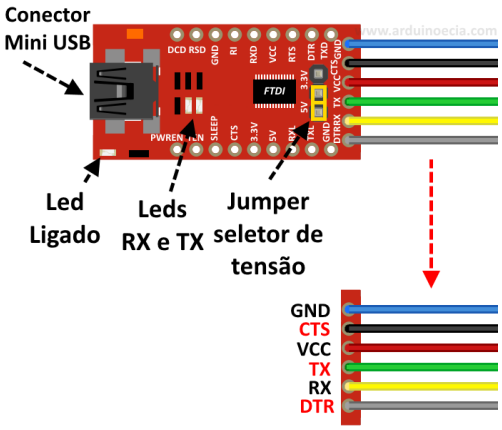


Figura 3-16: Legenda das portas do FDTI 232, Fonte: (Anexo 5)[7].

[7] ANEXO 5 – Sítios da Internet Relevantes

4. BASE DE DADOS

O *Gates Technology* baseia-se na existência de uma base de dados onde estão armazenados os dados dos clientes, dos convidados e do(s) administrador(es) da residência ou condomínio cujo acesso se pretende controlar. A facilidade de acesso e gestão da base de dados assume um papel muito importante, por um lado porque se pretende que seja de fácil utilização e, por outro, porque deve estar bem projetada de forma a permitir gerir os dados de forma eficiente.

Neste capítulo faz-se uma breve descrição sobre a construção e o funcionamento da base de dados desenvolvida. Informações adicionais, mais detalhadas, podem ser encontradas no Anexo 1.

4.1- Enquadramento

Para criar e entender o funcionamento de uma base de dados é preciso ter conhecimentos sobre determinadas ferramentas de *software* e entender o funcionamento de uma linguagem de programação.

Este trabalho utilizou várias ferramentas para a criação da base de dados nomeadamente o MySQL Workbench, o editor de textos Notepad++ e o Xampp, entre outras. Cada uma destas ferramentas desempenha um papel importante na criação da base de dados. O modelo *ER* (Relacionamento de Entidades) e a criação dos respetivos campos foram feitos utilizando o MySQL Workbench. A criação do *script* da base de dados foi feita com o Notepad++. O *software* Xampp foi utilizado para criar um servidor *Apache* no meu computador.

Um outro conceito que é importante ter presente é a definição de chave primária (*primary key*) e chave estrangeira (*foreign key*) (Jones et Tollervey, 2014).

Chave primária corresponde a um campo ou conjunto de campos que distingue cada registo de todos os restantes. Duas regras básicas são:

- Não pode ter duplicação – campo que numa tabela for definido como chave primária não pode ser igual em dois registos diferentes;
- Não pode ser nula – o campo que é a chave primária sempre terá que ter um valor inserido. Não pode ser nulo, nem assumir valor zero.

Uma chave estrangeira (*foreign key*) é um campo que vai apontar para a chave primária de uma outra tabela ou da mesma. A finalidade da chave estrangeira é garantir a integridade dos dados referenciais, pois, somente serão permitidos valores que supostamente vão aparecer na base de dados. Este tipo de atributo não permite exclusão, modificação e/ou inserção de dados em tabelas que estejam dependentes umas das outras, o que requer modificadores especiais.

4.2- Organização, Estrutura e Criação da Base de dados

Para que o carro de um cliente (isto é, morador no condomínio) possa ter acesso à garagem, o *VIN* tem de constar na base de dados. É este código (*VIN*) que o carro deve enviar para o servido quando se aproxima da porta da garagem. O servidor, localizado na garagem, tem a tarefa de aferir junto da base de dados (situada na *cloud* ou localmente para o caso de não haver ligação *wireless*), se esse *VIN* está ou não registado. Se for validado, então o servidor dará autorização para a abertura da porta da garagem.

A base de dados da *Gates Technology* assenta na utilização de 8 Tabelas, nomeadamente:

1. Tabela Pessoas
2. Tabela Carros
3. Tabela Alertas
4. Tabela Eventos
5. Tabela Carros_2
6. Tabela Convidados
7. Tabela Conv_Act
8. Tabela Pessoas_Tipo
9. Início Sessão

As tabelas comunicam entre si, estando interligadas direta ou indiretamente como se mostra na Figura *4-1*.

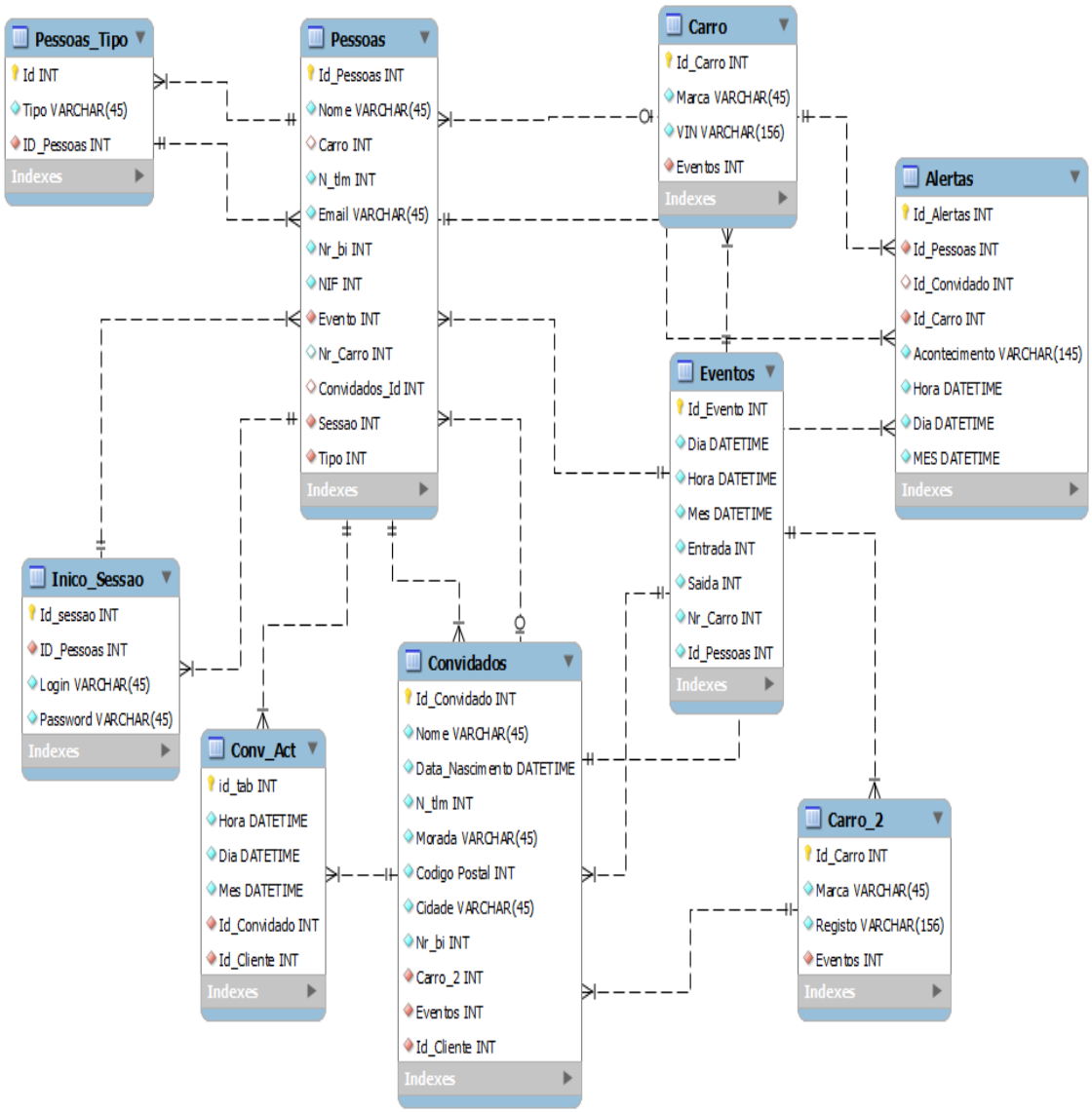


Figura 4-1: Comunicação das tabelas que constituem a base de dados (modelo *ER*).

A base de dados foi desenvolvida de modo a ter um funcionamento simples, seguro e que facilite a tarefa de fazer pesquisas na mesma. A interligação entre as tabelas da base de dados é feita à custa de *primary keys* e de *foreign keys*, o que nos permite fazer diferentes pesquisas com diferentes referências.

A nossa plataforma compreende duas versões da base de dados. Uma das versões é guardada localmente no servidor da garagem e ocupa 12,0 KB (12.288 bytes). A outra base de dados, mais completa, encontra-se na *cloud* e ocupa 340 KB (348.160 bytes).

4.3- Constituição

Nesta secção falaremos com mais algum detalhe de cada uma das tabelas e dos campos que as constituem.

As tabelas “Carro 2”, “Convidados” e “Convidado Act” vão gerir todas as informações relacionadas com os convidados das pessoas que vivem nos condomínios (ou dos proprietários no caso de se tratar de uma residência).

- **Tabela Pessoas**

É nesta tabela que se guardam os dados do cliente. É constituída pelos seguintes campos:

- **Id**
- Nome
- Número de Telefone
- Número de BI
- Número de Identificação Fiscal
- E-mail
- Número de Carros
- **Carro**
- **Eventos**
- **Convidados**
- **Sessão**
- **Tipo**

O campo “Id” é a chave primária da tabela. Os campos “Carro”, “Eventos, “Convidados”,” Sessão” e “Tipo” funcionam como chaves estrangeiras.

O campo “Convidados”, sendo uma chave estrangeira, identifica os dados da Tabela “Convidados” que será onde estará toda a informação das pessoas com o estatuto de convidados, isto é, que não vivem no condomínio/casa e que são convidados das pessoas que vivem nos apartamentos, quando se trata de um serviço num condomínio, ou pelo dono da casa, quando é um serviço presente numa casa/vivenda. Os campos “Sessão” e “Tipo” são as chaves estrangeiras para as tabelas “Inicio Sessão” e “Pessoas Tipo”. Estes campos contêm os *Ids* da Tabela Inicio Sessão onde estão guardados o *Login* (Nome de utilizador e Palavra passe) de cada utilizador e também a informação para saber se a pessoa está registada como um cliente ou como um administrador, na Tabela Pessoas Tipo (o ambiente de trabalho disponível é diferente para clientes e para administradores).

Os dados pessoais dos clientes são guardados nos campos “Nome”, “Número de Telefone”, “Número de BI”, “Número de Identificação Fiscal” e “*E-mail*”. O campo “Número de Carros” é utilizado para o administrador saber quantos carros estão associados a um mesmo cliente. Este campo tem um limite máximo, pois cada pessoa só terá direito a um determinado número de carros com direito a estacionamento na garagem. Este campo foi criado para o caso de a plataforma ser usada por um condomínio. Se for por uma residencial, trata-se de um campo um pouco redundante.

- **Tabela Carros**

Nesta tabela é onde são guardados todos os dados dos carros dos clientes, nomeadamente o número *VIN* do carro. Para além do *VIN*, é constituída pelos seguintes campos:

-Id
-Marca
-VIN
-Eventos

O campo "Id" é a chave primária da tabela. O campo "Eventos" é uma chave estrangeira para a Tabela Eventos.

O campo "Eventos" tem a identificação do registo onde estão guardados os eventos de cada carro. Colocamos este campo, pois tivemos como preocupação o caso de ser necessário saber os eventos da garagem associados a um determinado carro sem ser necessário ir pesquisar através de cada pessoa que está registada na base de dados. Acreditamos que esta abordagem facilita a organização da pesquisa, tanto para o cliente como para o administrador.

O campo "Marca" deve ser preenchido com a marca do carro, podendo ser útil para facilitar alguma pesquisa na tabela "Carro" através da marca do mesmo. O campo principal da tabela é o campo "*VIN*", identificador único de cada carro.

- **Tabela Alertas**

Esta tabela é muito importante para a manutenção e identificação de erros do nosso sistema. É constituída pelos seguintes campos:

-Id
-Id Pessoas
-Id Convidado
-Id Carro
-Acontecimento
-Hora
-Data
-Mês

O campo "Id" funciona como chave primária. Os campos "Id Pessoas", "Id Convidado" e "Id Carro" são chaves estrangeiras das tabelas "Pessoas", "Convidados" e "Carro", respetivamente. "Id Pessoas" contém a informação da pessoa presente na hora do acontecimento. Só será guardada alguma informação nos campos "Id Convidado" e "Id Carro" (informação sobre o convidado e sobre o carro) no caso de estar presente no momento que for registado o acontecimento. Temos ainda o campo "Acontecimento" onde será registada de forma detalhada toda a informação de um determinado acontecimento. Também temos ainda os campos que registam a hora, o dia e o mês do acontecimento que são os campos "Hora", "Dia" e "Mês", respetivamente.

- **Tabela Eventos**

Esta tabela vai organizar todos os eventos efetuados pelos diferentes clientes quando acedem à garagem. Por evento entenda-se a porta ser aberta ou fechada por um determinado carro. A tabela é constituída pelos seguintes campos:

-**Id**
-Hora
-Dia
-Mês
-Entrada
-Saída
-Nr_carro
-**Id Pessoas**

Os dados guardados nesta tabela são a hora, o dia, o mês e também o *Id* do cliente associado ao evento. Isto é feito para aumentar o nível de segurança da garagem. Desta forma, quando algum cliente necessite de consultar os seus eventos, através da aplicação *Android* ou do *website*, só terá acesso apenas aos seus respetivos dados. Para, além disso, esta organização facilita o trabalho do cliente ou administrador caso pretenda fazer uma pesquisa, pois a pode fazer simplesmente através da hora, dia ou até mês.

Campos "Entrada", "Saída" são campos de modo Binário que vão dizer em cada evento se o carro estava a sair ou a entrar. É utilizado binário, pois se o carro estiver a sair o campo "Entrada" automaticamente guardará o valor zero e o campo "Saída" ficará 1 (um). O campo "Nr_carro" representa quantos carros, associados ao mesmo cliente, estavam presentes no evento, ou seja, é também possível entrar/sair pela garagem vários carros ao mesmo tempo e nesse caso vai ser guardado o evento único com o número de carros que passaram pela garagem e também do cliente associado.

Por último temos o campo "Id Pessoas" que funciona como uma chave estrangeira da tabela "Pessoas". Assim, se fizermos uma pesquisa somente na tabela "Eventos", podemos chegar à pessoa que estava presente no evento. Esta abordagem acredita, que dá mais segurança ao serviço e ajuda a controlar melhor o funcionamento/acesso à garagem.

- **Tabela Carros_2**

Esta tabela é utilizada quando os clientes querem dar autorização de acesso à garagem a um convidado. Nesta tabela são guardados os dados dos veículos dos convidados, juntamente com um '*Login*' que é atribuído, caso o cliente pretenda aceder ao servidor da garagem através da aplicação *Android* ou do *website*. Todos os eventos da porta da garagem efetuados por um convidado podem ser consultados no *website* pelo administrador ou pelo cliente que deu acesso ao convidado. É constituída pelos seguintes campos:

-**Id**
-Marca

-Registo
-Eventos

O campo Id é usado para a identificação de cada coluna da tabela funcionando como a chave primária. O Campo "Marca" regista a marca do carro de cada convidado, para ajudar na identificação, na segurança e na pesquisa. No campo "Registo" está guardado o nome da pessoa em que o carro está registado, isso caso o carro esteja registado com um nome diferente do convidado. O campo "Eventos" funciona como uma chave estrangeira da tabela "Eventos" para que seja possível guardar cada evento realizado para cada carro de cada convidado.

- **Tabela Convidados**

Esta tabela é constituída pelos seguintes campos:

-Id Convidado
-Nome
-Data Nascimento
-Número Telefone
-Morada
-Código Postal
-Cidade
-Número BI (Identificação)
-Carro2
- Eventos
-Id Cliente

A chave primária é o campo "Id Convidado". Os campos "Nome", "Data Nascimento", "Número de Telefone", "Morada", "Código Postal", "Número de BI" e "Cidade", são os campos essenciais desta tabela, pois são aqueles que contêm toda a informação para a identificação de cada convidado. Os campos "Carro2", "Eventos" e "Id Cliente" funcionam como chave estrangeira para as tabelas "Carro2", "Eventos" e "Pessoas", respetivamente, através das quais conseguimos saber os eventos associados a cada convidado e quem é o cliente associado a cada convidado.

- **Tabela Conv_Act**

A Tabela "Conv_Act" torna possível a opção de tornar um convidado ativo ou passivo para determinada hora, dia ou mês. É constituída pelos seguintes campos:

-Id
-Hora
-Dia
-Mês
-Id Convidado
-Id Cliente

A chave primária da tabela é o campo "Id", pelo facto de ser único para cada registo. Nos campos "Hora", "Dia" e "Mês" o cliente vai definir a data específica que pretende deixar o convidado ativo, ou seja, com autorização para aceder à garagem. Os campos "Id Convidado" e "Id Cliente" funcionam como chave estrangeira para as tabelas "Convidados" e "Pessoas", respetivamente e vão permitir identificar cada um dos convidados e o cliente que lhe deu permissão a aceder à garagem.

- **Tabela Pessoas_Tipo**

Trata-se de uma tabela simples que serve apenas para identificar cada utilizador como cliente ou administrador. É constituída pelos campos:

-Id
-Tipo
-Id Pessoa

A chave primária é o campo "Id" que representa a identificação para cada coluna da nossa tabela. O Campo "Id Pessoas" é uma chave estrangeira para a tabela "Pessoas", que serve como identificação para cada coluna da tabela "Pessoas" de forma a agilizar a identificação do Administrador e dos Clientes.

- **Tabela Inicio_Sessão**

A tabela "Inicio Sessao" é constituída por:

-Id
-Id Pessoa
-Login (nome de utilizador)
-Password

A chave primária é o campo "Id" que vai ser a identificação para cada coluna da nossa tabela. O campo "Id Pessoa" é uma chave estrangeira da tabela "Pessoas". Decidimos incluir este campo nesta tabela para, de forma simples, identificar o *login* de cada pessoa sem ter de pesquisar na tabela "Pessoas" (é uma tabela mais complexa). Os campos "*Login*" e "*Password*" têm os dados que permitem a cada utilizador fazer o *login* na plataforma *online.* Consideramos que esta caraterística é uma das vantagens da nossa aplicação (voltaremos a falar deste aspeto mais à frente, neste relatório).

5. PLATAFORMA *ONLINE/WEBSITE*

Foi desenvolvida uma plataforma *online* http://www.gatestechnology.eu/ que permite ao utilizador, através do acesso ao *website*, aceder aos dados, tornando o controlo da garagem mais simples e seguro. Neste capítulo vamos falar da composição do *website* e de todo o percurso para a sua criação. Os tópicos referidos são a constituição do *website*, o seu funcionamento, bem como *softwares* e códigos utilizados para a criação do mesmo. Informações adicionais e mais detalhadas sobre a plataforma *online* do *Gates Technology* podem ser encontradas no Anexo 2.

5.1- Visão Geral

Foi desenvolvida uma plataforma *online* de acesso, o que representa uma forma importante de interagir com os utilizadores. Desta forma, os clientes conseguem aceder à garagem (controle direto do portão), consultar eventos diferentes, fazer atualizações na base de dados em relação aos convidados, carros registados, ou então adicionar novos clientes (processo esse que só é feito pelos administradores).

Numa fase inicial de criação do *website*, foi utilizado um *CMS* (*Content Management System*), ou seja, Sistema de Gerenciamento de Conteúdo. O *CMS* utilizado foi o *Joomla* (Figura 5-1). Representa um dos principais *CMS* utilizados atualmente e está disponível em mais de 10,071 mil extensões e milhares de *templates*. Sendo *open source*, tornou-se uma opção ideal para ser utilizado nesta fase do trabalho.

Figura 5-1: Logotipo Joomla.

No entanto, na fase final do projeto, optámos por criar um *website* de "raiz", de modo a conseguir criar um *BackOffice* com o controlo total da base de dados do *Gates Technology*. Foi então criado um *website*, tendo como base as linguagens de programação *html5*, *Css*, *Js* e *php* (Schengili-Roberts, Keith, 1997). Este *site* representa a plataforma *online* de acesso e controlo do *Gates Technology*, tendo sido escolhido pelo facto do *CMS Joomla* já vir acompanhado de uma *BackOffice*, pelo que não se teria o controlo de tudo e também não seria possível verificar o nível de segurança do mesmo. Assim, criando o nosso próprio *site* de raiz, tornou-se possível garantir o nível de segurança desejado.

Através da plataforma *online* é possível efetuar o *login* com uma conta que, ao ser registada pelo administrador, é atribuída a cada cliente. Depois, este tem a possibilidade de efetuar mudanças na conta, nomeadamente alterar a palavra passe ou o nome de utilizador necessário para efetuar o *login*. O acesso pode ser feito pelos clientes ou pelos

administradores, havendo no *website* dois botões que redirecionam cada utilizador para a página de início de sessão adequada ao seu perfil.

Após efetuar o *login*, o utilizador depara-se com um leque variado de opções. Esta interface está organizada de forma a tornar fácil o seu entendimento por qualquer utilizador. Uma das opções suprarreferidas é a possibilidade de o cliente conseguir abrir ou fechar a garagem através do *website*, estando em casa ou em qualquer parte do mundo. É também possível registar carros, apagar o registo de um carro e consultar os seus eventos. É possível também ter acesso à parte em que o cliente tem a possibilidade de registar um convidado atribuindo a esse um registo para efetuar um *login* ou também indicar a data (dia, hora ou mês) em que esse convidado tem a possibilidade de aceder à garagem. O administrador tem um maior número de opções nomeadamente à possibilidade de consultar os dados de qualquer pessoa que está registada na base de dados, os dados de qualquer carro registado, remover ou adicionar clientes à base de dados, consultar a informação sobre os convidados de todos os clientes, a possibilidade de ter acesso aos eventos realizados pelo portão da garagem, entre muitas outras (Brooks, 2011).

5.2- Constituição do *Website* "*índex*"

Como já mencionado anteriormente, foram desenvolvidas duas plataformas de acesso *online* durante a realização deste projeto. Foi um processo de aprendizagem e de desafio, porém, com esforço e dedicação, consegui desenvolver uma base de dados de raiz com as características desejadas. Por isso, iremos focar-nos apenas na explicação do *website* criado após a fase inicial, utilizando o *html5*, *Css* e *php* (Nixon, 2014).

No arranque desta etapa do trabalho, identificámos os seguintes desafios:

- **Nome** do *website* - Após algumas ideias, decidiu pelo nome "*Gates Technology*".
- **Domínio** para o *website* – tivemos de adquirir um domínio para o nosso *website*, que ficou definido como "*www.gatetechnology.eu*" em que o '.eu' significa que a empresa se situa na União Europeia.
- **Servidor** *online* – foi necessário comprar um servidor *online* para poder alojar o *website* na *web* (*internet*) de modo a ser possível aceder ao *website* em qualquer parte do mundo.
- **Página** inicial - A página inicial é a "cara" do *website* e por isso mereceu especial atenção. Tivemos de ponderar com muita atenção para decidir os conteúdos que se iriam apresentar (breves descrições sobre o trabalho, informação sobre a aplicação *Android*, etc). Nesse seguimento, colocámos ao dispor um *demo* que os utilizadores podem usar para interagir com a aplicação para conhecê-la melhor. Existe um menu, através do qual os utilizadores podem ter acesso ao que precisam e é constituído por: Início, Serviços, Aplicação, Sobre, *Download* e Contatos, que a seguir se descrevem:
    - Serviços - fala-se um pouco acerca da aplicação *Android*, das instalações e também um pouco sobre o produto em geral.

- Aplicação - existe dois botões que dão a possibilidade de conhecer a aplicação ou de ir diretamente à diretoria do *download*.
- *Download* - a pessoa pode aceder diretamente para fazer o download da aplicação.
- Contatos - é uma parte muito importante, pois é a forma que os utilizadores têm para entrar em contato com o administrador do *website* e também, se pretenderem conhecer mais sobre o produto, existe um mapa e um formulário para efetuar o contacto. O mesmo também pode ser feito através da página do Facebook, Twitter ou Youtube, ou também, pelo contacto móvel do administrador que a pessoa tem ao seu dispor.

5.3- Funcionamento

Nesta secção será apresentado o funcionamento da plataforma *online*, tendo presente que pode ser acedida por qualquer pessoa que (ainda) desconhece a aplicação ou por alguém que já seja um cliente ou o administrador. Assim, qualquer pessoa que aceda ao *website*, por exemplo, para ter conhecimento do produto, encontrará toda a informação que vai desde a explicação sobre o produto, como o instalar, o acesso ao *website* até à aplicação *Android*. Incluímos um *demo* que auxilia neste percurso. Caso se trate de um cliente ou um administrador, consegue ter acesso a um *backend* onde poderão ter acesso a diversas opções de controlo e acesso à garagem.

Um cliente, ao efetuar o *login*, será redirecionado para uma página em que pode Abrir ou Fechar a porta da garagem diretamente a partir do *website* independentemente de onde estiver, poderá também adicionar ou remover carros que lhe estão associados, ver ou registar convidados e também ainda consegue ver os eventos dos carros (somente os carros registados pelo cliente). No caso dos Administradores, o *website* representa uma ferramenta muito importante na medida em que é a porta de entrada para editar a base de dados. O administrador consegue ter acesso a praticamente tudo e por isso, na criação do mesmo do *website*, foi levado em consideração muito o nível de segurança para qualquer pessoa registada como administrador.

Após o *login*, a página que abre é a que dá acesso para consultar as pessoas com acesso à garagem, independentemente se é o administrador, um cliente ou um convidado. Tem acesso a uma página de registo de novos clientes ou também de remover/adicionar novos carros. É possível consultar os Eventos, pesquisando por apartamento, data, pessoa, entre outros. Consegue-se também ver o registo de todos os Alertas feitos pela garagem de forma automática registando. A Figura *5-2* mostra um *sitemap* do *website,* isto é, um mapa que permite para conhecer melhor a constituição da plataforma *online*.

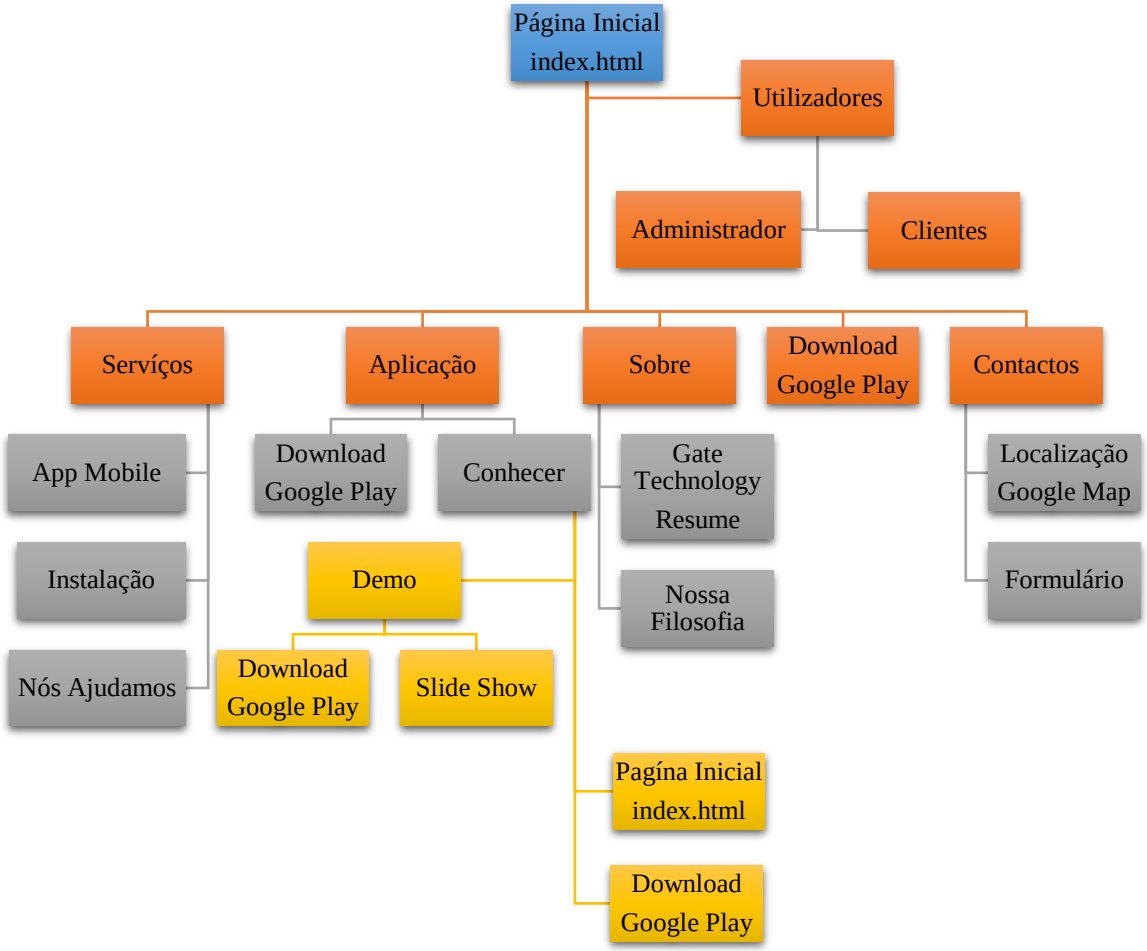


Figura 5-2: *Sitemap* da página inicial "*índex*".

5.4- *BackOffice* Clientes e Administradores

Nesta secção é feita uma descrição da constituição do *website* na perspetiva do cliente (Secção 5.4.1) e do administrador (Secção 5.4.2)**.**

5.4-1. Clientes

A Figura *5-3* apresenta um *sitemap* com as páginas do *BackOffice* Clientes.

Após uma pessoa (condómino) de um prédio efetuar um pedido do produto, é feito um registo na base de dados com os dados dessa pessoa, nomeadamente o nome, o *email* e o registo do carro. É-lhe atribuído um "*login*" (*login* esse que é o nome de utilizador e uma palavra passe) que pode ser utilizado para aceder ao *backend* do *website*. O *login* é o mesmo que deve ser utilizado para aceder à aplicação *Android*.

Figura 5-3: *Sitemap*, páginas do *BackOffice* Clientes.

De seguida será feita uma descrição de tudo o que o cliente pode fazer através da plataforma *online*.

Ao efetuar o *login* através do "*login Page*" (Figura *5-4*) será redirecionado para a página principal (Figura 5-5) de acesso do cliente.

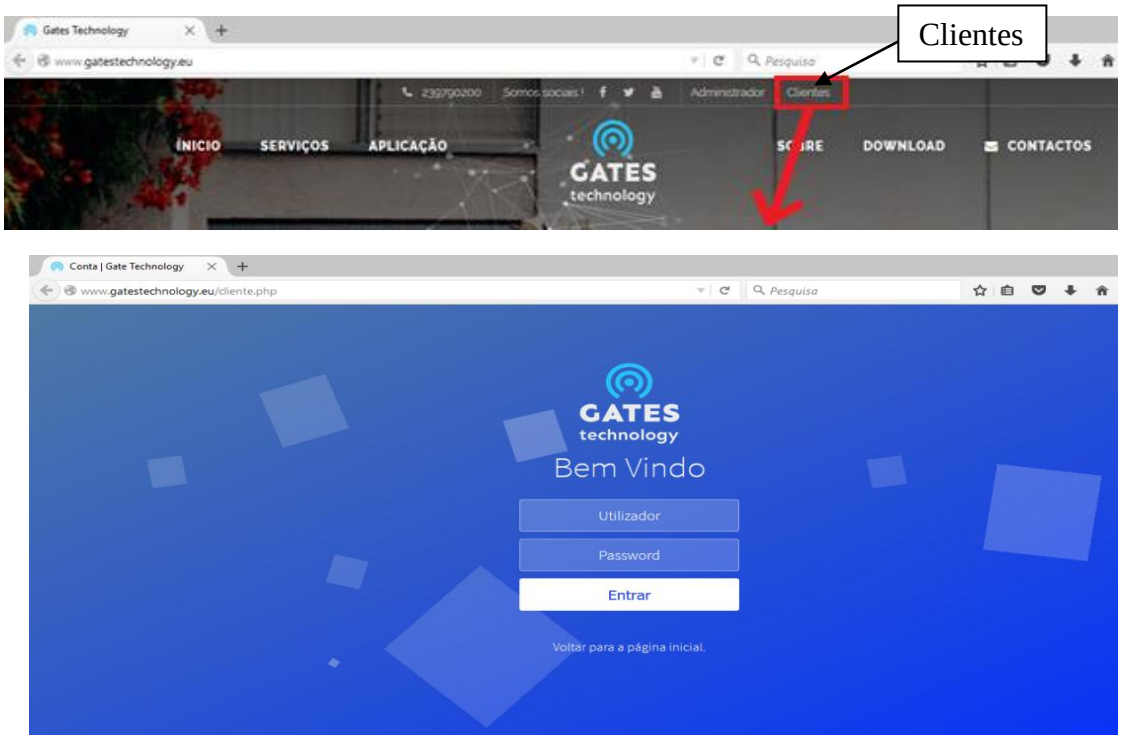


Figura 5-4: Página para efetuar o Início da Sessão (Clientes).

É possível ter acesso direto à porta da garagem através dos botões "Abrir" e "Fechar", controlando, assim, a abertura e o fecho da porta através do *website*, o que pode ser feito estando em qualquer parte do mundo, desde que se tenha acesso à *internet.*

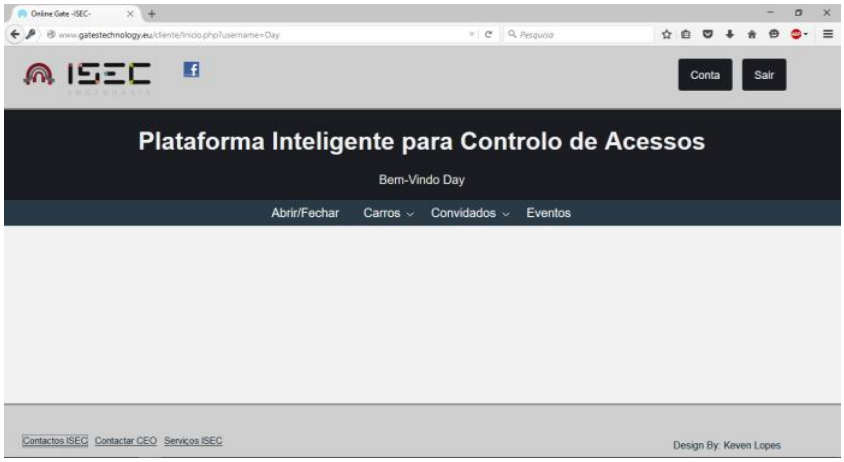


Figura 5-5: Página principal de acesso e controle dos clientes.

Este é o "*style*" utilizado nas páginas de controlo e acesso dos clientes e dos administradores. Como é possível observar, o cliente tem a possibilidade de efetuar diferentes ações através do *BackOffice*. Através do botão "Carros", o cliente tem a possibilidade de registar novos carros, sob a confirmação do administrador.

Pode também consultar os carros que estão registados com o nome e o *ID* do mesmo cliente (Figura 5-6). Caso necessite, o cliente pode ainda remover os registos dos carros que estão registados na base de dados da garagem, acedendo a configurações da conta (utilizando o botão "conta").

Figura 5-6: Página dos carros registados para um cliente.

O cliente consegue adicionar convidados à base de dados (Figura 5-7), o que também é feito sob a confirmação final do administrador. Neste caso, basta navegar utilizando o botão "Convidados". Dentro deste separador, ainda há a possibilidade de consultar os

eventos da garagem efetuados pelos convidados que esse mesmo cliente adicionou. Pode ainda definir um convidado como ativo ou não. Caso seja um convidado ativo, o cliente consegue definir uma determinada data em que esse consegue ter acesso à garagem. Como a Figura *5-7* indica, para efetuar o registo de um convidado é preciso algumas informações importantes sobre o mesmo, isto para aumentar o nível de segurança de todos. Dados como o nome completo do convidado, data de nascimento, morada, código postal e cidade, número de telefone, e número de identificação, são informações sobre o convidado que têm de ser inseridas. Serão também necessários a marca e o número *VIN* do veículo do convidado para o registo. Por fim, é atribuído um nome de utilizador e um *password*, para ser possível ao convidado ter o controlo da garagem através da plataforma *online* ou então através da aplicação *Android* (Harris, 2014).

Figura 5-7: Página de registo de um novo convidado.

O botão “Eventos” pode ser utilizado sempre que o cliente precisar consultar os eventos realizados pelo mesmo. O botão “Conta” é onde o cliente consegue configurar e alterar os seus dados da conta, como por exemplo, mudar o nome de utilizador, *password* a utilizar no início de sessão, ou ainda, alterar o *email* de contato.

O botão “Conta” é onde o cliente consegue configurar e alterar os seus dados da conta, como, por exemplo, mudar o nome de utilizador, *password* a utilizar no início de sessão, ou ainda, alterar o *email* de contato.

5.4-2. Administrador

No desenvolvimento das páginas do Administrador, a preocupação com a segurança foi algo que sempre esteve muito presente, pois é o administrador que consegue controlar toda a base de dados e os acessos à garagem.

A Figura *5-8* e Figura *5-9* complementam um *sitemap* com as páginas do *BackOffice* do Administrador.

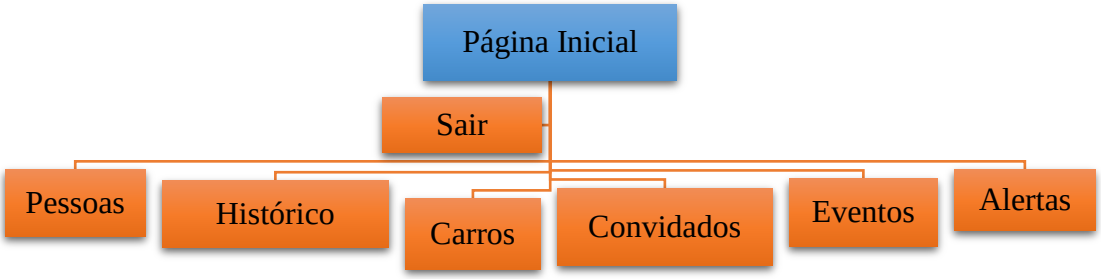


Figura 5-8: *Sitemap*, páginas do *BackOffice* administrador (fase1).

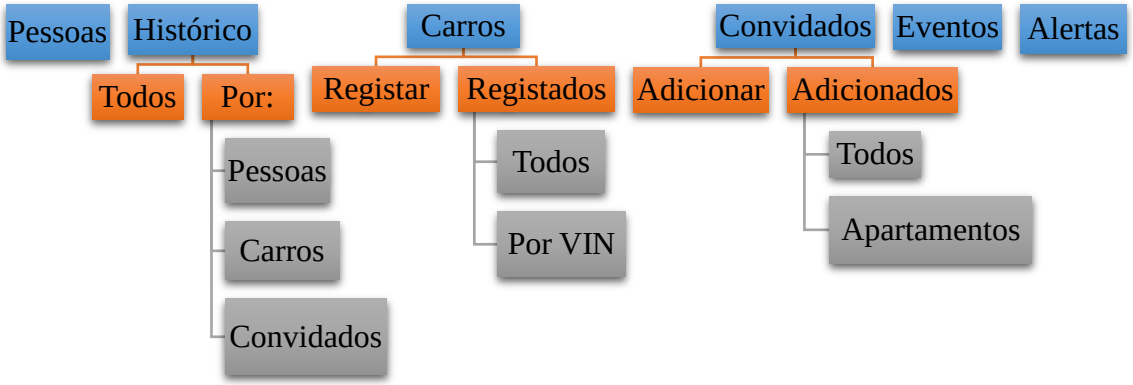


Figura 5-9: *Sitemap*, páginas do *BackOffice* administrador (fase2).

Após ser feito o *login* pelo administrador (Figura 5-10), este será redirecionado para a página inicial de controlo (Figura *5-11*). O administrador também precisa de um *login* e de uma palavra passe para aceder ao *BackOffice* do *website*. Os registos dos administradores são feitos numa base de dados externa ao da garagem, a qual vai ter os registos de todos os administradores de modo a conseguirem aceder uma primeira vez ao *BackOffice*, sem antes terem feito qualquer registo.

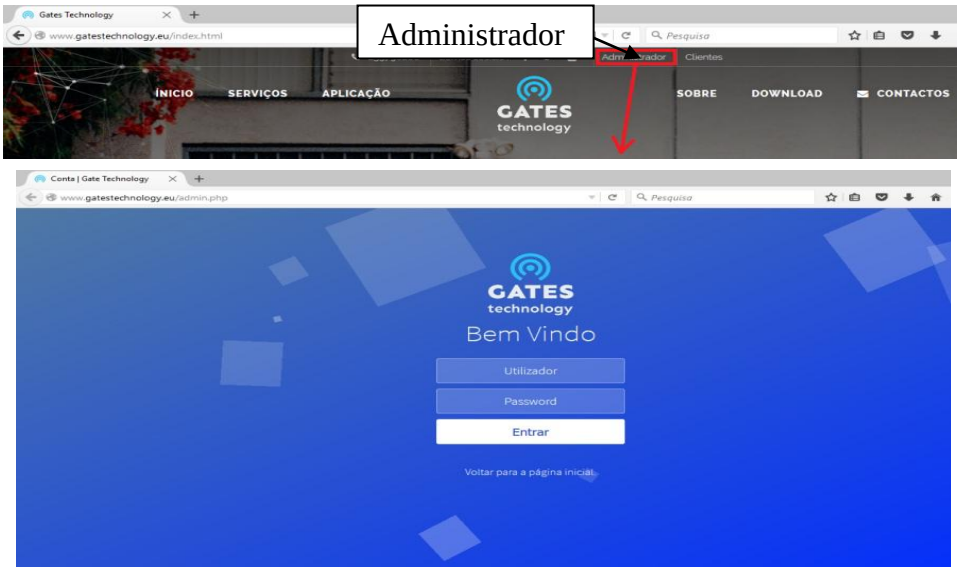


Figura 5-10: Página de Inicio de Sessão (Administrador).

Depois de aceder, o administrador pode registar todos os clientes na base de dados da garagem e também, registar a mesma informação nessa base de dados para efeitos de segurança.

A Figura *5-11* representa a página principal de acesso dos administradores.

Figura 5-11: Página de Controlo do Administrador.

Esta dispõe do maior nível de acesso, segurança e privacidade para o controle da base de dados da garagem. Na parte final da página do lado direito, temos três importantes botões. Estes permitem que o utilizador seja redirecionado para diferentes páginas. Caso o utilizador precise entrar em contato com o Instituto Superior de Engenharia de Coimbra (ISEC), pode fazê-lo através do botão "Contatos ISEC", sendo direcionado para a página em que pode deixar uma mensagem ao ISEC. Caso queira simplesmente contatar os administradores do *Gates Technology*, deve apenas utilizar o botão "Contatar *CEO*" e é redirecionado para a página em que preenche o formulário de contato. O botão "Serviços ISEC" redireciona para a página principal do Instituto Superior de Engenharia de Coimbra (*www.isec.pt*), para o separador dos serviços.

O estilo das páginas do *BackOffice* é diferente da página principal (*índex.html*) do *Gates Technology*, isto porque optámos por simplificar o máximo possível, para uma mais fácil compreensão por parte dos utilizadores. A *Gates Technology* foca-se na simplicidade, e por isso tentámos ter um *BackOffice* simples, acessível e que não exigisse muitos conhecimentos por parte dos utilizadores.

Os administradores têm diferentes alternativas de navegação dentro do *BackOffice*, concentrando-se principalmente na organização dos clientes, registos e históricos da garagem, organização e registos de novos carros, ou, também, de convidados de qualquer cliente registado na base de dados, avarias, ou alertas emitidos pela garagem (servidor presente na garagem), entre muitas outras possibilidades.

Através do botão "Pessoas", o administrador tem a possibilidade de adicionar novos clientes, ou então, consultar todas as pessoas que estão registados na base de dados da garagem, sendo esses clientes ou outros administradores. A Figura *5-12* representa o processo de registo de um novo cliente, indicando então os principais dados que serão precisos para registar esse novo cliente.

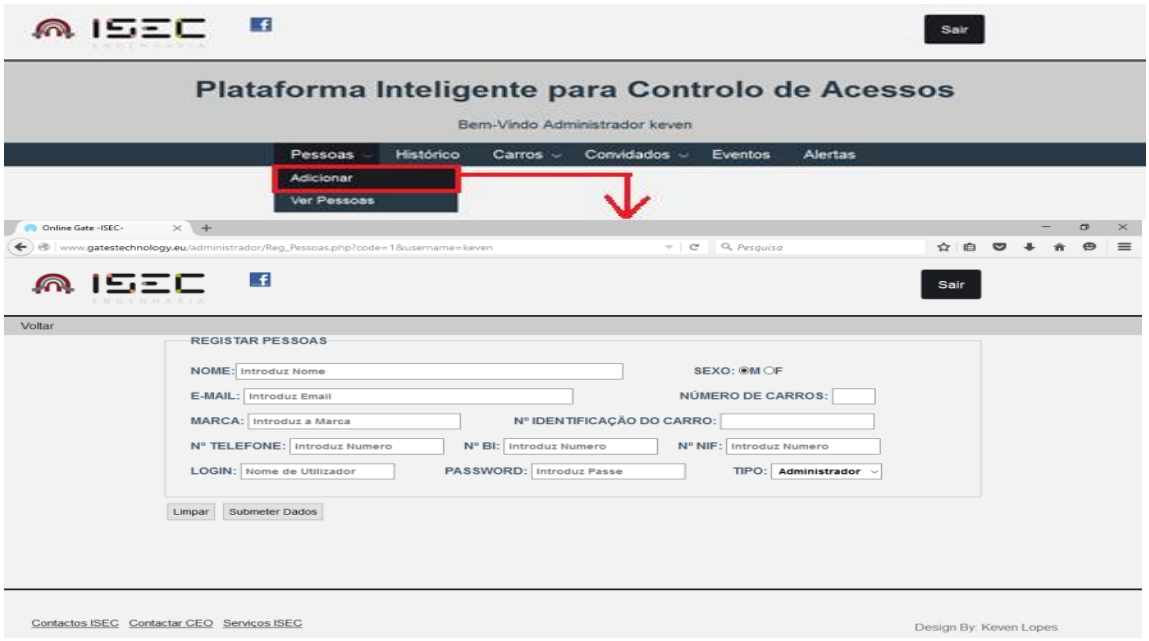


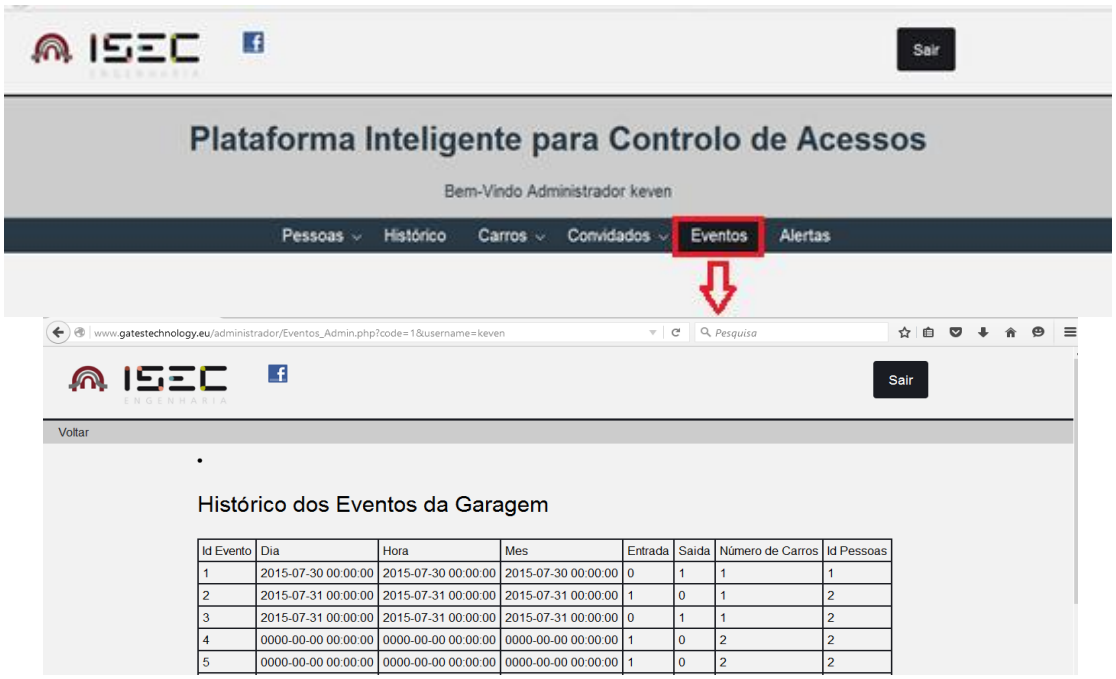
Figura 5-12: Página utilizada para registar um novo cliente ou administrador.

Para registar um novo cliente, o administrador vai precisar de alguns dados que são indispensáveis para efetuar um registo com sucesso de um novo cliente na base de dados. Esses são alguns dos campos que constituem a tabela "Clientes" da base de dados do *Gates Technology*. É preciso o nome completo do cliente, o sexo, o *e-mail* do cliente para algum caso de contatar esse cliente, o número de carros que o mesmo cliente possui (neste caso para um prédio, que depende da política do funcionamento do condomínio), para conseguir organizar melhor e ter uma estatística de quantos carros terão acesso à garagem. Vai ser preciso, igualmente: a marca do carro principal do cliente (para o caso de o cliente possuir mais do que um carro, basta que indicar a marca de um dos carros), o número de identificação do veículo, ou então, através do *VIN* (pois, no caso de possuir mais que um carro, poderá simplesmente indicar a identificação de um desses carros). O número de telefone, o número de identificação, e o número de identificação fiscal são, ainda, dados necessários para efetuar o registo. Por último, é atribuído um nome de utilizador e uma palavra passe à escolha do cliente (dados esse que podem ser posteriormente alterados), que são utilizados pelo utilizador para aceder ao *BackOffice*, assim o administrador só precisa indicar o tipo de utilizador, no caso pode ser cliente ou administrador.

Figura 5-13: Consultar o Histórico dos Clientes.

O administrador pode sempre consultar as ocorrências que são guardadas na base de dados quando de acede a garagem, através do botão "Histórico" (Figura *5-13*), o administrador tem diferentes opções. Onde consegue ver o geral dos históricos guardados, ou então, pode consultar focando diretamente na pessoa, no carro, ou ainda então, no convidado.

O administrador pode consultar todos os eventos registados pelo servidor da garagem, acedendo ao botão "Eventos" (Figura *5-14*), ou, pode ainda ver os registos dos alertas emitidos pelo servidor da garagem (Figura *5-15*).



Id Evento	Dia	Hora	Mes	Entrada	Saida	Número de Carros	Id Pessoas
1	2015-07-30 00:00:00	2015-07-30 00:00:00	2015-07-30 00:00:00	0	1	1	1
2	2015-07-31 00:00:00	2015-07-31 00:00:00	2015-07-31 00:00:00	1	0	1	2
3	2015-07-31 00:00:00	2015-07-31 00:00:00	2015-07-31 00:00:00	0	1	1	2
4	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	1	0	2	2
5	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	1	0	2	2

Figura 5-14: Exemplo de histórico emitido pelo servidor na garagem.

Número de Alerta	ID Pessoa	ID Convidado	ID Carro	Acontecimento	Hora	Dia	Mês
1	1	1	1	Erro ao Conectar a Internet	2015-07-30 00:10:00	2015-07-30 00:00:00	2015-07-30 00:00:00
2	2	2	2	Erro ao Reboot	2015-07-30 00:10:00	2015-07-30 00:00:00	2015-07-30 00:00:00

Figura 5-15: Alertas emitidas pelo servidor na garagem.

6. APLICAÇÃO *ANDROID*

Neste projeto foi desenvolvida uma aplicação *Android* como funcionalidade complementar para o controlo e acesso à garagem. A sua criação foi um dos maiores desafios encontrados durante o desenvolvimento deste projeto. Neste capítulo é descrita a sua criação e funcionamento. Informações adicionais, mais detalhadas, podem ser consultadas no Anexo 3 (Dufsfy, 2013).

6.1- Visão Geral

A decisão de criarmos uma aplicação *Android* deveu-se ao facto de nos dias de hoje existir um grande número de utilizadores de *smartphones.* A Figura *6-1* representa dados estatísticos dos últimos anos relativamente à utilização de *smartphones*.

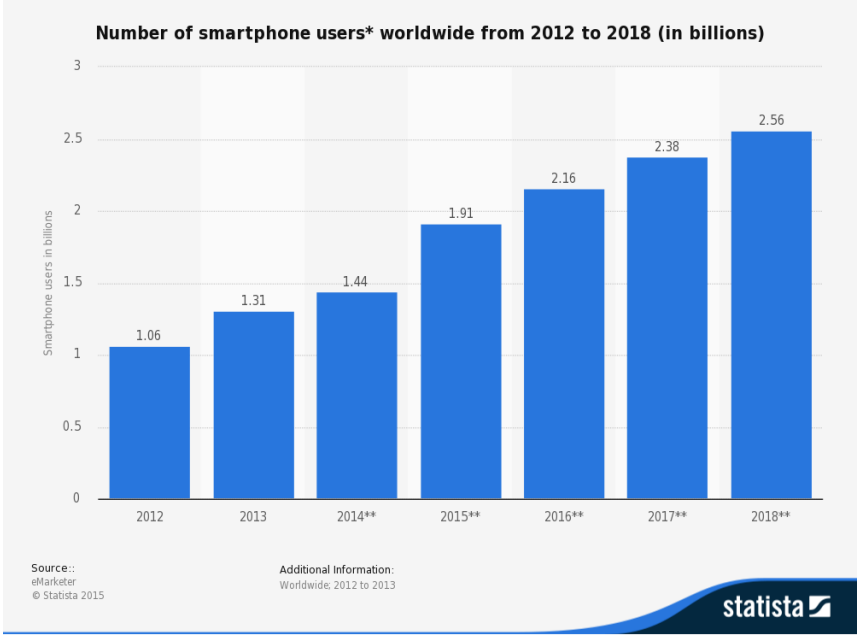


Figura 6-1: Estatística do número de utilizadores de *smartphones*, (Anexo 5 [w77]*).*

Com base nestes dados, podemos concluir que, a cada ano que passa, o número de utilizadores de *smartphones* vai aumentando substancialmente, representando, assim, uma grande vantagem para o caso de pretendermos injetar o nosso produto no mercado.

No desenvolvimento da aplicação *Android* tivemos a preocupação de garantir um nível de simplicidade elevado, boa qualidade de *design* e ser de fácil acesso.

6.1-1. *Download* da *App*

A aplicação que desenvolvemos encontra-se livre para o *download* na loja de aplicações *online* do Google, a qual pode ser acedida a partir de um *smartphone*, de um *tablet*, ou do *website* do produto.

Ao pressionar um dos botões de "*download*" existentes no *website*, é automaticamente redirecionado para o Google Play (loja *online* de aplicações), podendo, aí, efetuar o

download. Outra forma de fazer o *download* da aplicação consiste em ir diretamente ao Google Play e pesquisar pela aplicação. As palavras-chave para a pesquisa são: *gate*, *Gates Technology*, *remote gate*, garagem, controle de garagem, portões, ISEC.

6.1-2. Funcionalidades oferecidas pela *App*

Através da *App*, os utilizadores podem interagir com o sistema de diversas formas, nomeadamente:

- Abrir ou Fechar a porta da garagem. Isto pode ser feito com ou sem acesso à *internet*, desde que esteja conectado com a rede *wifi* da garagem.
- O Cliente ou Administrador consegue ter acesso aos dados que estão guardados na base de dados que está na *cloud*.

A *App* oferece também uma vertente de lazer, permitindo que as pessoas acedam à *internet*, façam pesquisas no *Google*, acedam a outras aplicações, ouçam músicas e joguem jogos.

Esta aplicação apresenta-se em dois idiomas: Português e Inglês.

6.2- Constituição da Aplicação

Nesta secção é apresentada a constituição da aplicação, nomeadamente os vários mapas de navegação.

Para o *design* foram utilizados o Adobe Photoshop e o Adobe Illustrator CS6.Todas as janelas da aplicação foram feitas com o *software* Adobe Photoshop. Este é caracterizado por ser um editor de imagens bidimensionais do tipo *raster* desenvolvido pelo Adobe Systems. É considerado líder no mercado dos editores de imagem profissionais, assim como o programa preferencial para a edição profissional de imagens digitais e trabalhos de pré-impressão (fonte: *https://pt.wikipedia.org/wiki/Adobe_Photoshop*). O Photoshop, tal como o Dreamweaver, pertencem à empresa Adobe.

Tivemos muita preocupação com o *design* da aplicação pois pretendíamos oferecer uma interface que fosse de fácil aceitação pelos potenciais utilizadores. Assim, desde o primeiro *layout* até ao último, manteve-se praticamente o mesmo *background*, o mesmo estilo de botão, a mesma configuração dos botões, de modo a ter uma confortabilidade estável e homogénea.

Pelo fato de haver atividades que se destacam mais do que outras, decidimos dividir o Mapa de Navegação em duas partes. A Figura *6-2* apresenta o mapa de navegação das atividades principais, isto é, as atividades mais importantes da aplicação (Liao, 2014).

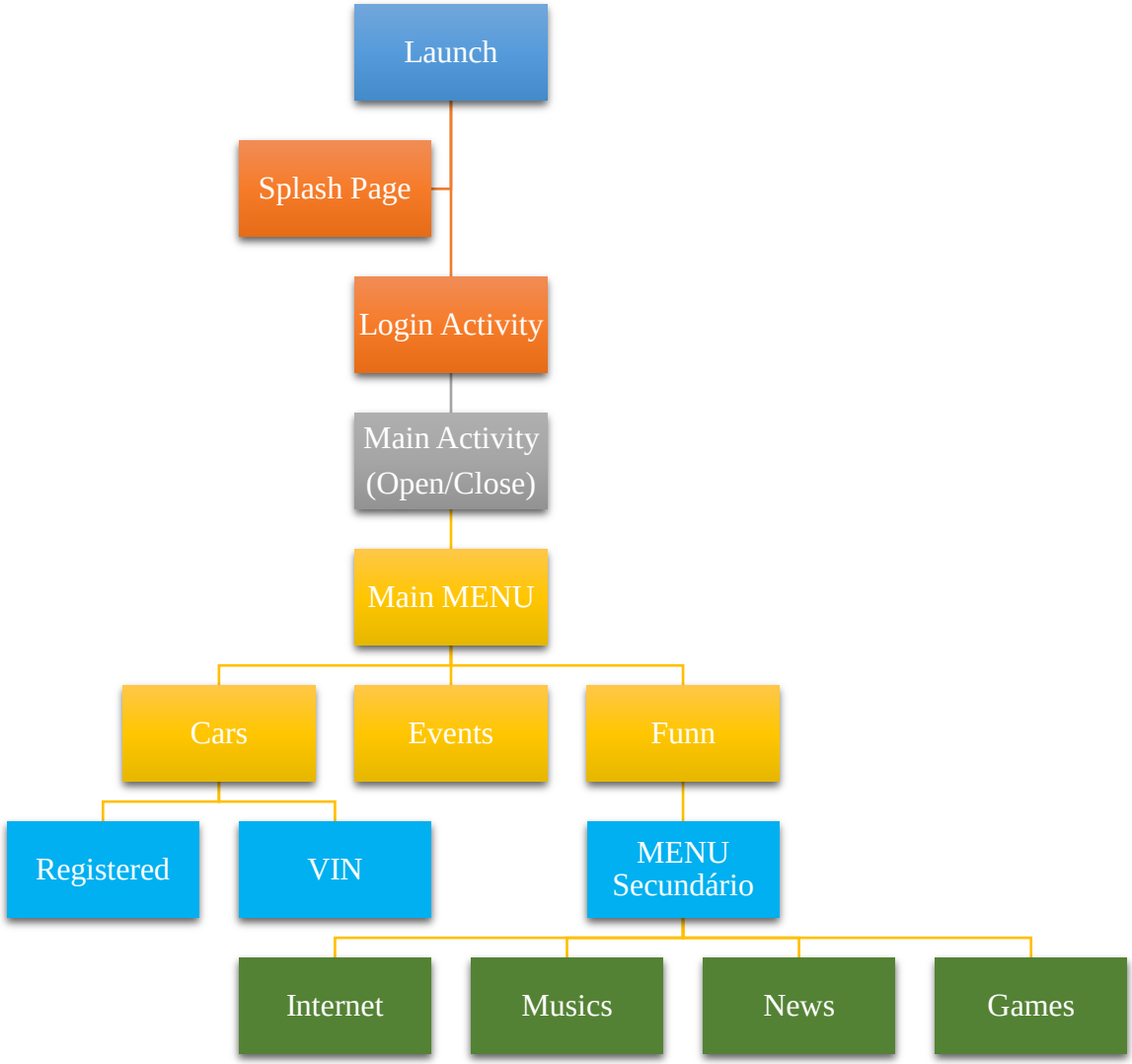


Figura 6-2: Mapa de navegação das atividades principais.

O mapa de navegação representado na Figura *6-3* representa as atividades extras que estão presentes nos respetivos *layouts* da aplicação.

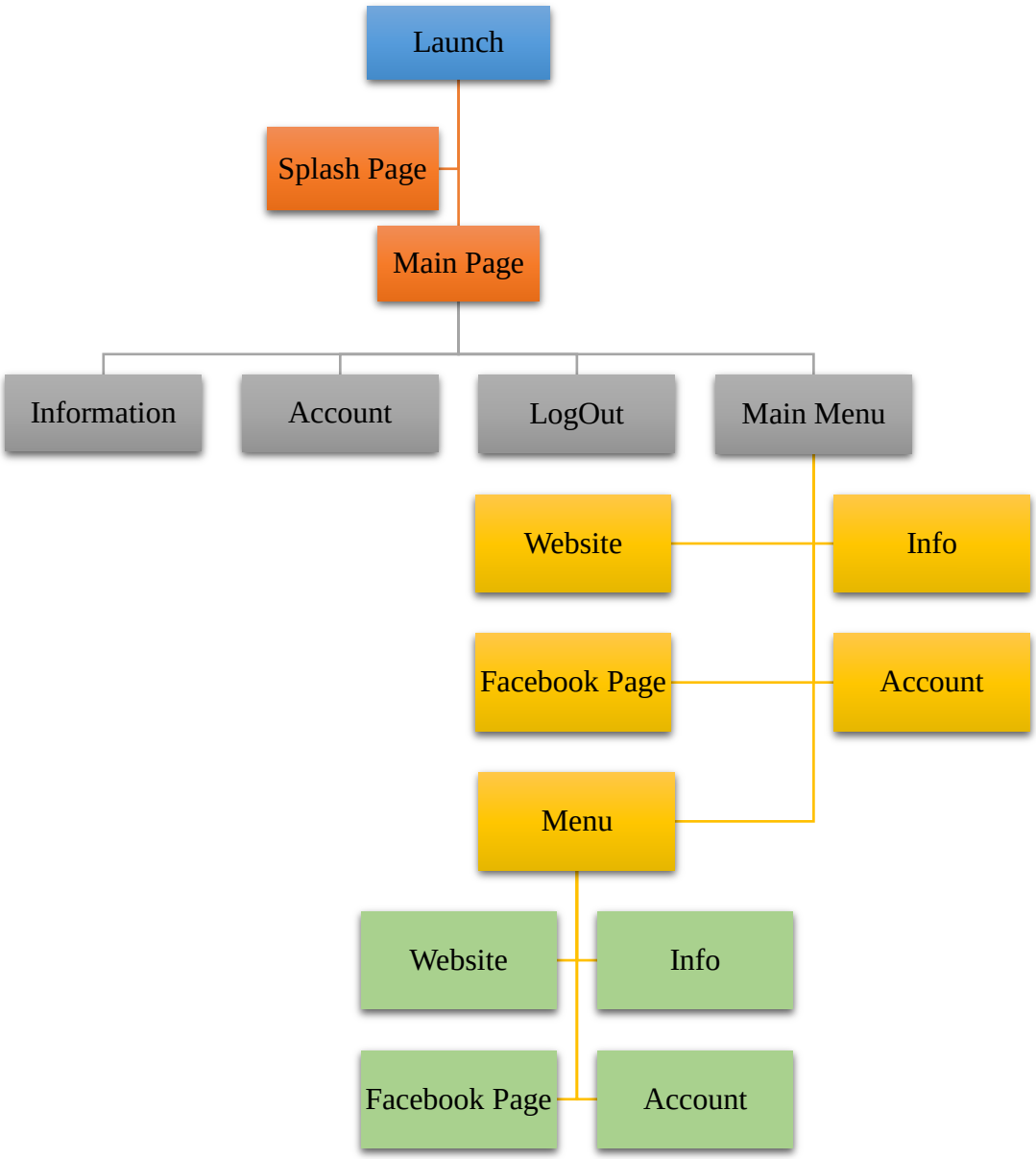


Figura 6-3: Mapa de navegação das atividades secundárias.

A Figura *6-4* representa o mapa de navegação do utilizador na página das definições de conta de cada utilizador. De realçar a possibilidade de contatar o Administrador (caso o utilizador seja um cliente), mudar a palavra passe ou o *email* associado à conta do utilizador.

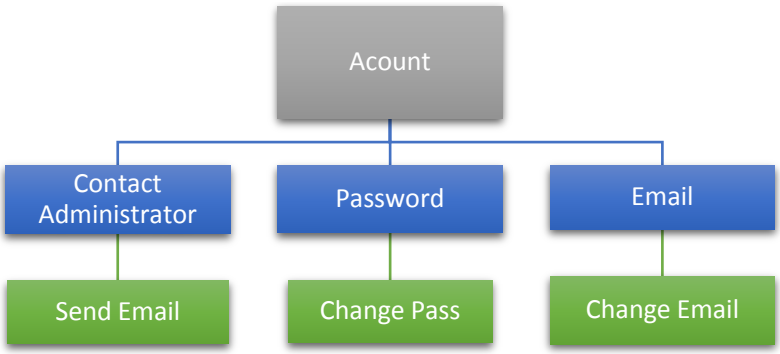


Figura 6-4: Mapa de navegação, definições de conta.

6.3- Funcionamento da Aplicação

Nesta secção é descrito o funcionamento da Aplicação *Android* desde que é iniciada (no *smartphone* ou *tablet)* e quais as atividades que a compõem.

A linguagem de programação utilizada foi o Java, tendo sido também utilizada a linguagem de marcação *XML* para a criação dos Layouts (páginas apresentadas pelas "*Activity*"). Cada atividade da aplicação é constituída por uma classe (onde está o código *Java*) e por um *Layout* (onde está o código *XML*), (Rogers, 2009), (Burd, 2011).

Ao iniciar a Aplicação no *smartphone* ou *tablet*, a primeira janela que abre mostra uma imagem de apresentação Figura *6-5* (esquerda), onde está presente o nome do Projeto "Plataforma Inteligente para Controlo de Acessos", juntamente com o logotipo que representa o produto e o logo do ISEC. Esta imagem permanece visível durante seis segundos - o que dá tempo para a leitura e para carregar todos os dados necessários para a aplicação.

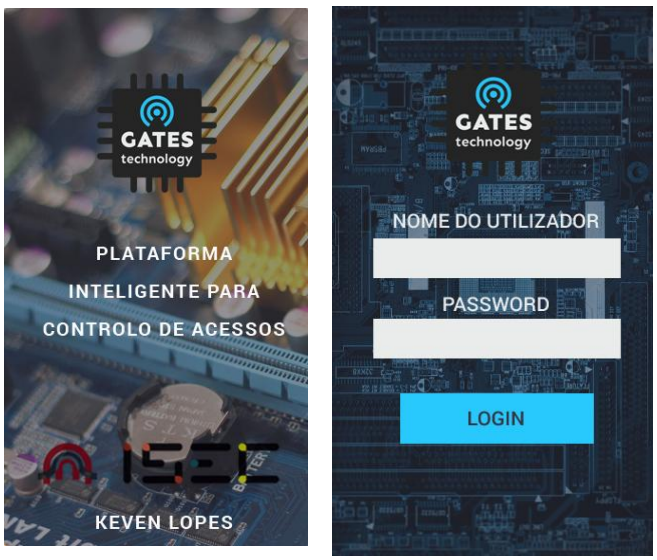


Figura 6-5: Maquete da página de apresentação e do Inicio de Sessão.

Depois de passar pela apresentação, será aberta a janela de *login*. Essa é a parte em que os utilizadores precisam do nome de utilizador e da palavra passe para terem acesso à totalidade das funções da aplicação. Caso o acesso seja feito por uma pessoa não

registada. Há também uma versão específica destinada para as pessoas que não possuem uma conta registada. Depois de efetuado o *login* é aberto a janela principal da aplicação (Figura 6-7), sendo aqui que o utilizador consegue aceder diretamente à garagem, abrindo ou fechando a porta. Dentro desta, o utilizador consegue ainda aceder às definições de conta e à página de informações. Por último, é também possível aceder ao Menu principal da aplicação.

O Menu principal (Figura 6-6) é a segunda mais importante Atividade da aplicação. Nesta parte o cliente ou administrador já consegue ter acesso a dados que estão guardados na base de dados.

Figura 6-7: Maquete da janela do Menu principal.

Figura 6-6: Maquete da janela principal da aplicação.

O utilizador tem três botões principais que lhe permitem abrir a janela dos Carros, a janela dos Eventos ou ir ao Menu secundário. É também possível ao utilizador aceder as definições de conta e partes extras tais como ir para o *website*, visitar a página do Facebook do *Gates Technology* ou ir à janela de informações.

Se o utilizador pressionar no botão "Carros" será aberta uma janela em que o utilizador consegue também visitar a página do Facebook do *Gates Technology*, ir ao *website* ou à janela de informações. Nessa janela, os botões mais importantes são "Registados" e "*VIN*". Caso pressione no botão "Registados", são apresentados todos os carros que estão registados na base de dados da garagem – no caso de um cliente este consegue somente ver os carros associados à sua conta; no caso de um administrador, este já consegue ver a informação de todos os carros. No caso de pressionar o botão "*VIN*", é aberta uma nova janela em que são apresentados os números de identificação de cada carro. De forma análoga à janela anterior, esta também funciona de modo diferente para clientes e administradores. Para o botão "Eventos" são apresentados todos os eventos registados na base de dados da garagem.

Em todas estas páginas, é possível consultar as definições de conta, aceder ao *website*, consultar a página do *Facebook* e também ir à janela de informação.

6.4- Criação da Aplicação e *Softwares* utilizados

A criação da Aplicação *Android* foi um grande desafio, pelo facto de exigir muitos conhecimentos de programação. Esta etapa do Projeto foi ultrapassada com sucesso, o que exigiu dedicar muito tempo de estudo a linguagens de programação, principalmente ao *Java* e a linguagem de marcação *XML*.

Na versão atual do trabalho, a aplicação suporta somente dispositivos de versão *Android*, ou seja, ainda não foi desenvolvida para versões *IOS*, *Blackberry* ou *Windows Phone*, que são os objetivos para concretizar como trabalho futuro.

O desenvolvimento desta aplicação utilizou diferentes ferramentas de *software* tais como o *Eclipse* e o *Android Studio*:

- O *Eclipse* é um *Integrated Development Environment* (IDE) para desenvolvimento com Java, porém suporta várias outras linguagens a partir de plugins como *C/C++*, *PHP*, *ColdFusion*, *Python*, *Scala* e plataforma *Android*. Ele foi feito em *Java* e segue o modelo *Open Source* de desenvolvimento de *software*. Atualmente faz parte do *kit* de desenvolvimento de software recomendado para criadores de *App's Android* (*https://pt.wikipedia.org/wiki/Eclipse_(software)*).
- O *Android Studio* é um *IDE* para o desenvolvimento para a plataforma *Android*. Foi anunciado em 16 de maio de 2013 na conferência do Google I/O de 2013 pelo gerente de produto. É grátis e está disponível sobre a licença do Apache 2.0 (fonte: *https://en.wikipedia.org/wiki/Android_Studio*).

Existem pequenas diferenças entre o *Eclipse* e o *Android Studio*. Apenas por preferência, utilizámos mais o *Eclipse* por o considerarmos mais organizado, mais fácil de utilizar e também por ser a ferramenta com que mais nos identificámos para a programação da aplicação. No entanto o *Android Studio* foi também muito utilizado, sobretudo devido à rapidez do emulador que utilizámos para fazer os testes, e também pelo facto de algumas bibliotecas de *design* não serem suportadas pelo *Eclipse*.

Os códigos que constituem a aplicação são extensos (Anexo 3) e por isso de seguida, vamos referir apenas as partes mais importantes, realçando o ciclo de vida de cada atividade que constitui a aplicação (Figura *6-8*).

Depois de ser iniciada, temos três funções principais que são "chamadas" - o "*onCreate*()", o "*onStart*()" e o "*onResume*()", sendo estas as funções que iniciam a atividade. Depois de se proceder ao início de outra atividade, temos as seguintes funções: "*onPause*()" e "*onStop*()". Nesta parte, a atividade já não é visível por parte do utilizador, no entanto, ainda está ativo no *background*. Se outra atividade precisar de utilizar a memória *RAM*, então o processo é terminado, e assim, para iniciar a aplicação novamente será chamada a função "*onCreate*()" através da função "*onRestart*()". Para terminar a atividade, é utilizada a função "*onDestroy*()".

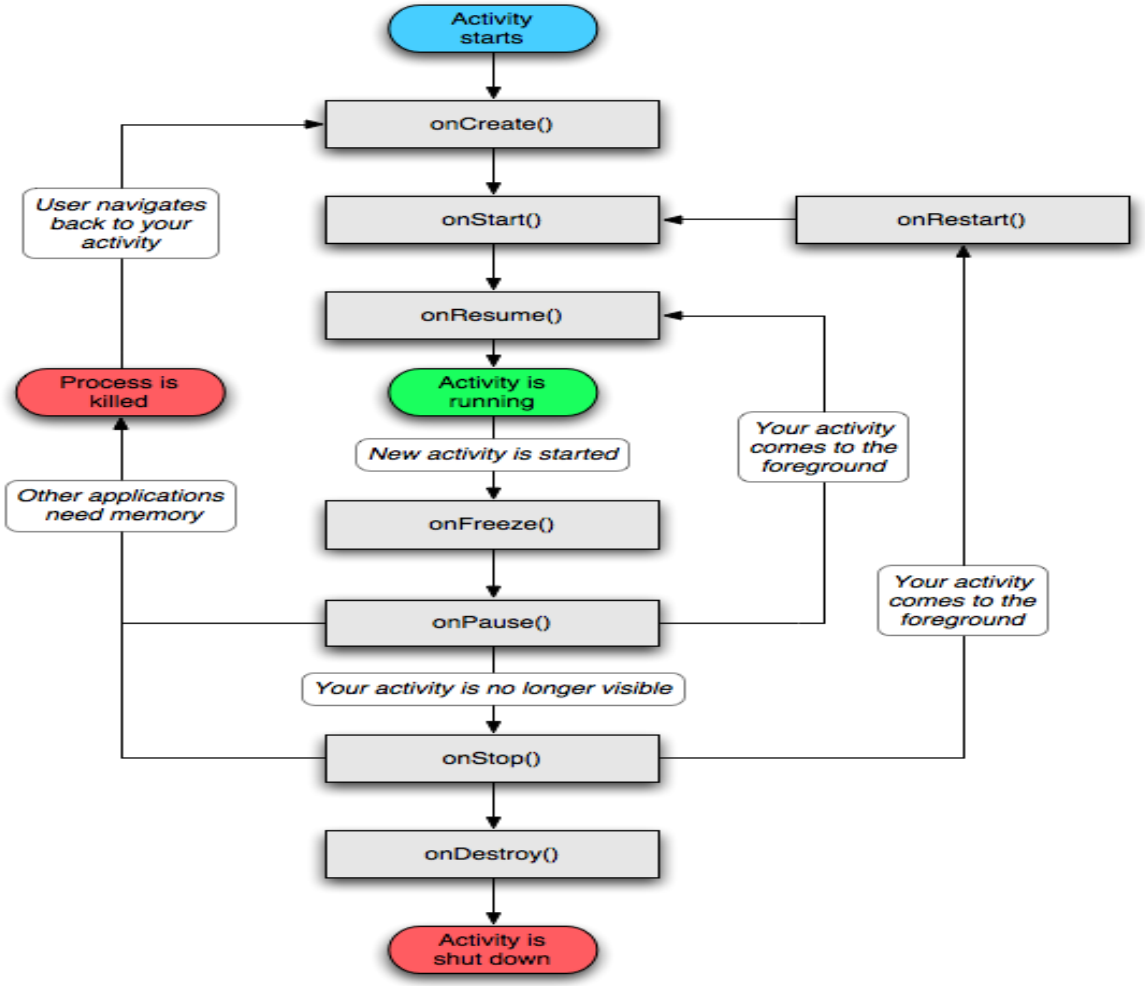


Figura 6-8: Ciclo de vida de uma Atividade de uma aplicação, Fonte (Anexo 5 [79])[8].

Mais informações sobre a Aplicação *Android* do *Gates Technology* podem ser encontradas no Anexo 3.

[8] ANEXO 5 – Sítios da Internet Relevantes

7. SISTEMA *Gates Technology*

No funcionamento do sistema *Gates Technology* é essencial a comunicação entre diferentes dispositivos que o constituem. Foram desenvolvidos dois protótipos de comunicação, em que um desses protótipos se destina ao veículo de um cliente e o outro é utilizado como um servidor, que está instalado na garagem. Informações detalhadas podem ser consultadas no Anexo 4.

7.1- Visão Geral

Foram utilizados diferentes módulos de comunicação e módulos baseados em microcontroladores para efetuarem os diferentes pedidos, que serão efetuados durante as comunicações, esses equipamentos que compõem os protótipos desenvolvidos; são eles:

- *Arduino Uno*,
- Módulo *Wireless* ESP8266,
- Módulo *Wireless* ESP8266 E12,
- FTDI 232,
- Fonte de Alimentação de 3.3v.

Para o *Hardware* também vamos ter um simulador de barramento *CAN* (Figura *7-1*) e um conversor *Serial-CAN*. A placa SBC2680 é constituída por diferentes componentes.

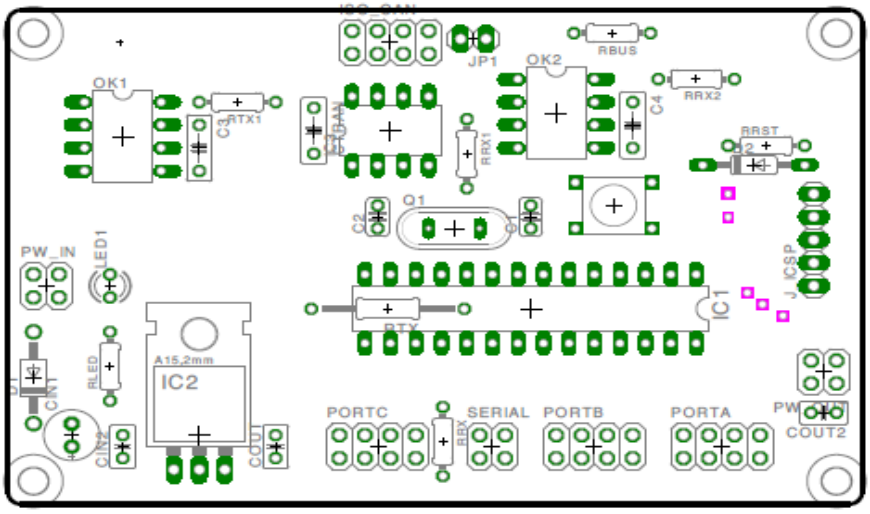


Figura 7-1: Simulador *CAN* (ECU2680).

Composição placa SBC2680:

- Controlador PIC18F2680, com controlador *CAN* integrado;
- Cristal de 10 MHz;
- Portos A, B e C disponíveis em conectores de 8 pinos;
- Porto serie num conector de 4 pinos (TX, RX, VCCe GND);

- *Transceiver CAN*, com isolamento *galvanico* do bus *CAN*, disponível num conector de 4 pinos (+5V, TX, RX, GND);
- Regulador de 5V (aceita alimentação DC de 7 a 12V);
- Pino para programação externa (ICSP).

Dentro deste capítulo é referido cada um desses componentes utilizados, descrevendo e analisando o funcionamento de cada um deles em específico. Também será referido para os diferentes circuitos criados a programação desses módulos. As diferentes linguagens de programação utilizadas são: *Lua*, *C++*, entre outras. Na programação dos módulos, utilizou ainda Comandos AT, (Schuytema & Manyen, 2005).

Esses protótipos desenvolvidos são os pontos principais que se referem à *Gates Technology*, pois, através deles, temos diferentes tipos de acesso e controlo da garagem, tornando assim, o nosso produto algo diferente de tudo o que já existe no mercado atualmente.

Para a parte do cliente, vamos utilizar o *Arduino*, o ESP8266 E12, utilizado também o FTDI232. O equipamento desenvolvido permite fazer uma leitura do número de identificação do veículo (*VIN*), e envia-o através de *wifi* para o servidor presente na garagem. Este procedimento é feito sem nenhuma interferência do utilizador. (Espressif Systems IOT Team, 2015).

Para a parte do servidor vamos ter também um *Arduino*, um ESP8266, assim como, um FTDI232. Esse é o equipamento que vai estar conectado à *internet*, oferecendo, desta forma, o controle e acesso à garagem para qualquer cliente, estando esse em qualquer parte do mundo.

Este protótipo é utilizado, também, para verificar o número de identificação do veículo (*VIN*) recebido entre a comunicação do cliente e o servidor, que por sua vez vai analisar a base de dados. Caso corresponda a algum *VIN* já previamente registado, será atribuído o acesso ao cliente. O servidor também vai fazer "*host*" a uma página *web* que controla a garagem, a qual pode ser acedida mesmo estando sem conexão à *internet*.

A comunicação dos dois dispositivos *wireless*, para diferentes testes realizados (em espaço aberto), pode-se dizer que tem um funcionamento que abrange uma vasta área, indo de 150 a 300 metros no máximo. Essa área depende de diversos fatores tais como: a temperatura, a humidade, existência ou não de obstáculos, entre outros. Essa distância, que é abrangida pelos módulos de comunicação *wireless*, pode diminuir quando estamos numa cidade. O *Hardware* presente nos veículos dos clientes foi programado para que, sempre que o "*Access Point*" esteja ao alcance, o cliente vai tentar estabelecer comunicação. A rede definida para a comunicação é a "*Gates Technology*". O cliente estará sempre a fazer uma leitura das redes disponíveis, até se interligar com a garagem. Para evitar uma falha de comunicação quando estamos perante duas garagens que se utilizam o *Gates Technology*, o cliente apesar de estar sempre a tentar comunicar com uma rede de nome "*Gates Technology*", mesmo encontrando duas redes desse nome disponível, só comunicará com uma em específico. Isto acontece, pois, o cliente consegue

identificar sempre com qual é que se pode comunicar através do *MAC Adress* do Servidor.

O que mais preocupou na parte dos clientes foi o consumo de energia por parte do dispositivo, que é algo de muito crítico, pois depende muito da bateria do carro. O problema que se tem é o facto de o cliente estar sempre a fazer uma leitura de sinais *wifi*, para tentar comunicar com a garagem, o que levaria a um constante consumo e desperdício de energia. Desta forma, sabendo que não é necessário estar sempre a tentar estabelecer contacto com o servidor, demos muita importância a um dispositivo inteligente, disponibilizando ao cliente condições de saber propriamente quando comunicar com o Servidor. Outra referência a ser feita é o fato de conseguirmos controlar de forma *online* e *offline* (com ou sem conexão a *internet*). Isto é feito pelo fato de o Servidor (Protótipo instalado na garagem) contem uma página *web*, que somente é possível aceder se o utilizador estiver conectado a rede *wifi* da garagem. Deste modo pode abrir ou fechar a porta da garagem (funcionamento sem conexão a *internet*). Para o funcionamento com *internet*, o dispositivo (protótipo instalado na garagem) vai ler através da página *www.gatestechnology.eu* os valores de dois botões (Abrir ou Fechar) para efetuar os pedidos dos diferentes clientes. A diferença é que quando o controle é feito através da página *online*, é automaticamente feito o registo de todos os eventos desse cliente. Quando é feito *offline* já não é possível fazer esses registos.

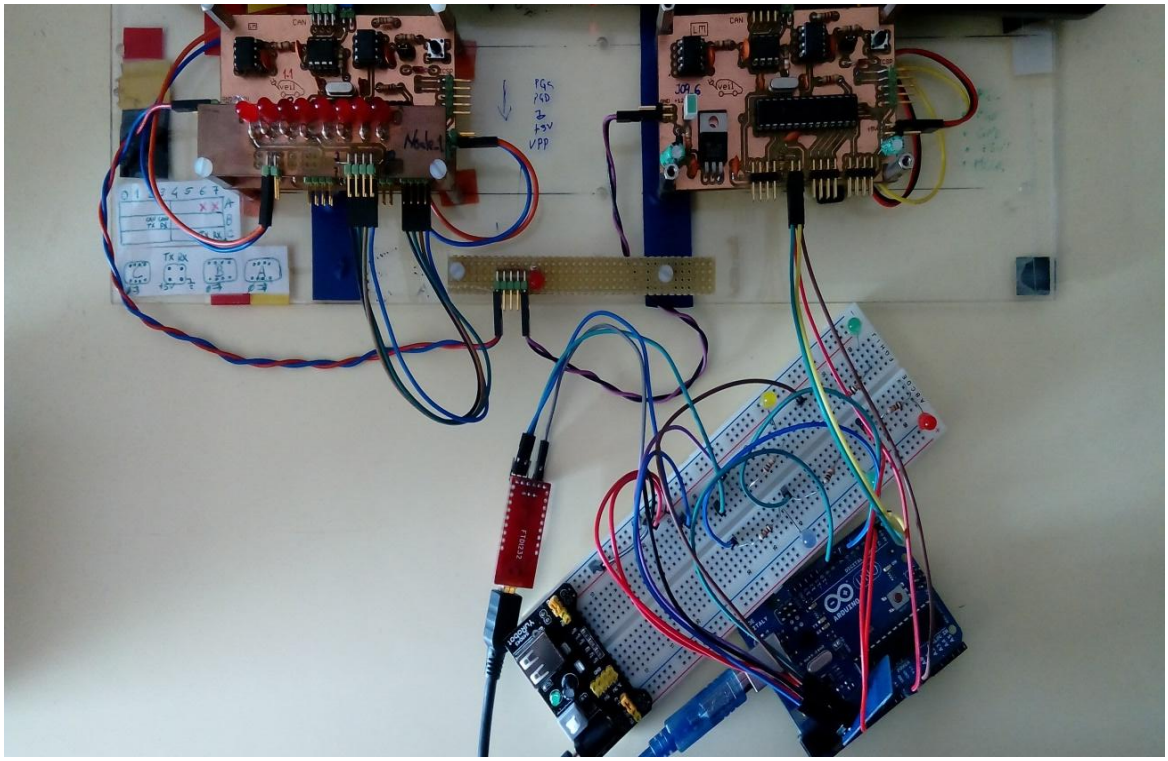


Figura 7-2: *Setup* experimental cliente.

7.2- Procedimento Inicial – *Firmware update*

Este é o procedimento efetuado antes da programação dos módulos de comunicação *wireless* ESP8266. Depois de adquirir esses módulos da fábrica, verificámos que estes já vêm com uma frequência de comunicação diferente da qual vamos utilizar diferentes programas dos protótipos de comunicação. Para termos então esses módulos configurados à frequência desejada - 9600 *bps* - temos um procedimento a seguir.

Vamos utilizar nesse caso o módulo ESP8266 juntamento com o FTDI 232 que é o conversor *USB-série*.

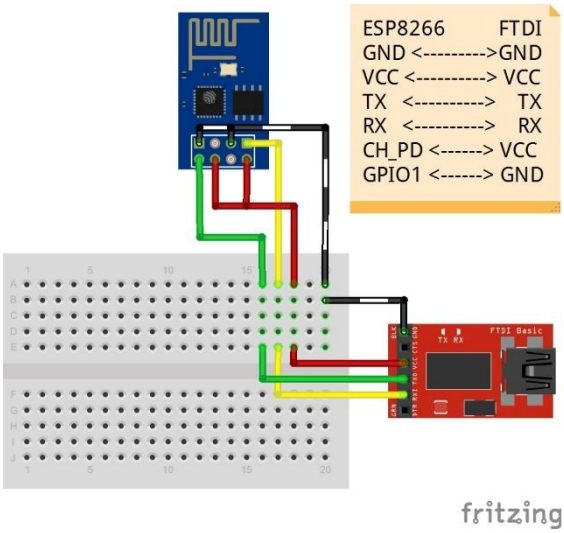


Figura 7-3: Esquema de ligação para fazer o *flash* do *firmware*.

A Figura *7-3* indica o esquema de ligação entre o módulo de comunicação ESP8266 e o FTDI 232 quando se pretende fazer o *flash* do *Firmware*, uma vez que o ESP8266 funciona a uma tensão de 3.3 V. Se o FTDI232 não estiver configurado a 3.3 V teremos que ter um circuito para converter o sinal enviado do FTDI, através do pino "TX" (pino utilizado pelo FTDI 232 para enviar os dados). Esse sinal geralmente vem com uma tensão de 5 V. Como já se referiu, podemos utilizar qualquer regulador de tensão à escolha. No nosso caso vamos utilizar o regulador representado pela Figura *7-4* para converter esse valor de tensão.

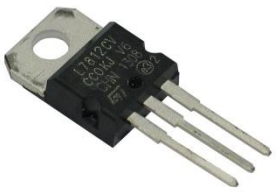


Figura 7-4: Regulador de tensão LM7812.

Falando agora da ligação, temos o condutor de cor preto que indica o GND (*ground*). Este se liga ao GND do ESP8266 e do FTDI 232; o condutor vermelho do FTDI 232 indica o VCC, que vai ligar ao VCC do ESP8266 e ao CH_PD; o condutor verde do ESP8266 é o

pino TX para enviar dados para o FTDI 232, que liga ao pino RX do FTDI 232. Por fim, temos o condutor de cor amarela que no FTDI 232 representa o TX e vai ligar ao pino RX do ESP8266.

Dependendo da linguagem de programação que se pretende utilizar, assim vai ser o *firmware* a utilizar. Neste caso temos "*nodemcu_integer_0.9.6-dev_20150704*", que é a versão mais recente do *firmware* do NodeMCU. Este *firmware* está disponível *online*, e, ao fazer-se o *upload* desse *firmware* para o ESP8266, vai fazer com que o módulo seja programável em Lua e que funciona a 9600 *bps*.

Para o caso de utilizar o *firmware* "*v0.9.2.2 AT Firmware*", que representa a versão mais recente do *firmware* para comandos AT, disponível também *online*, vai fazer com que o nosso módulo somente seja programável com comando AT e que funciona a 9600 *band* juntamente com o *Arduino*.

A Figura *7-5* e Figura 7-6 indicam os programas utilizadas para fazer o *upload* dos *firmwares* para os módulos ESP8266 através do FTDI 232 que vai estar ligado a porta *USB*. Escolhemos, então, a porta *COM* que está ligado o FTDI e o *firmware* desejado.

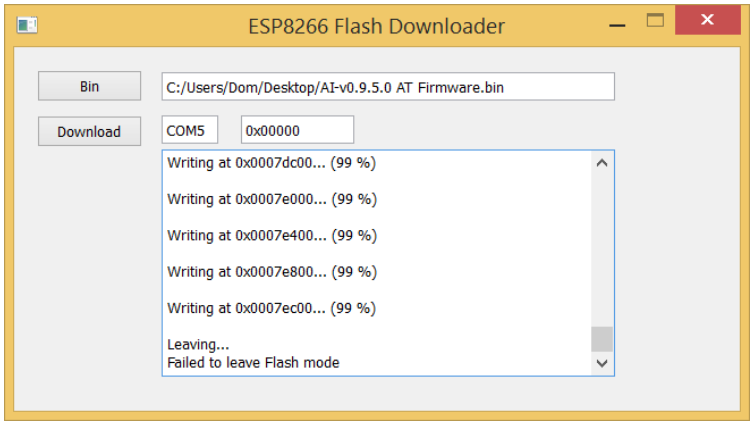


Figura 7-5: ESP8266 *flasher*.

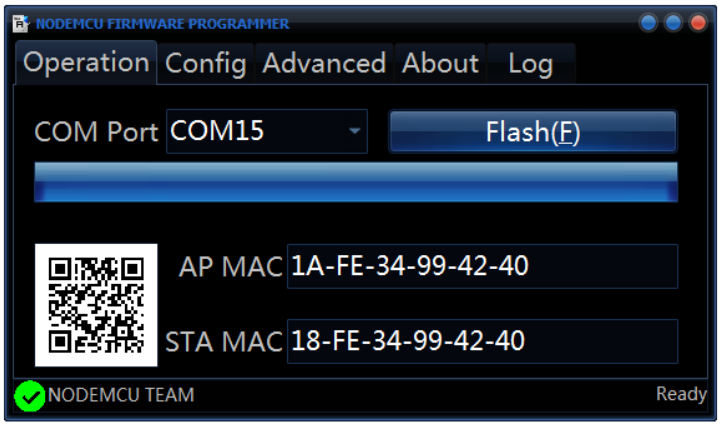


Figura 7-6: NodeMCU *Firmware Programmer*.

7.3- Enquadramento

7.3-1. Simulador Rede *CAN* programação

A programação do simulador da rede *CAN* é feita seguindo um procedimento estabelecido para o melhor funcionamento do mesmo. Em primeiro vai ser preciso o *IDE* que é utilizado para o desenvolvimento dos programas. O *software* que teremos que utilizar é o "*mikroC PRO for PIC*". Este *software* está disponível no sítio da *internet* presente no Anexo 5[30]. Depois de ter instalado e pronto para trabalhar, criamos um novo projeto, como mostra a Figura *7-7*.

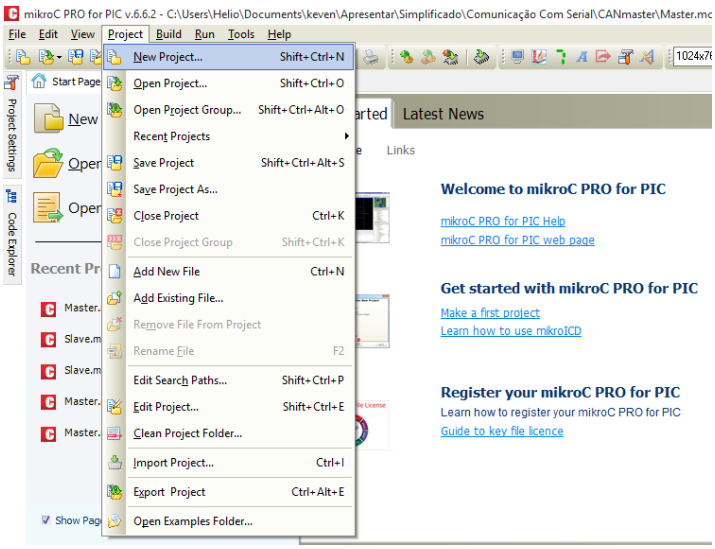


Figura 7-7: Criação de um novo projeto no mikroC PRO for PIC.

Depois disso só teremos que escolher o nome do projeto e o destino onde será guardado. Ao ter então desenvolvido o *script* desejado, é preciso agora fazer o "*Build*" do código, isto que vai gerar um ficheiro de formato "*.hex*" que será carregado para o microcontrolador.

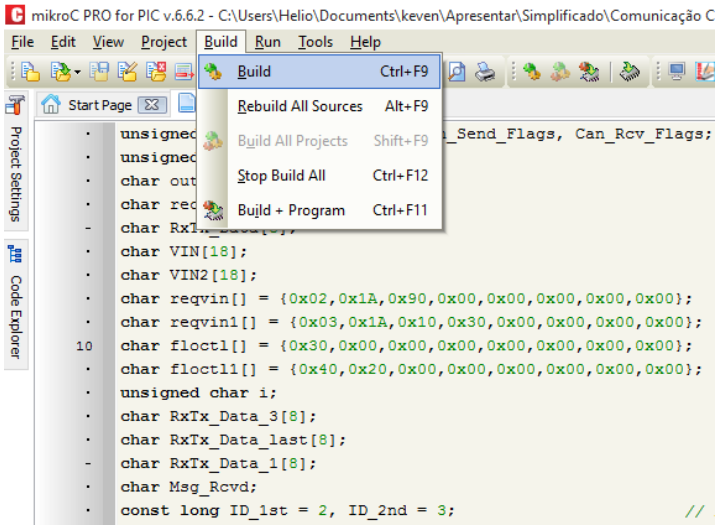


Figura 7-8: *Build* do *script* para gerar ficheiro "*.hex*".

Agora para carregar o ficheiro para o simulador vamos utilizar o MPLAB ICD 2, (Figura *7-9*).

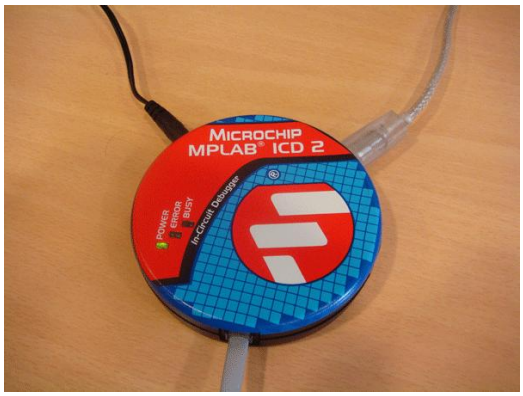


Figura 7-9: MPLAB ICD 2.

Também vai ser preciso o MPLAB *IDE*, em que foi utilizado em específico à versão 8.92. Esse *software* está disponível na *internet*. Depois de ter o nosso ficheiro "*.hex*" vamos então começar o procedimento de fazer o *upload* do código para o simulador. A Figura *7-10* e Figura *7-11* representam a ordem a seguir para fazer o *upload*. Em que primeiro temos que importar o ficheiro para o ambiente de trabalho do MPLAB, depois selecionamos o tipo de programador a utilizar (no nosso caso é o MPLAB ICD2) e por fim é só carregar no botão da Figura *7-12* para terminar o *upload*.

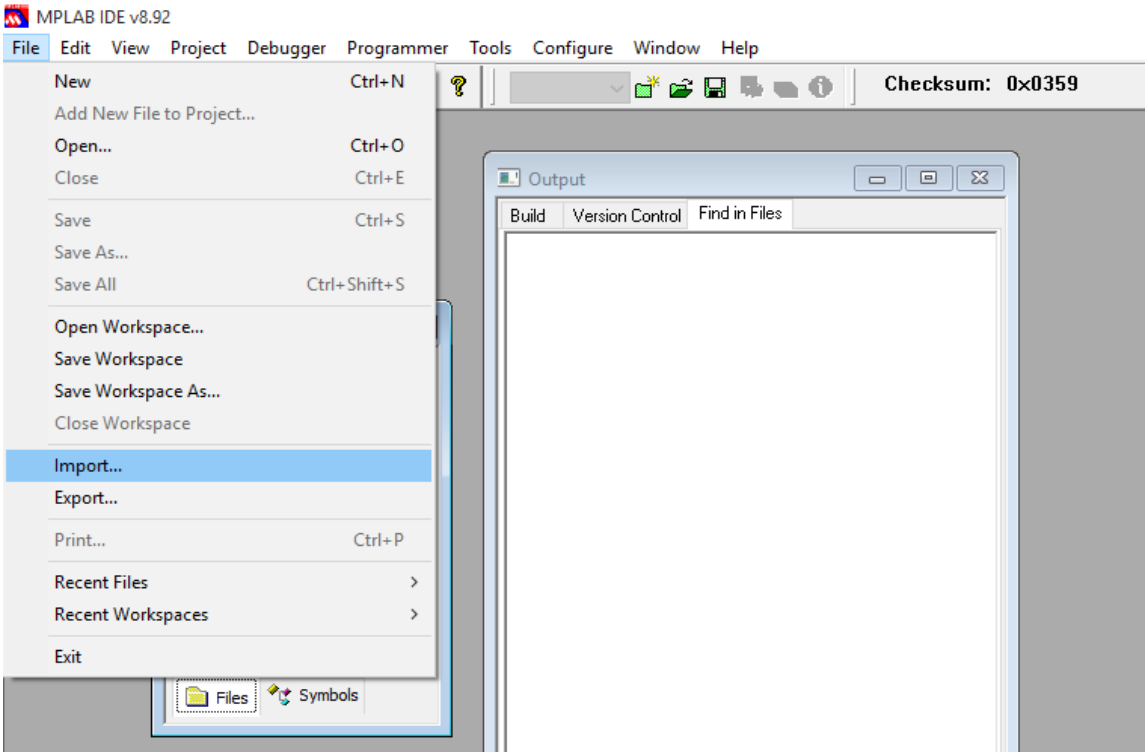


Figura 7-10: Fazer a importação do *Script*.

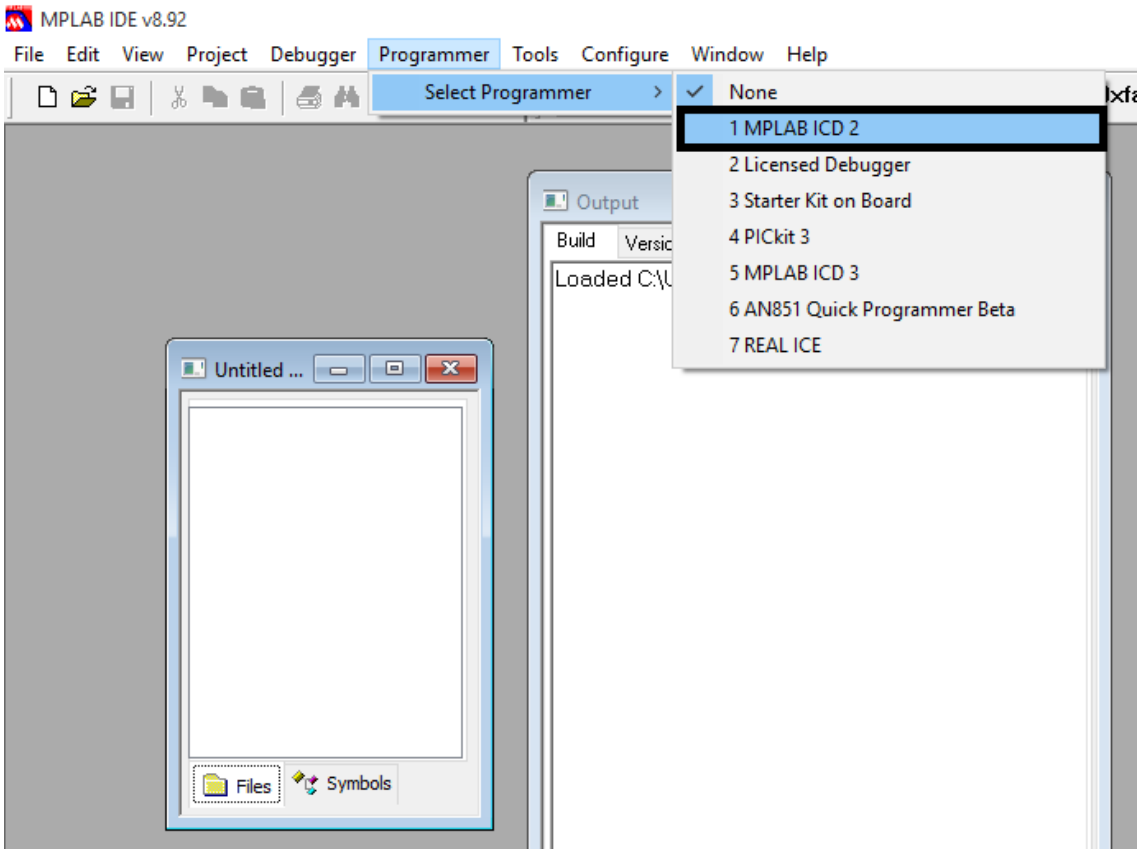


Figura 7-11: Selecionar o programador a utilizar.

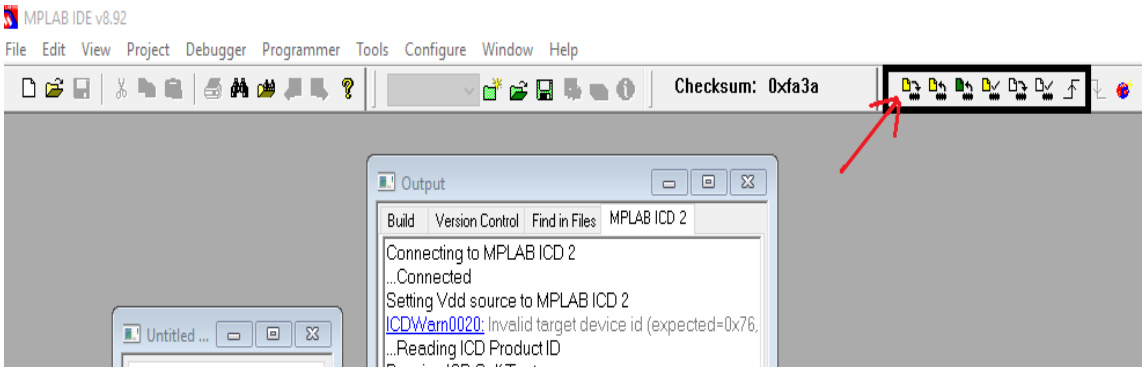


Figura 7-12: Fazer *download* do *script* para o simulador.

7.4- Clientes

O dispositivo que vai ser instalado no carro dos clientes é constituído por um *Arduino*, um módulo de comunicação *wireless* ESP8266, um FTDI 232 e ainda um módulo para fazer a interface série-*CAN* (para se ligar à rede *CAN* do carro, através da ficha *OBD* 2). A Figura *7-13* representa o diagrama de comunicação entre o cliente e o servidor, destacando neste caso a parte que constitui o cliente.

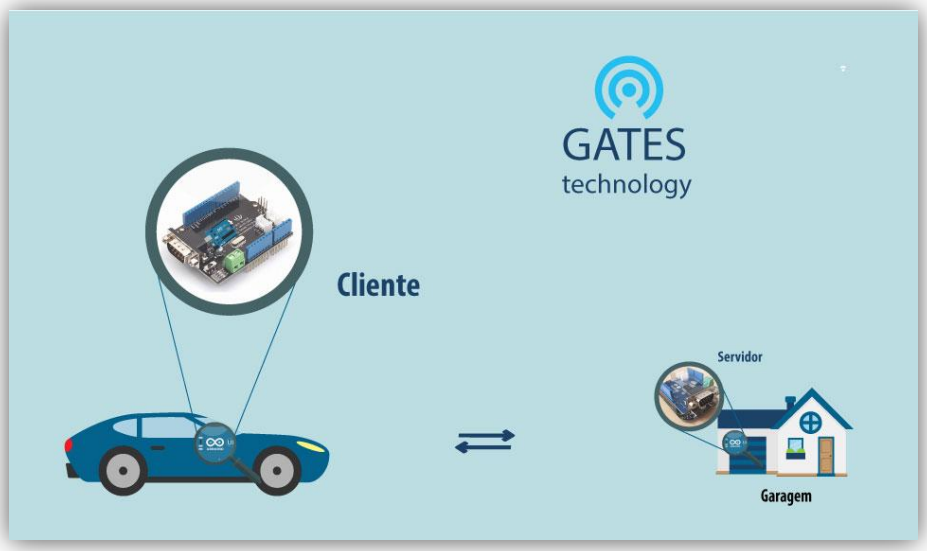


Figura 7-13: Diagrama de comunicação cliente para servidor.

A Figura *7-14* representa o circuito de comunicação presente no veículo dos clientes, onde que temos *Arduino* ligado juntamente com o módulo de comunicação ESP8266.

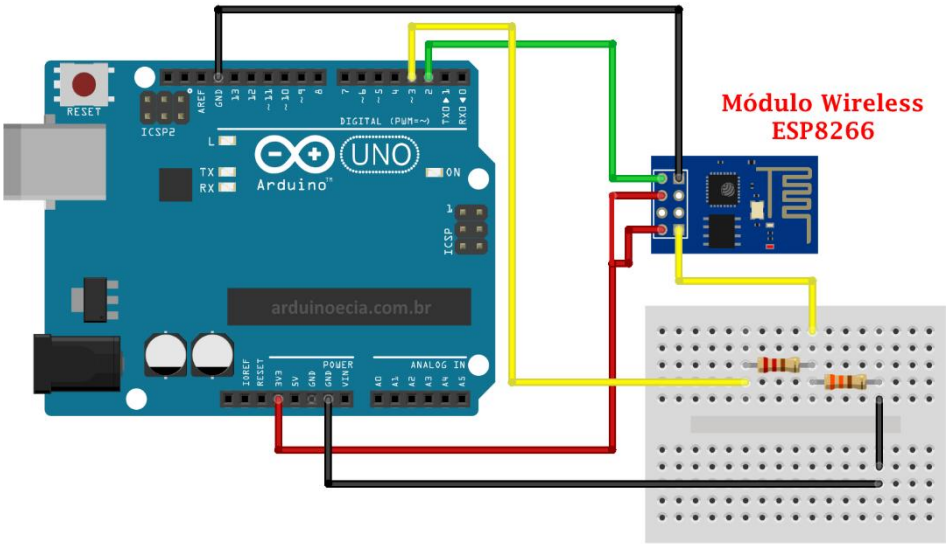


Figura 7-14: Circuito *Arduino* com o ESP8266.

Falando agora, um pouco, sobre as ligações do *Arduino* e o ESP8266, começamos por falar sobre o condutor representado pela cor amarela que está ligado no pino 3 do *Arduino* e o pino "RX" do ESP8266. RX é o pino de comunicação do ESP8266 em que recebe os dados. Sabendo que o ESP8266 funciona a 3.3 V, então, é feito um circuito para reduzir a tensão de 5 V que vem do *Arduino* para 3.3 V. O condutor da cor verde é o utilizado para enviar dados do ESP8266 para o *Arduino*, ligado ao pino "TX" do módulo (porta serial essa utilizada para enviar dados), e, o pino digital 2 do *Arduino* é a que está configurada para receber os dados. Por último, temos os condutores vermelho e preto que representam a alimentação (3.3 V) e o GND (*ground*), (Evans, 2011).

No carro, vai estar presente esse protótipo em que, é utilizado o *Arduino* para comunicar em tempo real com a rede *CAN Bus*, através da porta *OBD* II de modo a estar constantemente a fazer a leitura do número de identificação do veículo. De seguida, o *Arduino* inicia uma comunicação com o módulo ESP8266, enviando como variável o valor do número de identificação do veículo (*VIN*). O ESP8266, da mesma forma que vai recebendo o *VIN*, vai sempre atualizando esse valor. Quando o ESP8266 estabelecer uma comunicação cliente-servidor, o dispositivo cliente vai enviar por *wireless* o valor atualizado do número de identificação do veículo para o servidor.

```
while (FOREVER){
1. escuta mensagens no bus;
2. se recebeu mensagem, verifica se pede VIN;
3. se pediu VIN envia as mensagens CAN com ID, cujo
conteúdo é (descrever o conteúdo e como este constitui o VIN);
}
```

Script 7-1: Algoritmo do simulador do carro.

Agora para o conversor Série – *CAN*, composto por duas *boards* – uma SBC2680 com porto série ligada a um *Arduino*. A sua programação segue o seguinte algoritmo, *Script 7-2*.

```
while (FOREVER){
1. escuta porto serie
2. se recebeu letra ´1' então envia mensagem CAN com ID a pedir VIN
3. se recebeu mensagem CAN verifica qual é e assembleia o VIN
(quando receber as duas mensagens) e ativa flag VIN_recebido = 1
4. se flag VIN_recebido == 1 envia o VIN pela porta série para
Arduino
}
```

Script 7-2: Algoritmo do conversor Série - *CAN* ligado ao *Arduino*.

7.4-1. Comunicação Módulo - Carro

Esta é a parte em que comunicamos diretamente com a rede *CAN Bus* do veículo. A comunicação é feita de modo a obter o número de identificação do veículo (*VIN*).

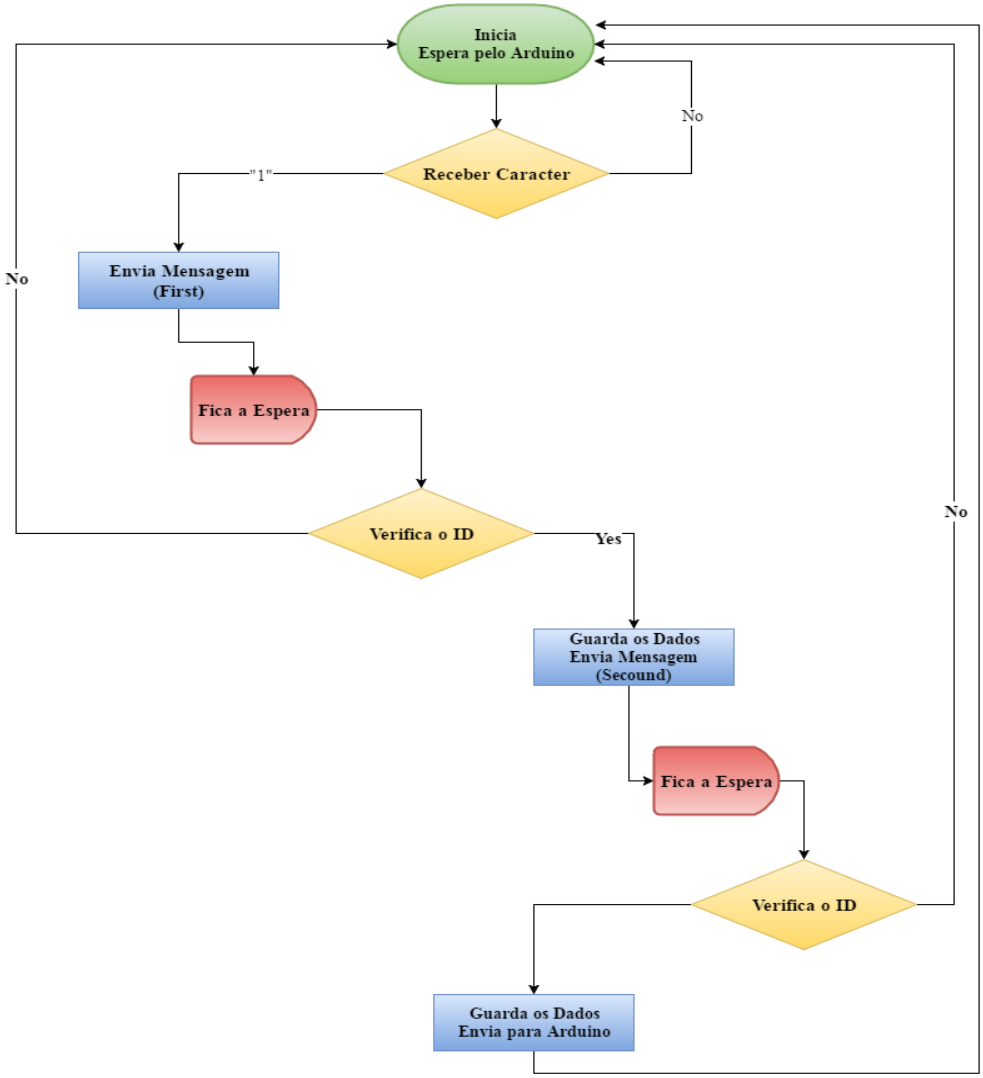


Figura 7-15: Fluxograma do Conversor Serial - CAN.

- **Pedido do *VIN* (Simulador)**

No protocolo há uma sequência entre o nó que pede o *VIN* e o módulo que responde, que é a seguinte (Forma real de fazer o *request* ao *VIN*):

- Pedido de envio do *VIN* – com *CAN ID* 0x7E0 e *payload* 0x02, 0x09 e 0x02
- Resposta do carro – 1ª mensagem; tem *CAN* ID 0x7E8 e contém primeiros 3 caracteres do *VIN* (nas 3 últimas posições do *payload*)
- Pedido de envio do resto dos dados do *VIN* – mensagem com *CAN* ID 0x7E0 e com *payload* 0x30 (é um mensagem de controlo de fluxo que solicita o envio da restante informação do *VIN*)

- Resposta do carro – 2ª mensagem; tem *CAN ID* 0x7E8 e no *payload* tem 7 caracteres do *VIN* (últimos 7 bytes do *payload* da mensagem CAN)
- Resposta do carro – 3ª mensagem, tem *CAN ID* 0x7E8 e no *payload* tem 7 caracteres do *VIN*.

Apresenta-se agora um pequeno exemplo de uma troca de mensagem entre o simulador e o *Arduino*, essa representação somente indica uma sequência das mensagem e de como é feito, é simplesmente um exemplo do mesmo. A descrição precisa do protocolo de comunicação real é o descrito anteriormente.

Primeiramente vai ser enviada uma mensagem para a rede *CAN* para saber qual "*controller*" que guarda o *VIN*, desta forna a primeira mensagem que enviamos vai ter uma identificação dada por "0X7E0" em hexadecimal, e com a restante da mensagem correspondendo a "02 09 02 00 00 00 00 00" tendo então no final uma mensagem igual a representada pelo *Script 7-3*. Sabendo então que o número de identificação do veículo é constituído por 17 carateres, vamos ter mais do que uma resposta do mesmo para no fim ter todos os carateres que representam o *VIN*. Ao receber essas mensagens, organizamos, assim, o *VIN* numa única variável.

```
char reqvin[] = {0x7E0,0x02,0x09,0x02,0x00,0x00,0x00,0x00,0x00};
```

Script 7-3: *Payload* de pedido do *VIN*.

Depois de enviar a primeira mensagem e se o VIN estiver presente no "*controller*" então vamos ter uma resposta do mesmo acompanhado da primeira parte do *VIN*. As respostas são geralmente da seguinte forma **Mensagem= [7E8 00 00 00 00 23 32 98]**, podemos reparar que aqui agora o *ID* é "0X78E" em hexadecimal. Depois de receber a resposta a essa primeira mensagem, é então enviada uma segunda mensagem com a mesma identificação da primeira, mas com um *payload* diferente. Esse *payload* é utilizado para fazer o pedido dos restantes carateres de uma "*multi frame message*" (Script *7-4*).

```
char floctl[] = {0x7E0,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
```

Script 7-4: *Payload* para pedido dos restantes *bytes*.

É feito um pedido do número de identificação do veículo pelo *Arduino*, que o faz dependendo de um período de tempo definido no *Arduino*. A comunicação entre o *Arduino* e o simulador é uma Comunicação Serial, onde vamos conectar o pino Tx do simulador ao pino Rx do *Arduino* de seguida, o pino Rx do simulador ao pino Tx do *Arduino*. O pedido é feito quando o *Arduino* enviar o carater "1" para o simulador que logo dá início à comunicação.

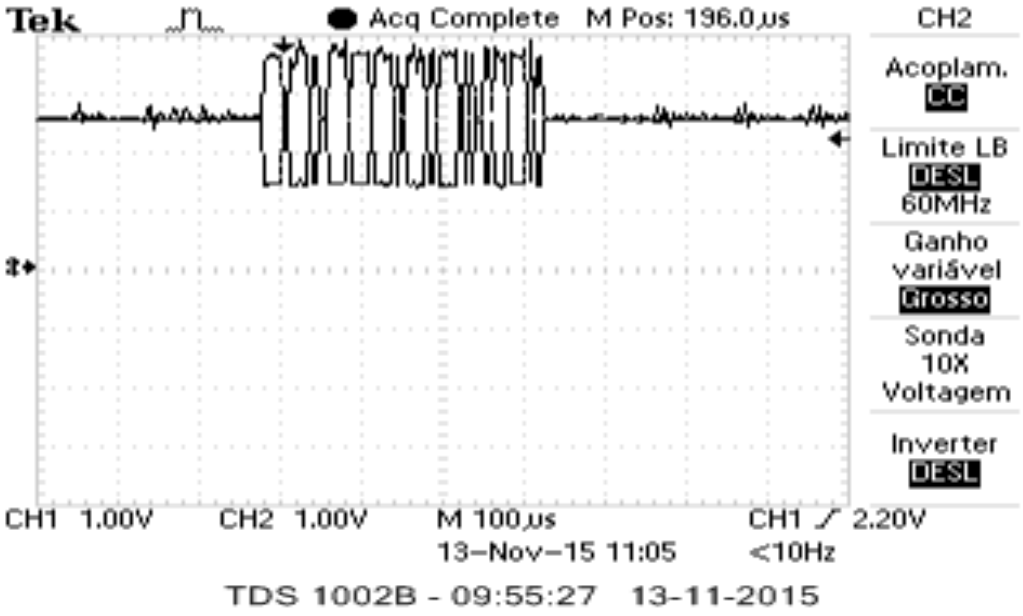


Figura 7-16: Imagem do Osciloscópio para o pedido do *VIN*.

A *Figura 7-16* é uma imagem retirada do osciloscópio, na simulação de quando faz um pedido do número de identificação do veículo na rede *CAN*. Quando se utiliza o simulador do barramento *CAN*.

7.4-2. Resposta do *VIN* Simulador - Módulo

Nesse caso o simulador está programado de modo que está sempre a espera que receba uma mensagem. Essa terá obrigatoriamente a estrutura referida anteriormente da constituição de uma mensagem da rede *CAN*.

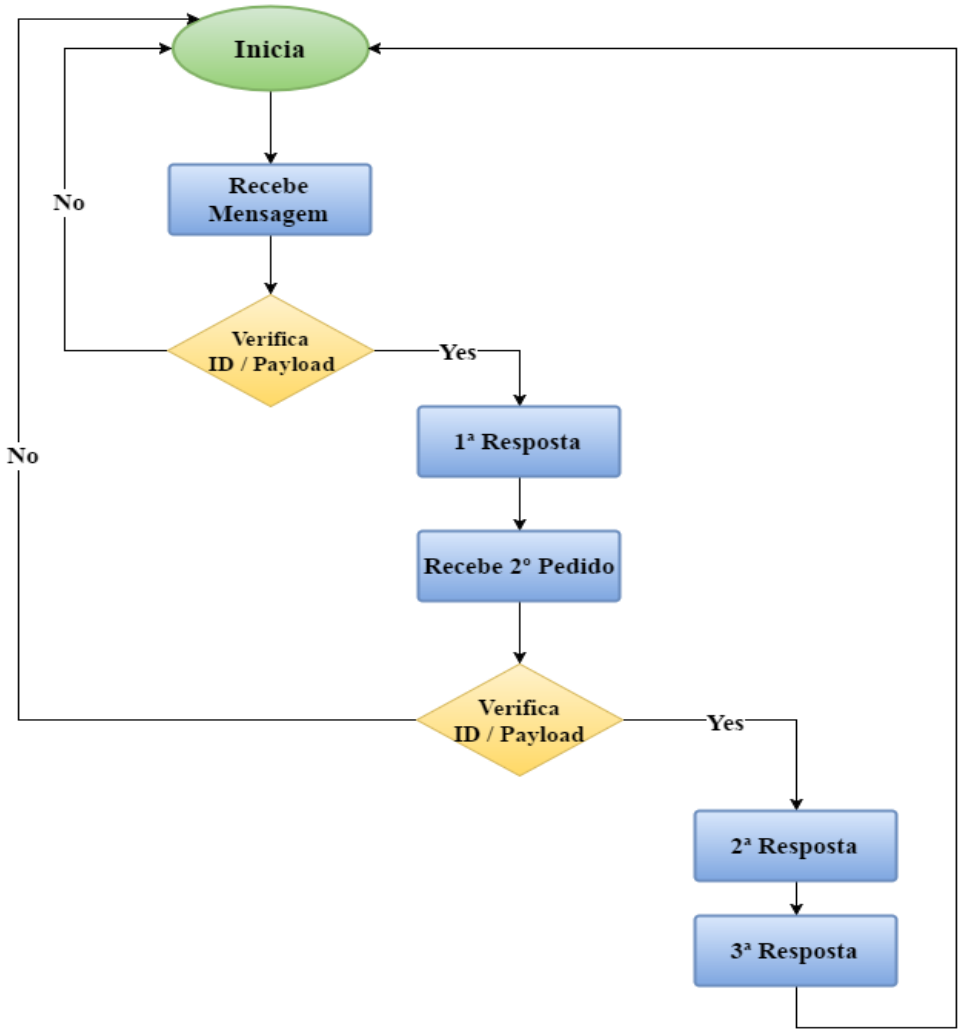


Figura 7-17: Fluxograma do Simulador *CAN*.

A Figura *7-17* que representa um pequeno exemplo de como é feito a troca das mensagens. Quando receber uma mensagem, primeiramente vai analisar o *ID* da mensagem juntamente com o *payload*. Deste modo, ao analisar essa primeira mensagem, consegue identificar que se trata de um pedido do *VIN*, envia então a primeira resposta, e fica aguardando que o cliente envia o segundo pedido. Ao receber o segundo pedido, vai novamente analisar o *ID* e o *Payload* da mensagem. Caso for os corretos, o simulador agora vai enviar duas repostas, estas que contêm todos os restantes carateres que compõem o *VIN*.

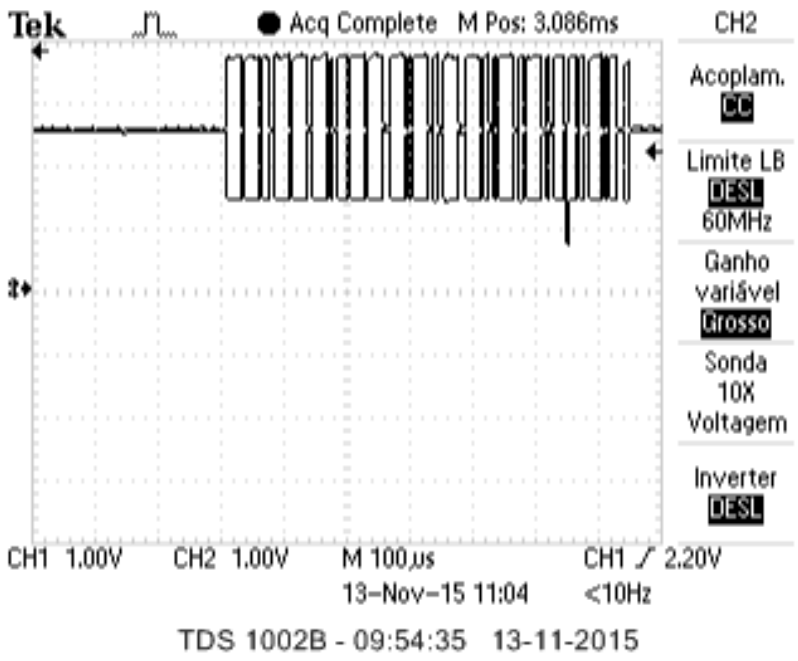


Figura 7-18: Imagem do Osciloscópio para a resposta do *VIN*.

A Figura *7-18* apresenta uma imagem retirada do osciloscópio para a simulação de uma resposta da rede *CAN* (do carro para o módulo).

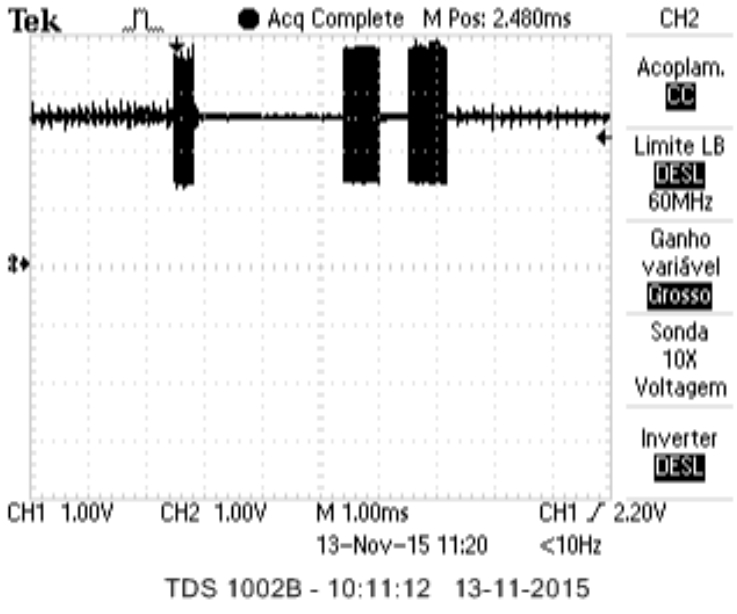


Figura 7-19: Imagem do Osciloscópio quando se faz um pedido e envia a resposta.

A Figura *7-19* é uma imagem retirada quando se faz uma comunicação entre a rede *CAN* e o *Arduino*, onde o *Arduino* envia uma mensagem e recebe duas respostas, como indica a figura, (Ibrahim, 2011).

7.4-3. Arduino

Á cada 30 segundos o *Arduino*, vai enviar para o simulador o sinal para dar início à comunicação. Neste caso, o *Arduino* que comunica por serial, envia o carater "1" que indica o início da comunicação. Após iniciar a comunicação, o *Arduino* vai estar em modo de "escuta" via porta serial para fazer a leitura do número de identificação do veículo.

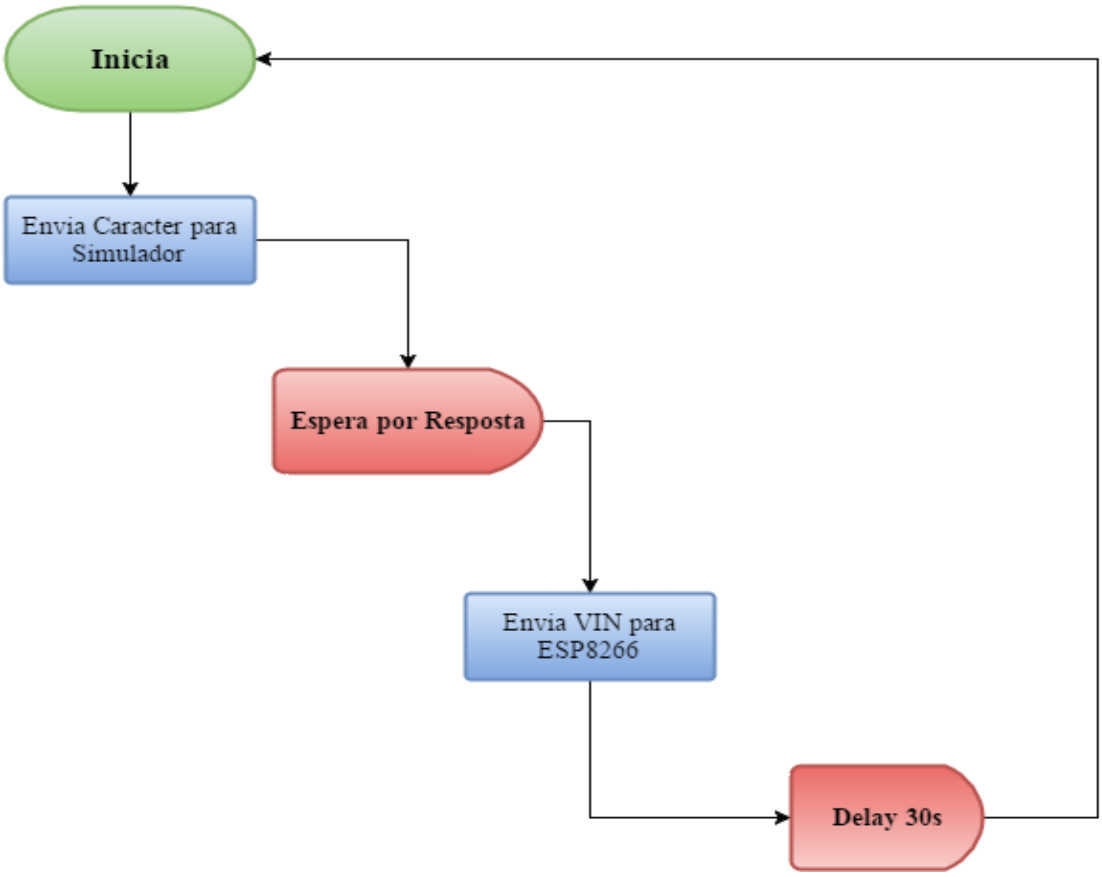


Figura 7-20: Fluxograma da comunicação do *Arduino*.

Quando é feita a leitura do *VIN*, o *Arduino* inicia uma comunicação, também via serial, com o módulo de comunicação *wireless* ESP8266, enviando para o mesmo o *VIN* recebido do simulador. A Figura *7-21* representa o esquema a seguir entre a ligação do *Arduino*, o módulo de comunicação *wireless* ESP8266 e o Simulador, que neste caso vai ligar ao pino Tx e ao pino Rx do *Arduino*.

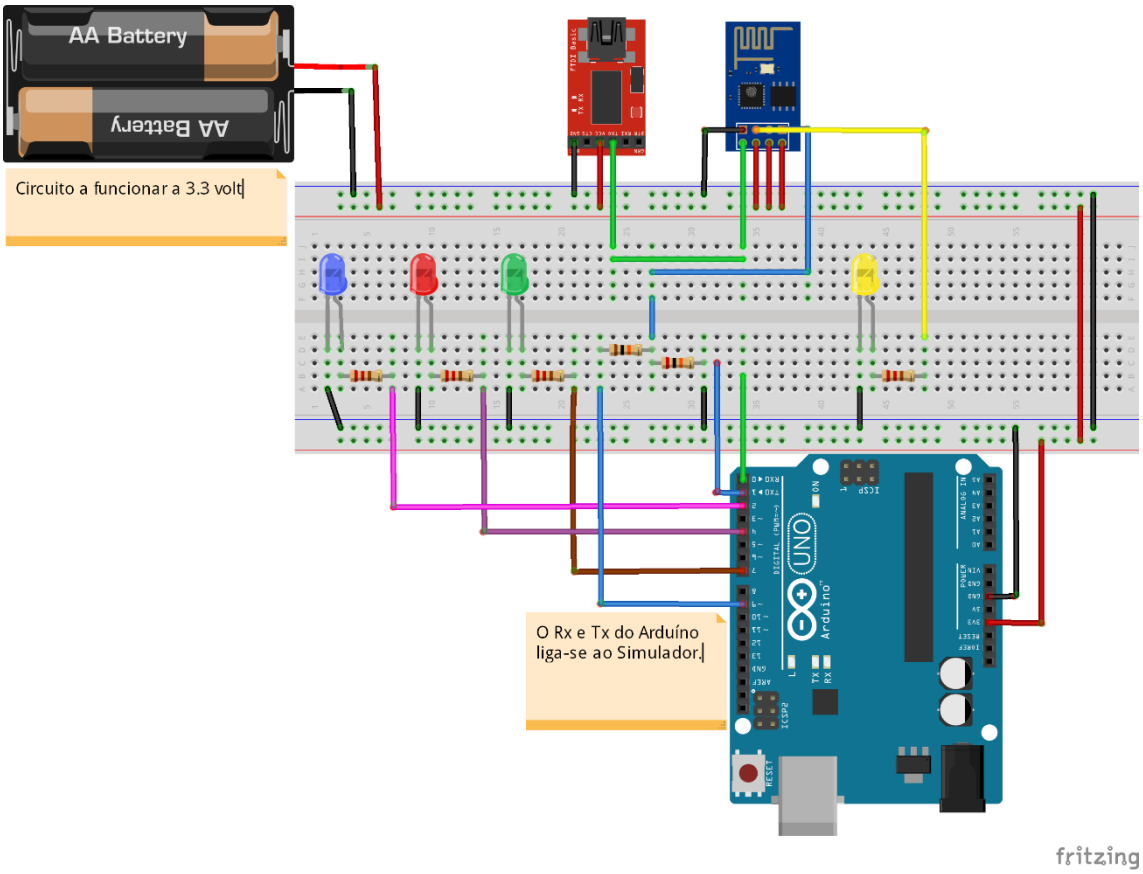


Figura 7-21: Esquema de ligação *Arduino*, ESP8266 e Simulador.

7.5- Servidor

Este trata-se do dispositivo instalado na garagem do prédio ou residência. Como já foi antes referido, este protótipo é constituido por um *Arduino*, um módulo de comunicação ESP8266, utiliza também um FTDI 232. A Figura *7-22* representa o diagrama de comunicação entre o carro e a garagem, indicando os diferentes importantes blocos que constituiem o *Gates Technology*.

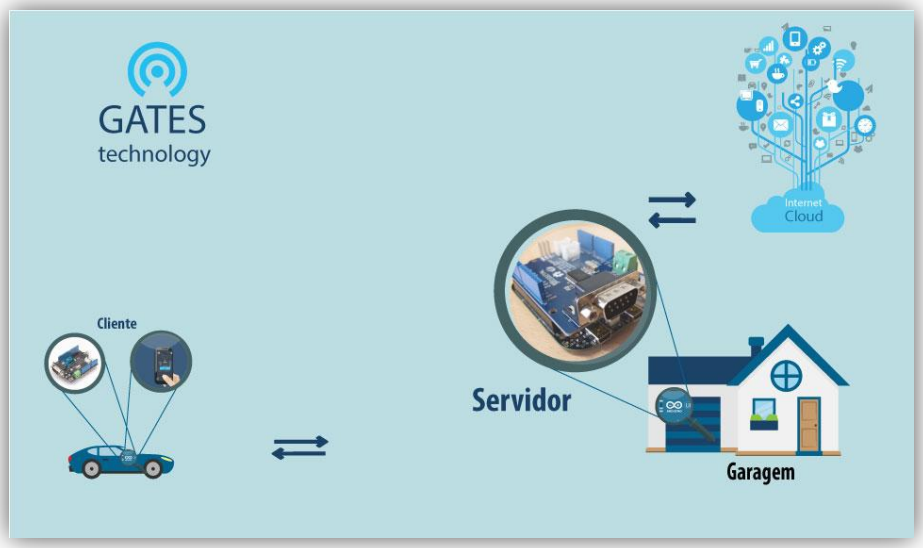


Figura 7-22: Diagrama de comunicação entre o carro e a garagem.

Neste caso a forma mais importante da ligação também é idêntica ao do que vai estar presente no carro do cliente. A Figura *7-23* representa neste caso a ligação entre o *Arduino* e o módulo ESP8266.

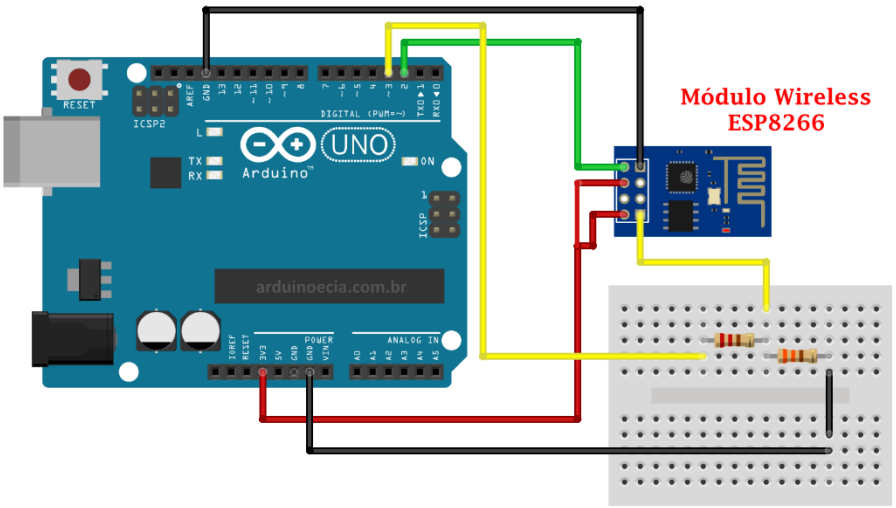


Figura 7-23: Ligação *Arduino*, ESP8266 presente na garagem.

Nesta situação, as ligações seguem o mesmo padrão já referido para o protótipo presente no carro do cliente. O *Arduino* é utilizado para comunicar com o módulo de comunicação, e, também, para enviar os comandos para a abertura ou fecho da porta da garagem. O módulo ESP8266, no caso do servidor, serve para comunicar com o carro do cliente, e, igualmente, através desse módulo, o servidor consegue aceder à *internet* para verificar os dados que estão presentes na base de dados. O FTDI 232 é aqui utilizado para configurar o módulo de comunicação para o modo "*Stand Alone*".

A principal funcionalidade do Servidor é estar sempre em modo de leitura, à espera que um cliente se tente conectar para atribuir ou não acesso à garagem. Como foi já referido, o Servidor (ESP8266) funciona, neste caso, como um *Access Point* e também como um *Station* para todas as formas de comunicação possível. O Servidor vai também fazer *Host* a uma página *web* para o controlo direto do portão da garagem em modo *offline* (sem ligação a *internet*).

Quando o Servidor detetar que um cliente está a tentar comunicar, vai estar sempre em ciclo, esperando dados do cliente, que, neste caso, é o número de identificação do veículo. Depois que receber esse número, o módulo vai primeiramente verificar através do *IP Adress* do cliente se é a primeira vez que se estabelece comunicação. Para o caso de ser a primeira vez, e de o *VIN* não estar presente na base de dados, passa-se para o procedimento do registo automático de um novo cliente. Para o caso de ser um cliente que já se comunicou outra vez com esse servidor, vai passar, então, ao procedimento de verificação do número e identificação do veículo. O dispositivo vai percorrer a tabela "Carros" (tabela essa onde é efetuado o registo dos *VIN*) na base de dados para verificar se está ou não registada. Caso já esteja registada, o ESP8266 envia um comando para o *Arduino*, que, por sua vez, vai enviar o comando *PWM* para o motor do portão para abrir ou fechar a porta. Para o caso de não estar registado, o ESP8266 envia uma mensagem de erro para o cliente, e é efetuado um registo de alerta na base de dados indicando como acontecimento que um cliente desconhecido estava a tentar aceder à garagem.

Um caso crítico que temos é o facto de o servidor perder conexão à *internet*, em que não seria possível aceder à base de dados para verificar os números de identificação dos veículos. Foi, então, programada de modo a ultrapassa esse obstáculo, ou seja, mesmo sem conexão à *internet*, o carro pode comunicar com o servidor e ter acesso à garagem. De modo a ser possível, tivemos que programar o ESP8266 da seguinte forma: quando o Servidor detetar a falha na conexão, a *internet* passa a funcionar no modo passivo. O ESP8266 já vem programado para fazer uma leitura de forma periódica dos dados que estão presentes na base de dados - tabela "Carros" (em específico os números de identificação dos veículos). Ao fazer a leitura desses dados, vai guardando automaticamente esses valores na sua memória interna. Quando se perder a ligação à *internet*, o servidor já não tenta verificar o *VIN* na base de dados, mas percorre uma tabela secundária de identificação, para ver se o *ID* do cliente está presente na base de dados.

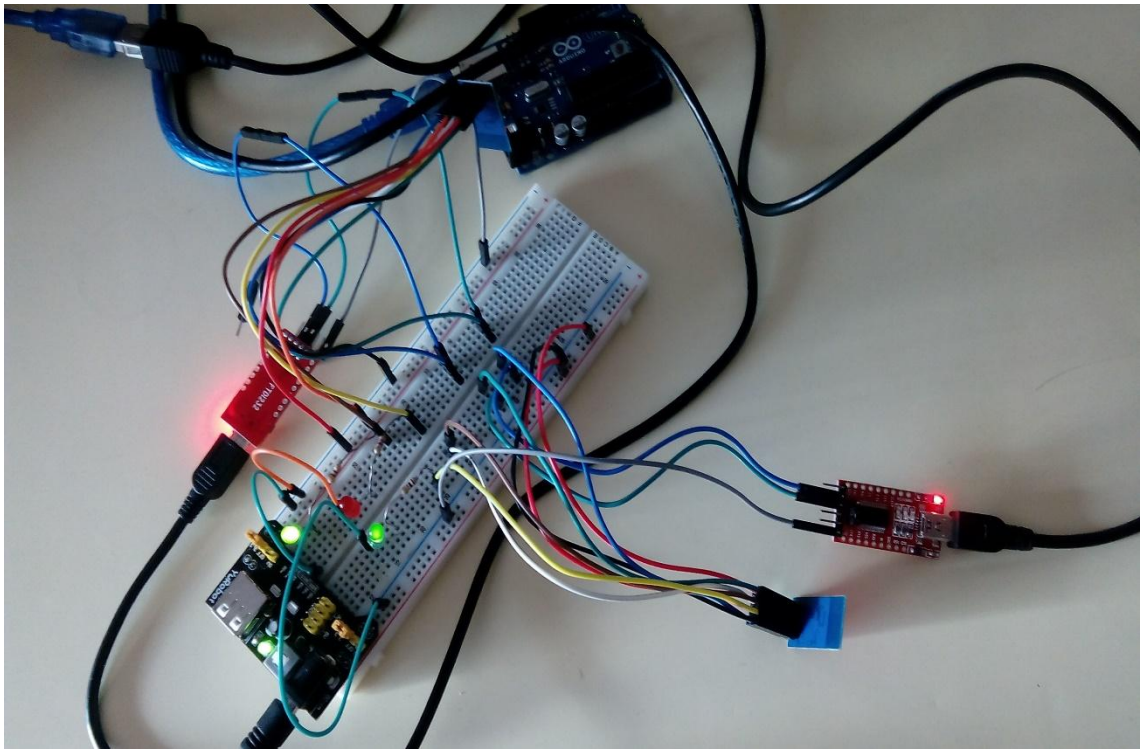


Figura 7-24: *Setup* Experimental do Servidor.

7.5-1. ESP8266 Circuito e Programação

Como já foi mencionado, o servidor, ao receber o número de identificação do veículo, passa para o processo de verificação da mesma, para atribuir ou não acesso à garagem. Essa parte é feita utilizando a linguagem de programação Lua. A Figura *7-25* representa o esquema do circuito utilizado.

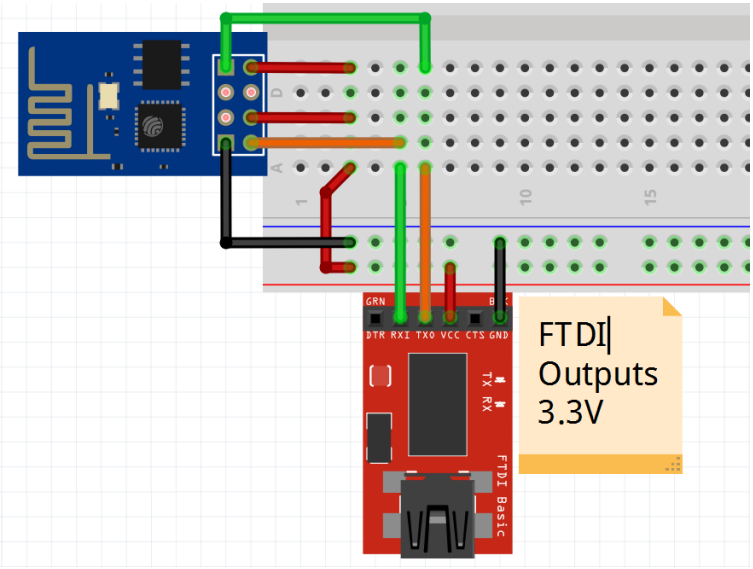


Figura 7-25: Circuito para a programação do servidor.

Depois de desenvolver o *Script*, é feita o *upload* direto para o ESP8266. Para fazer o *upload* do código para o módulo, utiliza o *IDE* (*Integrated Development Environment*) LuaUploader (Figura *7-26*) que é um dos *softwares* mais utilizados para a programação dos módulos de comunicação *wireless* ESP8266.

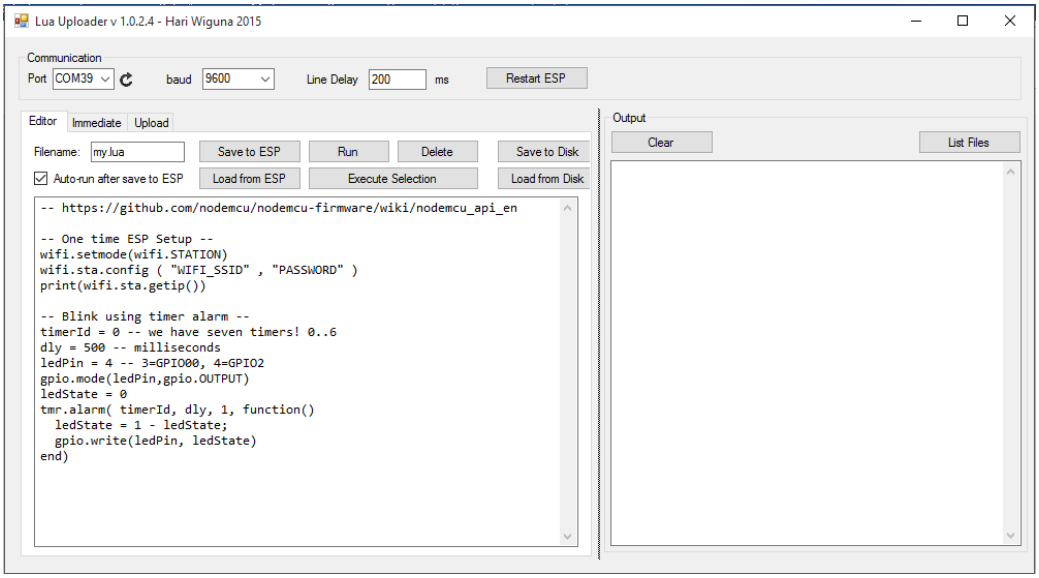


Figura 7-26: Ambiente de trabalho do LuaUploader.

Este *IDE* está disponível *online* (pode ser encontrado no Anexo 5 [72]) e o processo de carregar o programa é dado da seguinte forma: Através do FTDI, que se liga à porta *USB*, vamos, então, ter disponível uma porta *COM* do computador. Selecionamos essa porta, Figura *7-27* - a vermelho. Depois vamos selecionar o programa (o código que se vê na imagem é somente um exemplo que já vem predefinido no LuaUploader), pretendido através do botão "*Load from disk*", e, no final (Figura *7-27* representando pelo retângulo a azul), para carregar o código ao módulo é só pressionar o botão "*Save to ESP*".

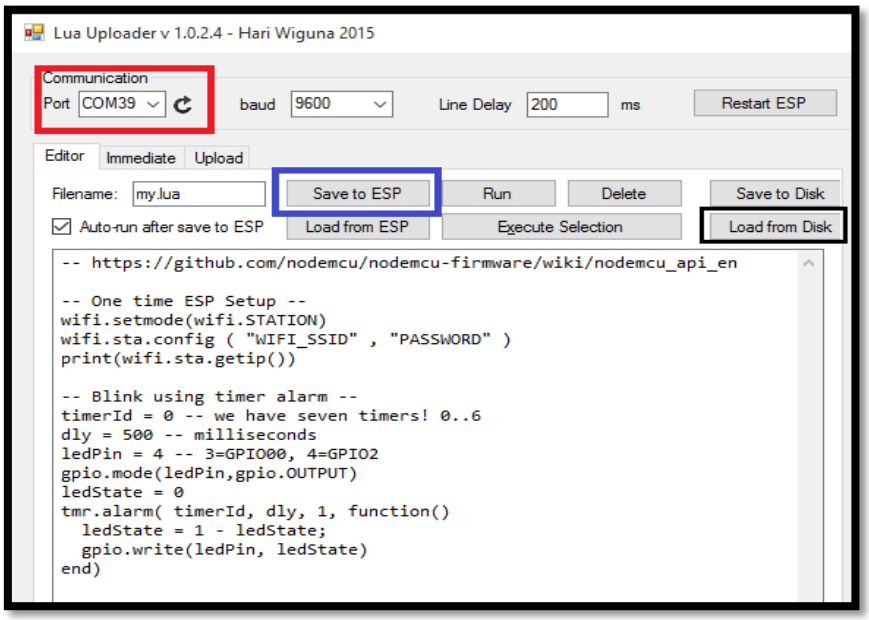


Figura 7-27: Procedimento de *upload* do código através LuaUploader.

7.5-2. ESP8266 como *Webserver*

A programação desse módulo é feita da mesma forma do presente no servidor. Que nesse caso é feita utilizando também o *IDE LuaUploader*, seguindo o mesmo procedimento já referido anteriormente, onde está explicado de forma detalhada como fazer o *upload* de um *script* de código para o módulo de comunicação *wireless*.

No entanto, temos o caso em que vamos ter o módulo a funcionar como um *webserver*. Este é o caso que torna possível controlar o portão da garagem de forma *offline*, ou seja, sem conexão a *internet*. Que pode ser acedido quando o dispositivo estiver conectado á rede do servidor. Para ter o funcionamento desse *website* e o funcionamento da aplicação *Android*, vamos ter que programar este módulo através dos comandos AT. Para fazer isso, primeiramente teremos que fazer o *flash* do *firmware* de comandos AT para o ESP8266. Depois disso, passamos a parte da programação, que neste caso vai ser feito utilizando o *IDE* do *Arduino* seguindo o esquema da Figura *7-28*.

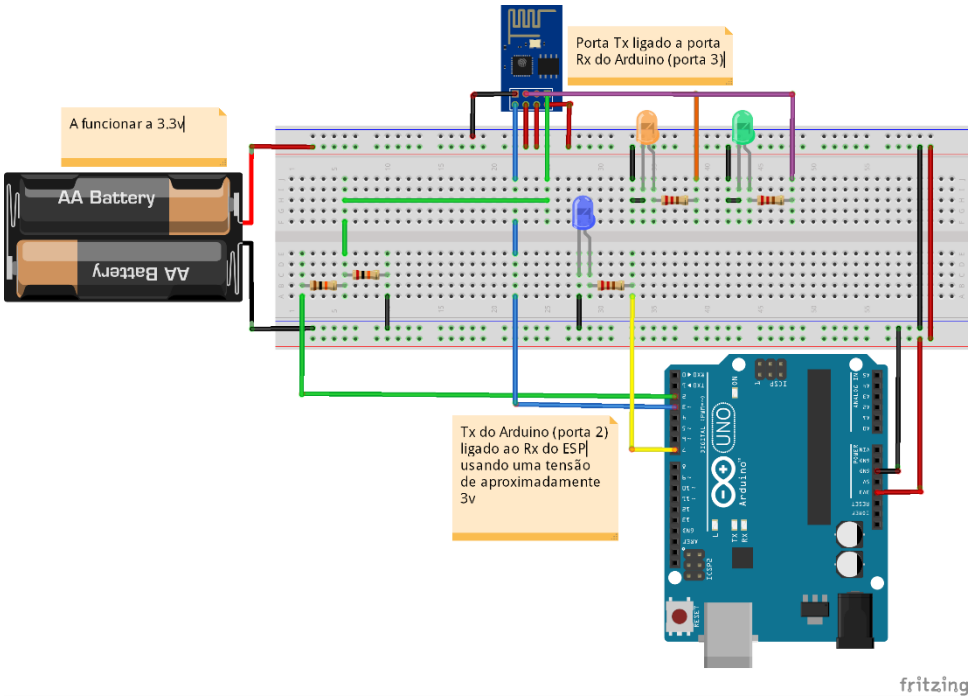


Figura 7-28: Esquema de ligação para a programação do ESP8266 *Webserver*.

Em que o *Arduino* vai configurar o ESP8266, nesse caso como o *webserver*. O *Arduino* envia os comandos AT para o módulo, programando de forma automaticamente esse módulo. O *Arduino* consegue fazer isso enviando esses comandos pela porta digital 2 que está programado para funcionar como uma porta de comunicação serial. O pino digital 2 do *Arduino* está ligado ao pino Rx do módulo. O pino digital 3 do *Arduino* (funcionando como uma porta de comunicação serial) está neste caso ligado então ao pino Tx do módulo ESP8266. Os pinos GPIO 0 e GPIO 2 que estão ligados aos *LED's* vão ligar e desligar conforme o comando enviado da *webpage*[9].

[9] ANEXO 5 – Sítios da Internet Relevantes

8. CONCLUSÕES E TRABALHO FUTURO

Neste capítulo faz-se um resumo muito breve do trabalho desenvolvido e das principais conclusões. Também se apresentam algumas ideias para trabalho futuro.

8.1- Conclusões

Facilitar e simplificar a vida das pessoas é o principal objetivo subjacente ao *Gates Technology*. A ideia base partiu de tentar desenvolver uma solução que permitisse ao utilizador chegar a casa com o carro para estacionar na garagem e ter o portão desta a abrir automaticamente, de forma segura, sem ser necessário qualquer interferência explícita do utilizador.

A solução desenvolvida, designada por *Gates Technology*, possui módulos de comunicação *wireless* nos carros e na garagem, uma aplicação *Android* e uma plataforma *online* de controlo. Foi também desenvolvido um *website* para tratar da gestão das contas dos utilizadores.

Os principais blocos que constituem a solução desenvolvida são:

- Relativamente ao *hardware* desenvolvido, temos diferentes modelos de comunicações: para os clientes e para as garagens. Para a parte do cliente, a comunicação é feita com o simulador da rede *CAN*, onde é feito o pedido do número de identificação do veículo, que posteriormente se envia ao servidor. Na parte do servidor, este vai fazer a leitura dos dados enviados pelos clientes e verificar se estes estão presentes na base de dados. É no servidor que está alocada uma *webpage* para o controlo da garagem.
- Na aplicação *Android* desenvolvida, é possível fazer o controlo direto do portão da garagem (onde é possível abrir ou fechar a porta). Também é possível ler diferentes dados que estão presentes na base de dados, tais como: o *VIN*, os carros registados e os eventos da garagem. Através da aplicação, o cliente ainda pode alterar os seus dados de registo na base de dados.
- O *website* distingue se o acesso está a ser feito por um cliente ou pelo administrador da garagem. O cliente consegue controlar a garagem (abrindo ou fechando a porta), adicionar ou remover carros da base de dados, consultar os eventos ou ainda adicionar convidados. O administrador pode adicionar os clientes, visualizar todos os registos (eventos) e alertas da garagem.

Para o cumprimento dos objetivos inicialmente propostos, deparei-me com muitos desafios e alguns obstáculos, mas, com muito trabalho e dedicação, conseguimo-los ultrapassar, o que se revelou muito gratificante.

8.2- Trabalhos Futuro

Uma ideia para trabalho futuro seria acrescentar uma funcionalidade que permitisse ao controlador da garagem saber se o cliente pretende mesmo estacionar dentro da garagem ou se está simplesmente a passar ou a estacionar na rua. Neste sentido propomos a instalação de quatros sensores no veículo (*Figura 8-1*). Cada sensor fica instalado nos quatros vértices do carro, a comunicarem para o mesmo ponto. Desta forma, através do cálculo da posição do carro, será possível identificar se o carro está somente a passar pela rua, a estacionar em frente da casa ou então se realmente se está a aproximar do portão para entrar na garagem. Assim sendo, o cliente não correrá o risco de ter a porta da sua garagem aberta sempre que passar pela rua. Integrar esta funcionalidade, tornaria o *Gates Technology* um produto que conferia maior segurança.

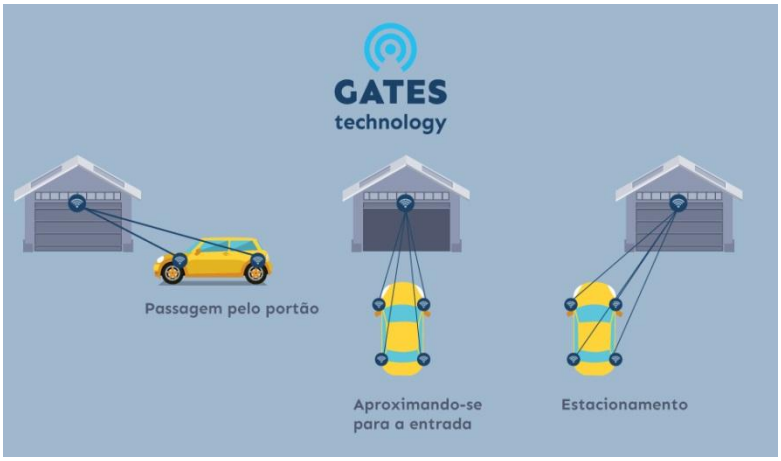


Figura 8-1: Garagem Inteligente.

Outra ideia para trabalho futuro é permitir o registo automático do *VIN*, isto é, sem ser necessário ser o cliente ou o administrador a inserir esse dado à mão na base de dados (Figura *8-2*). Uma sugestão para implementar essa funcionalidade passaria, por exemplo, por ter o módulo de comunicação *wireless* (ESP8266) a funcionar como um *Access Point* e também como um dispositivo terminal. Isto quer dizer que pode ser um "cliente" e um "servidor" ao mesmo tempo. Ao receber o *VIN* do carro a partir da placa com o *Arduino* instalada no carro, vai guardar esse valor. A primeira vez que comunicar com a garagem, o servidor vai ver se já está ou não registado esse *VIN* na sua base de dados. Caso não esteja registado e for a primeira vez que comunica com esse cliente, o servidor vai fazer um pedido de um código de acesso. Para ter esse determinado código de acesso, o utilizador precisa de se conectar (utilizando qualquer dispositivo como, por exemplo, *smartphone* ou *tablet*) à rede *wifi* do ESP8266 presente no carro. Aí, ser-lhe-ão disponibilizados os dados que serão utilizados para ter acesso a essa rede. Depois de se conectar, vai aceder a uma página *web* através da qual só é preciso pressionar no botão que vai gerar um número aleatório. Depois disso, o cliente (ESP8266) vai enviar esse número gerado para o servidor. Posteriormente, o servidor vai verificar se já existe um número aleatório igual registado na base de dados. Para o caso de já ter registado esse número, o servidor vai guardar o número de identificação do veículo, ficando assim com o registo de um novo cliente.

Os números aleatórios gerados pela página *web*, que podem estar já registados na base de dados do servidor, são diferentes para cada caso. Ou seja, para cada garagem a lista dos números é alterada de forma a aumentar o nível de segurança.

Figura 8-2: Registo automático de um novo cliente.

Uma outra ideia para trabalho futuro prende-se com o aspeto da eficiência energética, isto é, a poupança da bateria dos carros. Apesar dos módulos de comunicação instalados nos carros não consumirem muita energia, deve-se, no entanto, pensar numa forma de minimizar os gastos. Programando neste caso o ESP8266, utilizando uma função "*sleep*" que faz com que o módulo desligue o *CPU* e a antena *wifi* por um determinado período de tempo. Para reiniciar o módulo, pode ter-se um sensor na garagem que quando deteta a presença do carro envia um comando para o *Arduino* que então reiniciará o ESP8266 para efetuar a comunicação com o servidor.

9. REFERÊNCIAS BIBLIOGRÁFICAS

Adobe Systems (2001). *Adobe Photoshop 6.0 and Illustrator 9.0.* Adobe.

Adobe Systems (2008). *Adobe Dreamweaver CS3: basic.* Adobe: Peachpit Press.

António Júlio Morais Pires (2005). *Comunicação de tempo-real em barramentos CAN baseados no controlador SJA1000.* Dissertação para obtenção de Grau de Mestre em Engenharia Eletrotécnica e de Computadores (Área de especialização Informática e Automação), Faculdade de Engenharia da Universidade do Porto.

Baichtal, John (2014). *Arduino for beginners: essential skills every maker needs.* Que

Biancuzzi, Federico et Warden, Shane (2009). *Masterminds of programming.* O'Reilly.

Barclay, Kenneth A. et Gordon, Brian J. (1994). *C++ problem solving and programming*. Prentice Hall.

Böhmer, Mario (2012). *Beginning Android ADK with Arduino.* Apress; Distributed to the book trade worldwide by Springer Science+Business Media.

Brooks, David R. (2011). *Guide to HTML, JavaScript and PHP: for scientists and engineers.* Springer.

Buchanan, William (1997). *Software development for engineers with C, Pascal, C++, Assembly Language, Visual Basic, HTML, JavaScript, and Java*. Arnold; Copublished in North, Central, and South America by John Wiley & Sons, Inc.

Burd, Barry A. (2011). *Java for dummies.* Wiley.

Cooper, Dave et Boyer, Arlet Iris (2001). *C++ programming for Technology.* Delmar.

da Costa, E. T. et al. (2014). *Getting started with open-hardware: development and control of microfluidic devices.* Electrophoresis, Vol. 35.

Dale, Nell B. et Weems, Chip (2004). *Programming in C++*. Jones and Bartlett Publishers.

David, Matthew (2014). *Designing apps for success: developing consistent app design practices.* Focal Press, Taylor & Francis Group.

Davis, Stephen R. (2015). *Beginning programming with C++ for dummies*. John Wiley and Sons.

Devarakonda, K. et al. (2015). *ROBucket: A low cost operant chamber based on the Arduino microcontroller.* Behav Res Methods.

Doecke, S. et al. (2015). *The real-world safety potential of connected vehicle technology.* Traffic Inj Prev, Vol. 16 Suppl 1.

Durbin, T. D. et Norbeck, J. M. (2002). *The effects of repairs on tailpipe emissions for on-board diagnostics II-equipped vehicles with the malfunction indicator light illuminated.* J Air Waste Manag Assoc, Vol. 52.

Duffy, Thomas J. (2013). *Programming with mobile applications: Android, iOS, and Windows Phone 7.* Course Technology/Cengage Learning.

D'Ausilio, A. (2012). *Arduino: a low-cost multipurpose lab equipment.* Behav Res Methods, Vol. 44.

Emanuel Nunes Baldissera (2011). *Decodificador de dados usando Protocolo CAN – Padrão SAE J1939*. Projeto de Graduação em Engenharia Elétrica, com Ênfase em Eletrônica Telecomunicações, Faculdade de Engenharia e Arquitetura, da Universidade de Passo Fundo.

Espressif Systems IOT Team (2015). *ESP8266 AT Instruction SET*. Espressif Systems Inc.

Evans, Brian (2011). *Beginning Arduino programming.* Apress; Distributed to the book trade worldwide by Springer Science+ Business Media.

Fan, Gong et al. (2014). *RICA: A reliable and image configurable arena for cyborg bumblebee based on CAN bus.* Conf Proc IEEE Eng Med Biol Soc, Vol. 2014.

Farsi, Mohammad et Barbosa, Manuel Bernardo Martins (2000). *CANopen implementation: applications to industrial networks.* Research Studies Press.

Firnkorn, D. et al. (2014). *Alignment of high-throughput sequencing data inside in-memory databases.* Stud Health Technol Inform, Vol. 205.

Gamble, K. H. (2010). *Wireless Tech Trends 2010. Trend: smartphones.* Healthc Inform, Vol. 27.

Goodman, Danny (2010). *JavaScript bible.* Wiley.

Göransson, Andreas et Cuartielles Ruiz, David (2013). *Professional Android Open Accessory programming with Arduino.* Wiley.

Harris, Andrew (2011). *HTML, XHTML, and CSS all-in-one for dummies.* Wiley; John Wiley distributor.

Harris, Andrew (2014). *HTML5 and CSS3 all-in-one for dummies.* John Wiley & Sons.

Haseman, Chris (2012). *Creating Android applications: develop and design*. Peachpit Press.

He, Z. et al. (2014). *Who sits where? Infrastructure-free in-vehicle cooperative positioning via smartphones.* Sensors (Basel), Vol. 14.

Huebner, K. D. et al. (2006). *Validation of an electronic device for measuring driving exposure.* Traffic Inj Prev, Vol. 7.

Ibrahim, Dogan (2011). *Controller Area Network projects*. Elektor International Media,

James, Derek (2013). *Android game programming for dummies.* Wiley; John Wiley distributor.

Jamsa, Kris et al. (1996). *Web programming.* Jamsa Press.

Jara, A. J. et al. (2013). *IPv6 addressing proxy: mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6).* Sensors (Basel), Vol. 13.

Jones, Terry et Tollervey, Nicholas H. (2014). *Learning jQuery deferreds.* O'Reilly.

Jung, Kurt et Brown, Aaron (2007). *Beginning Lua programming.* Wiley/Wrox.

Lee, Wei-Meng (2014). *Beginning android programming.* John Wiley and Sons.

Lerdorf, Rasmus et al. (2006). *Programming PHP*. O'Reilly.

Liao, Sean (2014). *Migrating to swift from android.* Apress.

Luiz Roberto Guimarães Barbosa (2003). *Rede CAN*. Universidade Federal de Minas Gerais, Belo Horizonte.

Matthews, Robbie (2011). *Beginning Android tablet programming: starting with Android Honeycomb for tablets.* Apress.

Matthews, Martin S. (2015). *PHP and MYSQL web development: a beginner's guide.* McGraw-Hill Education.

McFarland, David Sawyer et Grover, Chris (2013). *Dreamweaver CC.* O'Reilly.

Mednieks, Zigurd R. (2012). *Programming Android.* O'Reilly.

Monk, Simon (2012). *Arduino + Android projects for the evil genius: control Arduino with your smartphone or tablet.* McGraw-Hill.

Nixon, Robin (2009). *Learning PHP, MySQL, and JavaScript.* O'Reilly.

Nixon, Robin (2014). *Learning PHP, MySQL, JavaScript, CSS & HTML5*. O'Reilly Media, Inc.

Olsen, Steven (2007). *Ajax on Java.* O'Reilly.

Pedro Venda (NA). *Controller Area Network – FUNDAMENTOS.* Instituto Superior Técnico de Lisboa, Portugal.

Purdum, Jack J. (2012). *Beginning C for Arduino.* Apress; Distributed to the book trade worldwide by Springer Science+Business Media.

Rehn, K. et al. (2002). *[Development of a CAN (controller area network) bus for modules of a perfused cell culture system]*. Biomed Tech (Berl), Vol. 47 Suppl 1 Pt 2.

Riley, Mike (2012). *Programming your home: automate with Arduino, Android, and your computer.* Pragmatic Bookshelf.

Roberto Brauer Di Renna et al. (2013). *Introdução ao kit de desenvolvimento Arduino*. Programa de Educação Tutorial, Universidade Federal Fluminense, Curso de Engenharia de Telecomunicações.

Rogers, Rick (2009). *Android application Development.* O'Reilly.

Schengili-Roberts, Keith (1997). *The advanced HTML companion.* AP Professional.

Schubert, T. W. et al. (2013). *Using Arduino microcontroller boards to measure response latencies.* Behav Res Methods, Vol. 45.

Schuytema, Paul et Manyen, Mark (2005). *Game development with Lua.* Charles River Media.

Shih, G. et al. (2010). *Is android or iPhone the platform for innovation in imaging informatics.* J Digit Imaging, Vol. 23.

Ueno, Y. et al. (2003). *Processing sequence annotation data using the Lua programming language.* Genome Inform, Vol. 14.

Ullah, N. et al. (2011). *A very low power MAC (VLPM) protocol for Wireless Body Area Networks.* Sensors (Basel), Vol. 11.

Wang, F. et al. (2012). *Development of an automatic subsea blowout preventer stack control system using PLC based SCADA.* ISA Trans, Vol. 51.

Wilcher, Don (2012). *Learn electronics with Arduino.* Apress; Distributed to the book trade worldwide by Springer-Verlag New York.

Wilhite, S. E. (2010). *Android ('an, droid).* HDA Now.

Woody, D. P. et al. (2007). *Controller-area-network bus control and monitor system for a radio astronomy interferometer.* Rev Sci Instrum, Vol. 78.

A. ANEXO – BASE DE DADOS

A.1 Criação e Desenvolvimento da Base de Dados

Começamos a pesquisar um *software* que seria mais acessível e de fácil acesso, usando então na criação das nossas tabelas da base de dados o MySQL Workbench versão 5.2.47 que era a versão que adaptei da melhor forma. Onde criamos as tabelas e as suas respetivas interligações entre as chaves primárias e estrangeiras. Nesta ferramenta optamos para trabalhar na parte gráfica da nossa base de dados, e para editar código foi utilizado o Notepad++ (*Notepad++ Je suis Charlie edition*) e o Sublime Text3 (*Sublime Text Build 3083 edition*), todos os sítios acedidos podem ser consultados no Anexo 5[10] [69,70,71].

Figura A-1: Mysql Workbench, Notepad++ e Sublime Text 3.

A.2 Mysql Workbench

Sendo esse um programa desenvolvido pela Oracle Corporation, é uma ferramenta "*Open Source*", ou seja, está gratuito no mercado (*Internet*) para qualquer pessoa que quiser utilizá-lo para desenvolvimento de projetos. A primeira versão do *Mysql Workbench* foi lançada em setembro de 2005 onde a primeira versão era a 5.0. Dando continuidade ao longo dos anos, em que nos dias de hoje a última versão que foi lançada em junho de 2015 que é a 6.3.4. Esse que funciona no sistema designado por *Cross-plataform* (ver mais em: *https://en.wikipedia.org/wiki/Cross-platform*). "Mysql Workbench é uma ferramenta de *design* visual de base de dados que integra o desenvolvimento *SQL*, administração, design de base de dados, criação e manutenção em um único ambiente de desenvolvimento integrado para o sistema de base de dados *Mysql*."

"É o sucessor do DBDesigner 4 de fabFORCE.net, e substitui o anterior pacote de *software*, Mysql GUI Tools Bundle."
(Fonte: *https://en.wikipedia.org/wiki/MySQL_Workbench*).

Começamos a desenvolver as tabelas utilizando o Workbench, isso porque sendo o início era mais fácil visualizar o que estaria a ser criada, esta ferramenta dá a possibilidade de criar as tabelas, cada coluna e o seu devido tipo de dado (*INT*, *VARCHAR*, *DATA*, Binário, etc.), e também é possível inserir dentro das tabelas os dados para as diferentes colunas.

[10] ANEXO 5 – Sítios da Internet Relevantes

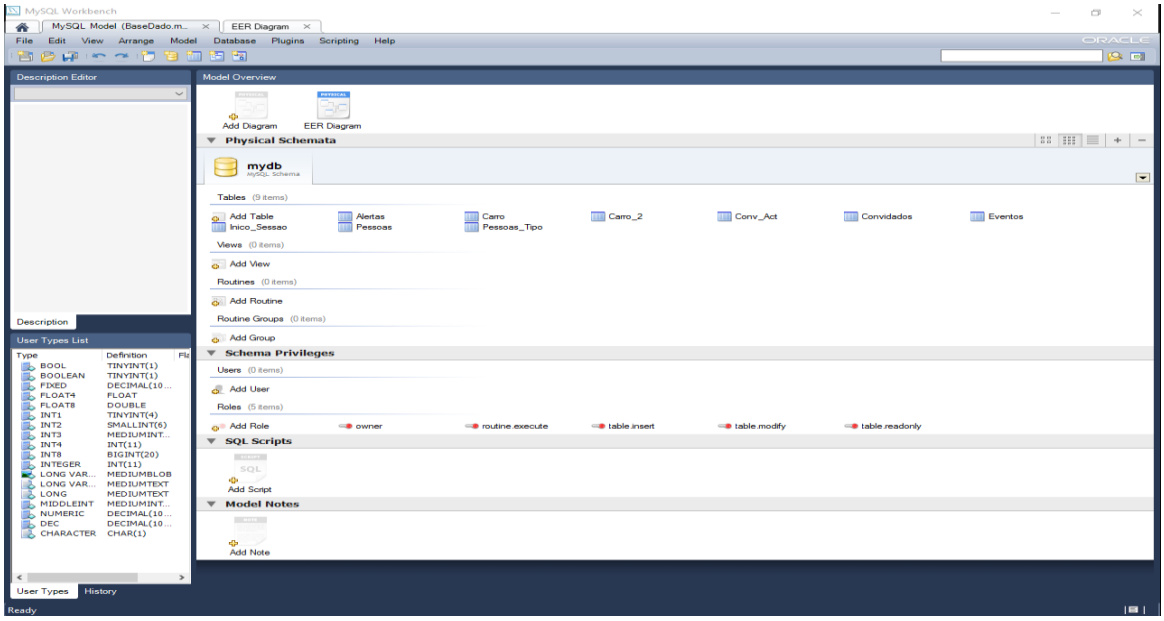


Figura A-2: Ambiente de trabalho de Mysql Workbench.

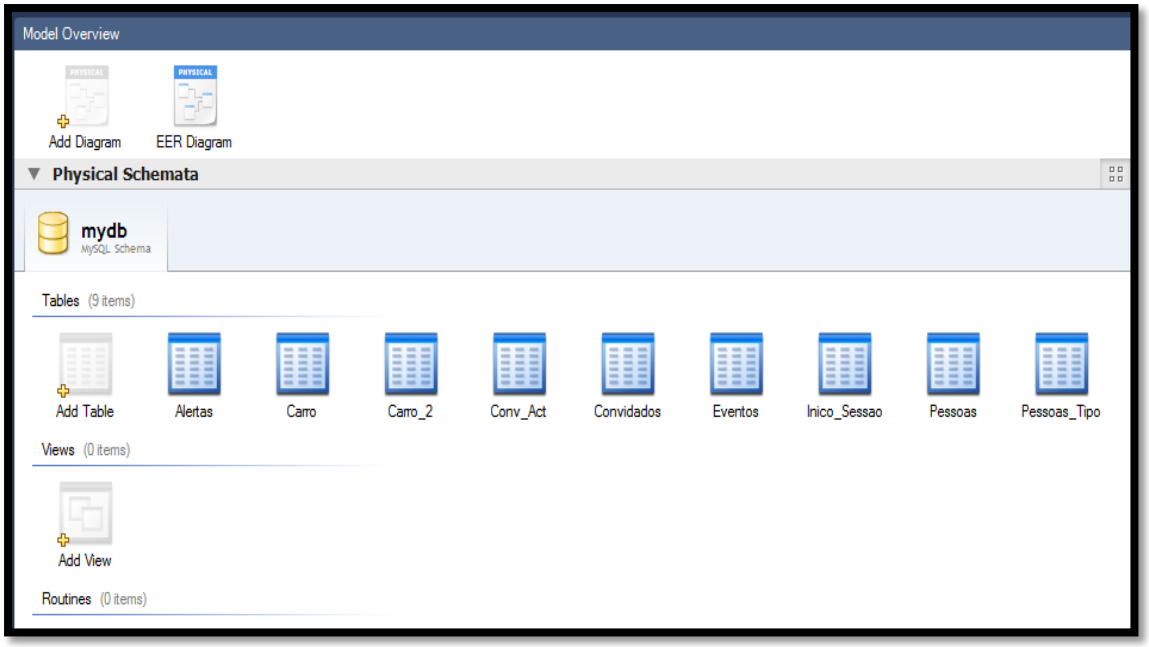


Figura A-3: Mysql Workbench, tabelas do *Gates Technology*.

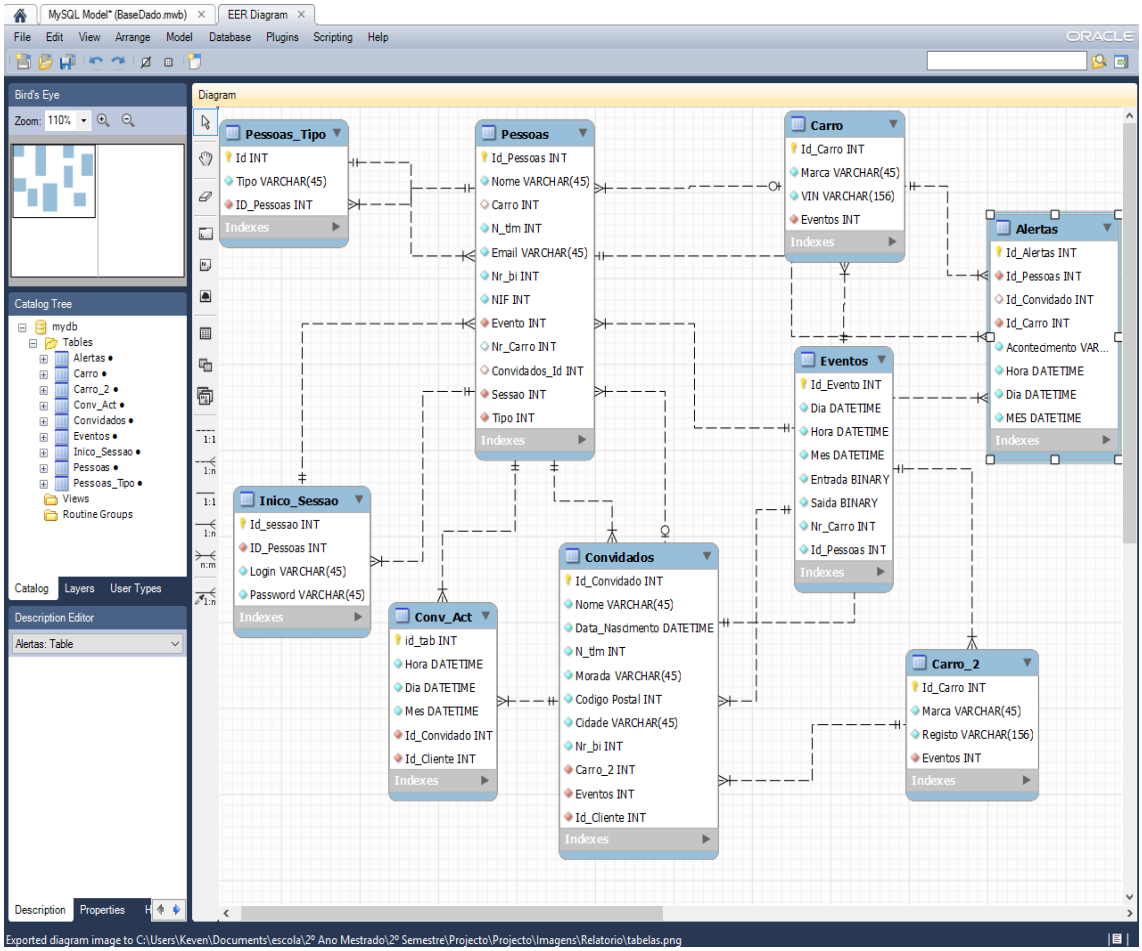


Figura A-4: Comunicação entre as diferentes tabelas.

É possível fazer uma exportação do *script* (Figura *A-6*) das tabelas que foram criadas, escolhendo o lugar a guardar o *script*, tem também as opções que podes escolher para o teu *script* e também ainda e possível escolher somente as tabelas que queres exportar o *script*. Por escolha própria e pelo facto de estar sempre a editar as tabelas não optei por trabalhar no *script* da base de dados com outras ferramentas, essas já referidas acima no relatório.

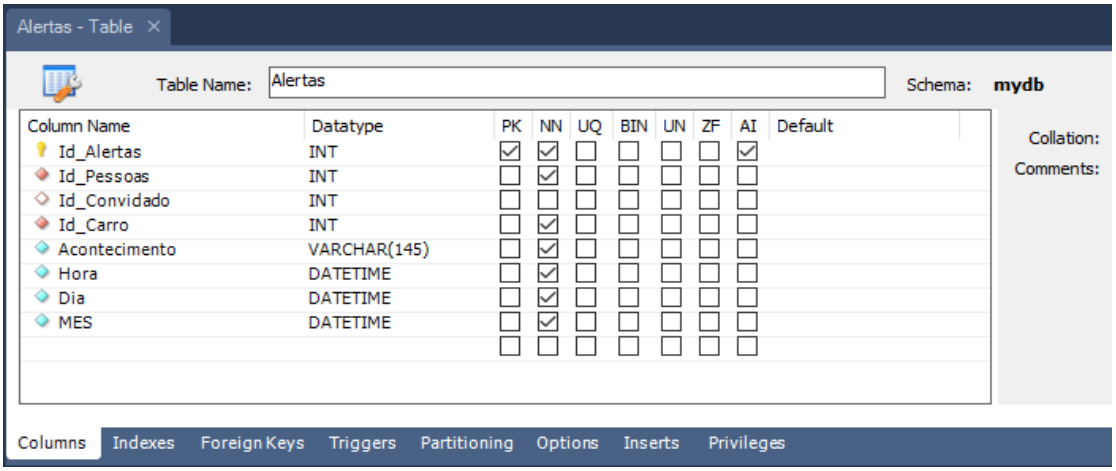


Figura A-5: Criação de uma das tabelas da base de dados.

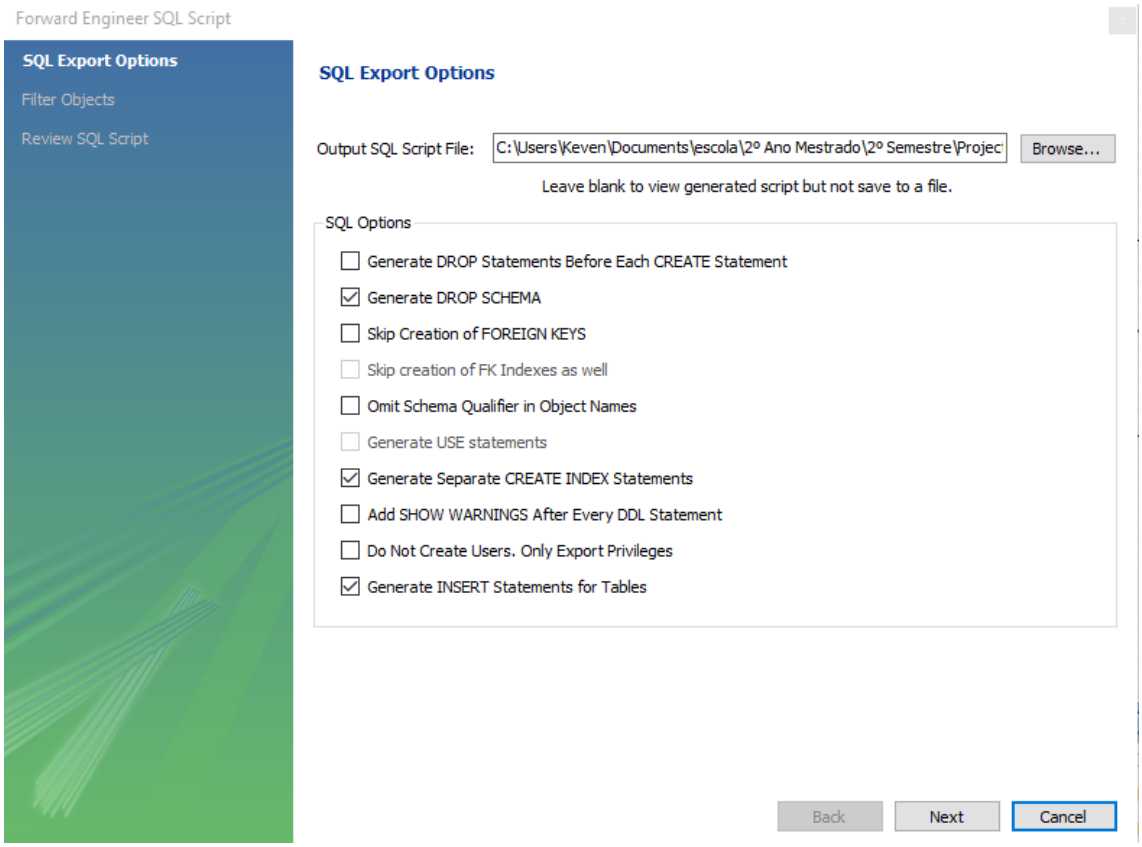


Figura A-6: Exportação do *Script* da Base de Dados.

Concluindo dessa forma que o Mysql Workbench é uma das ferramentas mais indicadas para trabalhar no nosso projeto, pois sendo "*open Source*" dá a possibilidade de trabalhar com essa ferramenta sem ser pirateada sendo que hoje em dia maioria das pessoas utilizam *softwares* pirateados e também contem inúmeras formas de trabalhar na criação de base de dados, trás muitas vantagens quanto ao facto de disponibilizar o código gerado pelas tabelas criadas graficamente no ambiente de trabalho e com os respetivos dados inseridos para cada coluna que constitui as tabelas.

A.3 NotePad++

Esse é um dos *softwares* utilizado no nosso projeto quando era precisa criar/editar as partes dos códigos que compõem o nosso projeto. É um *software* "*Open Source*" isto é que está disponível na *Web* (*Internet*) para fazer *download* gratuitamente. No nosso trabalho foi utilizada a versão *Notepad++ Je suis Charlie*, esse que era a versão disponível quando se desenvolvia o nosso sistema.

Notepad++ é um programa com um ambiente de trabalho de grande facilidade de trabalho, em que, já vêm incorporadas diversas linguagens de programação e marcação tais como (*C, C++, CSS, HTML, Java, Java script, PHP, XML*, entre muito outras), ainda tem algumas funcionalidades como executar em algumas dos *browsers* tal como Google Chrome, Firefox, entre outras. Também pelo facto de ter diversos temas já instaladas que ajudam na programação, pois tornam mais fácil identificar partes nos códigos e também na programação em si.

O Notepad++ foi lançado em 24 de novembro de 2003, e no momento a versão mais estável é a 6.7.3 que foi lançada á 1 de janeiro de 2015. Foi desenvolvido por Don Ho. "Notepad++ é um editor de texto e de código fonte de código aberto sob a licença GPL. Suporta várias linguagens de programação rodando sob o sistema Microsoft Windows. As linguagens de programação suportadas pelo Notepad++ são: C, C++, Java, C#, XML, HTML, PHP, Java Script, makefile, ASCII art, doxygen, ASP, VB/VBScript, Unix Shell Script, BAT, SQL, Objective-C, CSS, Pascal, Perl, Python, Lua, Tcl, Assembly, Ruby, Lisp, Scheme, Smalltalk, PostScript e VHDL. Além disto, usuários podem definir suas próprias linguagens usando um "sistema de definição de linguagem" integrado, que faz do Notepad++ extensível, para ter realce de sintaxe e compactação de trechos de código." (Anexo 5 [W80]), (Jung & Brown, 2007).

Podemos agora avançar para a parte do nosso código onde serão apresentadas somente as partes mais importantes do mesmo, pois não seria profissionalismo apresentar de forma extensa a totalidade do código da base de dados. Focando nessa parte somente a base de dados do sistema da garagem que contem as tabelas acima indicadas. Das partes do código cá apresentado iremos comentar e indicar o funcionamento.

```
--
-- Estrutura da tabela `alertas`
--
CREATE TABLE IF NOT EXISTS `alertas` (
  `Id_Alertas` int(11) NOT NULL,
  `Id_clientes` int(11) NOT NULL,
  `Id_Convidado` int(11) DEFAULT NULL,
  `Id_Carro` int(11) NOT NULL,
  `Acontecimento` varchar(145) NOT NULL,
  `Hora` datetime NOT NULL,
  `Dia` datetime NOT NULL,
  `MES` datetime NOT NULL
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Script A-1: Criação da Tabela "Alertas".

O Script *A-1* acima referido indica-nos o código utilizado para a criação da tabela Eventos da nossa base de dados e também o código para inserir dados nas colunas da nossa tabela. O código começa com "*CREATE TABLE IF NOT EXISTS `eventos`*" isto que é o comando utilizado para a criação da tabela e o que se segue são as colunas a serem introduzidas, temos o '*Id_Evento'* que é uma variável de forma Inteiro, ou seja só é possível guardar em '*Id_Evento*' valores reais, nunca pode tomar o valor 0 (*NULL*), este que vai incrementando com cada introdução de novo valor na coluna. Já 'Hora', 'Dia' e 'Mês' são variáveis do tipo Data, pois se regista uma data. A codificação utilizada é "*CHARSET=utf8*" que é um tipo de codificação que pode representar qualquer carácter universal padrão do Unicode, sendo também compatível com o ASCII. Como também é possível reparar no código é que a nossa base de dados utiliza o mecanismo de armazenamento *Innodb*, que é representado por "*ENGINE=InnoDB*". De seguida, introduzimos alguns dados na nossa base de dados, para fazer isso, utilizando a função "*INSERT INTO*" de seguida o nome da tabela e os determinados valores utilizando a função "*VALUE*", como se vê no Script *A-2*.

```
--
-- Extraindo dados da tabela `alertas`
--
INSERT INTO `alertas` (`Id_Alertas`, `Id_clientes`, `Id_Convidado`,
`Id_Carro`, `Acontecimento`, `Hora`, `Dia`, `MES`) VALUES
```

Script A-2: Código que se utiliza para inserir dados em uma tabela.

Como podemos ver os dados são introduzidos de acordo com o tipo de variável que vai guardar os dados, por exemplo, no caso de introduzir dados no campo "Saida" tem um formato diferente quando introduz uma data.

Temos ainda outra parte importante do código que é a ligação das tabelas utilizando as chaves estrangeiras, isto que é representado no Script *A-3*.

```
-- Indexes for table `alertas`
--
ALTER TABLE `alertas`
  ADD PRIMARY KEY (`Id_Alertas`),
  ADD KEY `fk_Alertas_Pessoas1_idx` (`Id_clientes`),
  ADD KEY `fk_Alertas_Convidados1_idx` (`Id_Convidado`),
  ADD KEY `fk_Alertas_Carro1_idx` (`Id_Carro`);
--
-- Indexes for table `carro`
--
ALTER TABLE `carro`
  ADD PRIMARY KEY (`Id_Carro`),
  ADD UNIQUE KEY `VIN_UNIQUE` (`VIN`),
  ADD KEY `fk_Carro_Eventos1_idx` (`Eventos`),
  ADD KEY `fk_Carro_clientes_idx` (`Id_Cliente`);
--
-- Indexes for table `carro_2`
--
ALTER TABLE `carro_2`
  ADD PRIMARY KEY (`Id_Carro`),
  ADD UNIQUE KEY `Eventos_UNIQUE` (`Eventos`);
```

Script A-3: Identificação das chaves primárias e estrangeiras das tabelas.

Como podemos reparar o *script* começa com a função "*ALTER TABLE*", este que serve para alterar uma determinada tabela que está presente na base de dados. Depois escolhemos o campo mais adequado para ser a chave primária, ou seja, o campo que será único e que servirá com identificação para cada coluna da nossa tabela, isto é feito com a função "*ADD PRIMARY KEY*", também para a tabela "Alertas" tínhamos chaves estrangeiras, por código é um pouco mais complexo, esse que é feito com a função "*ADD KEY*", depois criamos um campo que o guardará, nesse caso '*fk_Carro_Eventos1_idx*' e de seguida indicamos o valor que tem dado nesse caso por ('Eventos'), e assim são feitas para os outros campos.

Repare-se também que em determinados campos foi necessário ter campos únicos, ou seja, que não poderia ter nenhum campo com valor igual. Para a tabela "Carro" que também tem chave estrangeira e chave primária, criamos um campo único através da função "*ADD UNIQUE KEY*" seguindo do campo desejado.

B. ANEXO – PLATAFORMA *ONLINE/WEBSITE*

B.1 Criação e Desenvolvimento do *Website*

Para a criação da Plataforma *online*, tive que aprender diversas linguagens de programação, linguagens essas tais como, *HTML*, *CSS*, *PHP* e *JS*. Em que, a linguagem *HTML* é utilizada especialmente para o *design* das páginas do nosso *website*, o *CSS* é o que faz o *style* das páginas, e o *PHP* é utilizado precisamente para a troca de dados entre as páginas *web* e a base de dados, todos os sítios da *internet* consultados em relação aos mesmos podem ser consultados no Anexo 5[11].

O desenvolvimento dessas páginas foi feito utilizando diferentes ferramentas de programação, os principais que foram utilizados são:

- Dreamweaver;
- Notepad++;
- Subline3;
- Xampp.

B.1.2 Xampp

Na fase inicial do projeto, de criação do *website*, por ausência de um servidor para alocar a nossa base de dados, deste modo tive que criar um servidor local, onde fiz isso utilizando o Xamapp. ”XAMPP é um servidor independente de plataforma, *software* livre, que consiste principalmente na base de dados MySQL, o servidor *web Apache* e os interpretadores para linguagens de *script*: *PHP* e *Perl*. O nome prove da abreviação de X (para qualquer dos diferentes sistemas operativos), *Apache*, *MySQL*, *PHP*, *Perl*. O programa está liberado sob a licença *GNU* e atua como um servidor *web* livre, fácil de usar e capaz de interpretar páginas dinâmicas.”.

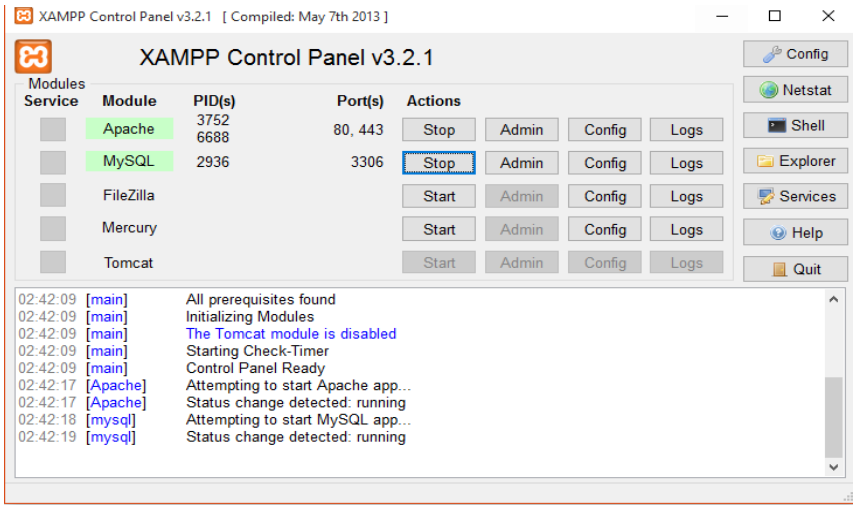


Figura B-1: Painel de Controlo do Xampp.

[11] ANEXO 5 – Sítios da Internet Relevantes

Utilizando o Xampp criou-se uma base de dados usando o meu computador como servidor, isto que foi possível através do *script* já demonstrado no capítulo de criação da base de dados. É também possível criar uma base de dados e diferentes tabelas de modo manual a partir do ambiente de trabalho do Xampp. Para aceder à página que dá à possibilidade de modificar e criar tabelas tem que ir a diretoria de *http://localhost/phpmyadmin/* Figura *B-2* onde consegues criar as tabelas e editar os mesmos. Para isso temos que primeiro aceder ao painel de controlo do Xampp para iniciar *o Apache e o Mysql* Figura *B-1*.

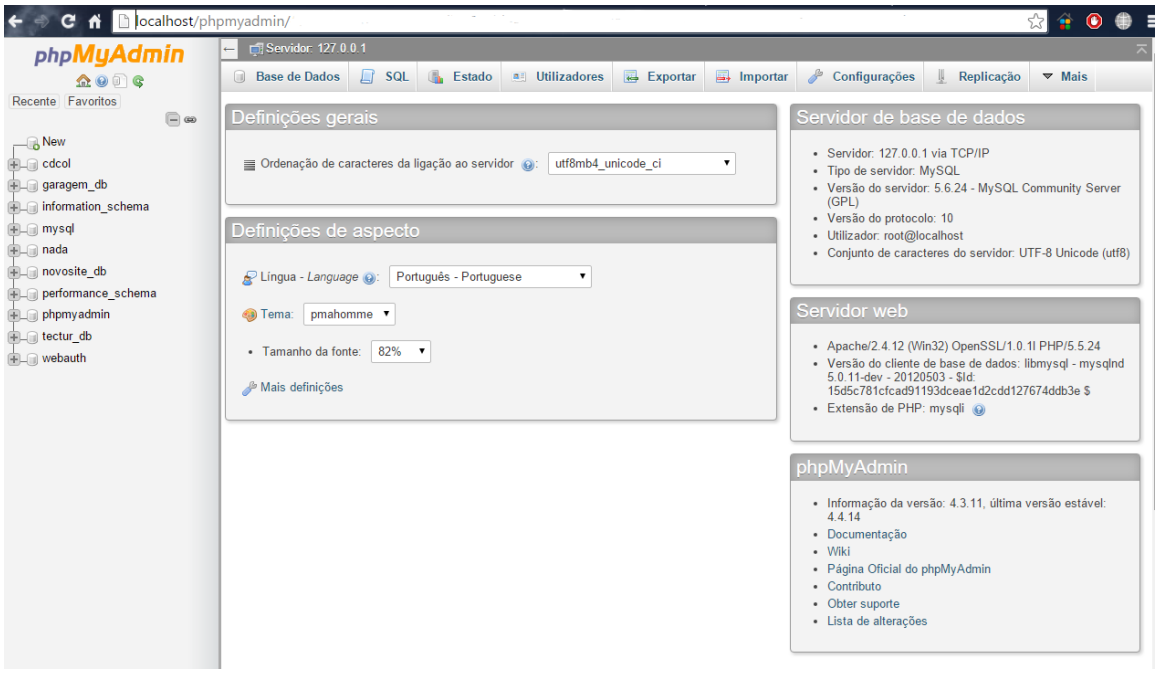


Figura B-2: Pagina *web* para o controlo do servidor.

B.1.2 Adobe Dreamweaver

Falando agora um pouco do Dreamweaver Figura *B-3*, que foi uma ferramenta essencial na criação do *website*, pois é uma ferramenta, que ao princípio é um pouco complicado de trabalhar, mas ao adaptar a essa ferramenta podemos fazer bons trabalho e com uma grande facilidade quando comparada com outras ferramentas de criação de *website*s. Hoje em dia o Dreamweaver representa uma das maiores e mais utilizadas ferramentas de *web design*, pois pertence a Adobe (Adobe Systems 2000, 2001, 2008), que é uma grande empresa quando se refere a *design* de um modo geral.

Figura B-3: Dreamweaver Logo.

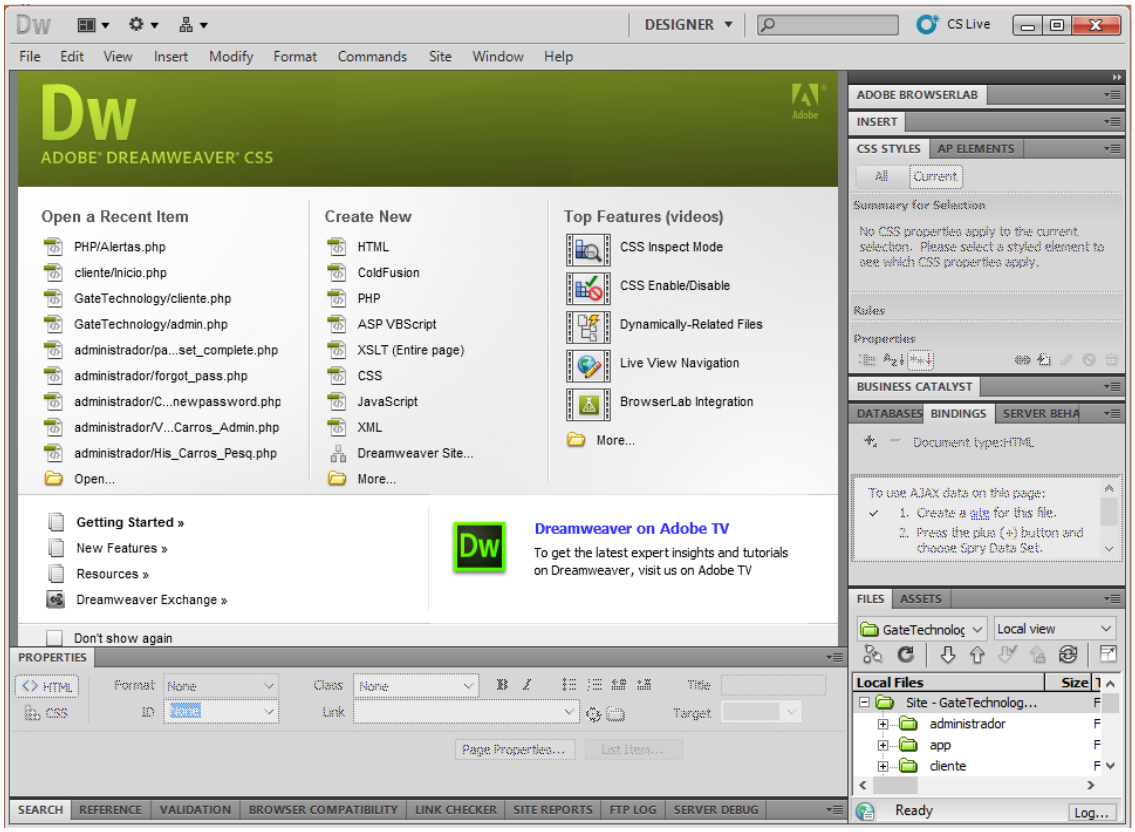


Figura B-4: Ambiente trabalho do Dreamweaver.

Na imagem Figura *B-4* pode ver à página inicial do Dreamweaver, com várias opções de trabalho, caso seja a vez a utilizar o Dreamweaver, esse já vem com um tutorial para ajudar a adaptar a essa ferramenta. No caso do Notepad++ e o Subline3 está explicado todo o modo de funcionamento dos mesmos no Anexo 1, (McFarland & Grover, 2013).

B.2 Criação da Pagina Inicial

Partindo agora para o processo de criação da página inicial, neste caso o "*index.html*", esse nome porque é o nome que os servidores leem como padrão para ser à página principal. A página inicial é do tipo "*One Page*" de modo que tudo se encontra nessa página, tem como exceção a página do demo da Aplicação *Android*. Página essa que foi desenvolvida de modo a ser o mais simples possível e com o máximo de informação sobre o produto.

No início do *script* de um *website*, primeiro são "chamados" os ficheiros que contém o *Style* (*CSS*), o *fav-icon* (é a imagem 16X16 que aparece no início no browser), e no caso o *website* seja responsivo (adapta ao tamanho do dispositivo em que é aberto o *website*), e também é onde se atribui o título do *website*, isto tudo está contido no "<*head*>" dado no *Script B-1*, (Harris, 2011).

```
<head>
  <meta charset="utf-8">
  <!-- PAGE TITLE -->
  <title>Gate Technology</title>
  <!-- MAKE IT RESPONSIVE -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- SEO -->
  <meta name="description" content="">
  <meta name="author" content="">
  <meta name="keywords" content="">
  <!-- FAVICON -->
    <link rel="shortcut icon" href="images/favicon.ico" />
  <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
  <link href="css/animate.min.css" rel="stylesheet" media="screen">
  <link href="css/font-awesome.min.css" rel="stylesheet" media="screen">
  <link href="style.css" rel="stylesheet" media="screen">
  <link href="css/options.css" rel="stylesheet" media="screen">
  <link href="css/responsive.css" rel="stylesheet" media="screen">
</head>
```

Script B-1: Definição do "*head*" do *website*.

Depois disso começa a trabalhar no "*<body>*" do *website*, neste caso onde está o Menu inicial *Script B-2*, o contato, e diferentes links para contato como por exemplo o do Facebook (*https://www.facebook.com/RemoteGate*) *Figura B-5*, foi criado uma página de Facebook para contatar os produtores e também para Marketing. Avançando agora mais no *Body* do *website* criei diferentes "*boxes*" para introduzir algum texto sobre o trabalho, esse que está no *Main Container*.

Figura B-5: Página do Facebook do *Gates Technology*.

```
<nav id="navigation">
<!-- DISPLAY MOBILE MENU -->
<a href="#" id="show-mobile-menu"><i class="fa fa-bars"></i></a>
<a href="#" id="close-navigation-mobile"><i class="fa fa-long-arrow-
left"></i></a>
<ul id="left-navigation" class="animate-me fadeInLeftBig">
<li class="menu-item menu-item-has-children">
<a href="#header" class="inside-link">Ínicio</a>
</li>
<li class="menu-item menu-item-has-children">
<a href="#skillsfor" class="inside-link">Serviços</a>
</li>
<li class="menu-item"><a href="#aplicacao" class="inside-
link">Aplicação</a></li>
</ul>
<div  class="animate-me flipInX" data-wow-duration="3s">
<a href="#header" id="logo-navigation" class="inside-link">
</a>
</div>
<ul id="right-navigation" class="animate-me fadeInRightBig">
<li class="menu-item"><a href="#team-container" class="inside-link">
Sobre</a></li>
<li class="menu-item menu-item-has-children">
<a href="#blog-container" class="inside-link"> Download</a>
</li>
<li class="menu-item"><a href="#contact-container" class="inside-link"><i
class="fa fa-envelope"></i> Contactos</a></li>
</ul>
</nav>
```

Script B-2: Criação do Menu principal.

Como referido antes criei um pequeno demo para levar as pessoas a conhecer mais da aplicação *Android*, de seguida apresentarei algumas imagens dessas páginas e os *scripts* que compõem os mesmos. Na página principal existe um botão "Conhecer" que redireciona para uma nova página onde está o demo, a função que faz isso é "*href="app.html*" *target="_blank*" onde o *href* é onde está o *link* da página e o "*Target blank*" faz com que seja aberto um novo separador no browser para abrir essa página.

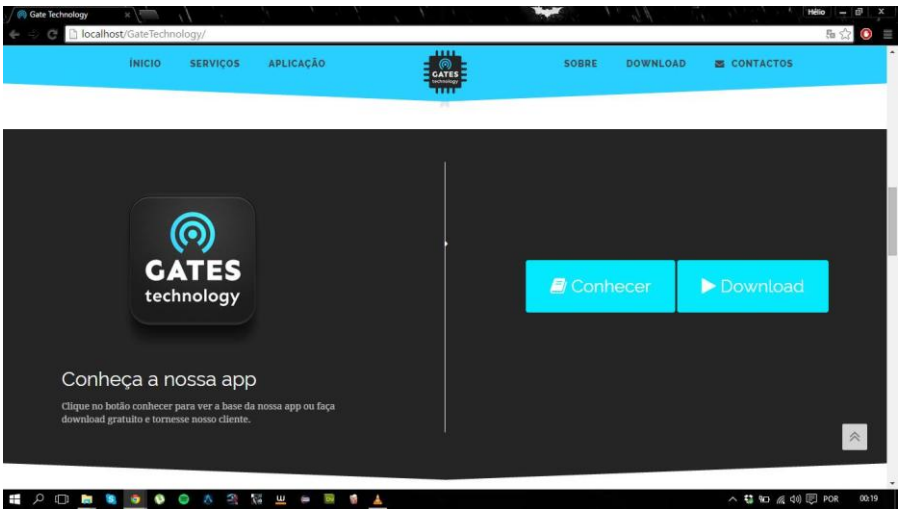


Figura B-6: Apresentação do "*body*" da página.

```
<div id="aplicacao">
<h2 class="with-breaker animate-me fadeInUp">
Conheça a Aplicação <span>Sim, desenvolvida para si totalmente grátis
!</span><br><br>
</h2>
</div>
<div class="appshow">
<section class="custom-section-container with-separation-bottom with-
separation-top">
<div class="container">
<div class="custom-section-text">
<img src="images/logoappgt.png" id="logoapp">
<h2>Conheça a nossa app</h2>
<p>Clique no botão conhecer para ver a base da nossa app ou faça download
gratuito e tornesse nosso cliente.</p>
</div>
<div class="custom-buttons-edit">
<div class="custom-section-buttons">
<a href="app.html" target="_blank" class="btn btn-app"><i class="fa fa-
book"></i> Conhecer</a>
<a href="#" class="btn btn-app"><i class="fa fa-play"></i> Download</a>
</div>
      </div>
            </div>
                  </section>
```

Script B-3: *Script* do *Custom Container*, criação da página apresentada anteriormente.

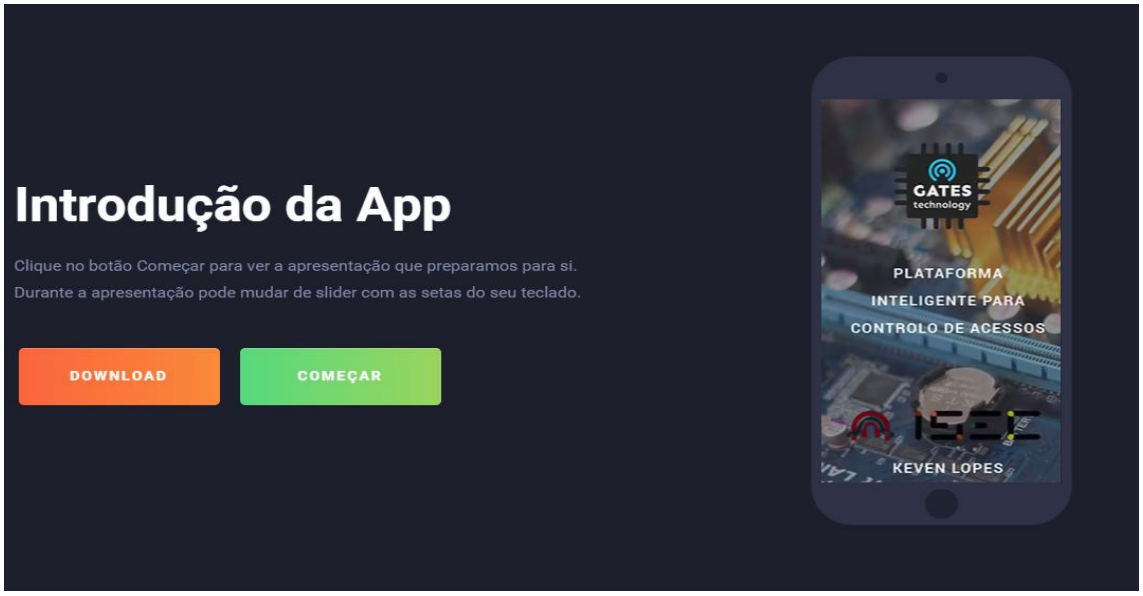


Figura B-7: Página do *demo* da aplicação.

```
<!doctype html>
<html lang="en" class="no-js">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="shortcut icon" href="images/favicon.ico" />
<link rel="stylesheet" href="app/css/reset.css"> <!-- CSS reset -->
<link rel="stylesheet" href="app/css/style.css"> <!-- Resource style -->
<script src="js/modernizr.js"></script> <!-- Modernizr -->
<title>Introdução da App | Gate Technology</title>
</head>
<body>
<header class="cd-header">
<div id="cd-logo"><a href="index.html"><img src="images/logo-header.png"
alt="Logo"></a></div>
<div id="cd-action"><a href="#" class="btn">Download</a></div>
<div id="cd-action"><a href="index.html" class="btn salmon">Gate
Technology</a></div>
</header>
<main class="cd-main-content">
<div class="cd-product-intro">
<h1>Introdução da App</h1>
<p>Clique no botão Começar para ver a apresentação que preparamos para
si.</p>
<div class="cd-triggers">
<a href="#" class="btn">Download</a>
<a href="#cd-product-tour" class="btn salmon" data-type="cd-
tour">Começar</a>
</div>
</div> <!-- cd-product-intro -->
```

Script B-4: Criação do demo.

Foi adicionada também na página do *website* pequenas informações sobre a comunicação entre o carro, à garagem e a aplicação *Android* de seguida pode-se verificar através de diferentes imagens da página e dos respetivos *scripts*, mas como os textos presentes nas páginas encontram-se unicamente em uma linha não será possível apresentar os mesmo de uma forma a conseguir esses nos. Depois foi desenvolvido um *script* que contém um mapa de localização do criador do produto, neste caso o endereço referido é o do ISEC. Do mesmo foi criado um formulário de contato diverso, desde uma pergunta, pedido de suporte ou também para adquirir o produto. Para isso o que se precisa é o nome, um *email*, um assunto e a respetiva mensagem. Em seguida através da Figura *B-8* e do Script *B-5* pode-se ver como se deu a criação do mesmo.

Figura B-8: Mapa e formulário de contato.

```
Contactar <span>Entre em contacto connosco</span>
</h2>
<p class="center"></p>
<div class="row">
<div class="col-md-6"><!-- Google Map -->
<div id="GoogleMap"></div>
</div>
<div class="col-md-6"><!-- FORMS -->
<form id="contact-form" method="post" action="contact_processing.php"
class="form-horizontal"><!-- NAME -->
<div class="control-group row">
<label class="control-label col-md-3" for="contact-name">Nome</label>
<div class="controls col-md-9">
<input id="contact-name" name="contact-name" type="text"
placeholder="Introduza o seu nome" class="input-xlarge form-control"
required="">
</div>
</div><!-- EMAIL -->
<div class="control-group row">
<label class="control-label col-md-3" for="contact-email">Email</label>
<div class="controls col-md-9">
<input id="contact-email" name="contact-email" type="text"
placeholder="introduza o seu email" class="input-xlarge form-control"
required="">
</div>
```

Script B-5: Criação do formulário de contato.

B.4 Criação da Pagina de Início de Sessão

Nas páginas em que os utilizadores efetuam o *login*, utilizam *HTML* e *PHP*, pois como já antes referido é necessário o *HTML* juntamente com o *CSS* para fazer o *design* e atribuir estilo aos componentes das páginas. Para essas páginas, utilizo o *PHP* porque tenho que fazer comunicação entre as páginas *web* e a base de dados. Para que as páginas acedem à base de dados, primeiro tenho que criar um ficheiro *PHP* para efetuar a comunicação.

```
<?php
# FileName="Connection_php_mysql.htm"
# Type="MYSQL"
# HTTP="true"
$hostname_conecao = "localhost";
$database_conecao = "gate_db";
$username_conecao = "root";
$password_conecao = "";
$conecao = mysql_pconnect($hostname_conecao, $username_conecao,
$password_conecao)
or trigger_error(mysql_error(),E_USER_ERROR);
?>
```

Script B-6: Código utilizado para aceder a base de dados.

Nesse ficheiro o que temos é primeiro temos o nome do "*Host*" que nesse caso é o nosso computador então se denomina de "*localhost*", depois precisamos do nome da base de dados em que temos "*database*" é "*garagem_db*", por fim precisamos de um utilizador e a palavra passe do mesmo. Depois, através da função "*Mysql_connect*", conseguimos aceder a nossa base de dados enviando como parâmetro as variáveis antes declaradas. Caso existe algum erro, através da "*trigger_error*" ou "*die*" bloqueasse a tentativa de conexão.

Para essas páginas criei uma animação de pequenos blocos a percorrer o ecrã, levando assim à necessidade de utilizar o *JS* ou pelo nome mais conhecido *JavaScript*, é um código simples em que somente efetua um "*fade*" e também para alguma das animações dos botões, (Olsen, 2007), (Lerdorf et al, 2006).

```
$("#login-button").click(function(event){
             event.preventDefault();

       $('form').fadeOut(500);
       $('.wrapper').addClass('form-success');
});
```

Script B-7: Código em *JavaScript* para as animações.

O formulário para efetuar o *login*, neste caso o nome de utilizador e a palavra passe, foi desenvolvido à base de *HTML* através do seguinte código.

```
<a href="index.html"><img src="images/logo-header.png" id="logologin"></a>
<h1>Bem Vindo</h1>
<form ACTION="<?php echo $loginFormAction; ?>" METHOD="POST" class="form">
<input type="text" name="username" placeholder="Utilizador">
<input type="password" name="passe" placeholder="Password">
<button type="submit" id="login-button">Entrar</button>
</form>
<form class="form">
<a href="index.html" id="login-link">Voltar para a página inicial.</a>
</form>
```

Script B-8: Formulário de Início de Sessão.

```php
<?php
// *** Validate request to login to this site.
if (!isset($_SESSION)) {
  session_start();
}
if (isset($_POST['username'])) {
  $loginUsername=$_POST['username'];
  $password=$_POST['passe'];
  $MM_fldUserAuthorization = "";
  $MM_redirectLoginSuccess =
"administrador/Inicio_Admin.php?username=$loginUsername";
  $MM_redirectLoginFailed = "admin.php";
  $MM_redirecttoReferrer = false;
  mysql_select_db($database_Conecxao, $Conecxao);
  $LoginRS__query=sprintf("SELECT Login, Password FROM inico_sessao WHERE
Login=%s AND Password=%s",
    GetSQLValueString($loginUsername, "text"), GetSQLValueString($password,
"text"));
  $LoginRS = mysql_query($LoginRS__query, $Conecxao) or die(mysql_error());
  $loginFoundUser = mysql_num_rows($LoginRS);
  if ($loginFoundUser) {
     $loginStrGroup = "";
      if (PHP_VERSION >= 5.1) {session_regenerate_id(true);} else
{session_regenerate_id();}
    //declare two session variables and assign them
    $_SESSION['MM_Username'] = $loginUsername;
    $_SESSION['MM_UserGroup'] = $loginStrGroup;

    if (isset($_SESSION['PrevUrl']) && false) {
      $MM_redirectLoginSuccess = $_SESSION['PrevUrl'];
    }
    header("Location: " . $MM_redirectLoginSuccess );
  }
  else {
    header("Location: ". $MM_redirectLoginFailed );
  }
}
```

Script B-9: Aceder à base de dados e iniciar sessão.

Como pode reparar no Script *B-9* é utilizado diversas funções definidas em *php* para iniciar sessão utilizando o nome de utilizador e a palavra passe correta. No início é feito um inicio fazemos um inicio de sessão com a função "*session_strart();*" e depois definindo diferentes variáveis para o nome de utilizador e palavra passe já introduzida, vamos guardar esses valores para depois ver se essas estão corretas. Também são definidas as páginas caso o *login* seja efetuada com sucesso (*"$MM_redirectLoginSuccess = "administrador/Inicio_Admin.php";")* e caso não (*$MM_redirectLoginFailed = "admin.php";).* Em seguida é aberta a conexão a base de dados utilizando a função "*mysql_select_db($database_Conecxao, $Conecxao);*" passando nesse caso como parâmetros, os dados necessários para aceder a base de dados. Por último é feito uma leitura a tabela onde estão os dados dos utilizadores, temos então (*"SELECT Login, Password FROM inico_sessao WHERE Login=%s AND Password=%s",)* depois disso é feito a comparação dos valores, caso estejam corretos é redirecionado para a página definida e caso não fica na mesma página.

B.2 Página de Controlo do Administrador

Descrito antes o processo de fazer o *login*, agora irei falar da página que será redirecionada depois que for feito o *login* com sucesso, nesse caso em específico irei referir as páginas do administrador e também indicar os *scripts* que as compõem. Em termos de *HTML* essas páginas são mais simples, pois não queria que essas páginas fossem de certa forma complexas a vista das pessoas que irão utilizar essas páginas. Também por serem *HTML* e *PHP* irei focar ao máximo possível nas partes mais importantes como, por exemplo, as que consigo ter acesso à base de dados para adicionar, remover ou editar. Primeiro vou falar da página em que o administrador tem a possibilidade de adicionar novos clientes a base de dados para que esses terem acesso ao *backend* do *website*.

```
<form method="POST" action="<?php echo $editFormAction; ?>"
name="form">
<fieldset>
<legend>Registar Pessoas</legend>
Nome:<input type="text" name="nome" placeholder="Introduz
Nome"required size="60px;"/>
Sexo: <input type="radio" name="sex" value="male" checked>M
<input type="radio" name="sex" value="female">F<p></p>
E-mail:<input type="email" name="email" size="50" id="email"
placeholder="Introduz Email" required/>
Número de Carros:input type="num" name="nr_carros" size="1"
required/><p></p>
Marca:<input type="text" name="Marca" placeholder="Introduz a Marca"
required size="30px;"/>
N° Identificação do Carro:<input type="num" name="VIN" required
size="20px;"/><p></p>
N° Telefone:<input type="num"  name="telefone" placeholder="Introduz
Numero" required/>
N° BI:<input type="num"  name="bi" placeholder="Introduz Numero"
required/>
N° NIF:<input type="num" name="nif" placeholder="Introduz Numero"
required/><p></p>
Login:<input type="text" name="log" placeholder="Nome de Utilizador"
required />
Password:<input type="text" name="passe" placeholder="Introduz
Passe" required />
Tipo:<select name="elemento2">
     <option value="Administrador"> Administrador </option>
     <option value="Cliente"> Cliente </option>
</select>
```

Script B-10: Formulário para registar novos clientes na base de dados.

No Script *B-10* está representado o *script* em *HTML* do formulário que é preciso preencher para adicionar novos clientes na base de dados pelo administrador, e através do código definido logo no início como "*method="POST" action="<?php echo $editFormAction; ?>""* é a parte que faz como que seja possível relacionar *HTML* com *PHP*. Como podemos ver na Figura *B-9*, esse é o formulário criado com o *script* antes referido.

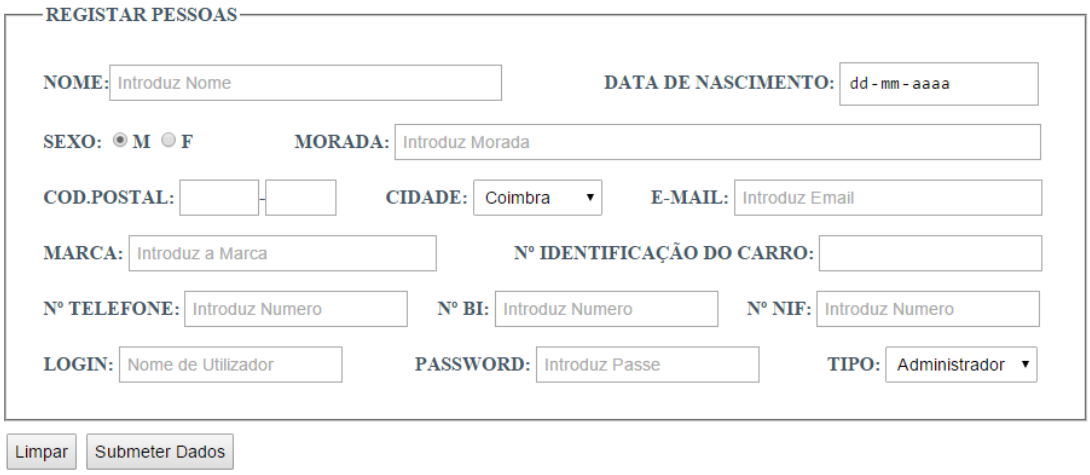


Figura B-9: Formulário para adicionar novo cliente a base de dados.

```
if ((isset($_POST["MM_insert"])) && ($_POST["MM_insert"] == "form")) {
  $insertSQL = sprintf("INSERT INTO inico_sessao (Login, Password) VALUES
(%s, %s)",
                       GetSQLValueString($_POST['log'], "text"),
                       GetSQLValueString($_POST['passe'], "text"));
  mysql_select_db($database_Conecxao, $Conecxao);
  $Result1 = mysql_query($insertSQL, $Conecxao) or die(mysql_error());
}
if ((isset($_POST["MM_insert"])) && ($_POST["MM_insert"] == "form")) {
  $insertSQL = sprintf("INSERT INTO pessoas_tipo (Tipo) VALUES (%s)",
                       GetSQLValueString($_POST['elemento2'], "text"));
  mysql_select_db($database_Conecxao, $Conecxao);
  $Result1 = mysql_query($insertSQL, $Conecxao) or die(mysql_error());
}
```

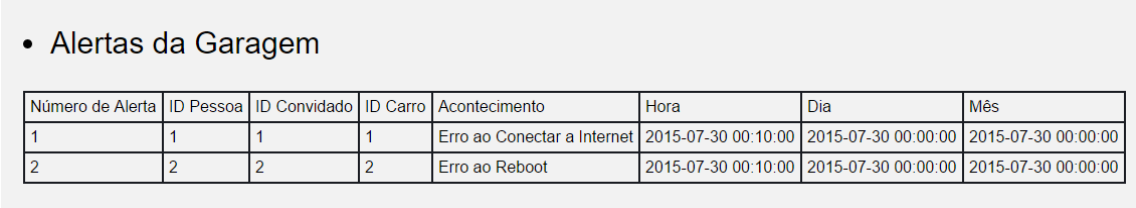
Script B-11: Representação do código que adiciona os dados a base de dados em *php*.

O Script *B-11* é o código em *PHP* para adicionar algum dos dados inseridos no formulário na base de dados, nesse caso são os dados do *login* e o tipo de pessoa (se é um cliente ou um administrador). Isto que em *HTML* faço a leitura de todos os dados guardados em variáveis no formulário e depois em *PHP* envia esses dados a partir da função "*INSERT INTO inico_sessao (Login, Password) VALUES (%s, %s)*", consigo inserir os dados do *login* na mesma tabela (inico_sessao). Sempre que vai realizar essa função é preciso conectar a base de dados e caso não seja possível através da função "*die(mysql_error();*" à função é interrompida. Os valores que serão adicionados na base de dados são obtidos através da seguinte função; *GetSQLValueString($_POST['log'], "text"),GetSQLValueString($_POST['passe'], "text"));*.

```
mysql_select_db($database_Conecxao, $Conecxao);
$query_Recordset1 = "SELECT * FROM alertas";
$Recordset1 = mysql_query($query_Recordset1, $Conecxao) or
die(mysql_error());
$row_Recordset1 = mysql_fetch_assoc($Recordset1);
$totalRows_Recordset1 = mysql_num_rows($Recordset1);
```

Script B-12: Selecionar dados da base de dados.

Referindo agora das páginas em que são apresentados dados que estão guardadas no servidor, irei então falar da página que apresenta as alertas feitas pela garagem, essa que é uma página que só o administrador tem acesso. Em primeiro é feito uma ligação à base de dados utilizando a função "*<?php require_once('../Connections/Conecxao.php'); ?>*" essa função que é declarada no início do *script* da página. Depois é feita uma leitura das tabelas em que estão os dados que se pretende mostrar, e por último através da função "*echo*" esses dados são apresentados na página.



- Alertas da Garagem

Número de Alerta	ID Pessoa	ID Convidado	ID Carro	Acontecimento	Hora	Dia	Mês
1	1	1	1	Erro ao Conectar a Internet	2015-07-30 00:10:00	2015-07-30 00:00:00	2015-07-30 00:00:00
2	2	2	2	Erro ao Reboot	2015-07-30 00:10:00	2015-07-30 00:00:00	2015-07-30 00:00:00

Figura B-10: Apresentação de algumas alertas registadas na base de dados.

```
<div class="alertas">
<li><a>Alertas da Garagem</a></li>
<table border="1"><tr>
<td>Número de Alerta</td>
<td>ID Pessoa</td>
<td>ID Convidado</td>
<td>ID Carro</td>
<td>Acontecimento</td>
<td>Hora</td>
<td>Dia</td>
<td>Mês</td></tr>
<?php do { ?>
<tr>
<td><?php echo $row_Recordset1['Id_Alertas']; ?></td>
<td><?php echo $row_Recordset1['Id_clientes']; ?></td>
<td><?php echo $row_Recordset1['Id_Convidado']; ?></td>
<td><?php echo $row_Recordset1['Id_Carro']; ?></td>
<td><?php echo $row_Recordset1['Acontecimento']; ?></td>
<td><?php echo $row_Recordset1['Hora']; ?></td>
<td><?php echo $row_Recordset1['Dia']; ?></td>
<td><?php echo $row_Recordset1['MES']; ?></td>
</tr>
<?php } while ($row_Recordset1 = mysql_fetch_assoc($Recordset1)); ?>
</table>
```

Script B-13: Apresentação dos dados lidos da base de dados.

C. ANEXO – APLICAÇÃO *ANDROID*

C.1 Menu Secundário da Aplicação

Na aplicação ainda vai estar, diversas funções disponíveis para o utilizador. O Menu secundário é acedido caso o utilizador pressionar o botão "Lazer", sendo esta a janela em que o será feito um display de um Menu com 4 (quatro) botões mais importantes, onde temos: botão de *Internet*, de Músicas, Notícias e Jogos. Para mais informações sobre os sítios da *internet* acedidos, podem ser consultados no Anexo 5[12].

Figura C-1: Maquete do Menu secundário.

Através do botão "*Internet*", será aberto o Google Chrome ou *Android Browser*, em que consegue ir à *internet* para fazer pesquisar ou qualquer coisa que o utilizador preferir. Com os botões, "Música", "Notícias", "Jogos" é aberta novas janelas *Figura C-2* onde é disponibilizada uma série de aplicações muito utilizadas hoje em dia. Essas Aplicações como, por exemplo, o Youtube, o Spotify, Perguntados, BBC News e muitos outros. Todas essas aplicações que já têm mais de 100 milhões de *downloads* atualmente destacam-se entre outras aplicações.

Essas foram incluídas na aplicação, para trazer de certa forma mais confortabilidade aos utilizadores, de forma que os utilizadores conseguem ouvir músicas, ver as notícias ou até jogar diferentes jogos a qualquer hora tudo isso a partir da aplicação, (Mednieks, 2012).

[12] ANEXO 5 – Sítios da Internet Relevantes

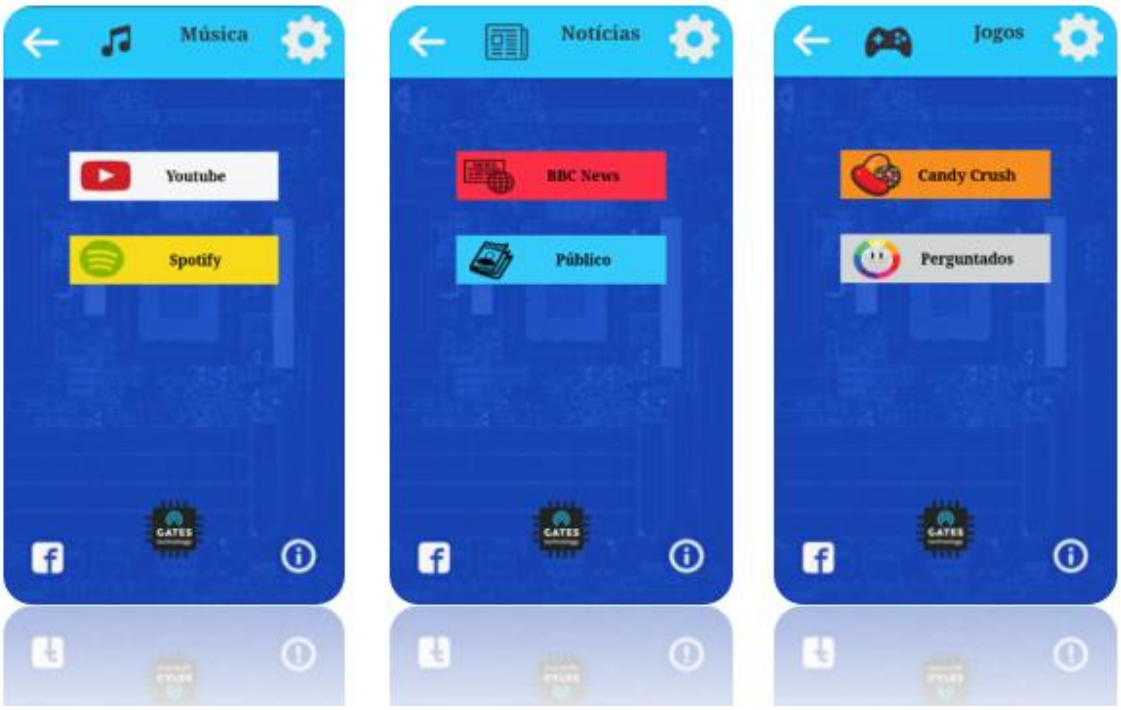


Figura C-2: Maquetes das páginas Música, Notícias e Jogos respetivamente.

A partir de todas essas janelas, é possível aceder à página do Facebook, do *Gates Technology*, consultar o *website* para conhecer melhor o produto, e, também, aceder à página de informação caso queira ler mais sobre a aplicação e o produto em si. As definições de conta (Figura C-3) consistem numa janela em que é possível aceder às seguintes páginas: Mudar Palavra passe; Mudar *Email* associada à conta; Contatar o administrador. O utilizador, através do botão "Administrador", consegue enviar um *Email* diretamente ao administrador, indicando questões, opiniões, ou a razão de contacto. Para, além disso, a aplicação está programada caso o produto esteja instalado numa residência, sendo possível receber um *SMS* pelo responsável, neste caso o administrador. Tal acontece sempre que a porta da garagem esteja aberta ou fechada através da aplicação, aumentando assim o nível de segurança do sistema.

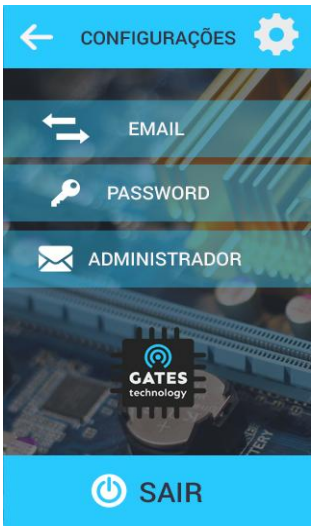


Figura C-3: Maquete da janela de Definições de Conta.

C.2 Criação e Desenvolvimento da Aplicação *Android*

Seguem nesse anexo, os diferentes procedimentos para a criação da aplicação *Android*, demonstrando pequenas partes dos códigos que constituem as diferentes classes da aplicação, (Lee, 2014). Os *scripts* serão comentados e explicados de uma forma simples para uma rápida compreensão.

C.2.1 Android MAINFEST

A parte principal de uma aplicação é o Android Mainfest, pois essa é a classe que controla toda a aplicação. A sua linguagem (de marcação) é *XML*, e é onde se define a versão de *Android* "*target*" da aplicação. Aqui, também se define a orientação dos *layouts* para a aplicação. Para a aplicação desenvolvida foi definida a sua orientação de "*Portrait*", isto significa que a aplicação só está em modo vertical nos dispositivos. Nesta parte da aplicação também é podemos atribui as permissões, como, no caso de: utilizar a *internet*, utilizar o *Wakelock*, utilizar a vibração do dispositivo, entre muitos outros. Aqui é onde a aplicação se inicia, onde é definida a atividade "*Main*" que é o *launcher*.

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
      <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
       </activity>
       <activity
            android:name=".Login"
            android:label="@string/title_activity_login"
            android:screenOrientation="portrait" >
            <intent-filter>
      <action android:name="com.gatestechnology.isec.Login" />
      <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
       </activity>
```

Script C-1: Sequência das classes.

O *Script C-1* indica qual a classe principal na aplicação, essa classe que inicia a aplicação onde é definido o "*MAIN*" e o "*LAUNCHER*". Ao iniciar a aplicação o *Android Mainfest* vai procurar primeiramente pelo "*launcher*". Depois disso, dependendo da organização das classes, quando se pretende abrir uma nova atividade, temos que referenciar a classe no *Mainfest*. Nesse caso a categoria da classe é "*DEFAULT*", (Haseman, 2012).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.gatestechnology.isec"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SEND_SMS" />
```

Script C-2: Adicionar permissões para a aplicação.

O Script *C-2* representa o código utilizado para adicionar permissões a aplicação, para o nosso caso vamos ter; permissão para utilizar a função "*Wake_Lock*", isto que não deixa a aplicação nunca entrar no "*sleep mode*", ou seja, quando a aplicação estiver aberta o *smarthphone* sempre estará ativo. Tem-se também a permissão para utilizar *internet*, pois, dentro da aplicação temos várias funções que precisam aceder a *internet*. Por último temos a função que dá permissão para enviar *SMS* para outro telemóvel. Define-se também o mínimo da versão do *Android* em que funciona a aplicação, (nesse caso é o "17") e também o "*target*" que é a versão principal desenvolvida a aplicação (versão "21").

```
<activity
            android:name=".Menu"
            android:label="@string/title_activity_menu"
            android:screenOrientation="portrait" >
            <intent-filter>
          <action android:name="com.gatestechnology.isec.Menu" />
          <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
</activity>
```

Script C-3: Declaração de uma atividade no *Mainfest*.

Como se pode reparar no Script *C-3* vamos ter uma atividade declarada no *Mainfest*. Onde o nome dessa classe é "*Menu*". Indica também a orientação da atividade é "*Portrait*", ou seja, a atividade só ficará no modo de retrato (em vertical). Depois indica o "*action*" que é o nome do pacote utilizado quando se pretende abrir essa atividade a partir de outra. Temos por último é a categoria que nesse caso, como já referido é "*DEFAULT*".

Para a programação do *Android Mainfest* utiliza a linguagem de marcação "*XML*", na Figura *C-4* apresenta a estrutura de uma aplicação quando essa é criada no *IDE Eclipse*, (James, 2013).

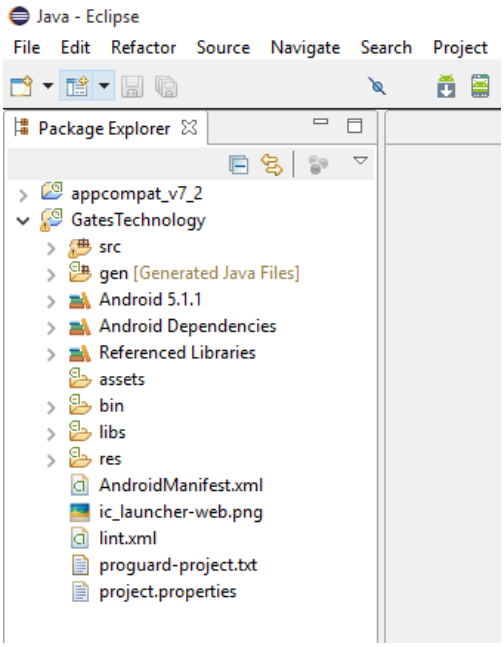


Figura C-4: Representação de uma aplicação, *Eclipse*.

Como pode ver na figura acima, ”*appcompat_v7_2*”, é utilizada para que a aplicação *Gates Technology* utlizada as diferentes bibliotecas disponibilizadas, para programações de aplicações.

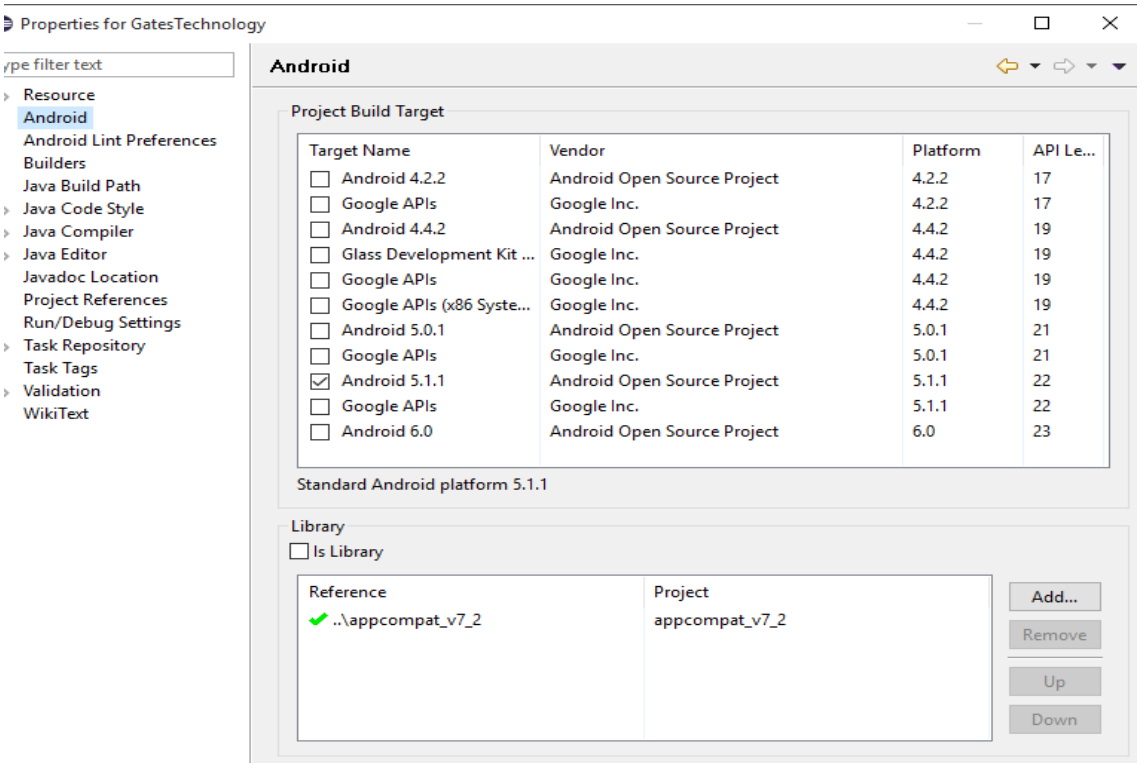


Figura C-5: Propriedades da aplicação *Gates Technology*.

C.2.2 Imagem *Splash Background*

Depois de ser iniciada a aplicação no dispositivo será apresentada uma imagem, durante um curto período de tempo, aproximadamente 6 (seis) segundos. Essa é a atividade em que o *Mainfest* chama pela primeira vez, quando é iniciada a aplicação. Neste caso, há um "*timer*" que conta o tempo para carregar os dados da base de dados, e, seguidamente chama a janela do *login*.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/splash"
    tools:context="com.isec.gate.MainActivity" >
</RelativeLayout>
```

Script C-4: Criação de um background.

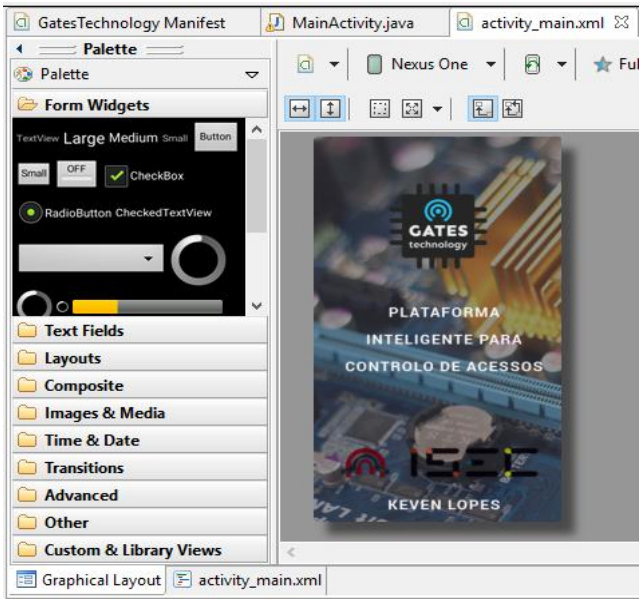


Figura C-6: *Background Splash*.

O Script *C-4* representa o código utilizado para a criação do "*background*" apresentada na Figura *C-6*. Onde declaramos um "*width*" (largura) e "*height*" (altura) como sendo "*match_parent*", isto é, a imagem importada através do comando "*Android*: *background='@drawable/splash"*" vai ocupar todo o ecrã, definido pelo *layout*.

Agora vamos indicar a constituição da classe que define a atividade do "*Splash Background*", o Script *C-5* representa então o início do mesmo.

```
package com.gatestechnology.isec;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.view.Window;
import android.view.WindowManager;

public class MainActivity extends Activity {
WakeLock wL;
@Override
protected void onCreate(Bundle savedInstanceState) {
//-----------------Wake-lock para nao fechar apl--------------------------
PowerManager pM = (PowerManager)getSystemService(Context.POWER_SERVICE);
             wL = pM.newWakeLock(PowerManager.FULL_WAKE_LOCK, "whatever");
             wL.acquire();
             super.onCreate(savedInstanceState);
//----------------Para ter em fullscreen e sem Titulo----------------------
-            this.requestWindowFeature(Window.FEATURE_NO_TITLE);
this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN
|WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED
|WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON,
 WindowManager.LayoutParams.FLAG_FULLSCREEN
|WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED
|WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON);
//----------------Abrir o Layout que se pretende usar----------------------
setContentView(R.layout.activity_main);
```

Script C-5: Inicialização e configuração da classe.

Inicia-se fazendo o importo das bibliotecas que serão utilizadas nessa classe. Defini a classe como sendo pública e é uma "atividade". Através do "*PowerManager*" e do "*WindowManager*", fazemos com que o "*layout*" referido nessa classe esteja sempre em "*fullscreen*" e com o ecrã sempre ativo.

```
//-----Ciclo de vida do  MainActivity-----------------------------
Thread timer  = new Thread(){
public void run(){
       try{
//-------Durante 5 segundos fica aberto depois fecha--------------
             sleep(5000);
       } catch (InterruptedException e){
       e.printStackTrace();
       }finally{
//-------Condição para abrir a janela Menu-----------------------
Intent janela_login =new Intent("com.gatestechnology.isec.Login");
startActivity(janela_login);
             }};
                    timer.start();
             }
@Override
protected void onPause() {
       super.onPause();
//------------termina a função-----------------------------------------
       finish();
}}
```

Script C-6: Ciclo de vida do *Main Activity*.

C.2.3 Página de Inicio de Sessão

A janela onde é feita o *Login*, é onde que o utilizador tem o primeiro contacto com a aplicação. Essa janela está conectada diretamente com a base de dados. De seguida temos a demonstração de alguns dos códigos mais importantes. Como já foi mencionada, a aplicação está desenvolvida em Inglês e Português; nessa janela o utilizador consegue escolher o idioma para a aplicação antes de iniciar sessão.

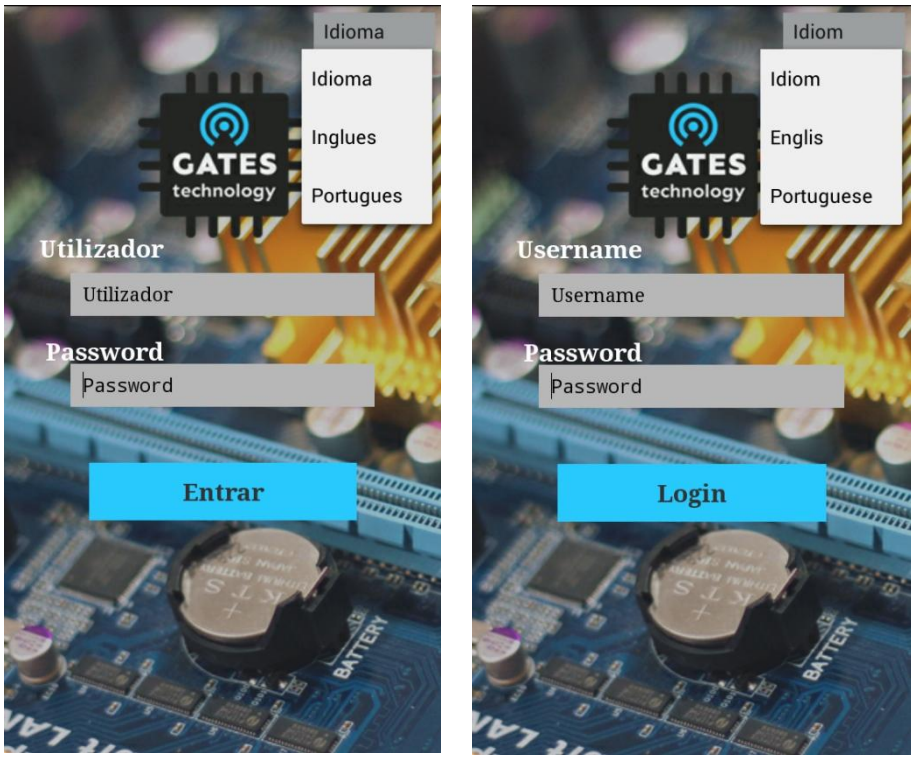


Figura C-7: Maquete do "*layout*" de início de sessão.

```
spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
public void onItemSelected(AdapterView<?> parent, View view, int pos, long
id) {
            if (pos == 1) {
Toast.makeText(parent.getContext(),
      "You have Selected English", Toast.LENGTH_SHORT)
      .show();
      setLocale("en");
            } else if (pos ==2) {
Toast.makeText(parent.getContext(),
      "Selecionaste Portugues", Toast.LENGTH_SHORT)
      .show();
      setLocale("pt");
      }}
public void onNothingSelected(AdapterView<?> arg0) {
}});
```

Script C-7: Escolha de idioma a utilizar.

C.2.4 Atividade Principal

Essa é a atividade mais importante da aplicação, pois é aqui que está o principal objetivo da mesma, que é o controlo da garagem. Aqui podem avançar para diferentes janelas ou atividades, como por exemplo: ir ao *Main Menu*, as definições de conta, janela de informações, ou, no caso de não fizer *logout* (desligar da aplicação). Como também já se sabe o facto de ser possível enviar um *SMS* para o administrador sempre que a porta da garagem for aberta ou fechada através da aplicação, serão explicados pelos respetivos códigos.

```
case R.id.buttonPin11:
// get the pin number
  String parameterValue = "";
// get the ip address
  String ipAddress ="192.168.1.147";
// get the port number
  String portNumber = "80";
// get the pin number from the button that was clicked
     if(v.getId()==buttonPin11.getId())
     {
        parameterValue = "11";
     }
// execute HTTP request
  if(ipAddress.length()>0 && portNumber.length()>0) {
       new HttpRequestAsyncTask(
       v.getContext(), parameterValue, ipAddress, portNumber, "pin"
         ).execute(); }
```

Script C-8: Enviar dados para o *Arduino* quando "*buttonPin11*" for pressionado.

O Script *C-8* apresenta uma parte do código da aplicação quando esse estiver a enviar comandos para o *Arduino* para abrir ou fechar a porta da garagem. Quando esse estiver conectado a *internet*, nesse caso através do *IP* "192.168.1.147".

```
private void sendSmsClose() {
String number ="5556";
String message ="Porta de Garagem Fechada";
SmsManager manager = SmsManager.getDefault();
manager.sendTextMessage(number, null, message, null, null);
Toast.makeText(getApplicationContext(), "Aviso de Porta Fechada Enviada",
Toast.LENGTH_LONG).show();
}
private void sendSmsOpen() {
String number ="5556";
String message ="Porta de Garagem Aberta";
SmsManager manager = SmsManager.getDefault();
manager.sendTextMessage(number, null, message, null, null);
Toast.makeText(getApplicationContext(), "Aviso de Porta Aberta Enviada",
Toast.LENGTH_LONG).show();
}
```

Script C-9: Envio de mensagens quando a porta for aberta ou fechada.

É dado agora o código das funções utilizadas nos testos para enviar uma mensagem de alerta a dizer que a porta foi aberta ou fechada. Nesse caso os números de telefones enviadas as mensagens, consiste nos emuladores utilizados.

C.1.1 Página do *VIN*

Refere-se aqui a constituição da atividade, que indica os números de identificação dos carros, pelo facto de ser uma atividade importante também da aplicação, pois está também conectada diretamente com a base de dados, aumentando o nível de segurança requerido para esta atividade na programação do mesmo.

```
String result = "";
InputStream isr = null;
try{
       HttpClient httpclient = new DefaultHttpClient();
       HttpPost httppost = new
HttpPost("http://www.gatestechnology.eu/phpAndroid/user_carros.php");
       HttpResponse response = httpclient.execute(httppost);
       HttpEntity entity = response.getEntity();
       isr = entity.getContent();
} catch(Exception e){
       Log.e("log_tag", "Error in http connection "+e.toString());
       resultView.setText("Couldnt connect to database");
}
//convert response to string
try{
       BufferedReader reader = new BufferedReader(new
       InputStreamReader(isr,"iso-8859-1"),8);
        StringBuilder sb = new StringBuilder();
        String line = null;
  while ((line = reader.readLine()) != null) {
          sb.append(line + "\n");
   }
    isr.close();
    result=sb.toString();
    }
catch(Exception e){
    Log.e("log_tag", "Error  converting result "+e.toString());
    }
//parse json data
   try {
          String s = "";
          JSONArray jArray = new JSONArray(result);
          for(int i=0; i<jArray.length();i++){
                JSONObject json = jArray.getJSONObject(i);
                s = s +
                "ID Carro : "+json.getString("Id_Carro")+"\n"+
                "VIN : "+json.getString("VIN")+"\n\n";
}
resultView.setText(s);
```

Script C-10*:* Código utilizado para quando se pretende ler dados da base de dados.

O Script *C-10* é utilizado nas classes que se pretender apresentar os dados que estão presentes na base de dados do *Gates Technology*. Para a atividade dos carros registados na base de dados também utiliza esse *script*. Através de um "*HTTP request*" vai aceder ao servidor *online*, utilizando o código presente no ficheiro "*user_carros.php*", para ler esses mesmos dados. O Script *C-11* indica o código presente nesse ficheiro.

```php
<?php
$con = mysql_connect("localhost","kevenlop_admin","kevenadmin");
if (!$con)
{
  die('Could not connect: ' . mysql_error());
}
mysql_select_db("kevenlop_nada", $con);
$result = mysql_query("SELECT * FROM carro");
while($row = mysql_fetch_assoc($result))
{
        $output[]=$row;
}
print(json_encode($output));
mysql_close($con);
?>
```

Script C-11: Aceder e ler da base de dados.

Figura C-8: Leitura de dados contidos na base de dados.

A Figura *C-8* apresenta as atividades de leitura dos carros que estão presentes na base de dados e os seus respetivos números de identificações.

D. ANEXO – SISTEMA *Gates Technology*

D.1 Clientes

Como já referido anteriormente, nesta parte será especificado a comunicação entre o *hardware* desenvolvido e o carro. Quando é feito o pedido do número de identificação do veículo. Apresentam-se então algumas partes do código utilizado para fazer o mesmo. Para mais informações sobre os sítios da *internet* acedidos, podem ser consultados no Anexo 5.

D.1.1 *Arduino*

Utiliza o *Arduino* para iniciar a comunicação com o Simulador (Conversor Serial – Barramento *CAN*). A comunicação somente é feita quando o *Arduino* comunica via serial com o simulador enviando para o mesmo o carater "1", que vai indicar o pedido do *VIN*. Vamos ter então:

```
#include <SoftwareSerial.h>
// software serial #2: TX = digital pin 8, RX = digital pin 9
SoftwareSerial portTwo(8, 9);
String inputString = "";// a string to hold incoming data
boolean stringComplete = false;// whether the string is complete
String VIN = "";// for incoming serial data
int led_envio = 4;
int led_erro = 7;
int cont;
char inData[25]; // Allocate some space for the string
char inChar; // Where to store the character read
byte index = 0; // Index into array; where to store the character
int led envio2 = 2;
```

Script D-1: Definição das Variáveis.

O *Script D-1* representa a parte inicial do código utilizado para a programação do *Arduino*. Começamos por importar a biblioteca do "*SoftwareSerial.h*" para conseguir utilizar mais do que uma porta de comunicação serial no *Arduino*. Que neste caso está definido para os pinos 8 e 9 sendo Tx e Rx respetivamente. Depois é inicializar as variáveis onde será guardado o *VIN*. Por fim definimos as saídas digitais 2, 4 e 7 como diferentes variáveis.

```
void setup() {
   portTwo.begin(9600);
  //reserve 200 bytes for the inputString:
  inputString.reserve(200);
   // initialize digital pin 7 as an output.
   pinMode(led_envio, OUTPUT);
   // initialize digital pin 4 as an output.
   pinMode(led_erro, OUTPUT);
   // initialize digital pin 2 as an output.
   pinMode(led_envio1, OUTPUT);
  Serial.begin(9600);// opens serial port, sets data rate to 9600 bps
}
```

Script D-2: Inicialização do *Arduino* e configuração.

Agora precisamos inicializar e configurar o *Arduino*. Isto tudo é feito dentro da função "*void setup* ()". Temos que inicializar as portas de comunicação serial para poder ter comunicação e no nosso caso foi especificada uma taxa de transmissão de dados a uma velocidade de 9600 *bps*. Por fim definimos as saídas digitais 2, 4 e 7 do *Arduino* como sendo "*output*" e reservamos 200 *bytes* para a variável que receberá o *VIN* do simulador. Tudo isso como mostra o *Script D-2*, (Wilhite, 2010), (Purdum, 2012).

```
void loop() {
inputString = "";
Serial.write('1'); // send a byte with the value 1
cont=0;
while(Serial.available()==0 && cont<50){
  delay(100);
  cont++;
}
  serialEvent(); //call the function
  // print the string when a newline arrives:
  if (stringComplete) {
    digitalWrite(led_erro, LOW);// turn the LED off
    digitalWrite(led_envio1, HIGH);// turn the LED on
    Serial.println("\nValor do VIN recebido do Carro e :");
    VIN = inputString;
    Serial.println(inputString);
    portTwo.print(inputString);
    // clear the string:
    inputString = "";
    stringComplete = false;
  }
delay(30000);
}
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
} } }
```

Script D-3: Ciclo utilizado pelo *Arduino*.

O Script *D-3* representa o ciclo que é feito constantemente pelo *Arduino*, isto que está tudo dentro da função "*void loop ()*". Começamos por enviar o carater "1" para o simulador para fazer o pedido do número de identificação do veículo. Logo de seguida entramos em um ciclo que vai identificar quando se receber uma resposta do simulador. Vai estar dentro desse ciclo enquanto não receber nada pela porta serial e o contador (definido pela variável "cont" for menor que 50). Tendo um atraso de 100 milissegundos dentro do ciclo então vai estar nesse ciclo por 5 segundos enquanto não receber resposta do simulador. Ao sair do ciclo vai chamar a função "*serialEvent*()" onde vamos ter um ciclo para receber todos os dados que estão a ser enviados pelo simulador. Quando recebermos o *VIN* o programa salta novamente para o ciclo principal e vai acender algumas *LED*, guardar o número de identificação do veículo na variável "*VIN*" e vai enviar esse mesmo valor para o módulo de comunicação *wireless*. Isto é feito quando o

Arduino faz o "*print*" na porta de comunicação serial 2 que foi definida inicialmente. Isto tudo acontece a cada 30 segundos, (Cooper & Boyer, 2001), (Barclay & Gordon, 1994).

D.1.2 Conversor Serial – Barramento *CAN*

O conversor serial para barramento *CAN* é utilizado para comunicar com a rede *CAN* do carro. Neste caso utilizado para fazer o pedido do número de identificação do mesmo. Vamos agora apresentar o *script* utilizado na programação do mesmo. Aonde vamos primeiramente definir as variáveis a utilizar nas comunicações e as configurações necessárias para o bom funcionamento.

```
unsigned char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags;// can flags
unsigned char Rx_Data_Len;// received data length in bytes
char output;
char receive;
char RxTx_Data[8];
char VIN[18];
char reqvin[] = {0x7E0,0x02,0x09,0x02,0x00,0x00,0x00,0x00,0x00};
char floctl[] = {0x7E0,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char i;
char RxTx_Data_last[8];
char RxTx_Data_1[8]; // can rx/tx data buffer
char Msg_Rcvd;// reception flag
const long ID_1st = 0x7E0, ID_2nd = 0x7E8; // node IDs
long Rx_ID;
char UARTx_Data_Ready();
char UARTx_Read();
```

Script D-4: Inicialização das variáveis do programa.

O Script *D-4* indica as variáveis que são utilizadas na programação do simulador. Em que no inicio temos os "*can flags*" que são precisos nas comunicações *CAN*. Importantes aqui também são as variáveis "*reqvin*" e "*floctl*" que são os utilizados no pedido do *VIN*. Em que "*reqvin*" é nesse caso o primeiro *payload* enviado na mensagem quando faz o pedido do *VIN*. O "*floctl*" é utilizado para fazer o pedido dos restantes *bytes* que constituem o número de identificação do veículo, pois, como já referido o *VIN* é constituído por 17 carateres e uma mensagem *CAN* só consegue transportar 8 *bytes* por mensagem. Definimos também os *ID's* que são utilizados em cada mensagem. Por fim temos a inicialização do *UARTx* para fazer a comunicação serial.

```
void main() {
unsigned char counter = 1;
  UART1_Init(9600);
  TRISA = 0;// set direction to be output
  PORTC = 0; // clear PORTC
  TRISC = 0; // set PORTC as output
  Can_Init_Flags = 0;
  Can_Send_Flags = 0;// clear flags
  Can_Rcv_Flags  = 0;
  Can_Send_Flags = _CAN_TX_PRIORITY_0 &
                   _CAN_TX_XTD_FRAME &
                   _CAN_TX_NO_RTR_FRAME;
  Can_Init_Flags = _CAN_CONFIG_SAMPLE_THRICE &
                   _CAN_CONFIG_PHSEG2_PRG_ON &
                   _CAN_CONFIG_XTD_MSG &
                   _CAN_CONFIG_DBL_BUFFER_ON &
                   _CAN_CONFIG_VALID_XTD_MSG;
   CANInitialize(1,3,3,3,1,Can_Init_Flags);
  CANSetOperationMode(_CAN_MODE_CONFIG,0xFF);
  CANSetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_XTD_MSG);
  CANSetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_XTD_MSG);
  CANSetFilter(_CAN_FILTER_B2_F4,ID_2nd,_CAN_CONFIG_XTD_MSG);
  CANSetOperationMode(_CAN_MODE_NORMAL,0xFF);
```

Script D-5: Configuração e Inicialização da rede *CAN*.

Na função "*main* ()" começamos por iniciar a comunicação *UART* a uma taxa de comunicação de 9600 *bps*. Depois se faz a configuração dos "*outputs*" que nesse caso é a Porta C. Avançamos agora limpando as *flags* (atribui-se o valor zero a cada uma delas). Para a variável "*Can_Send_Flags*", esses são os valores que vamos utilizar no *CANWrite*. Já na variável "*Can_Init_Flags"* vamos ter os valores que serão utilizados no *CANInit*. Seguindo agora da mesma ordem apresentando no código vamos ter; a inicialização do *CAN* Módulo, colocar em modo de configuração, colocar em todos os "*mask1*" o *bit* 1, colocar em todos os "*mask2*" o *bit* 1, definir o *ID* do filtro *B2_F4* para o segundo *ID* do *Node* 2 por fim definimos agora para o modo normal.

```
while(1) {
      if (UART1_Data_Ready() == 1) {
       receive = UART1_Read();
       }
         if(receive == '1'){
              if (counter < 7)
                     counter += 1;
              else
                      counter = 0;
for(i=0;i<7;i++){
       RxTx_Data_last[i]=0;
       RxTx_Data_1[i]=0;
}
       RxTx_Data[0] = counter;
CANWrite(ID_1st, reqvin, 4, Can_Send_Flags);
Msg_Rcvd = CANRead(&Rx_ID , RxTx_Data_1 , &Rx_Data_Len, &Can_Rcv_Flags);
       if ((Rx_ID == ID_2nd) && Msg_Rcvd){// if message received check id
              VIN[0] = RxTx_Data_1[0];
              VIN[1] = RxTx_Data_1[1];
              VIN[2] = RxTx_Data_1[2];
              VIN[3] = RxTx_Data_1[3];
              VIN[4] = RxTx_Data_1[4];
              VIN[5] = RxTx_Data_1[5];
              VIN[6] = RxTx_Data_1[6];
              VIN[7] = RxTx_Data_1[7];
CANWrite(ID_1st, floctl, 2, Can_Send_Flags);
Msg_Rcvd = CANRead(&Rx_ID , RxTx_Data_last , &Rx_Data_Len, &Can_Rcv_Flags);
       if ((Rx_ID == ID_2nd) && Msg_Rcvd){// if message received check id
             VIN[9] = RxTx_Data_last[0];
             VIN[10] = RxTx_Data_last[1];
             VIN[11] = RxTx_Data_last[2];
             VIN[12] = RxTx_Data_last[3];
             VIN[13] = RxTx_Data_last[4];
             VIN[14] = RxTx_Data_last[5];
             VIN[15] = RxTx_Data_last[6];
             VIN[16] = RxTx_Data_last[7];
       }
for(i=0;i<17;i++){
       if(i!=8)
             UART1_write(48 + VIN2[i]);
       else
             UART1_write(32);
} } } }  }
```

Script D-6: Comunicação *CAN*, envio e receção de mensagens.

O Script *D-6* representa o ciclo de comunicação utilizado no simulador (conversor serial barramento *CAN*). O simulador vai estar sempre a escuta pela porta serial ate receber algo. Se o dado recebido corresponder ao carater "1" nesse caso passamos agora a comunicação. O "*Master*" vai enviar uma mensagem ao "*Slave*" com um *ID* e um *payload* já especificado. Depois disso vai ficar à espera de uma resposta. Quando chegar essa resposta vai comparar os *ID's*, caso se verifica o mesmo vai guardar os primeiros 8 bytes do *VIN* recebidos no *payload*. Esses *bytes* vão ficar guardados na variável "*VIN*". Depois disso envia uma segunda mensagem com o mesmo *ID* mudando agora o *payload*, para o *floctl* que vai fazer o pedido dos restantes *bytes* que compõe o número de identificação do veículo. Compare novamente os *ID's* e se verificar vai guardar os restantes 8 bytes. Podemos ver que o nono *byte* é dispensado guardando na sua posição o carater espaço. No fim ao ter preenchido todo o *VIN* na variável definida, fazemos um

ciclo para enviar esses dados para o *Arduino* através da porta serial utilizando a função *UART1_write*.

Na figura seguinte (Figura *D-1*) encontra-se representada esta *board*, com indicação das ligações aos portos do micro e pontos de ligação à rede *CAN*.

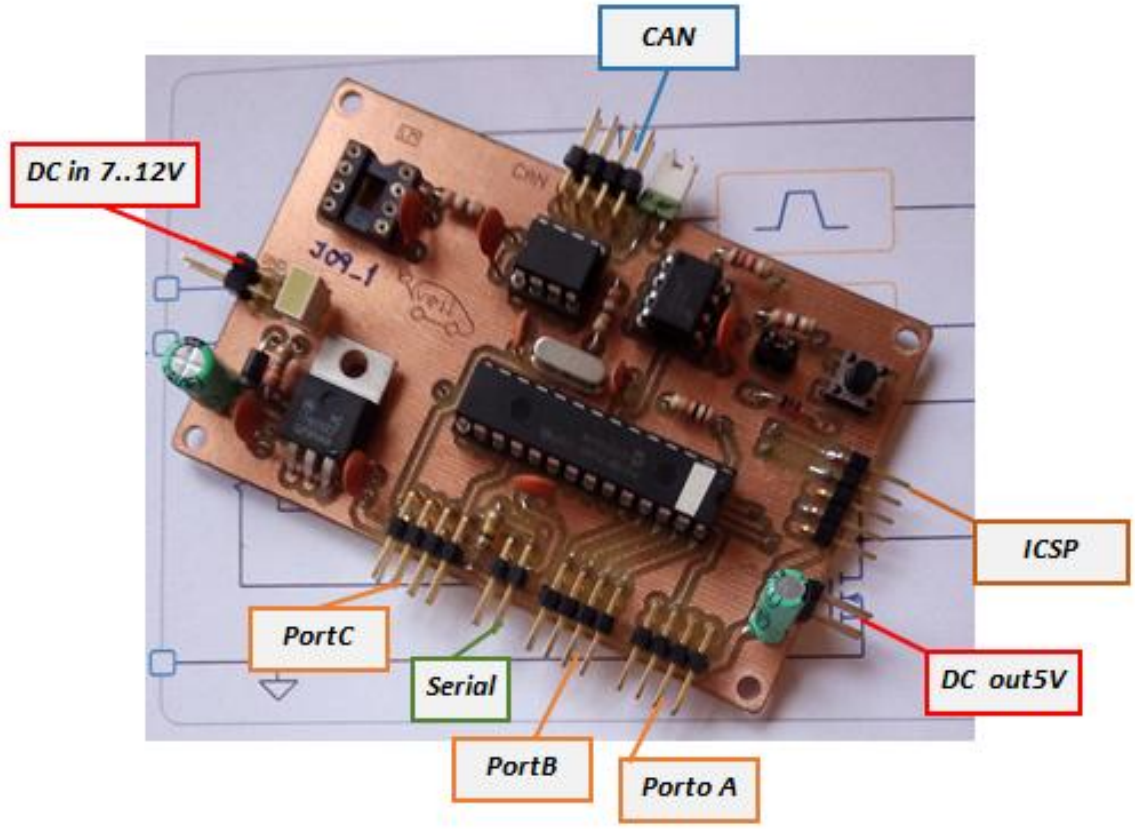


Figura D-1: *Board* SBC2680.

O módulo SBC2680 tem as seguintes características: µcontrolador PIC18F2680 – com interface série e controlador *CAN*; *Transceiver CAN*; Regulador de tensão 5V; Portos A, B. C e Série disponíveis no conector respetivo.

D.1.3 Simulador Resposta

Temos agora o simulador do carro que estamos a fazer a comunicação. Do mesmo modo foi programado para receber e enviar mensagens *CAN*.

```
unsigned char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags; // can flags
unsigned char Rx_Data_Len; // received data length in bytes
char RxTx_Data[8]; // can rx/tx data buffer
char VIN[17] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2, 3, 4, 5, 6, 7};
char reqvin[] = {0x7E0,0x02,0x09,0x02,0x00,0x00,0x00,0x00,0x00};
char floctl[] = {0x7E0,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
char Msg_Rcvd;// reception flag
const long ID_1st = 0x7E0, ID_2nd = 0x7E8;// node IDs
long Rx_ID;
```

Script D-7: Inicialização das variáveis.

Como se pode ver no Script *D-7* começamos por definir as variáveis que utilizaremos como os "*can flags*" e também a variável "*RxTx_Data*" que vai guardar os 8 bytes do *payload* de cada mensagem recebida. Depois se define o *VIN* exemplo a utilizar que está alocada na variável "*VIN*". Temos também os *payload* que estão presentes na rede *CAN* e os diferentes *ID's* que são utilizados nas comunicações.

```
void main() {
  PORTA = 0;
  TRISA = 0;
  Can_Init_Flags = 0; //
  Can_Send_Flags = 0; // clear flags
  Can_Rcv_Flags  = 0; //
  Can_Send_Flags = _CAN_TX_PRIORITY_0 &
                   _CAN_TX_XTD_FRAME &
                   _CAN_TX_NO_RTR_FRAME;
  Can_Init_Flags = _CAN_CONFIG_SAMPLE_THRICE &
                   _CAN_CONFIG_PHSEG2_PRG_ON &
                   _CAN_CONFIG_XTD_MSG &
                   _CAN_CONFIG_DBL_BUFFER_ON &
                   _CAN_CONFIG_VALID_XTD_MSG &
                   _CAN_CONFIG_LINE_FILTER_OFF;
  CANInitialize(1,3,3,3,1,Can_Init_Flags);
  CANSetOperationMode(_CAN_MODE_CONFIG,0xFF);
  CANSetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_XTD_MSG);
  CANSetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_XTD_MSG);
  CANSetFilter(_CAN_FILTER_B2_F3,ID_1st,_CAN_CONFIG_XTD_MSG);
  CANSetOperationMode(_CAN_MODE_NORMAL,0xFF);
  UART1_Init(9600);
```

Script D-8: Configuração e Inicialização da rede *CAN*.

Como já referido para o Script *D-5*, no Script *D-8* vamos ter a configuração e inicialização da rede *CAN*. Isto que é feito na função "*main* ()". Definimos as portas como saídas, limpamos as variáveis que são utilizadas como "*flags*" e atribuímos aos mesmos os respetivos valores de *CANWiret* e de *CANInit*. Depois agora vamos ter o processo de; a inicialização do *CAN* Módulo, colocar em modo de configuração, colocar em todos os "*mask1*" o *bit* 1, colocar em todos os "*mask2*" o *bit* 1, definir o *ID* do filtro *B2_F4* para o segundo *ID* do Node 2 e por último colocamos no modo de funcionamento normal. Por fim iniciamos a porta de comunicação serial, a uma taxa de transmissão e receção de 9600 *bps*.

O simulador vai agora estar em um ciclo infinito onde estará sempre a espera de uma comunicação. Ao receber uma mensagem vai analisar o *ID* e o *payload* recebido pela mensagem. Caso verifica vai enviar nesse caso em específico uma mensagem á quem está a comunicar com um determinado *ID* e com o *payload* de 8 bytes que nesse caso corresponde aos primeiros 8 carateres do *VIN*.

Tudo isto como é representado no Script *D-9*.

```
while (1) {// endless loop

Msg_Rcvd = CANRead(&Rx_ID , RxTx_Data , &Rx_Data_Len, &Can_Rcv_Flags);
if ((RxTx_Data[0] == reqvin[0]) && (RxTx_Data[1] == reqvin[1]) &&
(RxTx_Data[2] == reqvin[2]) && (Rx_ID == ID_1st) && Msg_Rcvd) {

      PORTA = 6;
      RxTx_Data[0] +=1 ;

      CANWrite(ID_2nd, VIN+0, 8, Can_Send_Flags);
}
if ((RxTx_Data[0] == floctl[0]) && (Rx_ID == ID_1st) && Msg_Rcvd) {

      PORTA = 9;
      RxTx_Data[0] +=1 ;

      CANWrite(ID_2nd, VIN+9, 8, Can_Send_Flags);
      PORTA = 15;
```

Script D-9: Ciclo de receção e de envio de mensagens *CAN*.

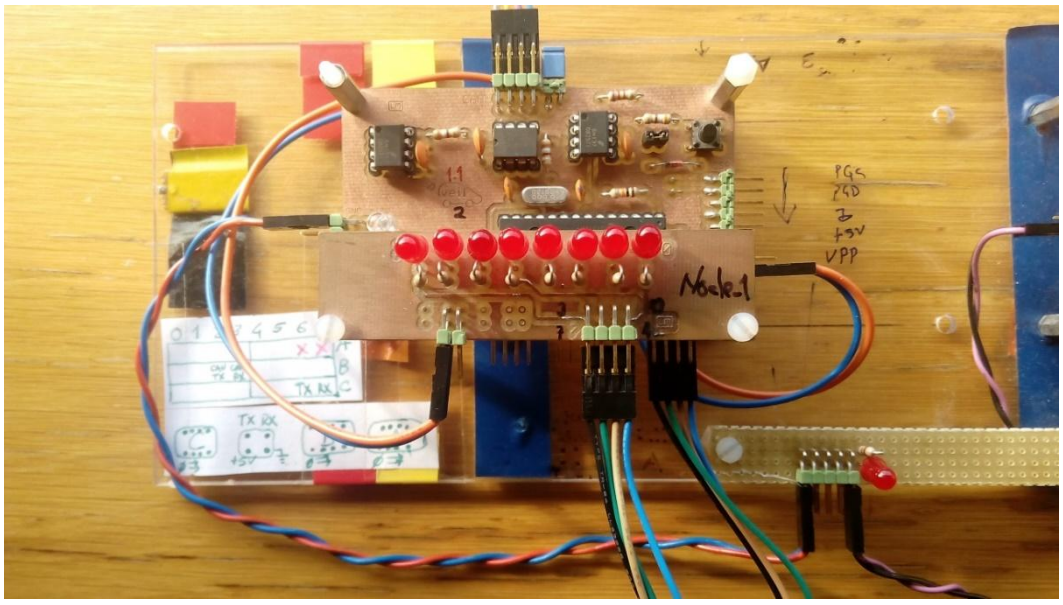


Figura D-2: Setup Experimental, Simulador *CAN*.

D.1.4 Comunicação Arduino – ESP8266

Vamos também ter o *Arduino* em comunicação via uma porta serial com o módulo de comunicação *wireless*, o ESP8266. Como já antes referido o *Arduino* vai estar a cada 30 segundo a enviar os dados recebidos do simulador da rede *CAN* para esse módulo.

```
  if (stringComplete) {
    digitalWrite(led_erro, LOW);
    digitalWrite(led_envio1, HIGH);
    Serial.println("\nValor do VIN recebido do Carro e :");
    VIN=inputString;
    Serial.println(inputString);
    portTwo.print(inputString);
    inputString = ""; stringComplete = false;  }
```

Script D-10: Envio de dados do *Arduino* para o módulo de comunicação *wireless*.

O *Arduino* vai estar sempre a ler os dados que são enviados do simulador, isso termina quando a variável "*stringComplete*" estiver a "*true*". Ao receber todos os dados do simulador, vai acender algumas *LED* e de seguida guarda esse valor na variável "*VIN*". O *Arduino* envia os dados para o módulo através do comando "*portTwo.print*". Em que está definida para uma taxa de comunicação de 9600 *bps* e é feito através das saídas 8 e 9. Isso como é apresentado no Script *D-10*, (Ueno et al, 2003).

```
ledOn = 0
VIN = 0
pin1=3
ledOn1=1
pin=4
aux=0
gpio.mode(pin,gpio.OUTPUT)
gpio.mode(pin1,gpio.OUTPUT)
gpio.write(pin,gpio.LOW)
```

Script D-11: Inicialização das variáveis para o ESP8266.

O Script *D-11* representa a inicialização das variáveis que serão utilizadas na programação do módulo. A programação é dada utilizando a linguagem de programação Lua. Essas variáveis serão utilizadas durante o programa presente no ESP8266. Tendo que o "*pin*" e o "*pin1*" estão configuradas como as duas saídas do módulo, o GPIO 0 e o GPIO 2. Depois disso vamos definir os mesmos como o modo de "*Output*". Começamos logo por colocar 0 volt a saída do "*pin*".

```
uart.on("data", 19, function(data)
print("Dados Recebidos do Arduino com Sucesso!!")
      if aux = 0 then
      aux=1
      print("A Atualizar os dados Recebidos do Arduino.\n")
            if ledOn==0 then
                  ledOn = 1
                  gpio.write(pin,gpio.HIGH)
                  gpio.write(pin1,gpio.LOW)
                  else
                  ledOn = 0
                  gpio.write(pin,gpio.LOW)
                  gpio.write(pin1,gpio.LOW)
            end
      else
      VIN = data
      aux = 1
      print("Dados atualizados!!!\n")
      print("O VIN do Carro e : ", VIN")
      data=0
            if ledOn1==0 then
                  ledOn1 = 1
                  gpio.write(pin1,gpio.HIGH)
                  gpio.write(pin,gpio.LOW)
                  else
                  ledOn1 = 0
                  gpio.write(pin1,gpio.LOW)
                  gpio.write(pin,gpio.LOW)
            end
      end
end, 0)
```

Script D-12: Receção dos dados pelo módulo enviado do *Arduino*.

Para o Script *D-12* vamos ter agora a função "*uart.on*" que indica o evento de recebimento de dados via serial. Nesta função define-se o número máximo de carateres que serão recebidos via serial, onde nesse caso o máximo serão 19 carateres e vai guardar esses dados na variável "*data*". Depois vai verificar se é a primeira vez que se comunica isto porque no início pode ocorrer em que o *buffer* ter já "*lixo*" guardado na memória então serão descartados esses dados. Seguidamente vai ser guardado o número de identificação do veículo recebido pelo *Arduino* na variável "*VIN*". Agora só vamos colocar 5 voltes ou 0 volte nas saídas do módulo utilizadas para acender ou apagar diferentes *LED's*, (Schuytema & Manyen, 2005), sítios relevantes consultados em Anexo 5[13].

D.1.5 Comunicação Cliente - Servidor

Como já anteriormente referido, um cliente vai estar sempre a fazer leitura de redes disponíveis ao redor. Quando esse estiver ao alcance da rede disponibilizado pela garagem (servidor) vai tentar comunicar com o mesmo.

[13] ANEXO 5 – Sítios da Internet Relevantes

```
led1 = 3
led2 = 4
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
gpio.write(led1, gpio.LOW);
gpio.write(led2, gpio.LOW);

print("ESP8266 Carro Cliente")
wifi.sta.disconnect()
wifi.setmode(wifi.STATION)

--Tentando ligar com o Gates Technology
wifi.sta.config("gatestechnology","12345678", "18:FE:34:9D:F3:D9")
wifi.sta.connect()
print("A conectar com a garagem . . .")
```

Script D-13: Inicialização e configuração do módulo.

No Script *D-13* vamos ter a configuração dos pinos 3 e 4 que representa o GPIO 0 e GPIO 2, colocando os mesmos no modo de "output". Através da função "*gpio.write*" vai colocar 0 volte a cada uma dessas saídas. Depois disso vamos configurar o ESP8266 como sendo um *Station*. Neste caso vamos ter o módulo a tentar ligar á um *Access Point*. Sabendo que o servidor está configurado para emitir uma rede de *SSID* (nome de uma rede sem fio) "*gatestechnology*" e com a palavra passe "*12345678*", então se configura o cliente com esses dados para conseguir ligar ao mesmo. O "*18: FE: 34:9D: F3: D9*" representa o *MAC Adress* do *Access Point* que se pretende ligar, isto é, mesmo que encontre uma rede com o mesmo nome, mas saberá realmente a qual se comunicar. A função "*wifi.sta.connect()*" vai fazer com que o módulo se tenta comunicar com o servidor.

```
tmr.alarm(1, 5000, 1, function()
     if(wifi.sta.getip()~=nil) then
          tmr.stop(1)
          print("Conectado com a Garagem!!")
          print("Client IP Address:",wifi.sta.getip())
          cl=net.createConnection(net.TCP, 0)
          cl:connect(80,"192.168.4.1")
          tmr.alarm(2, 7000, 1, function()
            cl:send("VIN")
                     gpio.write(led1 ,gpio.HIGH)
                     gpio.write(led2 ,gpio.LOW)
                     print("VIN Enviado para o sevidor")
          end)
      else
         print("Conectando ...")
               gpio.write(led2 ,gpio.HIGH)
               gpio.write(led1 ,gpio.LOW)
      end
end)
```

Script D-14: Ciclo de comunicação com o servidor.

No Script *D-14* vamos ter diferentes funções a serem utilizadas pelo módulo (Cliente). Primeiramente defina-se um timer que a cada 5 segundos vai-se tentar comunicar com o servidor. Se não for possível efetuar essa comunicação acende-se uma *LED* vermelha indicando uma mensagem de erro. No caso de conseguir comunicar com o servidor, vamos iniciar uma comunicação "*net.TCP*". Essa comunicação efetua-se na porta 80 e o cliente vai se conectar ao servidor através do *IP* do servidor que nesse caso é

"*192.168.4.1*". Depois disso utilizando um timer, a cada 7 segundos vai enviar então o *VIN* para o servidor através da função "*cl:send("VIN")*", acendendo uma *LED* que indica o sucesso na comunicação.

D.2 Servidor

A programação do servidor é feita através de Lua, vamos então agora identificar diferentes partes do *script* que constitui o programa a utilizar. Começamos com o Script *D-15* que representa a inicialização das variáveis, definindo então as variáveis led1 e led2 como sendo os pinos de saída do ESP8266. Esses que são o GPIO 0 e GPIO 2. Posteriormente e tendo diferentes dados a utilizar como número de identificação do veículo para efetuar as comparações, essas que são; *VIN1*, *VIN2*, *VIN3* e *VIN4*. Através da função "*gpio.mode*" vamos definir como sendo saídas os pinos 3 e 4, e através da função "*gpio.write*" iniciamos os mesmos em "*LOW*", ou seja, vai colocar 0 volt nos pinos. Depois, vamos configurar o ESP8266 para funcionar como "*STATIONAP*" que é uma combinação entre um *SOFTAP* e uma *Station* (funciona como um dispositivo terminal para conectar a um ponto de acesso). Quer dizer que para além do módulo estar a funcionar como um *Access Point* disponibilizando uma rede *wireless*, pode também ligar a outro *Access Point* no caso de pretender comunicar com outra garagem. Agora através da função "*wifi.ap.config*" vamos indicar o nome e a palavra passe da rede que o servidor vai estar a fazer *host*. Por fim utilizando as funções "*wifi.ap.getip()"* e "*wifi.ap.getmac()*", vamos conseguir visualizar o endereço de *IP* do servidor e o seu respetivo *Mac Adress*.

```
led1 = 3
led2 = 4
VIN1 = "ghhy4575424gjhj"
VIN2 = "ghjhgjykkmb4244"
VIN3 = "8989yjhjh1425"
VIN4 = "31561sdsds"

gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
gpio.write(led1, gpio.LOW);
gpio.write(led2, gpio.LOW);
print("ESP8266 Server")
wifi.setmode(wifi.STATIONAP);
wifi.ap.config({ssid="gatestechnology",pwd="12345678"});
print("Server IP Address:",wifi.ap.getip())
 -- print current mac address
print("Server MAC Address:",wifi.sta.getmac())
```

Script D-15: Inicialização das variava do servidor.

```
sv = net.createServer(net.TCP)
sv:listen(80, function(conn)
conn:on("receive", function(conn, receivedData)
print("A receber dados do Carro . . . ")
if receivedData == VIN1 or receivedData == VIN2 or receivedData == VIN3 or
receivedData == VIN4 then
       print("Identificacao correcta ->"  .. receivedData)
       print("Abrindo a Porta . . .")
       tmr.delay(200000)
              gpio.write(led1, gpio.HIGH);
              gpio.write(led2, gpio.LOW);
else
       print("Numero de identificacao do carro nao encontrado!!")
       tmr.delay(200000)
              gpio.write(led2, gpio.HIGH);
              gpio.write(led1, gpio.LOW);
end
end)
conn:on("sent", function(conn)
collectgarbage()
    end)
end)
```

Script D-16: Ciclo de comunicação do Servidor.

Script *D-16*, começamos por iniciar uma comunicação *TCP*, onde o servidor vai estar a "ouvir" pela porta 80. Ao verificar que um cliente estabeleceu a comunicação o servidor vai receber os dados que estão a ser enviados pelo cliente, que nesse caso vão ficar guardados na variável já antes definida "*receivedData*". Depois disso que já tiver recebido todos os dados, o servidor vai fazer a comparação com os diferentes números de identificação dos veículos presentes na base de dados. Caso a verificação for correta o servidor vai acender uma luz verde de confirmação que o cliente pode aceder à garagem. Ou se não acende uma luz vermelha e envia uma mensagem de erro.

```
-- load driver
local driver = require "luasql.mysql"
-- create environment object
env = assert (driver.mysql())
-- connect to data source
con = assert
(env:connect("garagem_database","administrador","admin","localhost")
)
```

Script D-17: Aceder a base de dados *online*.

O Script *D-17* indica o código utilizado para aceder a base de dados do *Gates Technology* que está alocada na web (*online*). Onde vamos importar a biblioteca "*luasql.mysql*", depois cria-se um ambiente de objeto e por último conectamos a base de dados, enviando como parâmetro na função o nome da tabela, o nome do utilizador e a palavra passe.

```
-- retrieve a cursor
cur = assert (con:execute"SELECT VIN from carro")
-- print all rows, the rows will be indexed by field names
row = cur:fetch ({}, "a")
while row do

print(string.format("VIN Recebido da base de dados e -> %s",
row.VIN))
  tipo=row.VIN

if Line == tipo then
      print("valores correspondem . . . certo")
            break
else
      print("valores nao presente na base de dados . . . ERRO")
end
  -- reusing the table of results
  row = cur:fetch (row, "a")
  print("-----------------------------------------")
  --a = io.read()

end
-- close everything
cur:close() -- already closed because all the result set was
consumed
con:close()
env:close()
```

Script D-18: Leitura e comparação com os dados presentes na base de dados.

O Script *D-18* conte a configuração necessária para fazer a leitura da tabela presente na base de dados do *Gates Technology* onde estão presentes os diferentes números de identificações dos carros dos clientes. A leitura é feita utilizando a função "*assert*", que vai selecionar os dados contidos na tabela "Carro".

D.3 Comunicação *Android* - Servidor

```
void setup()
{
Serial.begin(9600);
esp8266.begin(9600);
pinMode(11,OUTPUT);
digitalWrite(11,LOW);
pinMode(12,OUTPUT);
digitalWrite(12,LOW);
  sendCommand("AT+RST\r\n",2000,DEBUG); // reset module
  sendCommand("AT+CWMODE=1\r\n",1000,DEBUG); // configure as access point
  sendCommand("AT+CWJAP=\"SSID\",\"PASSWORD\"\r\n",3000,DEBUG);
  delay(10000);
  sendCommand("AT+CIFSR\r\n",1000,DEBUG);// get ip address
  sendCommand("AT+CIPMUX=1\r\n",1000,DEBUG);//configure for multiple
connections
  sendCommand("AT+CIPSERVER=1,80\r\n",1000,DEBUG);//turn on server port 80
  Serial.println("Server Ready");
}
```

Script D-19: Inicialização e configuração do *Arduino*.

Como já antes referido, vamos ter duas comunicações por serial do *Arduino*, um com o simulador e outro com o módulo. Inicia os dois a uma taxa de comunicação de 9600 *bps*. Depois se define as saídas e os respetivos valores (nesse caso coloca 0 *volt* em todas os pinos de saída). De seguida são enviados os Comandos AT para o módulo, como indica nos comentários, primeiramente reiniciamos, configura-se o módulo como um *Access Point*, liga ao ponto de comunicação *wireless*, depois vamos ler o *IP* e ligar o servidor na porta 80, (Göransson & Cuartielles Ruiz, 2013).

```
void sendHTTPResponse(int connectionId, String content)
{
// build HTTP response
     String httpResponse;
     String httpHeader;
// HTTP Header
httpHeader = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=UTF-
8\r\n";
     httpHeader += "Content-Length: ";
     httpHeader += content.length();
     httpHeader += "\r\n";
     httpHeader +="Connection: close\r\n\r\n";
     httpResponse = httpHeader + content + "
sendCIPData(connectionId,httpResponse);}
```

Script D-20: Função que define o *HTTP* 200 e o *HTML UTF-8*.

Função utilizada no *Arduino* para as comunicações *HTTP* e também juntamente com o *HTML* onde vamos definir a comunicação *HTTP* 200 e o tipo de leitura de *HTML* em *UTF-8*.

```
String sendData(String command, const int timeout, boolean debug)
{
String response = "";
int dataSize = command.length();
char data[dataSize];
command.toCharArray(data,dataSize);
esp8266.write(data,dataSize);//send the read character to the esp8266
if(debug)
{
Serial.println("\r\n====== HTTP Response From Arduino ======");
Serial.write(data,dataSize);
Serial.println("\r\n========================================");
}
  long int time = millis();
while( (time+timeout) > millis())
    {
      while(esp8266.available())
      {
// The esp has data so display its output to the serial window
        char c = esp8266.read(); // read the next character.
        response+=c;
      }
    }
if(debug)
   { Serial.print(response);  }
return response;}
```

Script D-21: Função utilizada para enviar dados para o ESP8266.

O Script *D-21* representa a função que envia dados para o ESP8266 que por sua vez envia a aplicação. Depois de estabelecer a comunicação e realizar os comandos a pedido da aplicação *Android*, o *Arduino* vai enviar a resposta para a aplicação. Utiliza então essa função para enviar a resposta *HTTP*, (Roberto et al, 2013), (Riley, 2012).

```
String sendCommand(String command, const int timeout, boolean debug)
{ String response = "";
esp8266.print(command); // send the read character to the esp8266
long int time = millis();
while( (time+timeout) > millis())
  {
      while(esp8266.available())
      {
      char c = esp8266.read(); // read the next character.
        response+=c;
  }    }
if(debug)
  {
     Serial.print(response);
   }
return response;
}
```

Script D-22: Função que envia os Comandos AT para o ESP8266.

O Script *D-22* é a função utilizada para enviar os diferentes Comandos AT para o módulo de comunicação *wireless*, para configurar o ESP8266.

E. ANEXO 5 – Sítios da *Internet* Relevantes

Sítios da *internet* consultados no período de junho á dezembro 2015:

[w1] http://www.ismartgate.com/ (iSmart Gate, página *internet* oficial)

[w2] http://www.gogogate.com/ (GogoGate, página *internet* oficial)

[w3] http://lutel.woese.com/page/7113 (Lutel, página *internet* oficial)

[w4] http://stock-off.com/produto/54-comando-universal-port%C3%B5esgaragens (Comando universal)

[w5] http://www.skoda-auto.com/en/models/new-octavia/technology (Skoda, página *internet* oficial)

[w6] http://elmelectronics.com/obdic.html#ELM327 (ELM Electronics, página *internet* oficial)

[w7] http://en.wikipedia.org/wiki/On-board_diagnostics (*Wikipedia*, página *internet* oficial l)

[w8] http://www.elmelectronics.com/obdsoftware.html#Windows(ELM Eletronics, página *internet* oficial)

[w9] http://en.wikipedia.org/wiki/CAN_bus (*Wikipedia*, página *internet* oficial)

[w10] http://en.wikipedia.org/wiki/OBD-II_PIDs (*Wikipedia*, página *internet* oficial)

[w11] http://www.obdpros.com/ (OBDPROS, página *internet* oficial)

[w12] https://developer.mbed.org/forum/mbed/topic/1786/?page=1#comment-9387 (*CAN Bus flow Control*)

[w13] http://www.drewtech.com/products/cardaqplus.html (Drewtech, página *internet* oficial)

[w14] http://www.drewtech.com/products/mongoose.html (Drewtech, página *internet* oficial)

[w15] http://en.wikipedia.org/wiki/ISO_15765-2 (*Wikipedia*, página *internet* oficial)

[w16] http://www.pctoledo.com.br/forum/viewtopic.php?f=39&t=4328&start=750 (Clipper On Line, página *internet* oficial)

[w17] http://www.drewtech.com/technician/ (Drewtech, página *internet* oficial)

[w18] http://www.drewtech.com/downloads/ (Drewtech, página *internet* oficial)

[w19]http://www.autocheck.com/consumers/car-facts/vin-numbers/find-vin.do (*Vehicle Identification Number*, car-facts)

[w20] http://www.dmv.org/vehicle-history/find-vin.php (DMV, página *internet* oficial)

[w21] http://en.wikipedia.org/wiki/Wired_Equivalent_Privacy (*Wikipedia*, página *internet* oficial)

[w22] http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access (*Wikipedia*, página *internet* oficial)

[w23] http://www.bluetooth.com/Pages/Basics.aspx (*Bluetooth Basics*)

[w24] http://en.wikipedia.org/wiki/Bluetooth (*Wikipedia*, página *internet* oficial)

[w25] http://pt.wikipedia.org/wiki/Arduino (*Wikipedia*, página *internet* oficial)

[w26] http://www.Arduino.cc/ (*Arduino*, página *internet* oficial)

[w27] https://infrastructure.gov.au/vehicles/imports/vins.aspx (DIRD, página *internet* oficial)

[w28]https://groups.google.com/forum/#!topic/openxc/FrZA8-pDs6A(Google Groups)

[w29] http://www.microchip.com/pagehandler/en-us/devtools/dev-tools-parts.html (MPLAB *Source*)

[w30] http://mplab-ide.software.informer.com/8.9/ (MPLAB, página *internet* oficial)

[w31] http://randomnerdtutorials.com/ (Randomnerdtutorials, página *internet* oficial)

[w32] http://zeflo.com/2014/esp8266-weather-display/ (ESP8266)

[w33] https://ricardo2aoc.wordpress.com/chave-primaria/ (Chave Primária)

[w34] https://pt.wikipedia.org/wiki/Chave_estrangeira (*Wikipedia*, página *internet* oficial)

[w35] https://web.fe.up.pt/~ee99058/projecto/pdf/Can.pdf (Faculdade de Engenharia, Universidade do Porto, página *internet* oficial)

[w36] http://microcontrollerkits.blogspot.pt/2015/05/esp8266-firmware-update-with-Arduino.html (Atualização do *firmware* do ESP8266 com *Arduino*)

[w37]http://www.electrodragon.com/w/ESP8266_NodeMCU_Dev_Board (Electrodragon página *internet* oficial)

[w38] http://benlo.com/esp8266/esp8266QuickStart.html (Módulo ESP8266)

[w39] http://nodemcu.com/index_en.html (NodeMCU, página *internet* oficial)

[w40] https://github.com/nodemcu/nodemcu-firmware (github, página *internet* oficial)

[w41] https://github.com/hwiguna/g33k/tree/master/ArduinoProjects (github, página *internet* oficial)

[w42] http://www.tutorialspoint.com/lua/lua_database_access.htm (tutorialspoint, página *internet* oficial)

[w43] https://coronalabs.com/blog/2012/04/03/tutorial-database-access-in-corona/ (Lua)

[w44] http://www.esp8266.com/viewtopic.php?f=24&t=1283 (ESP8266, página *internet* oficial)

[w45]https://github.com/hwiguna/g33k/tree/master/ArduinoProjects/Windows/ESP8266_Related (github, página *internet* oficial)

[w46]http://umoya-cottages.co.za/self-catering-guesthouse-automation-Arduino/ (*Arduino*)

[w47] http://fab.cba.mit.edu/classes/865.15/people/dan.chen/esp8266/ (Dan Chen, página *internet* oficial)

[w48] http://fab.cba.mit.edu/classes/863.14/tutorials/Programming/serialwifi.html (Dan Chen, página *internet* oficial)

[w49]http://williamdurand.fr/2015/03/17/playing-with-a-esp8266-wifi-module/ (ESP8266)

[w50]http://www.instructables.com/id/Using-the-ESP8266-module/?ALLSTEPS (Instructables, página *internet* oficial)

[w51] http://www.seeedstudio.com/wiki/WiFi_Serial_Transceiver_Module (seedwiki, página *internet* oficial)

[w52] http://fritzing.org/home/ (Fritzing, página *internet* oficial)

[w53] https://www.joomla.org/ (Joomla, página *internet* oficial)

[w54] http://www.w3schools.com/ (w3schools, página *internet* oficial)

[w55] http://www.bisecur-home.com/en/ (BiSecur, página *internet* oficial)

[w56] http://www.marktest.com/wap/a/n/id~1bfc.aspx (Marktest, página *internet* oficial)

[w57] http://www.marktest.com/wap/a/n/id~1952.aspx (Marktest, página *internet* oficial)

[w58]http://www.anacom-consumidor.com/em-destaque/estatisticas-sobre-os-servicos-de-comunicacoes-eletronicas.html (Anacom, página *internet* oficial)

[w59] https://www.youtube.com/ (Youtube, página *internet* oficial)

[w60]http://skpang.co.uk/catalog/Arduino-canbus-shield-with-usd-card-holder-p-706.html (*Arduino*, *CAN Bus Shield*)

[w61] https://code.google.com/p/webiopi/wiki/PASSWORD (Raspberry Pi)

[w62]https://learn.adafruit.com/adafruits-raspberry-pi-lesson-2-first-time-configuration (Adafruit, página *internet* oficial - Raspberry Pi)

[w63] https://learn.adafruit.com/adafruits-raspberry-pi-lesson-6-using-ssh (Adafruit, página *internet* oficial - Raspberry Pi)

[w64] http://allaboutee.com/2015/01/20/esp8266-android-application-for-Arduino-pin-control/ (Aplicações *Android* e *Arduino*)

[w65] https://eclipse.org/ (*Eclipse*, página *internet* oficial)

[w66]http://www.eletrica.ufpr.br/~james/Laboratorio%20V/arquivos/Mini%20Curso%20Arduino.pdf (*Arduino*)

[w67] http://developer.android.com/sdk/index.html (*Android Studio*, página *internet* oficial)

[w68] http://www.adobe.com/pt/products/catalog.html (Adobe, página *internet* oficial)

[w69] https://dev.mysql.com/downloads/workbench/ (MySQL Workbench, página *internet* oficial)

[w70] https://notepad-plus-plus.org/download/v6.8.8.html (Notepad++, página *internet* oficial)

[w71] http://www.sublimetext.com/3 (Sublimetext 3, página *internet* oficial)

[w72] http://www.lua.org/download.html (Lua, página *internet* oficial)

[w73] http://stackoverflow.com/ (StackOverflow, página *internet* oficial)

[w74] http://www.ebay.com/ (Ebay, página *internet* oficial)

[w75] http://www.amazon.com/ (Amazon, página *internet* oficial)

[w76] http://www.hpstecnologia.com.br/ (HPS Tecnologia, página *internet* oficial)

[w77] http://www.statista.com (Estatísticas Diversas).

[w78] http://www.dmv.org/vehicle-history/find-vin.php (Onde encontrar o NIV).

[w79] http://oster-lundqvist.com/karsten/?p=4898 (Ciclo de vida de uma Aplicação)

[W80] https://pt.wikipedia.org/wiki/Notepad%2B%2B (Notepad++)