

IPBeja
INSTITUTO POLITÉCNICO
DE BEJA

Escola Superior de Tecnologia e Gestão
Mestrado em Internet das Coisas

Relatório de Projecto de Web das Coisas

Web das Coisas

João Manuel Rafael Pica, Vasco Miguel Guerreiro Gomes

Beja, 9 de Julho de 2024

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Internet das Coisas

Relatório de Projecto de Web das Coisas

Web das Coisas

João Manuel Rafael Pica, Vasco Miguel Guerreiro Gomes

Orientado por :

Doutor Luís Carlos Bruno, IPBeja

Relatório de Projeto

Índice

Índice	i
Índice de Figuras	v
Índice de Listagens	vii
1 Introdução	1
1.1 Enquadramento	1
1.2 Objetivos	1
1.3 Organização do relatório	2
2 Sistemas Semelhantes	3
2.1 Parkopedia - Análise Visual	4
2.1.1 Utilizadores	4
2.1.2 Contextos de Utilização	4
2.1.3 Tarefas Executadas	4
2.1.4 Arquitetura e Tecnologias	5
2.1.5 Interface	5
2.1.6 Aspetos Positivos e Negativos	5
2.2 ParkWhiz - Análise Visual	6
2.2.1 Utilizadores	6
2.2.2 Contextos de Utilização	7
2.2.3 Tarefas Executadas	7
2.2.4 Arquitetura e Tecnologias	7
2.2.5 Aspetos Positivos e Negativos	8
2.3 <i>Low Cost Smart Parking System with IoT</i> - Análise Física	9
2.3.1 Arquitetura e Tecnologias	9
2.3.2 Funcionamento Geral	9
2.3.3 Descrição da Interface	10
2.3.4 Aspetos Positivos e Negativos	10

2.4	<i>IoT Based Cost Effective Car Parking Management for Urban Area - Análise Física</i>	11
2.4.1	Arquitetura e Tecnologias	11
2.4.2	Funcionamento Geral	12
2.4.3	Descrição da Interface	12
2.4.4	Aspetos Positivos e Negativos	12
3	Análise do Sistema	13
3.1	Casos de Uso	13
3.1.1	Procurar Vagas de Estacionamento	13
3.1.2	Registar-se na Plataforma	14
3.1.3	Reservar Lugar para Necessidades Especiais	14
3.1.4	Abrir Cancela do Lugar Reservado	15
3.1.5	Analisar a plataforma	15
3.2	Requisitos Funcionais	15
3.2.1	Autenticação e Registo	15
3.2.2	Pesquisa e Reserva de Lugares	16
3.2.3	Atualização em Tempo Real	16
3.2.4	Controle de Acesso	16
3.3	Requisitos Não Funcionais	16
3.3.1	Desempenho	16
3.3.2	Segurança	16
3.3.3	Usabilidade e Compatibilidade	16
3.3.4	Escalabilidade	16
3.3.5	Confiabilidade	17
4	Desenho do Sistema	19
4.1	Desenho da Interface Gráfica	20
4.1.1	Interface de Registo do Utilizador	20
4.1.2	Interface de Autenticação do Utilizador	21
4.1.3	Interface do Utilizador Autenticado	22
4.1.4	Interface do Utilizador Autenticado - Meus lugares reservados	23
4.2	Desenho da Arquitetura da Base de Dados	24
4.2.1	Login	24
4.2.2	UserToken	25
4.2.3	User	25
4.2.4	Park	25
4.2.5	Spot	26
4.2.6	Vehicle	26
4.2.7	Reserve	26

4.3	Desenho da Arquitetura Física	27
4.3.1	Utilizador	27
4.3.2	Interface <i>Web</i>	28
4.3.3	<i>Framework de APIs</i>	28
4.3.4	Base de Dados	28
4.3.5	<i>Servient</i>	28
4.3.6	<i>Broker MQTT</i> e <i>Scripts</i> Adjacentes	28
4.3.7	Micro-controladores	28
4.3.8	Sensores	28
5	Implementação do Sistema	29
5.1	Grupos e tecnologias para tratamento de dados	30
5.1.1	Utilizador	30
5.1.2	Câmara	30
5.1.3	Servidor 1 - <i>OnRender</i>	30
5.1.4	Cloud	30
5.1.5	Servidor 2 - Raspberry Pi 3B	30
5.1.6	Microcontroladores	31
5.2	Arquitetura do <i>W3C</i> no sistema <i>ParkLookUp</i>	31
5.2.1	Descrição de Coisas (<i>Thing Description</i>) & Modelos de Ligação (<i>Binding Templates</i>)	32
5.2.2	API de Scripting (<i>Scripting API</i>)	35
5.2.3	Diretrizes de Segurança e Privacidade (<i>Security and Privacy Guidelines</i>)	39
6	Testes ao Sistema	41
7	Conclusão	43
	Bibliografia	45

Índice de Figuras

2.1	Interfaces Gráficas - Parkopedia Parking	4
2.2	Interfaces Gráficas - Parkwhiz	6
2.3	Arquitetura Proposta Artigo 1	9
2.4	Arquitetura Proposta Artigo 2	11
4.1	Interface de Registo do Utilizador	20
4.2	Interface de Autenticação do Utilizador	21
4.3	Interface do Utilizador Autenticado	22
4.4	Interface do Utilizador Autenticado - Meus lugares reservados	23
4.5	Desenho da Arquitetura da Base de Dados	24
4.6	Desenho da Arquitetura Física - Abstrato	27
5.1	Desenho da Arquitetura Implementada no Sistema	29

Índice de Listagens

5.1	Descrição de Coisas (<i>Thing Description</i>)	33
5.2	API de Scripting (<i>Scripting API</i>) - <i>Producer</i>	36
5.3	API de Scripting (<i>Scripting API</i>) - <i>Client 1</i>	37
5.4	API de Scripting (<i>Scripting API</i>) - <i>Client 2</i>	38

Capítulo 1

Introdução

1.1 Enquadramento

No cenário atual, a crescente urbanização e a alta densidade populacional têm resultado em um aumento significativo no tráfego urbano e na dificuldade de encontrar vagas de estacionamento disponíveis. Para enfrentar esse desafio, várias soluções têm sido desenvolvidas para facilitar a busca por lugares de estacionamento, com o objetivo de melhorar a eficiência e a experiência dos utilizadores.

Uma dessas soluções é o sistema *ParkLookUp*. O seu objetivo principal é simplificar a procura por vagas em estacionamentos utilizando mapas que destacam a disponibilidade em tempo real. Além disso, o sistema permite que utilizadores com necessidades especiais, como deficiências ou demandas de carga e descarga, se registem na plataforma e reservem vagas adaptadas às suas necessidades.

1.2 Objetivos

Assim, o sistema pretende solucionar os seguintes problemas:

- **Disponibilidade de estacionamento:** A dificuldade para encontrar vagas de estacionamento disponíveis é um problema frequente em áreas urbanas densamente povoadas. **O sistema oferece informações atualizadas em tempo real sobre a disponibilidade de vagas.**
- **Acessibilidade para utilizadores com necessidades especiais:** Utilizadores com deficiências ou outras necessidades especiais frequentemente enfrentam desafios adicionais ao encontrar um estacionamento adequado. **O sistema oferece recursos específicos para assegurar que esses utilizadores tenham uma experiência de estacionamento acessível e sem complicações.**

Ao tratar desses problemas, o sistema *ParkLookUp* pretende aprimorar a experiência de estacionamento, tornando-a mais conveniente, acessível e eficiente para todos os utilizadores, além de beneficiar o meio ambiente.

1.3 Organização do relatório

Este relatório encontra-se estruturado em sete capítulos. Cada capítulo aborda aspectos específicos relacionados à elaboração do sistema. No seguimento, apresentamos uma breve descrição de cada capítulo:

1. **Sistemas Semelhantes:** Apresentação e análise de sistemas semelhantes ao *ParkLookUp*;
2. **Análise do Sistema:** Análise do sistema explorando casos de uso dos utilizadores e requisitos funcionais e não funcionais;
3. **Desenho do Sistema:** Desenvolvimento do desenho do sistema, apresentando as propostas das interfaces *web* e arquiteturas para os sistemas físicos;
4. **Implementação do Sistema:** Explicitar todas as tecnologias e abordagens seguidas para implementar o projeto;
5. **Conclusão:** Conclusão dos temas discutidos e aperfeiçoados ao longo do desenvolvimento do projeto;
6. **Bibliografia:** Listagem de todas as referências utilizadas na elaboração do relatório.

Capítulo 2

Sistemas Semelhantes

Para entender melhor o tema, foi realizada uma busca por sistemas semelhantes ao proposto neste projeto. Essa busca não permitiu apenas encontrar abordagens já exploradas por outros sistemas, mas também identificar problemas enfrentados.

Deste modo, foram analisados sistemas semelhantes em duas vertentes:

- **Visual** - Aqui pretende-se analisar os sistemas do ponto de vista de interfaces *Web*. Os sistemas aqui analisados denomina-se de *Parkopedia* [Par23a] e *ParkWhiz* [Par23d].
- **Física** - Os sensores, os microcontroladores, os sistemas de comunicação, entre outros, foram o foco nesta análise. Os sistemas aqui analisados denomina-se de *Low Cost Smart Parking System with IoT* [C S20] e *IoT Based Cost Effective Car Parking Management for Urban Area* [Pey+21].

Toda a análise foi elaborada com base na opinião de ambos os alunos, podendo alguns dos pontos não coincidirem com a análise de outros pesquisadores.

2.1 Parkopedia - Análise Visual

O sistema *Parkopedia* tem como objetivo ajudar os utilizadores a encontrar lugares de estacionamento próximos do seu destino. Isso permite economizar tempo e combustível, reduzindo também as emissões de CO2 por veículo. [Par23a]

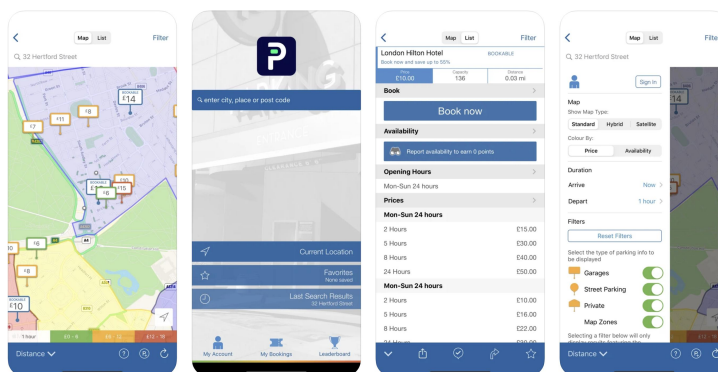


Figura 2.1: Interfaces Gráficas - Parkopedia Parking

2.1.1 Utilizadores

Os utilizadores do *Parkopedia* estão espalhados por mais de 90 países e buscam lugares de estacionamento para realizar suas tarefas de forma eficiente. [Par23b]

2.1.2 Contextos de Utilização

Os principais contextos de utilização da aplicação incluem:

- Deslocações diárias para o trabalho.
- Compras e lazer.
- Viagens para locais desconhecidos.

2.1.3 Tarefas Executadas

Os utilizadores podem realizar várias tarefas através da aplicação [Par23a]:

- **Encontrar Estacionamento:** Localizar lugares de estacionamento próximos ao destino.
- **Pré-Reserva do Lugar:** Reservar o lugar de estacionamento online antecipadamente.
- **Consultar Status da Reserva:** Verificar o status da reserva feita.
- **Realizar Pagamento da Reserva:** Efetuar o pagamento do estacionamento online.

2.1.4 Arquitetura e Tecnologias

Embora detalhes específicos não sejam fornecidos pela empresa, a aplicação utiliza tecnologias comuns em sistemas similares, como:

- Plataformas baseadas em *cloud* [Mic24].
- GPS para localização [Nat23].
- Sensores IoT [Zip23] para monitorizar a disponibilidade dos lugares.
- Sistemas de pagamentos online.

A combinação dessas tecnologias permite fornecer informações em tempo real sobre a disponibilidade e preços dos estacionamento.

2.1.5 Interface

A aplicação é projetada para ser amigável, facilitando tarefas como navegação em tempo real e reservas de lugares, especialmente na versão móvel. O *website* oferece informações detalhadas sobre os estacionamento. Para funcionar, utiliza uma API que facilita conexões e interações com outras aplicações [Par23c].

2.1.6 Aspetos Positivos e Negativos

Positivos:

- Interface gráfica fácil de usar.
- Promove a mobilidade urbana.

Negativos:

- Precisão dos dados em tempo real pode ser comprometida.
- Dependência de sensores externos e sua cobertura pode afetar a fiabilidade dos dados e a experiência do utilizador em algumas localizações.

2.2 ParkWhiz - Análise Visual

A aplicação **ParkWhiz** visa revolucionar a experiência de estacionamento, permitindo aos utilizadores procurar, comparar e reservar lugares de estacionamento em qualquer lugar e a qualquer momento. Isso facilita o planeamento antecipado de viagens ou a procura de lugares imediatos. [Par23d]



Figura 2.2: Interfaces Gráficas - Parkwhiz

2.2.1 Utilizadores

Os utilizadores do **ParkWhiz** dividem-se em três categorias:

- **Urbanos Ativos:** Necessitam de estacionamento próximo às suas necessidades diárias.

- **Viajantes:** Deslocam-se de cidade em cidade e precisam de estacionamento perto de destinos específicos.
- **Profissionais Ocupados:** Querem evitar o *stress* de procurar estacionamento, preferindo reservar antecipadamente para ganhar tempo e conveniência.

2.2.2 Contextos de Utilização

Os principais contextos de utilização incluem:

- Deslocações diárias para o trabalho.
- Compras e lazer.
- Viagens para locais desconhecidos.

2.2.3 Tarefas Executadas

Os utilizadores podem realizar várias tarefas através da aplicação [Par23f]:

- **Pesquisar Vagas de Estacionamento:** Busca por vagas disponíveis em áreas específicas.
- **Comparar Preços:** Comparação de preços entre diferentes locais de estacionamento.
- **Reservar Vaga:** Reserva de vagas com alguns cliques.
- **Pagamento Online:** Transações seguras de pagamento online.
- **Aceder ao Passe de Estacionamento Móvel:** Recebimento e acesso ao passe de estacionamento no dispositivo móvel.
- **Planeamento Antecipado:** Reserva de estacionamento com antecedência.
- **Economizar com Descontos:** Aproveitamento de descontos nas tarifas de estacionamento.
- **Aceder ao Histórico de Reservas:** Gestão e visualização do histórico de reservas.

2.2.4 Arquitetura e Tecnologias

Embora os detalhes específicos não sejam fornecidos pela empresa, a aplicação utiliza padrões comuns, como:

- Plataformas baseadas em *cloud* [Mic24].

- GPS para localização [Nat23].
- Sensores *IoT* [Zip23] para monitorizar disponibilidade.
- Sistemas de pagamentos online.

A combinação dessas tecnologias fornece informações em tempo real sobre disponibilidade e preços de estacionamento.

Interface

A interface é amigável e proporciona uma boa experiência de utilização [Par23e]:

- **Página Inicial:** Barra de pesquisa acessível, permitindo inserir locais específicos ou usar o GPS [Nat23] para encontrar estacionamentos próximos.
- **Resultados de Pesquisa:** Informações claras sobre disponibilidade, preços e descontos.
- **Processos de Reservas:** Simples, com opções de pagamento seguras integradas.
- **Detalhes de Reservas:** Visualização dos detalhes das reservas e acesso aos passes de estacionamento móvel.

2.2.5 Aspetos Positivos e Negativos

Positivos:

- Solução eficiente para diversas necessidades de estacionamento.
- Permite planeamento antecipado e utilização de passes de estacionamento móveis.
- Economiza tempo e dinheiro, com descontos de até 50
- Boa interface e integração com GPS para uma experiência de uso positiva.

Negativos:

- Fraca cobertura em várias áreas.
- Possível variação na disponibilidade em tempo real, causando falhas de dados em momentos de alta utilização.

2.3 Low Cost Smart Parking System with IoT - Análise Física

Este artigo apresenta uma arquitetura para um sistema de estacionamento inteligente de baixo custo que atualiza constantemente o estado de ocupação dos lugares de estacionamento [C S20].

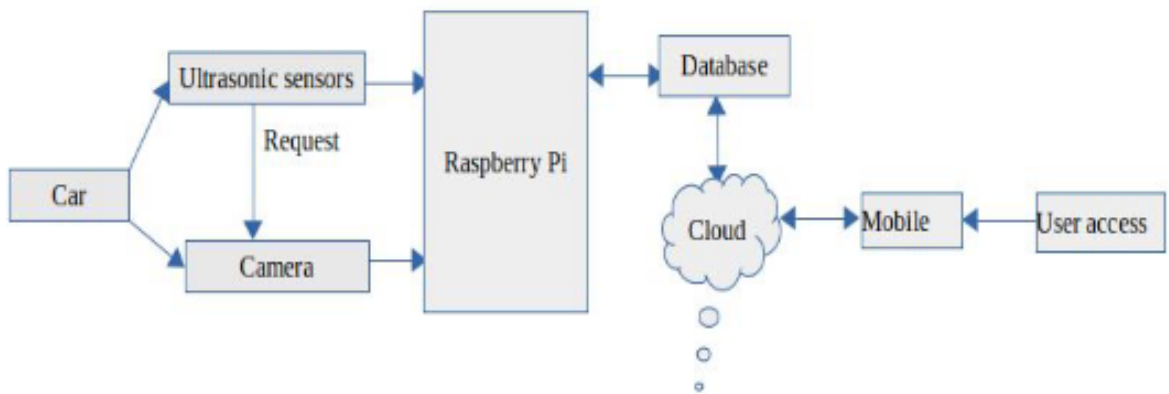


Figura 2.3: Arquitetura Proposta Artigo 1

2.3.1 Arquitetura e Tecnologias

A arquitetura proposta utiliza os seguintes componentes:

- **Sensores Ultrassônicos HC-SR04:** Detectam a ocupação dos lugares de estacionamento [Dig24].
- **Câmara:** Captura imagens dos lugares de estacionamento [Ras24b].
- **Raspberry Pi 3B:** Processa as imagens capturadas e envia-as para a nuvem [Ras24a].
- **Google Cloud Vision API:** Realiza o processamento de imagem e a detecção da ocupação dos lugares [Goo24b].
- **Cloud (Microsoft Azure):** Armazena os dados de ocupação e facilita o acesso aos utilizadores através de dispositivos móveis [Mic24].

2.3.2 Funcionamento Geral

- **Deteção da Ocupação:** Sensores ultrassônicos e uma câmara capturam o estado de ocupação dos lugares [Dig24].

- **Processamento de Imagem:** Imagens capturadas são enviadas para o *Raspberry Pi 3B* [Ras24a], que as transmite para a *Google Cloud Vision API* [Goo24b].
- **Atualização de Dados:** A *Google Cloud Vision API* [Goo24b] processa as imagens e determina a ocupação dos lugares.
- **Armazenamento e Acesso:** A informação é enviada para uma *cloud* (*Microsoft Azure*) [Mic24], onde os dados são armazenados e disponibilizados para acesso por dispositivos móveis.

2.3.3 Descrição da Interface

A interface do sistema permite aos utilizadores acessar facilmente as informações sobre a ocupação dos lugares de estacionamento através de dispositivos móveis, proporcionando uma experiência de uso conveniente e eficiente.

2.3.4 Aspetos Positivos e Negativos

Positivos:

- **Custo-efetivo:** Utiliza componentes acessíveis como o **Raspberry Pi** [Ras24a] e sensores ultrassónicos [Dig24].
- **Atualizações Constantes:** Proporciona informações em tempo real sobre a ocupação dos lugares de estacionamento.
- **Acesso Remoto:** Facilita o acesso aos dados através de dispositivos móveis.

Negativos:

- **Dependência de Conectividade:** Requer uma conexão constante à *internet* para o envio e processamento das imagens.
- **Precisão de Detecção:** A precisão depende da eficácia dos sensores e do processamento de imagem realizado pela *Google Cloud Vision API* [Goo24b].

2.4 *IoT Based Cost Effective Car Parking Management for Urban Area - Análise Física*

Este sistema visa monitorizar todos os lugares de um parque de estacionamento, fornecendo informações sobre a ocupação e controlando o acesso ao parque através de uma cancela operada por um motor servo. [Pey+21]

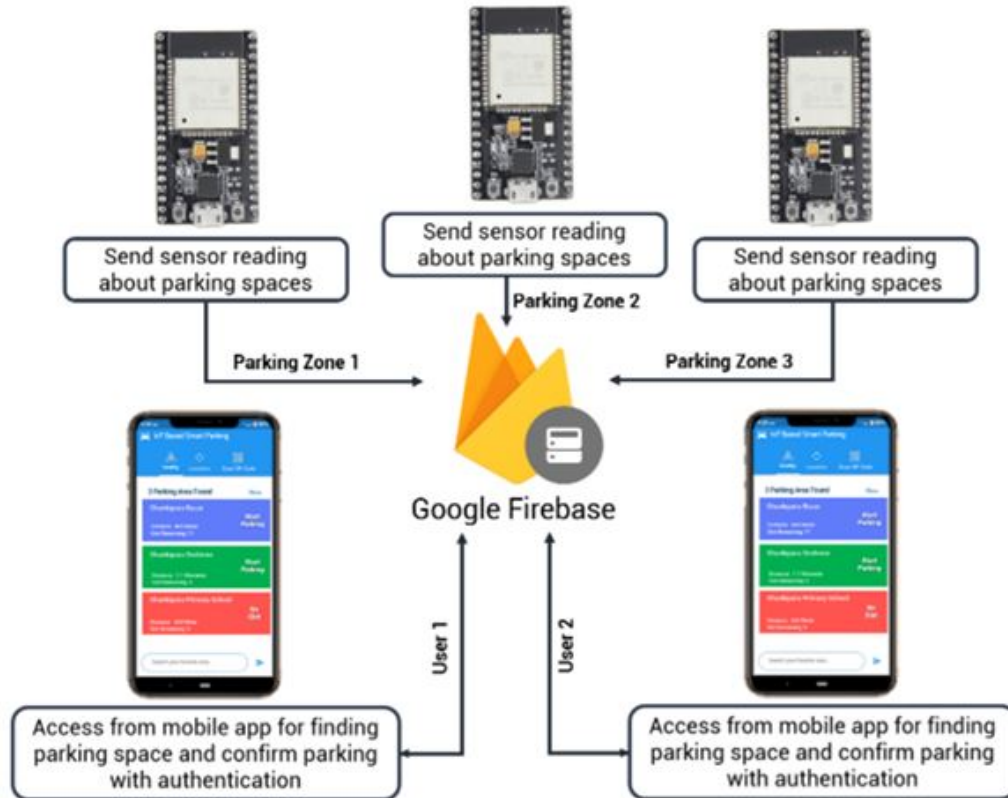


Figura 2.4: Arquitetura Proposta Artigo 2

2.4.1 Arquitetura e Tecnologias

A arquitetura do sistema é composta por componentes de hardware e software mais complexos, incluindo:

- **ESP32:** Três unidades de ESP32 são responsáveis por recolher informações de cada lugar de estacionamento [Esp24].
- **Motor Servo:** Opera a cancela que controla a entrada dos utilizadores no parque [Rea24].
- **Google Firebase:** Armazena os dados recolhidos pelos ESP32 e facilita o acesso em tempo real [Goo24a].

2.4.2 Funcionamento Geral

- **Monitorização da Ocupação:** Os ESP32 recolhem dados sobre a ocupação de cada lugar de estacionamento [Esp24].
- **Envio de Dados:** As informações de ocupação são enviadas diretamente para uma base de dados no *Google Firebase* [Goo24a].
- **Acesso aos Dados:** Utilizadores podem acessar os dados em tempo real através de uma aplicação móvel para verificar a disponibilidade de lugares.
- **Controlo de Acesso:** A entrada no parque é controlada por uma cancela operada por um motor servo, com base nas informações de ocupação.

2.4.3 Descrição da Interface

A interface do sistema permite que os utilizadores acessem facilmente os dados sobre a disponibilidade de lugares de estacionamento através de uma aplicação móvel, proporcionando uma experiência de uso eficiente e conveniente.

2.4.4 Aspetos Positivos e Negativos

Positivos:

- **Complexidade Avançada:** Utiliza tecnologias avançadas como ESP32 [Esp24] e *Google Firebase* [Goo24a] para uma monitorização preciso .
- **Controle de Acesso:** A cancela operada por motor servo melhora a gestão do estacionamento.
- **Acesso em Tempo Real:** Utilizadores podem verificar a disponibilidade de lugares instantaneamente através da aplicação móvel.

Negativos:

- **Dependência de Conectividade:** Requer uma conexão constante à *internet* para o envio e acesso aos dados.
- **Complexidade de Implementação:** A configuração e manutenção do sistema podem ser mais desafiadoras devido à complexidade do hardware e software utilizados.

Capítulo 3

Análise do Sistema

Com o objetivo de criar uma solução para melhorar a busca por vagas de estacionamento em áreas urbanas densas, é necessária uma análise que inclua uma descrição dos casos de uso, especificando as interações dos utilizadores com o sistema, requisitos funcionais e não funcionais.

3.1 Casos de Uso

Casos de uso descrevem interações entre utilizadores e sistema, detalhando passos para alcançar objetivos específicos, focando funcionalidades e requisitos do projeto [Gee24a].

3.1.1 Procurar Vagas de Estacionamento

- **Ator:** Utilizador
- **Pré-condições:** Utilizador está autenticado no sistema através da interface.
- **Fluxo Principal:**
 1. O utilizador abre a interface do sistema.
 2. O utilizador procura a cidade que pretende na secção de pesquisa.
 3. O sistema exibe um mapa com os parques disponíveis na cidade pesquisada.
 4. O utilizador seleciona, no mapa, o parque que pretende visualizar.
 5. O sistema exibe um *PopUp* [Get24] com estado dos lugares, em tempo real.
 6. O utilizador visualiza os lugares disponíveis e escolhe um.
 7. O utilizador, se pretender, seleciona a opção para mostrar o caminho até ao parque.

3.1.2 Registrar-se na Plataforma

- **Ator:** Utilizador
- **Pré-condições:** Nenhuma.
- **Fluxo Principal:**
 1. O utilizador abre a interface do sistema.
 2. O utilizador selecciona a opção **Registrar-se**.
 3. O utilizador preenche os dados pessoais.
 4. O utilizador confirma o registo.
 5. O sistema guarda os dados e confirma o registo.
- **Pós-condições:** Utilizador está registado na plataforma.

3.1.3 Reservar Lugar para Necessidades Especiais

- **Ator:** Utilizador com necessidades especiais
- **Pré-condições:** Utilizador está autenticado e registado com necessidades especiais.
- **Fluxo Principal:**
 1. O utilizador abre a interface do sistema.
 2. O utilizador procura a cidade que pretende na secção de pesquisa.
 3. O sistema exhibe um mapa com os parques disponíveis, com lugares para necessidades especiais, na cidade pesquisada.
 4. O utilizador selecciona, no mapa, o parque que pretende visualizar.
 5. O sistema exhibe um *PopUp* [Get24] com estado dos lugares, em tempo real, juntamente com a possibilidade de reserva.
 6. O utilizador visualiza os lugares disponíveis, escolhe um e, se pretender, reserva.
- **Pós-condições:** O lugar para necessidades especiais fica reservado para o utilizador.

3.1.4 Abrir Cancela do Lugar Reservado

- **Ator:** Utilizador com necessidades especiais
- **Pré-condições:** Utilizador está autenticado, o lugar está reservado e a hora atual está entre os tempos definidos na reserva.
- **Fluxo Principal:**
 1. O utilizador chega ao lugar reservado na hora da reserva.
 2. O utilizador abre a interface do sistema.
 3. O utilizador abre, na interface do sistema, a secção **As minhas reservas**.
 4. O utilizador seleciona a opção **Abrir Cancela** na reserva.
 5. O sistema abre a cancela.
- **Pós-condições:** A cancela do lugar reservado está aberta.

3.1.5 Analisar a plataforma

- **Ator:** Utilizador com permissões de administrador
- **Pré-condições:** Utilizador está autenticado e tem permissões de administrador.
- **Fluxo Principal:**
 1. O utilizador abre a interface do sistema.
 2. O utilizador abre, na interface do sistema, uma das secções **Arquivo** ou ***Dash-board*** [Tab24].
 3. O sistema disponibiliza os dados referentes à secção selecionada.
 4. O utilizador analisa os dados.

3.2 Requisitos Funcionais

Requisitos funcionais descrevem o que o sistema deve fazer, suas funcionalidades e comportamentos específicos [Gee24b].

3.2.1 Autenticação e Registo

- O sistema deve permitir que os utilizadores se registem e autenticem na plataforma.
- O sistema deve permitir que utilizadores registem suas necessidades especiais.

3.2.2 Pesquisa e Reserva de Lugares

- O sistema deve exibir num mapa com a disponibilidade de vagas em tempo real.
- O sistema deve permitir que os utilizadores reservem vagas adaptadas para necessidades especiais.

3.2.3 Atualização em Tempo Real

- O sistema deve atualizar a disponibilidade das vagas em tempo real.

3.2.4 Controle de Acesso

- O sistema deve permitir que utilizadores com necessidades especiais abram a cancela do lugar reservado através da interface.

3.3 Requisitos Não Funcionais

Requisitos não funcionais definem como o sistema deve operar, incluindo desempenho, segurança e usabilidade, entre outros [Gee24b].

3.3.1 Desempenho

- O sistema deve responder às solicitações do utilizador em menos de 2 segundos.
- O sistema deve processar a atualização de disponibilidade de vagas em tempo real com um atraso entre 10 e 30 segundos.

3.3.2 Segurança

- O sistema deve garantir que os dados dos utilizadores estejam protegidos contra acesso não autorizado.
- As transações de reserva e abertura de cancela devem ser seguras e encriptadas.

3.3.3 Usabilidade e Compatibilidade

- A interface do sistema deve ser intuitiva e fácil de usar, permitindo que utilizadores de todas as idades naveguem facilmente.
- O sistema deve estar disponível em múltiplas plataformas (*iOS*, *Android*, *Web*).

3.3.4 Escalabilidade

- O sistema deve ser capaz de suportar um grande número de utilizadores simultâneos sem degradação de desempenho.

3.3.5 Confiabilidade

- O sistema deve ter mecanismos de recuperação em caso de falhas.

Capítulo 4

Desenho do Sistema

Numa abordagem abstrata para preparar o sistema *ParkLookUp*, foram consideradas três vertentes fundamentais:

1. Interface Gráfica.
2. Arquitetura Física.
3. Arquitetura da Base de Dados.

Cada uma dessas vertentes desempenha um papel crucial no design e desenvolvimento do sistema, contribuindo para que se respeitem os pontos referidos no capítulo anterior (Casos de Uso, Requisitos Funcionais e Requisitos Não Funcionais).

4.1 Desenho da Interface Gráfica

Para demonstrar um desenho mais concreto do sistema proposto, seguem-se, através de grafismo, as interfaces que o sistema possui e ainda uma descrição dos mesmos.

4.1.1 Interface de Registo do Utilizador

The mockup shows a registration form titled "Criação de Conta". On the left is a logo for "PARK LOOK UP" with an image of two cars. The form contains the following fields: "Vasco" (first name), "Teste" (last name), "AA-02-AS" (license plate), "987654321" (phone number), "vascoteste@gmail.com" (email), "Morada Teste" (address), two password fields (each with "*****" placeholder), "UMM" (fuel type), "Diesel" (fuel type), and "Jipe" (vehicle type). A "Registrar" button is at the bottom. Below the button is the text "Já possui conta? Realize o Login".

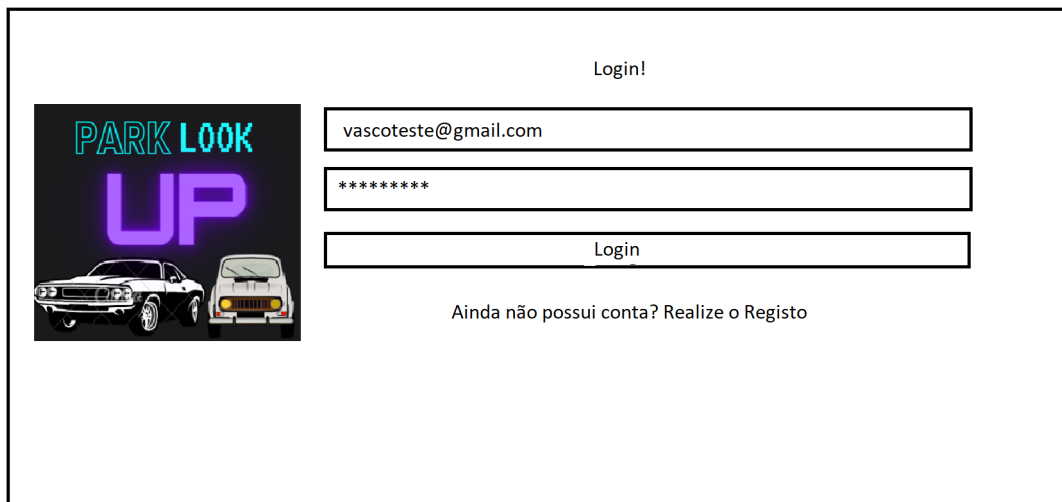
Figura 4.1: Interface de Registo do Utilizador

A interface de registo é intuitiva e amigável e permite aos utilizadores inserir suas informações pessoais e de seu veículo para criar uma conta. Os campos incluem:

- **Nome:** Campo de texto para inserir o nome.
- **Apelido:** Campo de texto para inserir o apelido.
- **Número de Telefone:** Campo de texto para inserir o número de telefone.
- **E-mail:** Campo de texto para inserir o endereço de *e-mail*.
- **Morada:** Campo de texto para inserir a morada completa.
- **Password:** Campo de *password* para adicionar segurança.
- **Matrícula do Veículo:** Campo de texto para inserir a matrícula do veículo.
- **Marca do Veículo:** Campo de texto para inserir a marca do veículo.
- **Combustível do Veículo:** Campo de *dropdown* para escolher o tipo de combustível do veículo.
- **Tipo do Veículo:** Campo de *dropdown* para escolher o tipo de veículo.

O formulário pode incluir validações em tempo real para garantir que todos os campos sejam preenchidos corretamente antes da submissão.

4.1.2 Interface de Autenticação do Utilizador



The image shows a user authentication interface within a rectangular frame. On the left side, there is a logo for 'PARK LOOK UP' featuring the text in a stylized font above two cars. To the right of the logo, the interface includes the following elements: the title 'Login!' at the top; an email input field containing 'vascoteste@gmail.com'; a password input field with masked characters '*****'; a 'Login' button; and a link at the bottom that reads 'Ainda não possui conta? Realize o Registo'.

Figura 4.2: Interface de Autenticação do Utilizador

A interface de autenticação é intuitiva e amigável, permitindo aos utilizadores inserir seu *email* e *password* para aceder às suas contas. Os campos incluem:

- ***E-mail***: Campo de texto para inserir o endereço de *e-mail* para efetuar a autenticação.
- ***Password***: Campo de *password* para efetuar a autenticação.

O formulário pode incluir validações em tempo real para garantir que todos os campos sejam preenchidos corretamente antes da submissão.

4.1.3 Interface do Utilizador Autenticado

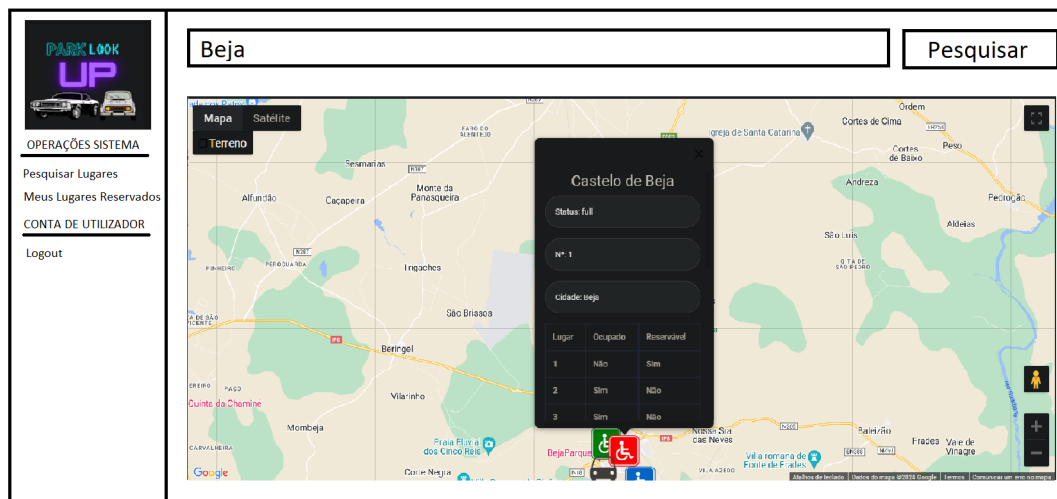


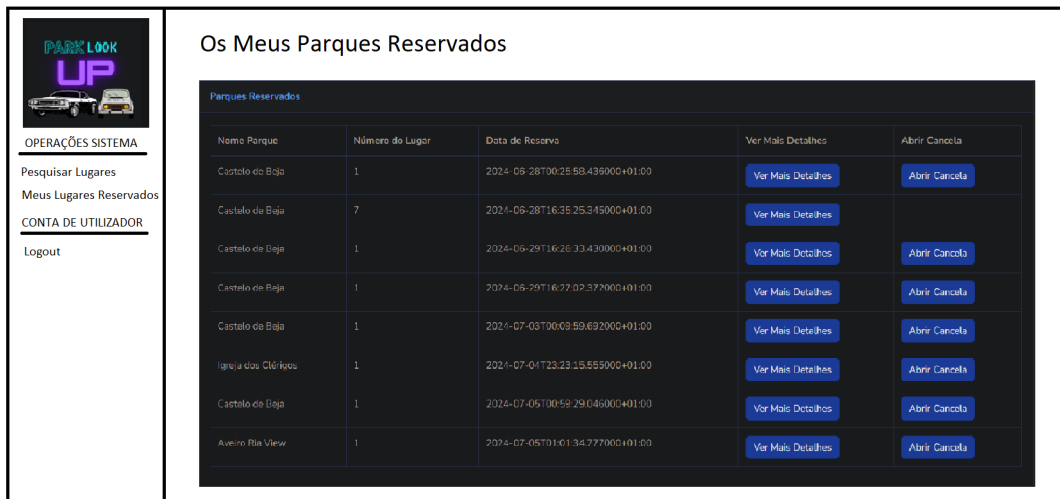
Figura 4.3: Interface do Utilizador Autenticado

A interface autenticada é projetada para proporcionar aos utilizadores uma experiência de navegação e gestão de estacionamento fácil. Os principais componentes da interface são:

- **Barra de Pesquisa por Cidade:** Localizada no topo da tela, permite que o utilizador pesquise rapidamente estacionamentos em diferentes cidades.
- **Mapa Interativo:** Ocupando a maior parte da tela, o mapa permite que os utilizadores explorem e localizem parques de estacionamento. O mapa exibe ícones que representam os estacionamentos disponíveis. Ao clicar em um ícone, o utilizador pode ver detalhes em tempo real sobre o estacionamento.
- **Menu Lateral Esquerdo:**
 - **Terminar Sessão:** Opção para sair da conta.
 - **Meus Lugares Reservados:** Permite ao utilizador visualizar e gerir os lugares de estacionamento que foram reservados.
- **Detalhes do Estacionamento:**
 - **Estado em Tempo Real:** Ao clicar em um pino no mapa, o utilizador pode ver quais lugares estão vagos e quais estão ocupados.
 - **Reserva de Lugar ou Caminho com GPS:** O utilizador pode selecionar um lugar específico do estacionamento através de um *dropdown* e clicar no botão "Reservar" para confirmar a reserva. Ou, para utilizadores sem necessidades especiais, clicar no botão "Mostrar no GPS" de modo a que a rota para o parque desejado apareça no mapa [Nat23].

Esta interface foi projetada para ser intuitiva e funcional, oferecendo informações e opções de gestão de estacionamento de forma clara e acessível.

4.1.4 Interface do Utilizador Autenticado - Meus lugares reservados



Os Meus Parques Reservados				
Parques Reservados				
Nome Parque	Número do Lugar	Data de Reserva	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	1	2024-06-28T00:25:53.439000+01:00	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	7	2024-06-28T16:35:25.315000+01:00	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	1	2024-06-29T16:26:33.430000+01:00	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	1	2024-06-29T16:27:02.372000+01:00	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	1	2024-07-03T00:03:59.692000+01:00	Ver Mais Detalhes	Abrir Cancela
Igreja dos Clérigos	1	2024-07-04T23:23:15.555000+01:00	Ver Mais Detalhes	Abrir Cancela
Castelo da Beja	1	2024-07-05T00:59:29.046000+01:00	Ver Mais Detalhes	Abrir Cancela
Avenida Rita View	1	2024-07-05T01:01:34.777000+01:00	Ver Mais Detalhes	Abrir Cancela

Figura 4.4: Interface do Utilizador Autenticado - Meus lugares reservados

A interface de gestão de reservas é projetada para oferecer aos utilizadores uma visão clara e organizada de suas reservas de estacionamento. Os componentes principais desta interface são:

- **Tabela de Reservas:** Ocupa a maior parte da tela e fornece informações detalhadas sobre as reservas do utilizador. A tabela contém as seguintes colunas:
 - **Nome do Parque:** Exibe o nome do parque de estacionamento.
 - **Lugar:** Indica o lugar específico reservado dentro do parque.
 - **Data de Saída da Reserva:** Mostra a data e a hora do término da reserva.
 - **Ver Mais Detalhes:** Botão ou link que permite ao utilizador acessar informações detalhadas sobre a reserva.
 - **Abrir Cancela:** Botão para abrir a cancela do estacionamento associado à reserva.
- **Menu Lateral Esquerdo:**
 - **Terminar Sessão:** Opção para sair da conta.

Esta interface é desenhada para ser eficiente e fácil de usar, proporcionando acesso rápido e direto às informações e funcionalidades essenciais para a gestão das reservas de estacionamento.

4.2 Desenho da Arquitetura da Base de Dados

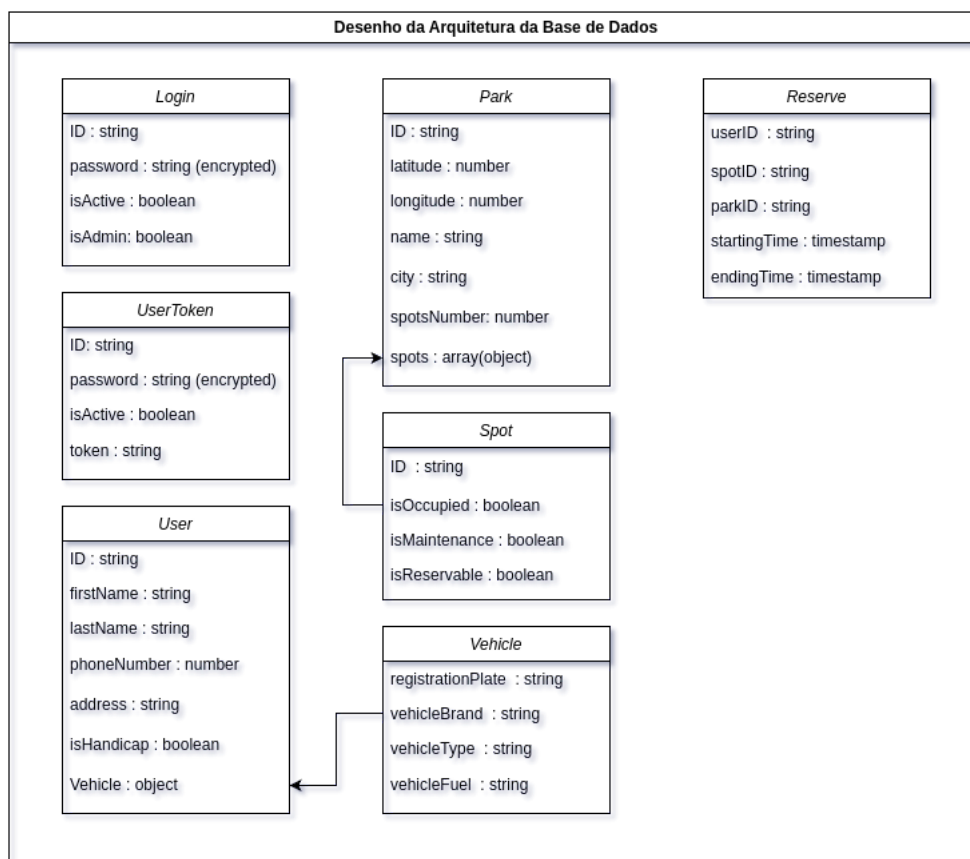


Figura 4.5: Desenho da Arquitetura da Base de Dados

A arquitetura da base de dados para o sistema de estacionamento inteligente *ParkLookUp* é composta por vários blocos de dados interligados, cada um desempenhando um papel específico na gestão das informações de estacionamento. Aqui está uma descrição simples das blocos de dados e os seus campos:

4.2.1 Login

Armazena as credenciais de acesso dos utilizadores, permitindo a autenticação segura no sistema.

- **ID:** Identificador único (string)
- **password:** Senha do utilizador (string criptografada)
- **isActive:** Indica se a conta está ativa (boolean)
- **isAdmin:** Indica se o utilizador é administrador (boolean)

4.2.2 UserToken

Gere *tokens* de autenticação para sessões de utilizadores, garantindo acesso seguro e autorizado.

- **ID**: Identificador único (string)
- **password**: Senha do utilizador (string criptografada)
- **isActive**: Indica se a conta está ativa (boolean)
- **token**: Chave de autenticação (string)

4.2.3 User

Contém informações pessoais dos utilizadores, como nome, contacto e detalhes sobre necessidades especiais.

- **ID**: Identificador único (string)
- **firstName**: Primeiro nome do utilizador (string)
- **lastName**: Sobrenome do utilizador (string)
- **phoneNumber**: Número de telefone do utilizador (number)
- **address**: Endereço do utilizador (string)
- **isHandicap**: Indica se o utilizador possui necessidades especiais (boolean)
- **vehicle**: Lista de veículos associados ao utilizador (object)

4.2.4 Park

Regista detalhes dos parques de estacionamento, incluindo localização, nome, cidade e número de vagas disponíveis.

- **ID**: Identificador único (string)
- **latitude**: Latitude do parque de estacionamento (number)
- **longitude**: Longitude do parque de estacionamento (number)
- **name**: Nome do parque de estacionamento (string)
- **city**: Cidade onde o parque está localizado (string)
- **spotsNumber**: Número total de vagas no parque (number)
- **spots**: Array de objetos de lugares de estacionamento (array(objetos))

4.2.5 Spot

Monitora o estado individual das vagas de estacionamento, indicando se estão ocupadas, em manutenção ou disponíveis para reserva.

- **ID**: Identificador único (string)
- **isOccupied**: Indica se a vaga está ocupada (boolean)
- **isMaintenance**: Indica se a vaga está em manutenção (boolean)
- **isReservable**: Indica se a vaga pode ser reservada (boolean)

4.2.6 Vehicle

Armazena informações sobre os veículos dos utilizadores, como placa de registo, marca, tipo e combustível.

- **registrationPlate**: Placa de registo do veículo (string)
- **vehicleBrand**: Marca do veículo (string)
- **vehicleType**: Tipo de veículo (string)
- **vehicleFuel**: Tipo de combustível do veículo (string)

4.2.7 Reserve

Regista as reservas de lugares de estacionamento, incluindo dados do utilizador, lugar reservada, parque e horários de início e término.

- **userID**: Identificador único do utilizador que fez a reserva (string)
- **spotID**: Identificador único da vaga reservada (string)
- **parkID**: Identificador único do parque de estacionamento (string)
- **startingTime**: Hora de início da reserva (timestamp)
- **endingTime**: Hora de término da reserva (timestamp)

Essa estrutura permite a gestão eficiente das informações de utilizadores, veículos, vagas de estacionamento e reservas, garantindo que o sistema funcione de maneira organizada e eficaz. A tipagem de variáveis seguiu o padrão da tecnologia *MongoDB* [Mon24a].

4.3 Desenho da Arquitetura Física

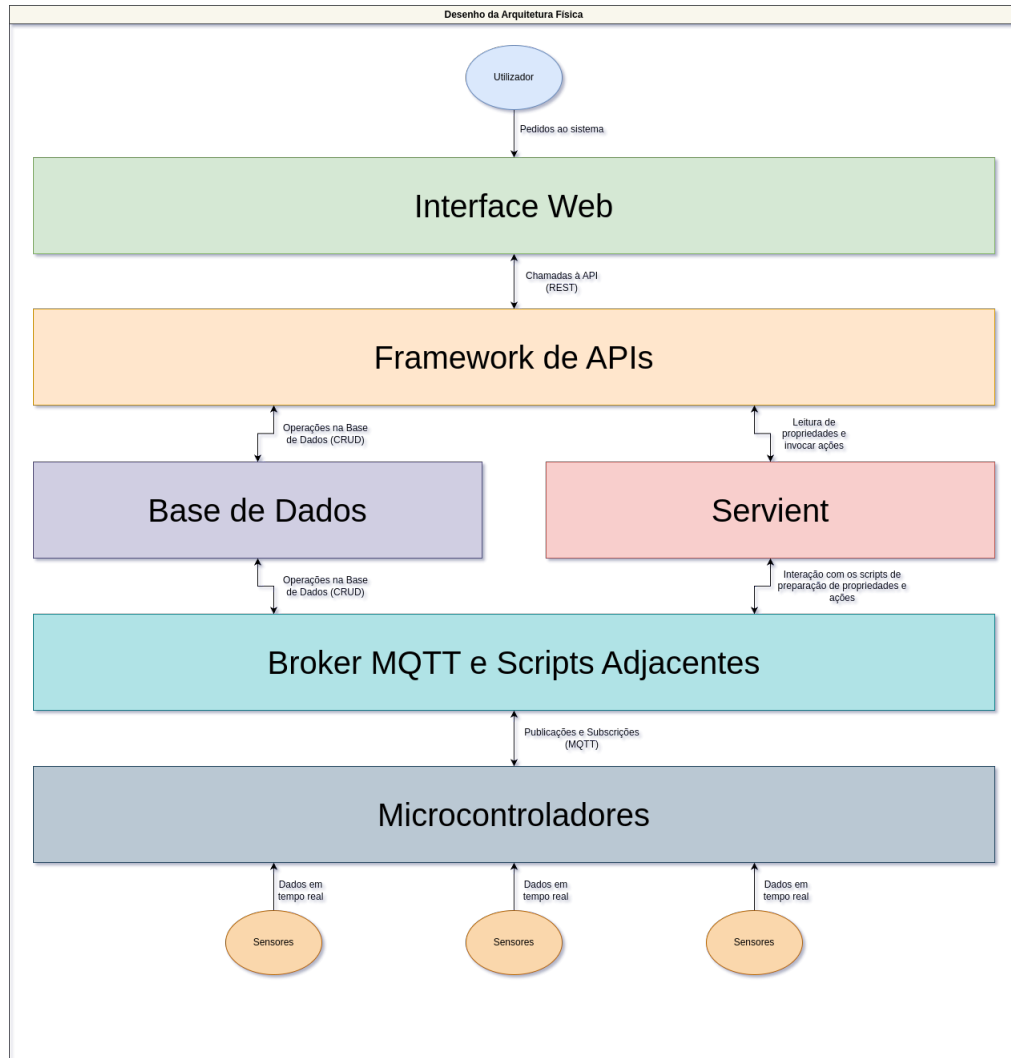


Figura 4.6: Desenho da Arquitetura Física - Abstrato

O desenho da arquitetura física, em formato abstrato, do sistema de *ParkLookUp* é composto por várias camadas interconectadas, cada uma desempenhando um papel essencial no funcionamento do sistema. Aqui está uma descrição simples de cada camada:

4.3.1 Utilizador

- Interage com o sistema através da Interface Web para acessar informações e realizar operações.

4.3.2 Interface *Web*

- Proporciona uma interface gráfica para os utilizadores, facilitando a interação com o sistema através de navegadores web.

4.3.3 *Framework de APIs*

- Garante a comunicação entre a Interface Web e outros componentes do sistema, permitindo o acesso a dados e serviços [AWS23].

4.3.4 Base de Dados

- Armazena todas as informações necessárias, como dados de utilizadores, veículos, reservas e estado das vagas de estacionamento.

4.3.5 *Servient*

- Gerência a comunicação com dispositivos *IoT* [IBM24] e processa os dados recebidos dos micro-controladores [Gee24c] e sensores [Zip23].

4.3.6 *Broker MQTT e Scripts Adjacentes*

- Facilita a comunicação eficiente entre os micro-controladores [Gee24c] e o sistema central, utilizando o protocolo *MQTT* [Emq24b] para a transmissão de dados.

4.3.7 Micro-controladores

- Controlam os sensores no campo, coletando dados sobre a ocupação das vagas de estacionamento [Gee24c; Zip23].

4.3.8 Sensores

- Monitorizam as vagas de estacionamento, detetando a presença ou ausência de veículos e enviando esses dados para os micro-controladores [Gee24c; Zip23].

Esta estrutura modular permite uma gestão eficiente do sistema de estacionamento, garantindo uma comunicação fluida entre utilizadores, dispositivos de campo e a central de dados.

Capítulo 5

Implementação do Sistema

A implementação do sistema baseia-se na leitura do desenho da arquitetura trabalhado no capítulo anterior e na aplicação do mesmo com tecnologias estudadas ao longo do mestrado.

Assim surgiu outro desenho de arquitetura, em contexto real, demonstrando como o sistema *ParkLookUp* se encontra implementado. Durante o planeamento desta arquitetura houve um elevado foco em respeitar arquitetura base *W3C* [W324].

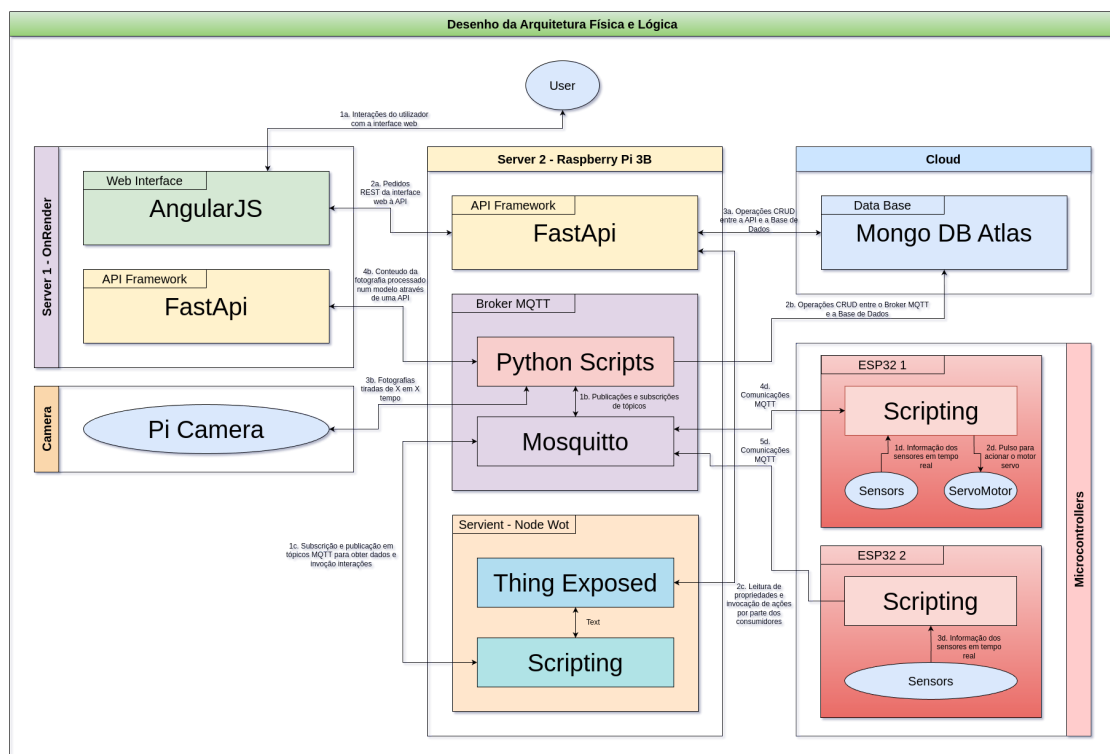


Figura 5.1: Desenho da Arquitetura Implementada no Sistema

5.1 Grupos e tecnologias para tratamento de dados

5.1.1 Utilizador

- Interage com o sistema através da Interface Web para visualizar a disponibilidade de vagas e realizar reservas.

5.1.2 Câmara

- **Pi Camera** - Captura imagens dos veículos e das vagas de estacionamento nos parques exteriores [Ras24b].

5.1.3 Servidor 1 - *OnRender*

«OnRender is a unified cloud to build and run all your apps and websites with free TLS certificates, global CDN, private networks and auto deploys from Git» [Ren24].

- **Web Interface(AngularJS)** - Proporciona a interface gráfica para os utilizadores [Ang24].
- **API Framework (FastApi)** - Expõe um *EndPoint* [AWS23] com protocolo *HTTP* [Clo24] onde as imagens são processadas através de um modelo de inteligência artificial para detetar os lugares vagos num parque de estacionamento exterior [Fas24].

5.1.4 Cloud

- **Data Base (MongoDB Atlas)** - Armazena informações sobre utilizadores, veículos, reservas, e estado dos lugares de estacionamento [Mon24b].

5.1.5 Servidor 2 - Raspberry Pi 3B

Raspberry Pi é um microcomputador de baixo custo, tamanho compacto, usado para aprendizado, projetos de programação e eletrónica, com diversas aplicações práticas [Ras24a].

- **API Framework (FastApi)** - Processa requisições vindas da interface *Web* através *HTTP* [Clo24], responde e realiza operações na Base de Dados [Fas24].
- **Broker MQTT (Mosquitto)** - Facilita a comunicação entre dispositivos através do protocolo *MQTT* [Emq24b; Mos24; Emq24a].
- **Python Scripts** - Executa *scripts* em *Python* para processar dados e gerir a lógica do parque de estacionamento [Pyt24].

- ***Servient - Node WoT***

- **Thing Exposed** - Expõe dispositivos IoT [IBM24] e suas capacidades para que possam ser lidos dados e disparadas ações [thi24].
- **Scripting** - Executa *scripts* para integrar e gerir dispositivos IoT [IBM24; thi24].

5.1.6 Microcontroladores

- **ESP32 1 & 2** [Esp24]

- **Scripting** - Executa scripts para coletar dados dos sensores e controlar atuadores.
- **Sensores** - Monitoram o estado das vagas de estacionamento [Zip23].
- **ServoMotor** - Controla a abertura e o fecho das cancelas [Rea24].

5.2 Arquitetura do W3C no sistema *ParkLookUp*

A *Web of Things (WoT)* [W324] visa melhorar a interoperabilidade e usabilidade da Internet das Coisas (IoT). Assim, a arquitetura W3C define um modelo abstrato da *WoT* [W324] e os seus componentes principais, explicando como eles se interligam. Assim temos 4 grupos principais:

- **Descrição de Coisas (*Thing Description*)**: Formato para descrever meta-dados e interfaces de coisas [W324].
- **Modelos de Ligação (*Binding Templates*)**: Diretrizes para definir interfaces de rede para protocolos específicos [W324].
- **API de Scripting (*Scripting API*)**: API JavaScript para desenvolver aplicações IoT de forma portátil [W324].
- **Diretrizes de Segurança e Privacidade (*Security and Privacy Guidelines*)**: Orientações para implementação segura e configuração de coisas, discutindo riscos e mitigação [W324].

A tecnologia usada como *Servient* [W324] neste sistema, foi *The Eclipse Thingweb node-wot* [thi24].

5.2.1 Descrição de Coisas (*Thing Description*) & Modelos de Ligação (*Binding Templates*)

O sistema *ParkLookUp* necessita de 2 interações principais:

- Leitura, em tempo real, do estado dos lugares de estacionamento.
- Abrir cancela para que os utilizadores com necessidades especiais possam estacionar.

Esta Descrição de Coisas (*Thing Description*) [W324] tem como representação um parque de estacionamento e prepara o sistema para trabalhar com as suas 2 interações principais.

```
1  //////////////////////////////////////
2  // THING DESCRIPTION
3  //////////////////////////////////////
4  const td = {
5    "@context": [
6      "https://www.w3.org/2019/wot/td/v1",
7      { "@language": "en" }
8    ],
9    "title": "park",
10   "description": "A parking lot with 3 spots
11     that provides the occupation in real time.",
12   "properties": {
13     "occupiedSpots": {
14       "title": "Occupied Spots",
15       "description": "Information of spots occupation.",
16       "type": "string",
17       "readOnly": true
18     }
19   },
20   "actions": {
21     "openDoor": {
22       "title": "Open Door",
23       "description": "Open the door of the provided park and spot.",
24       "uriVariables": {
25         "spotID": {
26           "type": "string",
27           "description": "Id of the spot.",
28         },
29         "parkID": {
30           "type": "string",
31           "description": "Id of the park.",
32         }
33       },
34       "forms": [
```

```

35         {
36             "href": "http://localhost:8080/park/actions/openDoor",
37             "htv:methodName": "POST"
38         }
39     ]
40 }
41 }
42 };

```

Listagem 5.1: Descrição de Coisas (*Thing Description*)

Propriedades (*properties*)

- *occupiedSpots*

- **title:** *"Occupied Spots"*: O título da propriedade.
- **description:** *"Information of spots occupation."*: Uma descrição da propriedade, indicando que fornece informações sobre a ocupação dos lugares.
- **type:** *"string"*: O tipo de dados da propriedade, que neste caso é uma string.
- **readOnly:** *true*: Indica que esta propriedade é só de leitura.

Ações (*actions*)

- *openDoor*

- **title:** *"Open Door"*: O título da ação.
- **description:** *"Open the door of the provided park and spot."*: Uma descrição da ação, indicando que abre a porta do parque e da vaga especificados.
- **uriVariables:** Define as variáveis necessárias para realizar a ação.
 - * **spotID**: Uma variável para identificar o número do lugar.
 - * **type:** *"string"*: O tipo de dados da variável, que é uma string.
 - * **description:** *"Id of the spot."*: Uma descrição da variável, indicando que é o identificador de um lugar.
 - * **parkID**: Uma variável para identificar o número do parque.
 - * **type:** *"string"*: O tipo de dados da variável, que é uma string.
 - * **description:** *"Id of the park."*: Uma descrição da variável, indicando que é o identificador do parque.
- **forms:** Define os detalhes da interação com a ação.

- * **href:** "*http://localhost:8080/park/actions/openDoor*": O endpoint para onde a solicitação **POST** [Red24] deve ser enviada para executar a ação.
- * **htv:methodName:** "**POST**": Define que o método **HTTP** [Clo24] a ser usado para esta ação é o **POST** [Red24].

Por fim, os Modelos de Ligação (*Binding Templates*) [W324], definem interações com determinado padrão para aceder às informações de ocupação de lugares e controlar a cancela do lugares para necessidades especiais. Assim, garante-se uma comunicação eficiente entre os elementos do sistema, conforme descrito na *Thing Description* [W324].

5.2.2 API de Scripting (*Scripting API*)

Para que os dados estejam sempre atualizados e disponíveis para todo o sistema, é necessário que se prepare os mesmos através de *scripting*. Para isso, neste sistema existem três operações, uma para preparar e expor os dados ao sistema, uma para ler o estado dos lugares do parque e outra para invocar a ação de abrir a cancela [W324].

Produtor

```

1  //////////////////////////////////////
2  // PRODUCE THING
3  //////////////////////////////////////
4  WoT.produce(td)
5    .then((thing) => {
6      //////////////////////////////////
7      // Connect Broker
8      const mqttClient = mqtt.connect("mqtt://192.168.1.83:1884", {
9        username: 'park',
10       password: '*****'
11     });
12
13     //////////////////////////////////
14     // Prepare readable property
15     let occupiedSpots = null;
16     thing.setPropertyReadHandler(
17       'occupiedSpots', async () => occupiedSpots
18     );
19
20     //////////////////////////////////
21     // Prepare action
22     thing.setActionHandler("openDoor", async (params, options) => {
23       //////////////////////////////////
24       // Read input provided by the client
25       const uriVariables = options.uriVariables;
26       const spotID =
27         "spotID" in uriVariables ? uriVariables.spotID : spotID;
28       const parkID =
29         "parkID" in uriVariables ? uriVariables.parkID : parkID;
30
31       //////////////////////////////////
32       // Prepare message to publish
33       const message =
34         '{ "spotID": ' + spotID +
35         + ', "parkID": ' + parkID +
36         + ' }';
37
38       //////////////////////////////////

```

```

39      // Prepare topic
40      const topic = "esp32/open_door";
41
42      //////////////////////////////////
43      // Publish message in the topic
44      mqttClient.publish(topic, message, { qos: 1 }, (err) => {
45          if (err) {
46              return { result: "Error publishing message!" };
47          } else {
48              return { result: "Door was asked to be opened!" };
49          }
50      });
51  });
52
53      //////////////////////////////////
54      // Recive the data of updated spots
55      thing.expose().then(() => {
56          mqttClient.on('connect', () => {
57              mqttClient.subscribe("rpi/update_spots", { qos: 1 });
58          });
59
60          mqttClient.on('message', (topic, message) => {
61              occupiedSpots = message.toString();
62          });
63      });
64  })
65  .catch((e) => {
66      console.log(e);
67  });

```

Listagem 5.2: API de Scripting (*Scripting API*) - *Producer*

De forma sintética, este código:

- **Produz a *thing*:** `WoT.produce(td)` cria a *thing* baseada na descrição fornecida [W324].
- **Conecta ao broker MQTT:** Conecta ao *broker MQTT* [Emq24a; Emq24b] em `mqtt://192.168.1.83:1884` com credenciais específicas.
- **Prepara uma propriedade legível:** Define o *handler* de leitura para a propriedade *occupiedSpots*.
- **Prepara uma ação:** Define o handler da ação `openDoor`, que lê parâmetros do cliente, prepara e publica uma mensagem no tópico `esp32/open_door` do *broker MQTT* [Emq24a; Emq24b].

- **Expõe a *thing* e atualiza a propriedade:** Inscreve-se no tópico `rpi/update_spots` para atualizar *occupiedSpots* quando mensagens são recebidas.

Cliente 1

```

1  //////////////////////////////////////////////////
2  // SYNC CLIENT WITH PRODUCER
3  //////////////////////////////////////////////////
4  WoT.requestThingDescription("http://localhost:8080/park")
5    .then(async (td) => {
6      try {
7        //////////////////////////////////////////////////
8        // Consume thing
9        const thing = await WoT.consume(td);
10
11       //////////////////////////////////////////////////
12       // Invoke the action with specific variables
13       const result = await thing.invokeAction("openDoor", undefined, {
14         uriVariables: { "spotID": "1", "parkID": "1" },
15       });
16     } catch (err) {
17       console.error("Script error:", err);
18     }
19   })
20   .catch((err) => {
21     console.error("Fetch error:", err);
22   });

```

Listagem 5.3: API de Scripting (*Scripting API*) - *Client 1*

De forma sintética, este código:

- **Requisita a descrição da "thing":** `WoT.requestThingDescription(...)` obtém a descrição da *thing* [W324].
- **Consome a *thing*:** `await WoT.consume(...)` consome a *thing* baseada na descrição obtida [W324].
- **Invoca uma ação:** `thing.invokeAction(...)` invoca a ação `openDoor` com variáveis específicas (`spotID` e `parkID`) [W324].

Resumindo, o código obtém a descrição de uma *thing*, consome essa *thing*, e invoca uma ação específica nela com variáveis fornecidas.

Cliente 2

```
1  //////////////////////////////////////
2  // SYNC CLIENT WITH PRODUCER
3  //////////////////////////////////////
4  WoT.requestThingDescription("http://localhost:8080/park")
5    .then(async (td) => {
6      try {
7        //////////////////////////////////
8        // Consume thing
9        const thing = await WoT.consume(td);
10
11        //////////////////////////////////
12        // Read the property
13        const read1 = await thing.readProperty("occupiedSpots");
14        console.log(await read1.value());
15      } catch (err) {
16        console.error("Script error:", err);
17      }
18    })
19    .catch((err) => {
20      console.error("Fetch error:", err);
21    });
```

Listagem 5.4: API de Scripting (*Scripting API*) - *Client 2*

De forma sintética, este código:

- **Requisita a descrição da *thing*:** `WoT.requestThingDescription(...)` obtém a descrição da *thing* [W324].
- **Consome a *thing*:** `await WoT.consume(...)` consome a *thing* baseada na descrição obtida [W324].
- **Lê uma propriedade:** `await thing.readProperty(...)` lê a propriedade `occupiedSpots` [W324].

Resumindo, o código obtém a descrição de uma *thing*, consome essa *thing*, lê a propriedade `occupiedSpots`, e exibe seu valor.

5.2.3 Diretrizes de Segurança e Privacidade (*Security and Privacy Guidelines*)

Sendo a **Segurança** um dos **Requisitos Não Funcionais** [Gee24b] procurados na implementação deste projeto, surge de encontro com a arquitetura **W3C** [W324] que revê este ponto como um dos seus pilares.

Assim, para que este sistema siga esta arquitetura, em termos de segurança seguimos três pontos principais:

- **Separação de conteúdo na base de dados** - Aquando do planeamento da base de dados, idealizou-se uma separação da informação do Utilizador, para evitar que os dados menos sensíveis, como *nome* e *apelido*, estivessem no mesmo local que a *password*. Com este método, criaram-se vários grupos de dados para separar os dados.
- **Encriptação das *passwords*** - Todas as *passwords* são guardadas na base de dados após serem encriptadas com o algoritmo **SHA256** [Nor24].
- **Autorização na API** - Para que todas as chamadas aos *endpoints* definidos na *API* [AWS23] sejam seguras é utilizado o sistema de autorização com base em *Bearer token* [Swa24].

Capítulo 6

Testes ao Sistema

Devido às limitações de transporte causadas pela maquete do sistema **ParkLookUp**, os testes foram difíceis de executar. No entanto, foi realizado um teste com um utilizador alheio ao desenvolvimento do projecto. Este teste seguiu a Técnica *Cognitive Walkthrough* [Int24] que se baseia em alguns pontos como:

- **Exploração Mental Empática:** Colocando-se nos sapatos do utilizador.
- **Procura ativa por obstáculos:** Descobrindo barreiras à usabilidade.
- **Padrões de Avaliação Perspicazes:** Critérios de análise afiados para descobrir falhas.
- **Processo Reflexivo e Iterativo:** Aprimorando continuamente a compreensão da usabilidade.

O teste gerou um vídeo onde se pode ver a interação do utilizador com o sistema tanto na interface *Web* como com a maquete real do parque de estacionamento. O vídeo seguirá em anexo a este relatório.

Capítulo 7

Conclusão

O sistema **ParkLookUp** visa solucionar os desafios decorrentes da urbanização crescente e do aumento da densidade populacional, que causam um tráfego urbano intenso e tornam difícil encontrar lugares de estacionamento disponíveis. Ao facilitar a busca por vagas em estacionamentos através de mapas e destacar a disponibilidade em tempo real, o sistema **ParkLookUp** busca aprimorar a eficiência e a experiência dos utilizadores. Adicionalmente, o sistema atende às necessidades de pessoas com necessidades especiais, permitindo que se registrem na plataforma e reservem vagas adaptadas às suas condições específicas.

Para adquirir um conhecimento profundo sobre o tema, foram analisados quatro sistemas semelhantes com o objetivo de preparar o sistema **ParkLookUp** da melhor forma possível para os utilizadores. Esta análise focou tanto na interface do utilizador quanto na implementação física em um estacionamento real.

O sistema pretendia atender aos requisitos funcionais, não funcionais e ainda aos casos de usos propostos durante a fase de análise, pois, em termos gerais, são os requisitos mínimos a contemplar por um sistema *IoT* que pretenda interagir com utilizadores. No entanto, os mesmos não foram aprofundados integralmente devido à carência de tempo para realizar o projeto.

Durante a análise do sistema, foram abordados aspetos cruciais como o desenho da interface gráfica, a arquitetura da base de dados e a arquitetura física. Estes elementos visam garantir uma interação intuitiva, uma gestão eficiente dos dados e uma implementação prática e robusta do sistema em ambientes reais.

7. CONCLUSÃO

Na implementação do sistema, foram considerados elementos da arquitetura **W3C**, como a descrição de coisas (*Thing Description*), modelos de ligação (Binding Templates), *scripts* de *API* e medidas de segurança. Esses componentes garantem a interoperabilidade, flexibilidade e proteção dos dados, essenciais para a eficiência e segurança do sistema **ParkLookUp**.

Em suma, o projecto, segue as ideias e planeamentos propostos, no entanto faltaram alguns pontos a melhorar:

- **A interface** - Sendo um dos pontos de contacto com o utilizador, deveria ter sido mais trabalhada, apesar de fazer todos os principais propositos do projecto.
- **A segurança** - Como um dos pontos fundamentais de um sistema com interface *web*, garantir que o sistema respeita o *Top 10 OWASP* [OWA24], seria um ponto a melhorar.
- **Os testes** - Seria a garantia que os utilizadores não encontrariam comportamentos estranhos no sistema, assim, uma bateria de testes mais exaustiva seria necessária.

Bibliografia

- [Ang24] AngularJs. Consultado em 2024/02/20. 2024 (citado na página 30).
- [AWS23] AWS. *API*. Consultado em 2023/11/17. 2023 (citado nas páginas 28, 30, 39).
- [C S20] Aswathy C S. «Low Cost Smart Parking System with IOT». Em: *International Journal for Research in Applied Science and Engineering Technology* 8 (fev. de 2020), pp. 273–278. DOI: 10.22214/ijraset.2020.2040 (citado nas páginas 3, 9).
- [Clo24] Cloudflare. Consultado em 2024/02/20. 2024 (citado nas páginas 30, 34).
- [Dig24] Digkey. *HC-SR04*. Consultado em 2024/02/20. 2024 (citado nas páginas 9, 10).
- [Emq24a] Emqx. *Broker*. Consultado em 2024/02/20. 2024 (citado nas páginas 30, 36).
- [Emq24b] Emqx. *MQTT*. Consultado em 2024/02/20. 2024 (citado nas páginas 28, 30, 36).
- [Esp24] Espressif. *ESP32*. Consultado em 2024/02/20. 2024 (citado nas páginas 11, 12, 31).
- [Fas24] FastApi. Consultado em 2024/02/20. 2024 (citado na página 30).
- [Gee24a] Geeksforgeeks. Consultado em 2024/02/20. 2024 (citado na página 13).
- [Gee24b] Geeksforgeeks. Consultado em 2024/02/20. 2024 (citado nas páginas 15, 16, 39).
- [Gee24c] Geeksforgeeks. Consultado em 2024/02/20. 2024 (citado na página 28).
- [Get24] Getresponse. Consultado em 2024/02/20. 2024 (citado nas páginas 13, 14).
- [Goo24a] Google. *Firebase*. Consultado em 2024/02/20. 2024 (citado nas páginas 11, 12).
- [Goo24b] Google. *Vision*. Consultado em 2024/02/20. 2024 (citado nas páginas 9, 10).
- [IBM24] IBM. Consultado em 2024/02/20. 2024 (citado nas páginas 28, 31).
- [Int24] Interaction. *Cognitive*. Consultado em 2024/02/20. 2024 (citado na página 41).
- [Mic24] Microsoft. *Cloud*. Consultado em 2024/02/20. 2024 (citado nas páginas 5, 7, 9, 10).

- [Mon24a] MongoDB. Consultado em 2024/02/20. 2024 (citado na página 26).
- [Mon24b] MongoDB. Consultado em 2024/02/20. 2024 (citado na página 30).
- [Mos24] Mosquitto. *Mosquitto*. Consultado em 2024/02/20. 2024 (citado na página 30).
- [Nat23] NatGeo. *GPS*. Consultado em 2023/11/17. 2023 (citado nas páginas 5, 8, 22).
- [Nor24] Nordvpn. *Sha256*. Consultado em 2024/02/20. 2024 (citado na página 39).
- [OWA24] OWASP. *OWASP*. Consultado em 2024/02/20. 2024 (citado na página 44).
- [Par23a] Parkopedia Parking. *Parkopedia Parking*. Consultado em 2023/11/17. 2023 (citado nas páginas 3, 4).
- [Par23b] Parkopedia Parking. *Parkopedia Parking Contextos de Utilização*. Consultado em 2023/11/17. 2023 (citado na página 4).
- [Par23c] Parkopedia Parking. *Parkopedia Parking Interfaces Gráficas*. Consultado em 2023/11/17. 2023 (citado na página 5).
- [Par23d] Parkwhiz. *Parkwhiz*. Consultado em 2023/11/17. 2023 (citado nas páginas 3, 6).
- [Par23e] Parkwhiz. *Parkwhiz Interfaces Gráficas*. Consultado em 2023/11/17. 2023 (citado na página 8).
- [Par23f] Parkwhiz. *Parkwhiz Tarefas Executadas pelos Utilizadores*. Consultado em 2023/11/17. 2023 (citado na página 7).
- [Pey+21] Md Peyal et al. «IoT Based Cost Effective Car Parking Management for Urban Area». Em: set. de 2021, pp. 70–75. DOI: 10.1109/ISAMSR53229.2021.9567826 (citado nas páginas 3, 11).
- [Pyt24] Python. *Python*. Consultado em 2024/02/20. 2024 (citado na página 30).
- [Ras24a] Raspberrypi. Consultado em 2024/02/20. 2024 (citado nas páginas 9, 10, 30).
- [Ras24b] Raspberrypi. *RBPiCamera*. Consultado em 2024/02/20. 2024 (citado nas páginas 9, 30).
- [Rea24] Realpars. *ServoMotor*. Consultado em 2024/02/20. 2024 (citado nas páginas 11, 31).
- [Red24] Redhat. *Rest*. Consultado em 2024/02/20. 2024 (citado na página 34).
- [Ren24] Render. Consultado em 2024/02/20. 2024 (citado na página 30).
- [Swa24] Swagger. *Bearer*. Consultado em 2024/02/20. 2024 (citado na página 39).
- [Tab24] Tableau. Consultado em 2024/02/20. 2024 (citado na página 15).
- [thi24] thingweb. *NodeWot*. Consultado em 2024/02/20. 2024 (citado na página 31).
- [W324] W3. Consultado em 2024/02/20. 2024 (citado nas páginas 29, 31, 32, 34–39).

- [Zip23] Zipitwireless. *IoT Sensors*. Consultado em 2023/11/17. 2023 (citado nas páginas 5, 8, 28, 31).