NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Simulação de Custos de Manutenção Automóvel

Pedro Daniel Ribeiro da Silva Valério

Projeto apresentado como requisito parcial para obtenção do grau de Mestre em Gestão de Informação

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

SIMULAÇÃO DE CUSTOS DE MANUTENÇÃO AUTOMÓVEL

por

Pedro Daniel Ribeiro da Silva Valério

Projeto apresentado como requisito parcial para a obtenção do grau de Mestre em Gestão de Informação, Especialização em Gestão de Sistemas e Tecnologias de Informação

Orientador: Vitor Duarte dos Santos

Novembro 2021

RESUMO

Desde que o automóvel foi inventado e lançado no mercado no século XIX que todos os anos são lançados novos modelos com novas funcionalidades e evoluções. Apesar de ser uma frota em constante renovação esta também precisa de ser mantida em boas condições de circulação, quer seja através de intervenções de rotina (revisões) ou mesmo reparação de avarias. Estas operações, reparações e revisões, independentemente de existirem períodos de garantia providenciados pelas marcas com diversas intervenções de manutenção incluídas, acabam por ter custos associados.

Apesar de muitas pessoas estarem acreditadas para conduzir, algumas possuindo viatura própria, poucas são aquelas que consideram ter os conhecimentos técnicos necessários para realizar a manutenção do seu carro, preferindo para isso recorrer a oficinas e técnicos especializados. Como referido anteriormente, estas operações têm custos associados, quer seja em peças, mão de obra ou em fluidos (óleo de motor e de caixa de velocidades, líquido de refrigeração). Dependendo da operação a realizar e da oficina, existe a possibilidade de alguns valores já estarem tabelados para o custo da mesma. Noutras situações é apresentado ao consumidor uma estimativa de gastos que este, por falta de conhecimento técnico, não tem capacidade de avaliar. Uma vez que estes custos estão suscetíveis a variações, seria útil para o cliente ter uma ferramenta tecnológica que lhe permitisse avaliar os orçamentos apresentados, evitando surpresas menos agradáveis no momento de pagar.

Assim, pretende-se desenvolver uma aplicação, suportada por uma Web API, que permita a qualquer pessoa, independentemente dos seus conhecimentos de mecânica, verificar e comparar o valor orçamentado aquando de qualquer intervenção mecânica.

De modo a realizar este projeto terá de ser estudado o enquadramento concetual associado às áreas da mecânica automóvel, bem como o enquadramento concetual associado ao desenvolvimento de uma Web API multiplataforma. Uma vez aprofundados estes conhecimentos, será feito o plano e desenho da arquitetura do sistema que servirá depois como guia para a implementação do sistema e respetivos testes.

PALAVRAS-CHAVE

Mecânica Automóvel; Orçamentação; Engenharia de Software; Simuladores

ÍNDICE

1. INTRODUÇÃO 1

1.1. Enquadramento e Definição do Problema 1
1.2. Objetivos do Projeto 3
1.3. Importância e relevância do Projeto 3

2. PLANO DE TRABALHO 5

2.1. Fases do Projeto / Etapas 5
2.2. Ferramentas / Recursos 5
2.3. Cronograma 7

3. ENQUADRAMENTO TEÓRICO 9

3.1. Mecânica Automóvel 9
3.1.1. Conceitos 9
3.1.2. Categorias de Operações Mecânicas 10
3.1.3. Diagnóstico de avarias 10
3.1.4. Orçamentação das reparações 11
3.2. Desenvolvimento de Software 11
3.2.1. Conceitos gerais 11
3.2.2. Metodologias de desenvolvimento de software 12
3.2.2.1. *Linear Sequential Model* (Modelo em Cascata) 13
3.2.2.2. *Prototyping* (Modelo de Prototipagem) 14
3.2.2.3. Modelo em Espiral 15
3.2.2.4. Metodologias Ágeis 16
3.2.3. Tipo de software 18
3.2.4. Tecnologias de desenvolvimento Web 18

4. DESENVOLVIMENTO DO PROJETO 20

4.1. Metodologia de Desenvolvimento 20
4.2. Levantamento e Análise de Requisitos Funcionais 21
4.2.1. RF.01 – Identificação um veículo 21
4.2.1. RF.02 – Detalhes de veículo 22
4.2.2. RF.03 – Listagem de operações de um veículo 23
4.2.1. RF.04 – Listagem de peças das operações selecionadas 24
4.2.2. RF.05 – Listagem de peças de um veículo 24
4.2.1. RF. 06 – Resumo do formulário 25
4.2.2. RF. 07 – Editar detalhes de peças 26
4.2.3. RF. 08 – Gerar simulação de orçamento 26
4.1. Diagrama de Atividades 27
4.2. Desenho do Sistema 27
4.2.1. Diagramas de Classes 27
4.2.2. Modelo Lógico de Dados 29
4.3. Desenvolvimento do Sistema 31
4.3.1. Arquitetura 31
4.3.2. Modelo Físico de Dados 32
4.3.3. Estrutura da Aplicação 33

4.3.4. Programação do Sistema *33*
4.3.4.1. Identificação do veículo (página inicial) 36
4.3.4.2. Lista de operações 37
4.3.4.3. Lista de peças 38
4.3.4.4. Resumo 40
4.3.4.5. Opções de peça 41
4.3.4.6. Orçamento 42
4.3.5. Testes *46*
5. CONSIDERAÇÕES FINAIS 48
5.1. SÍNTESE 48
5.2. CONTRIBUIÇÕES DO PROJETO 48
5.3. LIMITAÇÕES E TRABALHO FUTURO 49
BIBLIOGRAFIA 50
ANEXOS 53

ÍNDICE DE FIGURAS

Figura 1 – Cronograma do plano de trabalhos 7
Figura 2 – Ficha de diagnóstico OBD II 11
Figura 3 – Modelo em Cascata 13
Figura 4 – Modelo de Prototipagem 14
Figura 5 – Modelo em Espiral 16
Figura 6 – Fluxograma da metedologia do projeto 20
Figura 7 – Dropdown de identificação do veículo a tratar (wireframe de exemplo) 22
Figura 8 – Detalhes do veículo (wireframe de exemplo) 23
Figura 9 – Listagem de operações (wireframe de exemplo) 23
Figura 10 – Listagem de peças de operações (wireframe de exemplo) 24
Figura 11 – Listagem de peças (wireframe de exemplo) 25
Figura 12 – Detalhes do veículo (wireframe de exemplo) 25
Figura 13 – Listagem de peças selecionadas (wireframe de exemplo) 26
Figura 14 – Detalhes de peça (wireframe de exemplo) 26
Figura 15 – Diagrama de atividades do sistema 27
Figura 16 – Diagrama de classes I (Infrastructure) 28
Figura 17 – Diagrama de classes II (Business) 29
Figura 18 – Diagrama de classes III (DBManager) 29
Figura 19 – Modelo lógico de dados 30
Figura 20 – Arquitetura do sistema 31
Figura 21 – Modelo físico de dados 32
Figura 22 – Estrutura da solução .NET 34
Figura 23 – Mapa de interação entre páginas 35
Figura 24 – Página inicial 36
Figura 25 – Página de seleção e operações 37
Figura 26 – Página de seleção de peças 39
Figura 27 – Página de resumo 41
Figura 28 – Página de opções de peça 42
Figura 29 – Página de resultado da simulação de orçamento 44

ÍNDICE DE TABELAS

Tabela 1 – Endpoint de listagem de veículos 37
Tabela 2 – Endpoind de detalhes de veículo 38
Tabela 3 – Endpoint de listagem de operações 38
Tabela 4 – Endpoint de listagem de peças 40
Tabela 5 – Endpoint de listagem de peças para determinadas operações 40
Tabela 6 – Endpoint de listagem de peças por código 42
Tabela 7 – Endpoint de orçamento 45
Tabela 8 – Testes da camada de serviço 47

LISTA DE SIGLAS E ABREVIATURAS

API	Application Programming Interface
BPMN	Business Process Model and Notation
CRUD	Create, Read, Update and Delete
DTO	Data Transfer Object
MSRP	Manufacturer's Suggested Retail Price
OBD	On-board diagnostics
OE	Original Equipment
OEM	Original Equipment Manufacturer
RF	Requisito Funcional
SDLC	Software Development Life Cycle
UML	Unified Modeling Language

1. INTRODUÇÃO

1.1. ENQUADRAMENTO E DEFINIÇÃO DO PROBLEMA

Desde que foi inventado e patenteado no século XIX, por Karl Benz, que o automóvel tem vindo a sofrer diversas evoluções (Daimler, 2019), tendo-se tornado indispensável no quotidiano das pessoas. O automóvel começou por ser pouco mais que uma carroça acoplada de um motor de combustão, cuja utilização permitia deslocações mais rápidas comparativamente à solução existente até então e sem recurso a animais. Desde aí que muito se alterou. O foco deixou de ser exclusivamente a de ser um potenciador e facilitador da deslocação entre dois locais, tendo havido um reforço no conforto e segurança dos seus ocupantes. Passaram a existir automóveis específicos para diferentes números de ocupantes, mais citadinos, para grandes viagens ou até mais desportivos, o que levou à criação de diversos segmentos dentro daquilo que é considerado hoje em dia um carro. Um dos fatores que contribuiu grandemente para o desenvolvimento do automóvel foi a existência de desportos motorizados. Nestes, em muitas situações as próprias marcas chegam a estar envolvidas nas equipas de competição e, sendo estas uma boa mostra daquilo que cada marca é capaz, estas chegam a investir largas quantias monetárias no desenvolvimento de diversos componentes que posteriormente chegam aos veículos de produção. (Mercedes-Benz, n.d.; Renault, n.d.)

Ao longo deste último século a produção automóvel para uso pessoal tem vindo progressivamente a aumentar, sendo neste momento fabricadas mais de 90 milhões de unidades anualmente (Thomas & Maine, 2019), tendo já sido produzido um total até hoje em todo o mundo de mais de 1.8 milhares de milhões de veículos, quer seja para uso pessoal ou comercial (Bureau of Transportation Statistics, 2019). Na mesma linha, também o número de vendas de automóveis apresenta uma tendência geral de crescimento anual, tendo sido vendidos em todo o mundo mais de 65 milhões todos os anos desde 2014, e mais de 220 mil só no ano de 2019 em Portugal (OICA, 2019).

Um dos fatores que tem contribuído para este crescimento de mercado é exatamente a forma como o automóvel evoluiu (History.com Editors, 2010). A evolução tecnológica proveniente dos desportos motorizados e não só tornou os carros cada vez mais seguros e de fácil utilização, o que contribuiu para uma melhor aceitação pelas massas. Por outro lado, os diversos componentes que fazem parte de um carro, têm-se tornado cada vez mais complexos, sendo que alguns, que inicialmente eram apenas constituídos por um pequeno número de peças, chegam hoje em dia a ser compostos por milhares delas. Outra vantagem proporcionada pelo desenvolvimento tecnológico é o facto de o custo de produção de cada componente se ter tornado bastante mais reduzido, o que, nalgumas situações, leva a que seja preferível a substituição à reparação do mesmo (Côrtes, 2008).

Apesar desta maior complexidade dos componentes, a qualidade de produção automóvel tem vindo a melhorar, o que leva a que haja menos reparações a serem efetuadas dentro do período de garantia (Kachadourian, 2005). Isto tem levado a que, apesar de existirem períodos de garantia obrigatórios por lei (ACP, n.d.; Legislação, 2008), nos quais muitos destes custos estão normalmente cobertos pelas marcas, muitas chegam a oferecer extensões destes períodos (Rocha, 2017), por vezes de forma gratuita, como parte das suas estratégias de marketing, tentando assim demonstrar a confiança que têm na qualidade do produto que vendem (Afsahi & Shafiee, 2020).

A existência de viaturas pessoais acarreta diversos custos sendo que, deste ponto em diante, o principal foco recairá sobre as despesas de manutenção. Consideram-se despesas de manutenção todos os gastos necessários para manter uma viatura em circulação e dentro dos padrões de segurança exigidos. Nestes gastos inclui-se reparação de avarias, substituição de peças de desgaste, troca de fluidos (óleo de motor e de caixa de velocidades, líquido de refrigeração, entre outros) e toda a mão de obra associada a estas operações.

Normalmente, aquando do processo de compra de um carro, uma pessoa tenta estimar os custos que irá ter com o mesmo ao longo do período em que se estima que este estará na sua posse. Se por um lado existem valores diretos, como o custo do veículo, ou fáceis de estimar por estarem tabelados, como é o caso dos impostos e seguro (Impostosobreveiculos.info, 2019), por outro lado o valor de toda a manutenção mecânica já não é de cálculo fácil. Isto ocorre uma vez que, para se efetuar este cálculo, têm de ser considerados dois tipos diferentes de intervenções. Por um lado, existem as chamadas intervenções de revisão periódica e por outro, as reparações necessárias devido a avarias.

Uma revisão periódica consiste numa avaliação frequente do estado e desgaste dos diversos subsistemas, substituição de componentes já pré-estabelecidos e fluidos e, caso alguma anomalia ou desgaste excessivo sejam detetados durante o processo de revisão, é recomendado ao dono do automóvel que se proceda à troca ou reparação dos mesmos de forma preventiva. Já uma reparação por avaria é isso mesmo, a troca ou reparação de componentes danificados durante o uso do automóvel que, no seu estado atual, não possibilitam a utilização do veículo dentro dos padrões exigidos, ou seja, acabam por ser intervenções de remediação.

Estas operações acabam por ter custos associados, quer seja em peças, mão de obra ou em fluidos. Dependendo da operação a realizar e da oficina, poderão ou não existir valores tabelados para o custo da mesma. Estes mesmos custos estão suscetíveis a variações. Uma determinada peça ou fluido, dependendo da marca que a fabrica ou da empresa que a comercializa, pode ter uma grande variação no preço. A isto há que somar o valor da mão de obra que, apesar de, para a mesma

operação, em número de horas necessárias não dever ter uma grande variação entre oficinas, o valor a pagar por hora, este sim, depende do local onde é prestado o serviço.

Por outro lado, a reparação automóvel não está limitada a ser realizada em oficinas, existindo a possibilidade de cada um reparar a sua viatura. Porém, apesar de, de acordo com um estudo (Autotrader, 2015), 72% das pessoas que possuem um automóvel considerar que efetuam elas próprias a manutenção do seu veículo, a verdade é que as operações que realmente foram incluídas no questionário, são de certa forma simples. No que respeita a operações mais complexas, a manutenção passa a ser algo pouco aprazível devido ao conhecimento técnico necessário para as realizar. Este conhecimento técnico não só é importante no momento de efetuar as operações como também no momento de avaliar a necessidade e custo das mesmas (Andaleeb & Basu, 1994; Darley & Luethge, 2019) pelo que seria muito útil para o consumidor leigo que este tivesse disponível uma ferramenta tecnológica que lhe permitisse avaliar os orçamentos apresentados.

1.2. OBJETIVOS DO PROJETO

O objetivo do projeto é o desenvolvimento de uma aplicação, suportada por uma Web API, que permita a qualquer pessoa, independentemente dos seus conhecimentos de mecânica, verificar e comparar o valor orçamentado aquando de qualquer intervenção mecânica.

Para atingir este objetivo são definidos os seguintes objetivos intermédios:

- Estudar o enquadramento concetual associado às áreas da mecânica automóvel
- Estudar o enquadramento concetual associado ao desenvolvimento de uma Web API multiplataforma
- Fazer o plano e desenho da arquitetura do sistema
- Desenvolver o sistema
- Testar o desempenho do sistema

1.3. IMPORTÂNCIA E RELEVÂNCIA DO PROJETO

Este projeto ganha importância uma vez que resulta de uma necessidade direta que existe por parte de diversos donos de automóveis, sem conhecimentos de mecânica, em conseguirem avaliar os custos associados às diversas operações de manutenção e reparação das suas viaturas. Assim, este

projeto pretende colmatar essa mesma lacuna, fornecendo uma ferramenta que permita fazer essa análise de forma simples e rápida.

Recorrendo ao sistema proposto, os utilizadores ganham a capacidade de melhor analisar planos de manutenção/reparação e respetivos orçamentos, contribuindo para uma tomada de decisão mais informada/consciente e, possivelmente, conduzindo a uma diminuição de gastos.

Do ponto de vista das oficinas e respetivos mecânicos este projeto poderá também vir a servir como referência no momento de realizar orçamentos, permitindo-lhes consultar um valor de referência. Por outro lado, no caso de mecânicos não especializados em determinada marca ou modelo, este projeto almeja conseguir providenciar, num único sítio, acesso a um conjunto de informações que lhes permitam melhor realizar o seu trabalho.

Assim, uma vez que o projeto tanto se destina a prestadores de serviços como aos consumidores dos mesmos, este poderá também vir a ter impacto económico no mercado estabelecendo valores de referência para determinadas operações. Este impacto poder-se-á vir a refletir na relação entre as duas partes, uma vez que a falta de conhecimento do lado dos consumidores e a existência de diferentes valores para a mesma operação estabelecidos pelos prestadores de serviços, deixavam algum espaço para a desconfianças entre ambos.

2. PLANO DE TRABALHO

2.1. FASES DO PROJETO / ETAPAS

Este projeto encontra-se dividido nas seguintes fases:

- Identificação e Estruturação do Problema

Fase inicial do projeto, na qual é identificado e estruturado o problema a abordar.

- Estudo Teórico

Nesta fase é feito um estudo teórico aprofundado das áreas associadas ao projeto e ao problema a abordar, permitindo assim consolidar bases para o desenvolvimento do mesmo.

- Planeamento

Esta fase tem o propósito de planear, organizar e definir metas do desenvolvimento do projeto.

- Desenvolvimento do Projeto

É nesta fase que é executada toda a componente prática do projeto (desenvolvimento de código).

- Conclusões

Por fim é feita uma análise do trabalho realizado, apurando resultados e limitações e tirando as devidas conclusões.

2.2. FERRAMENTAS / RECURSOS

- **UML**

Unified Modeling Language (UML) é uma linguagem gráfica utilizada na representação de peças de software (UML, 2005).

Neste trabalho recorre-se à notação UML para representar os diagramas de classes, de atividades e relacional do sistema desenvolvido.

  - Diagrama de atividades: corresponde a uma representação simplificada do sistema através da qual é possível identificar as diferentes ações existentes neste, bem como a sequência e o modo como estas interagem entre si.

  - Diagrama de classes: permite identificar as classes programáticas existentes no sistema e as suas propriedades e de que forma é que estas se relacionam com outras.

- Diagrama Relacional: com o objetivo de se obter uma visão ampla da base de dados, este diagrama permite entender as relações existentes entre as diversas entidades, assim como os atributos que as compõem.

O diagrama de atividades irá ser implementado recorrendo à ferramenta Visio. Para obter o diagrama de classes irá utilizar-se a ferramenta de geração de diagrama de classes do Visual Studio, que gera automaticamente o modelo a partir do código desenvolvido, permitindo assim visualizar as associações e dependências das classes do projeto. Já para obter o diagrama relacional vai recorrer-se à função de *reverse engineering* da ferramenta PowerDesigner que converte os *scripts* de criação da base de dados no respetivo modelo.

- **BPMN**

O *Business Process Model and Notation* (BPMN) é uma notação que permite a descrição *standardizada* dos processos de uma empresa. (White & Miers, 2008)

Um processo representado através da anotação BPMN traduz-se num fluxograma, aproximado a um diagrama de atividades UML, com representações simples e objetivas de cada evento, atividade e respetivas relações, facilitando assim a compreensão destes por qualquer pessoa, independentemente do seu nível de conhecimento técnico.

Neste trabalho de projeto recorre-se à notação BPMN para representar o fluxo de utilização do sistema de informação desenvolvido, bem como o processo de obtenção e tratamento dos dados disponibilizados ao utilizador.

- **SQL**

SQL é uma linguagem de programação utilizada no desenvolvimento e gestão de bases de dados relacionais, com características padronizadas de acesso e comunicação com a Base de dados. Através de uma base de instruções CRUD - *insert, select, update* e *delete* – é possível construir e efetuar operações sobre um modelo relacional. (Date, 1987)

A construção da base de dados relacional a utilizar neste projeto irá ser realizada recorrendo ao Microsoft SQL Server Management Studio. Este programa permite a escrita e execução dos scripts SQL, bem como pré-visualizar as bases de dados existentes e a forma como estas estão organizadas.

- **C#**

C# é uma linguagem de programação orientada a objetos e orientada e a componentes desenvolvida pela Microsoft. Esta linguagem faz parte da *framework* .NET que, a partir da versão .NET 5, reúne em

si, de forma unificada, diversas ferramentas de desenvolvimento de aplicações. Será esta a versão da *framework* utilizada ao longo do projeto. (Microsoft, 2021)

É com recurso a esta linguagem de programação que é desenvolvida toda a componente de *backend* do projeto. Já a componente e *frontend* irá ser desenvolvida através de Razor Pages, também presentes no .NET. Para tal irá recorrer-se à ferramenta Visual Studio para servir como ambiente de desenvolvimento. Apesar de existirem versões pagas desta ferramenta (*Professional* e *Enterprise*) a versão usada ao longo deste trabalho (*Comunity*) já tem todos os recursos necessários e não tem custos associados. (Microsoft, n.d.)

2.3. CRONOGRAMA

Nesta secção é apresentado um cronograma correspondente ao plano de trabalho a realizar. Neste cronograma estão detalhadas todas as atividades do projeto, bem como o tempo previsto para a sua realização e respetiva calendarização.

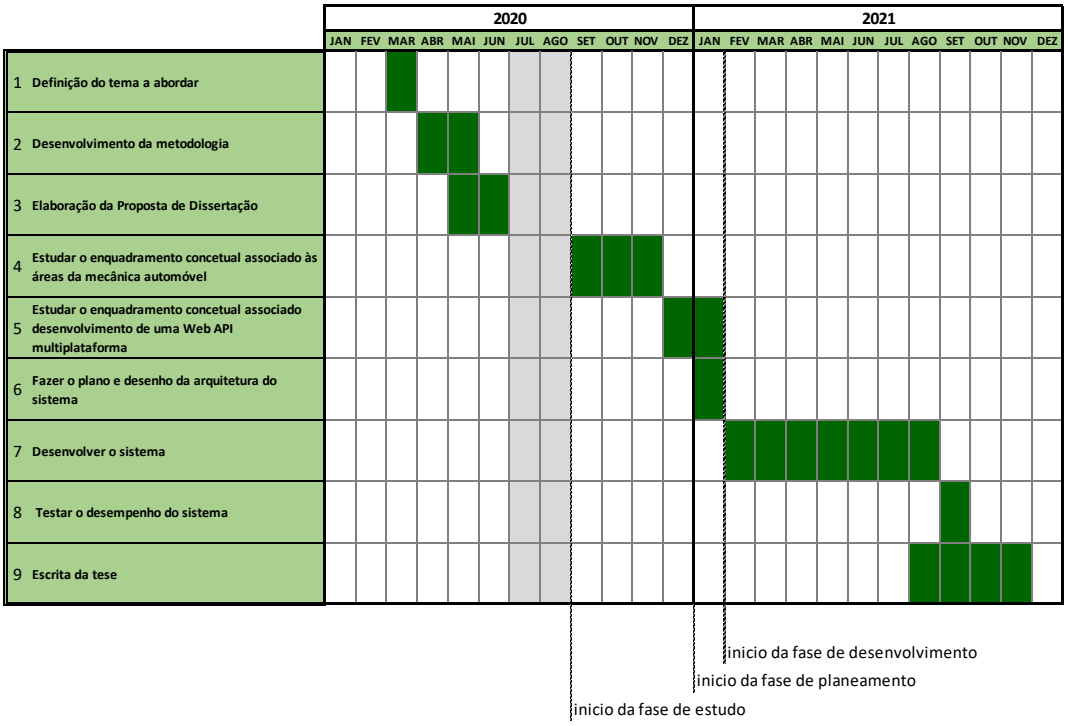

Figura 1 – Cronograma do plano de trabalhos

Neste projeto, tendo em conta a sua natureza e uma vez que terá todos os requisitos bem definidos antes do início da implementação, irá ser aplicado um modelo de desenvolvimento *Linear Sequential Model*.

3. ENQUADRAMENTO TEÓRICO

3.1. MECÂNICA AUTOMÓVEL

3.1.1. Conceitos

Mecânica, por um lado, e enquanto área da ciência, refere-se ao estudo das forças e movimentos (Goldstein et al., 2002), por outro é vulgarmente usado o termo para referir a aplicação deste ramo ao design, construção, operações e manutenção de máquinas e ferramentas (Kennedy, 1886). Assim, "mecânica automóvel" pode ser descrita como a ciência que se foca no estudo, manutenção, reparação e outras atividades relacionadas com veículos ligeiros (automóvel). A mecânica automóvel, normalmente, é referente à componente de *hardware*, elétrica e mesmo de *software* que compõem estes veículos (Denton, 2008). Uma vez que cada modelo automóvel tem as suas especificidades e sendo a mecânica uma área de estudo com diversas componentes, acaba por ser recorrente que um técnico ou uma oficina sejam especializados apenas numa área, tal como acontece nos casos das oficinas de pintura, oficinas especializadas em motores ou outro tipo de componentes, ou mesmo num determinado modelo (mais para casos premium) ou numa marca específica, como é o caso das oficinas oficiais e respetivos técnicos.

Da mesma forma que a tecnologia avança, também a mecânica automóvel se vê envolvida num processo de constante aperfeiçoamento. Diversos fatores afetam esta busca pela melhoria, quer sejam internos às empresas, de modo a alcançar objetivos económicos ou apresentar algo que os diferencie da concorrência, ou mesmo por fatores externos, como é o caso das normas de segurança e de emissões de gases que lhes são impostas, o que leva a um constante aumento de complexidade nos sistemas automóveis bem como à quantidade crescente de componentes eletrónicos em cada veiculo (Stjepandić et al., 2015).

Uma das mudanças que veio alterar radicalmente o paradigma automóvel e a forma como os carros se deslocam foi a introdução da eletricidade como única fonte energética na locomoção. Uma vez que este tipo de veículos não possui motores de combustão interna, muitas das operações normalmente executadas em viaturas com esta variante acabam por não existir, sendo também, por outro lado, necessárias determinadas precauções aquando de operações mecânicas, pelo que a nível de reparações e manutenção são abordados de forma diferente (Weldon et al., 2018). Por esta razão, veículos movidos a energia elétrica não serão considerados no estudo deste trabalho, podendo sê-lo, posteriormente, em desenvolvimentos deste ou em futuros projetos.

3.1.2. Categorias de Operações Mecânicas

Ao longo da sua vida útil cada veículo tem certas operações já determinadas pelo seu fabricante de modo a prolongar a sua vida útil, evitar avarias e garantir condições de segurança na sua circulação (Denton, 2008). Estas operações podem implicar a troca de peças, componentes ou fluidos, bem como a atualização de software. Em casos extraordinários em que são detetadas irregularidades no modelo quando já existem unidades vendidas pode mesmo dar-se o caso dessas unidades serem "chamadas" para proceder à troca do componente defeituoso por um de outro lote ou mesmo por uma versão melhorada (*recall*) (HOFFER et al., 1994).

No caso das operações de rotina pré-programadas, apesar dos fabricantes definirem quais as peças a trocar, não existe a obrigação de que estas sejam substituídas por peças da mesma marca, sendo que deve ser sempre respeitado o código descritivo do componente (Federal Trade Commission, n.d.; Insurance Information Institute, n.d.). Dependendo do local onde forem realizadas estas operações, ou das preferências do dono do carro, podem ou não ser usadas peças de fabricantes iguais às utilizadas inicialmente pelo construtor automóvel, com base nas quais este efetuou os seus testes, medições e cálculos de modo a determinar o intervalo de troca de cada componente. Assim, dependendo das características do material de reposição, a sua qualidade poderá ser variável, o que, de forma idêntica, levará, ou não, a potenciar mais avarias ou falhas ao longo do próximo intervalo entre operações de revisão. O mesmo se passa no caso dos fluidos, em que o líquido de reposição deve ter características semelhantes ao que estava anteriormente na viatura, pressupondo que este era o recomendado pela marca, de modo a garantir o seu bom funcionamento (Mobil, n.d.). Qualquer alteração de características afetará de maneira diferente a prestação da viatura (Castrol, n.d.; Total, n.d.).

Para além das operações mecânicas de manutenção, os veículos podem necessitar de ser intervencionados em caso de acidente ou de avarias espontâneas. Nestas situações, um dos desafios que se coloca no processo de reparação é o diagnóstico do problema bem como a sua causa.

3.1.3. Diagnóstico de avarias

A partir do momento em que se assume que o veículo tem uma avaria será necessário realizar um diagnóstico da mesma para se proceder corretamente à sua reparação. Em veículos mais recentes que possuam uma ficha de diagnóstico o primeiro passo nessa verificação será fazer um scan de software recorrendo a uma ferramenta de diagnóstico, também conhecida como leitor de códigos. De acordo com os problemas detetados, este leitor deverá apresentar uma série de códigos que,

contrariamente ao que normalmente se possa pensar, não correspondem diretamente a um determinado problema, mas sim indica que componentes se encontram fora do estabelecido, o que ajuda a restringir a "área de busca". (Baltusis, 2004)

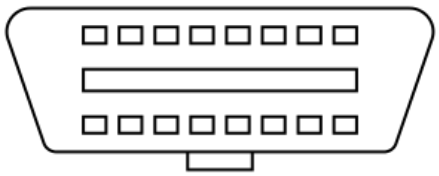

Figura 2 – Ficha de diagnóstico OBD II
(Xoneca, 2015)

3.1.4. Orçamentação das reparações

Uma vez identificado o problema, o próximo passo da parte da oficina/reparador para com o cliente passa por produzir um orçamento detalhado com o plano de reparações e valores. Este orçamento é calculado tendo em conta o tempo estimado para cada tarefa a realizar, a taxa de mão de obra (isto é a quantidade de dinheiro cobrada por hora de trabalho), o preço das peças necessárias e, por vezes, é aplicada uma a matriz de preços da loja, como é o caso quando existe algum desconto para clientes regulares ou preço especial sempre que algumas reparações ou ações de manutenção forem feitas em conjunto, sempre de forma a proporcionar lucro. (Shyun, 2017)

Dos pontos acima identificados, o mais difícil de estimar é exatamente o tempo do serviço, uma vez que cada operação terá o seu tempo específico e que este pode variar entre veículos. De modo a auxiliar nesse cálculo, muitas oficinas usam guias temporais, como o ALLDATA por exemplo, que fornecem estimativas com uma margem já precavendo a eventualidade de algo não correr de acordo com o planeado. (ALLDATA, n.d.)

3.2. DESENVOLVIMENTO DE SOFTWARE

3.2.1. Conceitos gerais

"Desenvolvimento de software" é descrito como sendo um "conjunto de atividades de ciência da computação dedicadas ao processo de criação, conceção, *deploy* e suporte de software" (IBM Research, 2014). No fundo é um conceito que se refere a todo o processo desde que surge a ideia para uma peça de software até que esta se materialize e, em alguns casos, também todo o processo

de suporte após estar concluído, ou seja, "desenvolvimento de software" não descreve apenas o momento em que determinado produto é implementado recorrendo a linguagens de programação, mas também as fases que precedem esse momento bem como as seguintes.

3.2.2. Metodologias de desenvolvimento de software

Desde que uma peça de software é idealizada até que esta se encontre completamente desenvolvida, testada e em produção, e, caso seja necessário, com as respetivas operações de manutenção, esta passa por diversas etapas. Ao conjunto dessas etapas dá-se o nome de *lifecycle* do projeto (SDLC). O SDLC consiste num plano detalhado que estabelece e descreve as diversas fases de desenvolvimento de software, definindo também como estas interagem entre si de modo a maximizar a qualidade do software produzido, indo também ao encontro das expetativas do cliente. (Avison & Fitzgerald, 2003)

Partindo do princípio de que não há dois projetos iguais, muitos fatores os podem distinguir:

- projetos que, desde que são idealizados até à sua conclusão, podem demorar apenas alguns dias, enquanto outros se podem prolongar por meses ou até anos;
- projetos desenvolvidos apenas por uma pessoa e outros em que participa uma equipa ou até diversas equipas;
- projetos completamente desenvolvidos pela mesma empresa ou em que os seus componentes são divididos por várias;
- projetos de software com ou sem interface gráfica;
- …

Na realidade, a mesma fórmula não pode ser aplicável a todos os tipos de projeto nas suas diversas variantes. De acordo com as características deste deve aplicar-se o modelo de *lifecycle* que mais se adapta a essa situação (Ruparelia, 2010; Santos, 2019).

Alguns dos modelos mais conhecidos e utilizados são:

- *Linear Sequential Model* (Modelo em Cascata)*;*
- *Prototyping* (Modelo de Prototipagem)*;*
- Modelo em Espiral*;*

- *Agile.*

3.2.2.1. *Linear Sequential Model* (Modelo em Cascata)

Linear Sequential Model, também conhecido como modelo em Cascata, foi concebido em 1956 por Benington e modificado por Winston Royce em 1970 (Awad, 2005; Ruparelia, 2010).

Este modelo tem como principal característica uma abordagem sequencial e linear que obriga a concluir a fase corrente para se poder passar para a seguinte.

De acordo com o modelo sugerido por Benington, o modelo em cascata é constituído pelas fases de análise/avaliação, definição de requisitos, design e especificação de código, desenvolvimento, testes e por fim *deploy* e respetiva avaliação. Tal como se pode observar na Figura 3, todas estas fases são independentes e apenas têm início após a conclusão da anterior (Ruparelia, 2010).

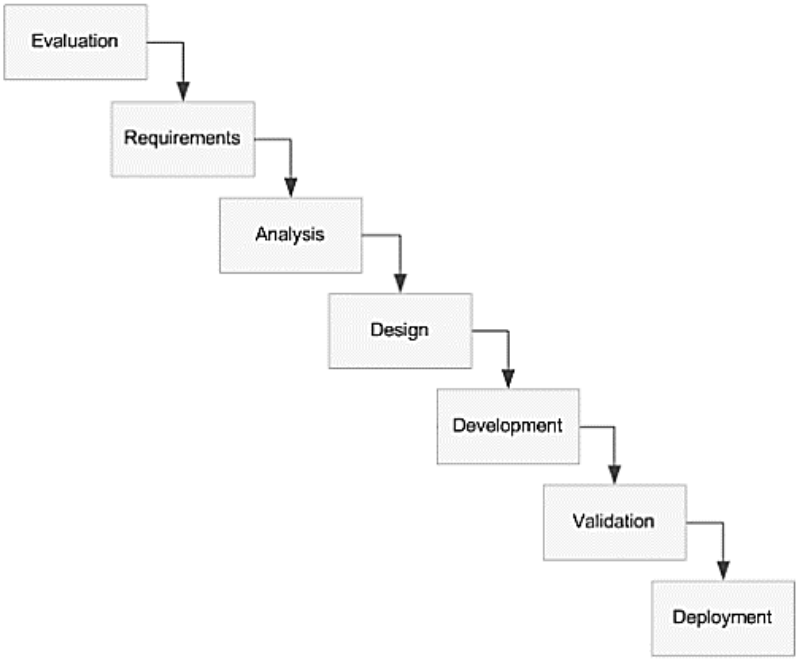

Figura 3 – Modelo em Cascata
(Ruparelia, 2010) - adaptado

No caso de projetos bem definidos e de pequena dimensão, este modelo tem vantagem por promover um rápido desenvolvimento e com baixa complexidade de gestão de projeto. O mesmo não acontece em projetos de média e grande dimensão onde os requisitos estão em constante mutação/evolução. Nestes casos é normal a existência de alterações ao longo do seu desenvolvimento, sendo difícil por parte do cliente especificar todos os requisitos numa fase inicial.

É de notar que, no caso de projetos reais nem sempre é seguido um fluxo sequencial, sendo também difícil por parte do cliente expor de forma clara os requisitos pretendidos, o que posteriormente se traduz em atrasos no desenvolvimento do projeto. Esta situação tem a agravante de que os requisitos podem ainda sofrer alterações ao longo do projeto.

Assim, podemos concluir que o modelo Linear Sequential Model é mais apropriado para casos em que já se têm todos os requisitos definidos no início do projeto, como por exemplo para base de dados relacionais, backend, ou outro tipo de peças de software que providenciem serviços a outras (Ruparelia, 2010).

3.2.2.2. *Prototyping* (Modelo de Prototipagem)

O modelo de Prototipagem surge como evolução do modelo em Cascata referido acima. Este modelo tem características evolutivas no sentido em que, apesar de também estar dividido por fases independentes, estas são agora cíclicas, como demonstrado na Figura 4.

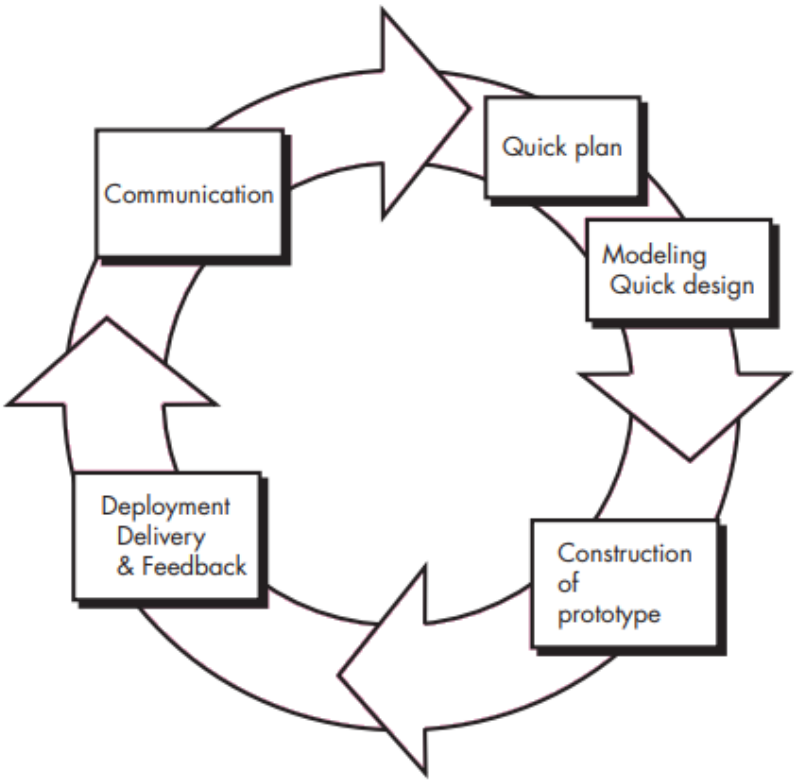

Figura 4 – Modelo de Prototipagem
(Pressman, 2014)

Este modelo é utilizado em situações em que o cliente não tem claros os requisitos do projeto a desenvolver, podendo assim ser definido um esboço inicial (protótipo) que permita ao cliente melhor compreender as suas necessidades e o que pretende do projeto (Pressman, 2014).

Por norma, o protótipo inicial serve de auxílio ao cliente no processo de identificação dos requisitos funcionais desejados, bem como na análise da sua viabilidade, mas permite também a deteção de erros, normais numa versão em bruto, permitindo assim, a sua correção em versões futuras do sistema (Avison & Fitzgerald, 2003; Pressman, 2014).

O modelo de prototipagem tem como ponto forte a compreensão do projeto e a sua viabilidade, mesmo em situações em que os requisitos não estão definidos à partida, uma vez que é possível através do protótipo ter um vislumbre do futuro do sistema. Por outro lado, o cliente pode incorrer em erro e considerar que o protótipo se trata do produto final, devendo-se, portanto, saber gerir expetativas

3.2.2.3. Modelo em Espiral

Proposto em 1986 por Barry Boehm, o modelo em espiral é um modelo evolutivo que combina a metodologia do modelo em cascata com as componentes iterativa e interativa da prototipagem (Pressman, 2014; Ruparelia, 2010).

Neste modelo o desenvolvimento de um sistema é visto como um processo contínuo e iterativo, representado por uma espiral, no qual são implementadas várias versões do sistema, cada vez mais completas e aprimoradas, até se chegar a um produto final.

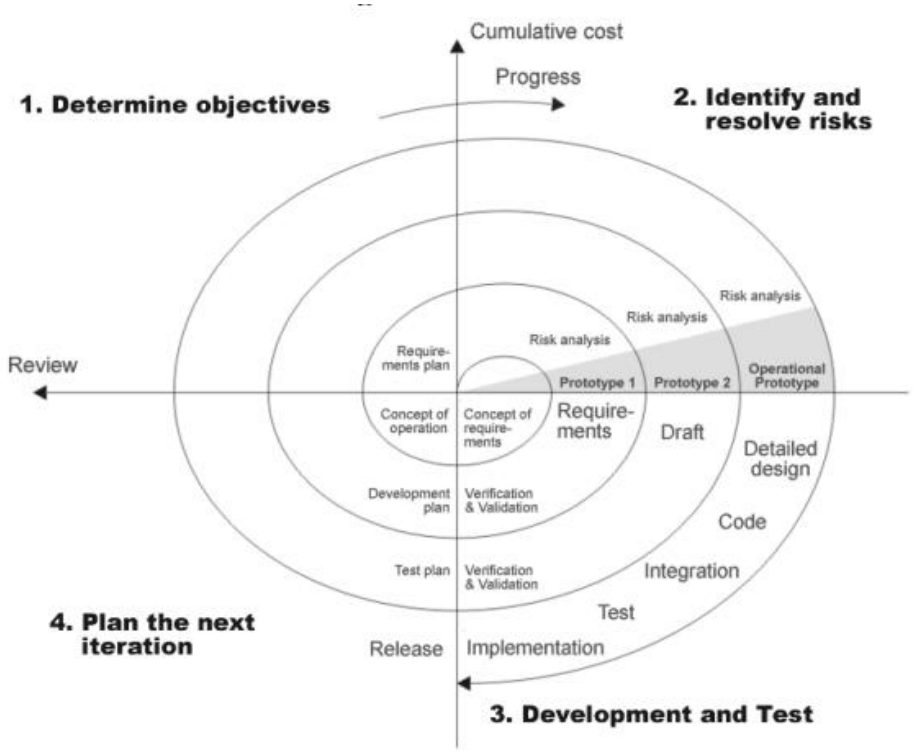

Figura 5 – Modelo em Espiral
(Ruparelia, 2010)

Tal como é apresentado na *Figura 5* cada ciclo deste modelo está dividido em quatro quadrantes: primeiro a determinação de objetivos, seguida da avaliação de riscos, o desenvolvimento (com os respetivos testes) e por fim o planeamento do ciclo seguinte. A cada volta da espiral é desenvolvido um protótipo. (Ruparelia, 2010)

Este modelo tem como ponto positivo a forma como aborda os riscos, permitindo que estes sejam mitigados em etapas posteriores do processo iterativo. Estas etapas também permitem que haja uma evolução sistemática e constante do sistema. Por outro lado, o modelo em espiral depende de uma gestão de projeto flexível e também de uma correta identificação dos riscos do ciclo seguinte.

3.2.2.4. Metodologias Ágeis

As metodologias Ágeis, ou conhecidas simplesmente por Agile, são, tal como o nome indica, um conjunto de métodos ágeis de desenvolvimento de projetos. Nestas metodologias alterações de âmbito de projeto e aumentos no número de requisitos são evitados dividindo o projeto global em

subprojetos de dimensão menor, cujo desenvolvimento é efetuado em curtos espaços de tempo (sprints), permitindo assim, que haja evoluções constantes e incrementais (Awad, 2005).

Estas metodologias assentam nos seguintes doze princípios (Beck et al., 2001):

1. Priorizar a satisfação do cliente por meio de entregas céleres e contínuas de *software*;
2. Aceitar mudanças de requisitos, mesmo no final do desenvolvimento. Os processos ágeis aproveitam a mudança para a vantagem competitiva do cliente;
3. Entregar *software* funcional com frequência, de algumas semanas a alguns meses, com preferência para a escala de tempo mais curta;
4. Cliente e equipa de desenvolvimento devem trabalhar juntos diariamente ao longo do projeto;
5. Construir projetos em torno de indivíduos motivados, dando-lhes ambiente, suporte e confiança necessários para a realização do seu trabalho;
6. O método mais eficiente e eficaz de transmitir informações para e dentro de uma equipa de desenvolvimento é a conversa face a face;
7. *Software* funcional é a principal medida de progresso;
8. Processos ágeis promovem o desenvolvimento sustentável. Todos os envolvidos devem ser capazes de manter um ritmo constante indefinidamente;
9. A atenção contínua à excelência técnica e uma solução bem definida aumentam a agilidade;
10. Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial;
11. As melhores arquiteturas, requisitos e designs surgem de equipas auto-organizadas;
12. A equipa reflete regularmente sobre como se tornar mais eficaz e ajusta o seu comportamento de acordo.

Algumas das metodologias Ágeis que mais se destacam são Scrum, Extreme Programming (XP), Lean Development (LD) e Joint Application Development (JAD) (Ruparelia, 2010). Um dos principais fatores que as destaca das metodologias tradicionais acima descritas é o facto das tarefas que as compõem não serem necessariamente isoladas, não sendo necessário fechar uma para iniciar outra.

Aplicar Agile a projetos de grande dimensão acarreta algumas limitações, nomeadamente por envolver a produção de pouca documentação durante a fase de desenvolvimento. Estas metodologias são mais indicadas para projetos que providenciem uma interface gráfica ao seu utilizador final.

3.2.3. Tipo de software

Existem diferentes tipos de *software* que, dependendo do propósito, podem ser agrupados em duas categorias diferentes: *software* de sistema e *software* de aplicação (*System Software* e *Application Software*). (Squareboat, n.d.)

Software de Sistema, na sua génese, é um intermediador, ou uma camada intermédia, entre o utilizador e o hardware, isto enquanto fornece uma base para o outro *software* trabalhar.

Software de aplicação está mais focado nas necessidades do utilizador final e, na maioria das vezes, ajuda o utilizador a completar tarefas específicas, como navegar online, preparar uma apresentação ou até mesmo conversas de vídeo. Um subtipo popular de software de aplicação são as aplicações Web. Uma das principais características que as distingue dos restantes tipos de aplicações é o facto de estas serem executadas normalmente através de um browser recorrendo a um servidor web, ao contrário de outras aplicações que funcionam localmente no Sistema Operativo. (Parahar, 2020)

3.2.4. Tecnologias de desenvolvimento Web

Uma aplicação web está normalmente dividida em duas partes: *frontend* e *backend* (IATE, 2018).

Frontend corresponde à interface do utilizador, ou seja, não é apenas o que este vê, mas sim tudo aquilo com que este interage diretamente. Esta componente é executada no terminal do utilizador, o que significa que está limitada às especificações e limitações do *software* que este utiliza e até mesmo hardware. É por esta razão que normalmente não vemos operações que exijam cálculos muito pesados a serem executadas no *frontend*. Caso isso aconteça é possível que a aplicação seja lenta (de forma percetível ao utilizador) na resposta ao pedido. Quem é responsável por executar estes cálculos é habitualmente o backend que, por estar a correr num servidor web, terá mais recursos disponíveis para proceder a este tipo de cálculos. É também no backend que, por intermedio de diversas camadas de abstração, é feito o acesso a dados. Uma aplicação web pode ser desenvolvida com estas duas partes separadas ou juntas (monólito).

A existência de camadas de abstração e desta separação entre *backend* e *frontend* levam à necessidade de recorrer a uma API para comunicar entre elas. API traduz-se literalmente por "interface de programação de aplicações" e faz exatamente o que o nome sugere, fornece uma interface que permite que peças de software independentes comuniquem entre si (RedHat, n.d.).

Existem diversas tecnologias e linguagens de programação que suportam cada uma das abordagens acima referidas (se não ambas), pelo que se deve escolher a mais apropriada para o projeto de acordo com as suas características, preferências do programador e exigências do projeto e respetivo cliente (Pierce, 2002).

4. DESENVOLVIMENTO DO PROJETO

4.1. METODOLOGIA DE DESENVOLVIMENTO

Tendo em conta as caraterísticas do projeto, do contexto em que este é realizado e considerando que a partir do momento em que se definem os requisitos estes se manterão ao longo do desenvolvimento, foi decidido que uma metodologia linear em cascata seria a mais apropriada para o desenvolvimento deste projeto.

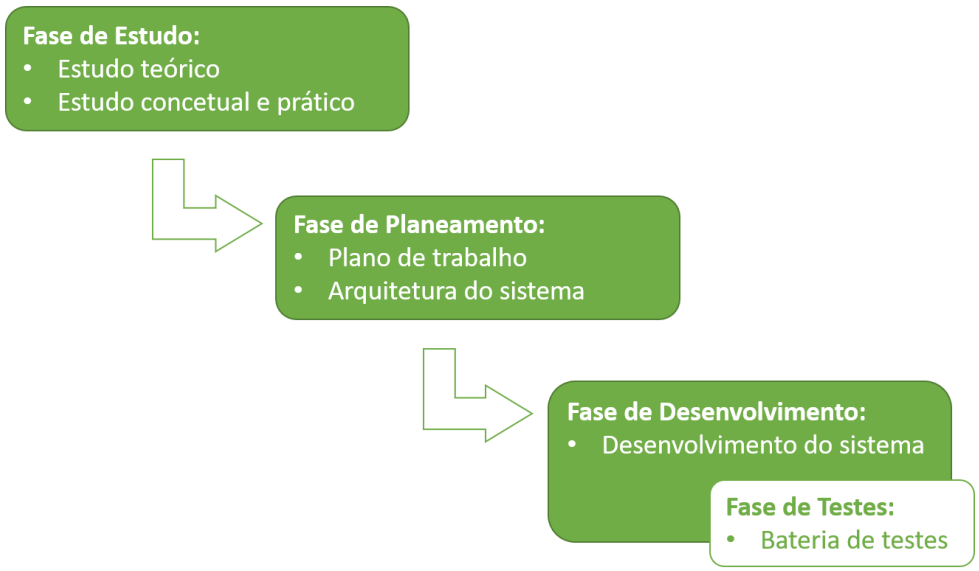

Figura 6 – Fluxograma da metedologia do projeto

Para a realização deste trabalho prático, em primeiro lugar será necessário realizar um estudo teórico que conduza a um melhor enquadramento conceptual associado às áreas da mecânica automóvel. Para a realização deste estudo deverão ser consultadas diversas fontes como livros, manuais, *papers* ou mesmo publicações *online*.

Está também previsto ser realizado um aprofundamento dos conhecimentos necessários para o desenvolvimento de uma Web API, nomeadamente em termos de normas e arquitetura de código, segurança e integração/alojamento da solução a implementar num serviço externo.

Posteriormente terá de ser realizado um plano do sistema proposto com a respetiva arquitetura. Neste plano será detalhada a composição do sistema, bem como as interações entre os diversos componentes do mesmo através de um diagrama UML. Fará também parte do plano um outro diagrama, este representativo do fluxo de todo o processo, recorrendo à notação BPMN.

Uma vez estudados os enquadramentos teóricos e práticos necessários, os conhecimentos adquiridos e/ou cimentados irão ser aplicados no desenvolvimento do sistema proposto, seguindo o plano e arquitetura previamente definidos.

Por fim, de modo a testar o desempenho e que o sistema implementado cumpre os requisitos propostos deverá ser realizada uma bateria de testes.

4.2. Levantamento e Análise de Requisitos Funcionais

O projeto foi decomposto e estruturado em diferentes requisitos funcionais. Estes requisitos funcionais, descritos abaixo, definem o *scope* do projeto. É com base nestes que é posteriormente realizado o trabalho de desenvolvimento de *software*.

- RF.01 – Identificação um veículo;
- RF.02 – Detalhes de veículo;
- RF.03 – Listagem de operações de um veículo;
- RF.04 – Listagem de peças das operações selecionadas;
- RF.05 – Listagem de peças de um veículo;
- RF.06 – Resumo de formulário;
- RF.07 – Editar detalhes de peças;
- RF.08 – Simulação de orçamento.

4.2.1. RF.01 – Identificação um veículo

De forma a identificar o veículo sobre o qual ira ser realizado o orçamento é necessário primeiro identificar a marca, modelo e submodelo que o caracterizam. Esta identificação deve ser efetuada por via de uma *dropdown*.

Em termos visuais o conteúdo da *dropdown* deverá encontrar-se agrupado por marca e respetivos modelos, tal como se pode observar na Figura 7, sendo apenas possível selecionar os submodelos/variantes. Uma vez que estes são diretamente dependentes dos agrupadores, ao selecionar um item da *dropdown* implicitamente estará também a selecionar-se a sua marca e modelo, tendo assim toda a informação necessária para identificar o veículo a tratar.

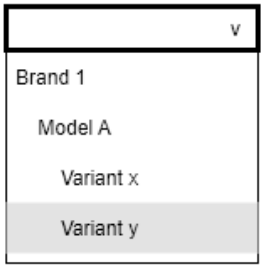

Figura 7 – Dropdown de identificação do veículo a tratar (wireframe de exemplo)

4.2.1. RF.02 – Detalhes de veículo

Após ser selecionado o veículo para o qual se está a efetuar a simulação de orçamento, deverá ser possível ver alguns detalhes do mesmo. Para tal, deverá existir e estar presente em múltiplas páginas um cartão, semelhante ao *wireframe* de exemplo da Figura 8, onde seja possível visualizar os seguintes dados do veículo:

- Marca;
- Modelo;
- Variante;
- Ano de lançamento;
- Capacidade do motor;
- Tipo de propulsão;
- Potência.

Vehicle Details	
Brand:	Brand 1
Model:	Model P
Variant:	Model P Sport
Release year:	2021
Engine capacity:	2300 cc
Propulsion type:	Gasoline
Power output:	303 kW

Figura 8 – Detalhes do veículo (wireframe de exemplo)

4.2.2. RF.03 – Listagem de operações de um veículo

Uma vez identificado o veículo sob o qual incidirá a simulação de orçamento deverá ser possível pesquisar e listar operações para o mesmo. Para tal, deverá existir um campo de pesquisa que, a partir do momento que esteja preenchido com pelo menos um caracter, deverá despoletar o preenchimento de uma tabela com os seguintes campos distribuídos por colunas:

- Operação;
- Tempo de Referência (em horas);

Para além das colunas acima indicadas deverá haver também uma coluna destinada à seleção do item. Abaixo, na Figura 9, apresenta-se um *wireframe* de exemplo da listagem pretendida.

Select Operations

Search...

Operation	Estimated Time
Operation A	1h11
Operation B	2h22
Operation C	3h33

Figura 9 – Listagem de operações (wireframe de exemplo)

O texto inserido no campo de pesquisa deverá refletir-se nos itens a apresentar na tabela filtrando pela coluna correspondente ao nome das operações.

Conforme forem selecionadas operações deverá ser compilada uma lista composta pelas peças necessárias para a realização destas.

4.2.1. RF.04 – Listagem de peças das operações selecionadas

Caso num passo inicial, correspondente ao requisito anterior, seja selecionada alguma operação para o veículo em análise, deverá posteriormente ser apresentado ao utilizador uma lista das peças que a estas correspondem. Estas peças já são consideradas como pré-selecionadas para efeitos da simulação, sendo, portanto, importante o utilizador ter esta informação presente. Assim, deverá existir um cartão no qual é apresentado o nome, código e quantidade para cada peça automaticamente selecionada.

Na eventualidade de duas operações requererem a mesma peça, esta só será apresentada uma vez neste resumo, assumindo a quantidade máxima das duas. Caso assim o desejar, o utilizador deverá ter hipótese, mais à frente, de retificar esta situação.

Operation Parts

Part	Code	Quantity
Part A	abcd123	1
Part B	efgh456	2
Part C	ijkl789	3

Figura 10 – Listagem de peças de operações (wireframe de exemplo)

4.2.2. RF.05 – Listagem de peças de um veículo

Uma vez identificado o veículo sob o qual incidirá a simulação de orçamento deverá ser possível pesquisar e listar peças para o mesmo. Para tal, deverá existir um campo de pesquisa que, a partir do momento que esteja preenchido com pelo menos um caracter, deverá despoletar o preenchimento de uma tabela com os seguintes campos distribuídos por colunas:

- Categoria;
- Nome;
- Código

Para além das colunas acima indicadas deverá haver também uma coluna destinada à seleção do item. Abaixo, na Figura 11, apresenta-se um *wireframe* de exemplo da listagem pretendida.

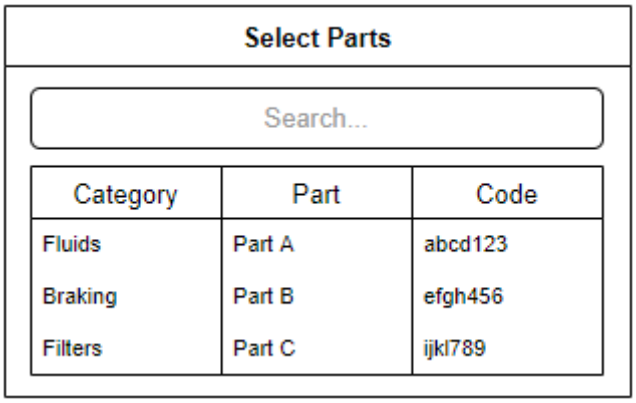

Figura 11 – Listagem de peças (wireframe de exemplo)

O texto inserido no campo de pesquisa deverá refletir-se nos itens a apresentar na tabela filtrando pela coluna correspondente ao nome das peças.

4.2.1. RF. 06 – Resumo do formulário

Antes do utilizador submeter os dados inseridos para a simulação de orçamento, deverá ter acesso a um resumo. Este resumo será composto por dois quadros: um com os detalhes do veículo e outro com as peças selecionadas.

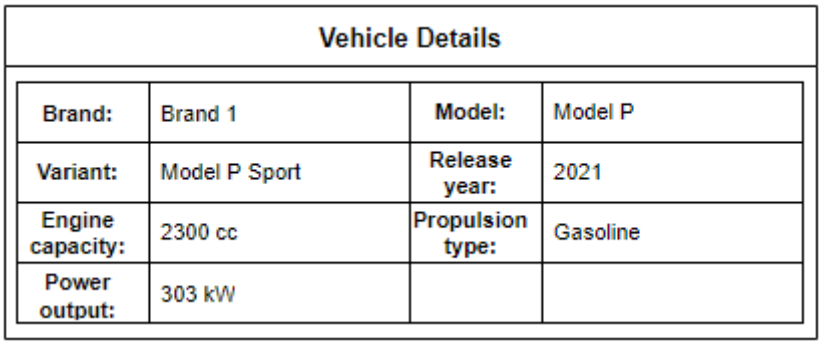

Figura 12 – Detalhes do veículo (wireframe de exemplo)

Parts

Part	Manufacturer	Code	Unit Price	Quantity	
Part A	xptoOils	abcd123	1,10 €	1	edit
Part B	SuperBrakes	efgh456	2,20 €	2	edit
Part C	-	ijkl789	-	3	edit

Figura 13 – Listagem de peças selecionadas (wireframe de exemplo)

O primeiro terá exatamente os mesmos dados do Requisito Funcional 02, apenas organizados de forma diferente. Já no resumo das peças deverá constar o seu nome, fabricante, código, preço unitário de referência e quantidade.

4.2.2. RF. 07 – Editar detalhes de peças

Uma vez que para cada peça poderão existir vários fabricantes, deverá ser possível escolher qual a considerar na simulação de orçamento. Para tal será necessário apresentar uma tabela com todas as hipóteses existentes, apresentando para cada uma o nome do fabricante, o preço de referência (MSRP) e se esta é a que vem equipada de fábrica (OE). Na mesma área deverá também ser possível definir a quantidade da peça em questão.

Part Options

Part C

Manufacturer	Reference Unit Price	OE	Quantity
CarParts	50 €	No	3
AirFiltersX	15 €	No	
SuperParts	45 €	Yes	

Figura 14 – Detalhes de peça (wireframe de exemplo)

4.2.3. RF. 08 – Gerar simulação de orçamento

O sistema deverá ser capaz de calcular um orçamento para as opções previamente selecionadas pelo utilizador. Para tal deverá ter em conta o veículo pré-selecionado, identificado através da sua marca, modelo e submodelo, bem como as peças selecionadas, quer individualmente, quer por fazerem

parte de uma operação de mecânica a realizar. O utilizador deverá também submeter um valor de referência para o custo horário de mão de obra.

Uma vez identificados todos os pontos a avaliar, o sistema deverá ser capaz de extrapolar de forma otimizada quais as operações a realizar e, tendo como base o tempo de referência de cada uma, calcular um valor final estimado.

Pode haver casos em que algumas das peças selecionadas ainda não estejam associadas a nenhuma operação para aquele veículo. Nestes casos, no momento do cálculo do orçamento, estas peças deverão ser identificadas sendo, de seguida, pedido ao utilizador que submeta manualmente uma duração estimada para a sua instalação. A inserção deste valor deverá afetar de forma dinâmica o valor final calculado.

4.1. DIAGRAMA DE ATIVIDADES

Na Figura 15 podemos observar o diagrama de atividades do sistema. Através deste é possível compreender o fluxo de utilização do sistema.

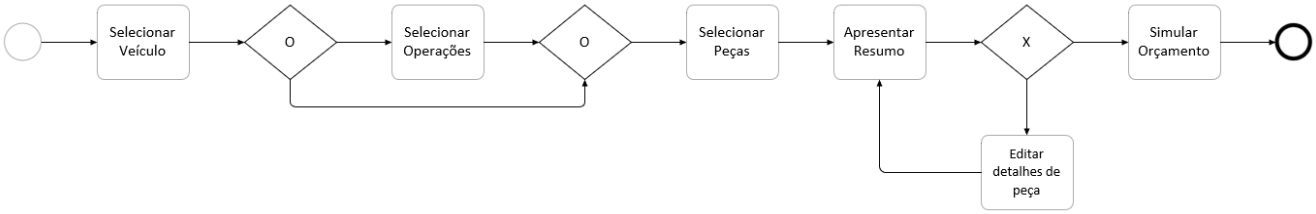

Figura 15 – Diagrama de atividades do sistema

4.2. DESENHO DO SISTEMA

Neste capítulo é apresentado o diagrama de classes e o modelo logico de dados representativos gráficos do sistema do projeto.

4.2.1. Diagramas de Classes

As Figura 16, 17 e 18 correspondem aos diagramas de classes do sistema, organizados por subprojectos. Estes diagramas correspondem a um mapeamento das classes existentes, as suas propriedades e de que forma é que estas se relacionam com outras. Nestes diagramas podemos

observar as classes que compõem a solução do sistema, nomeadamente as correspondentes às entidades da base de dados, a DTOs, entre outras.

Figura 16 – Diagrama de classes I (Infrastructure)

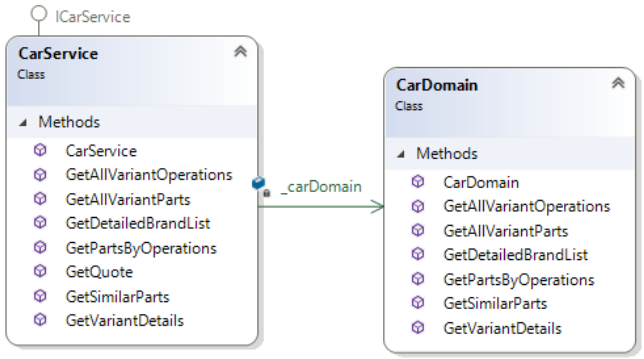

Figura 17 – Diagrama de classes II (Business)

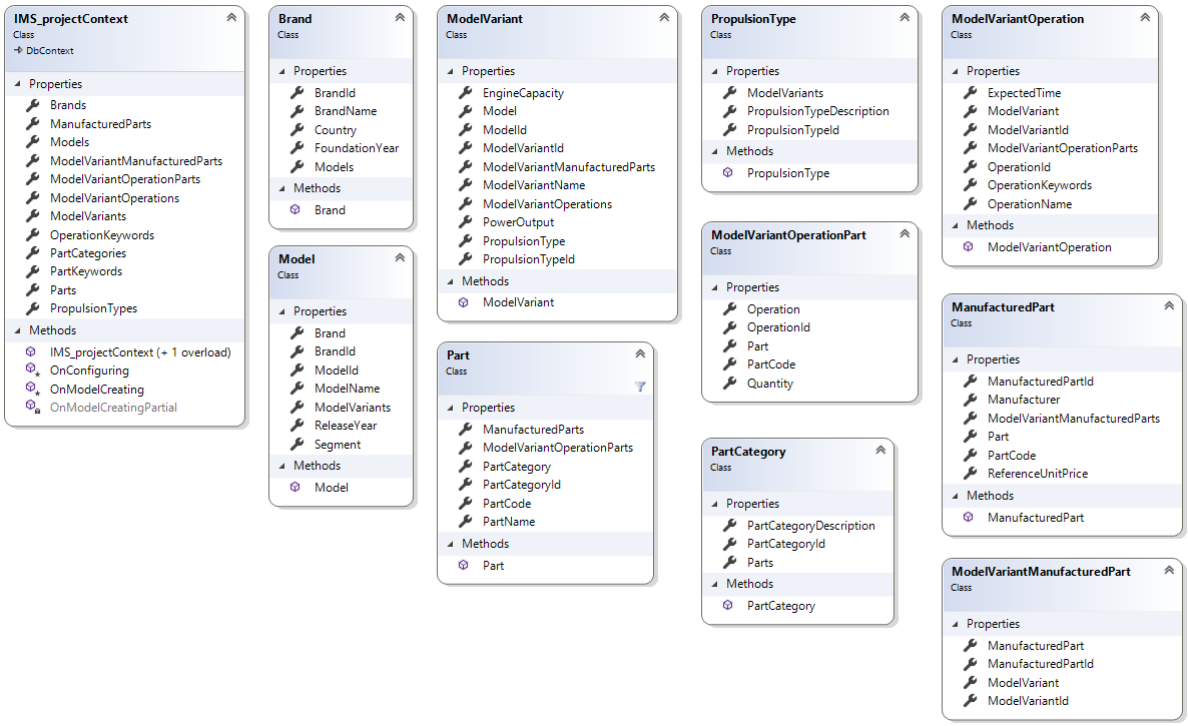

Figura 18 – Diagrama de classes III (DBManager)

4.2.2. Modelo Lógico de Dados

O seguinte modelo logico de dados (Figura 19) é uma representação gráfica da base de dados do sistema. Este diagrama permite entender as relações existentes entre as diversas entidades, assim como as propriedades que as compõem. A título de exemplo, podemos observar que um modelo (Model) terá de ter uma e só uma marca (Brand), mas uma marca poderá ter vários modelos ou mesmo nenhum.

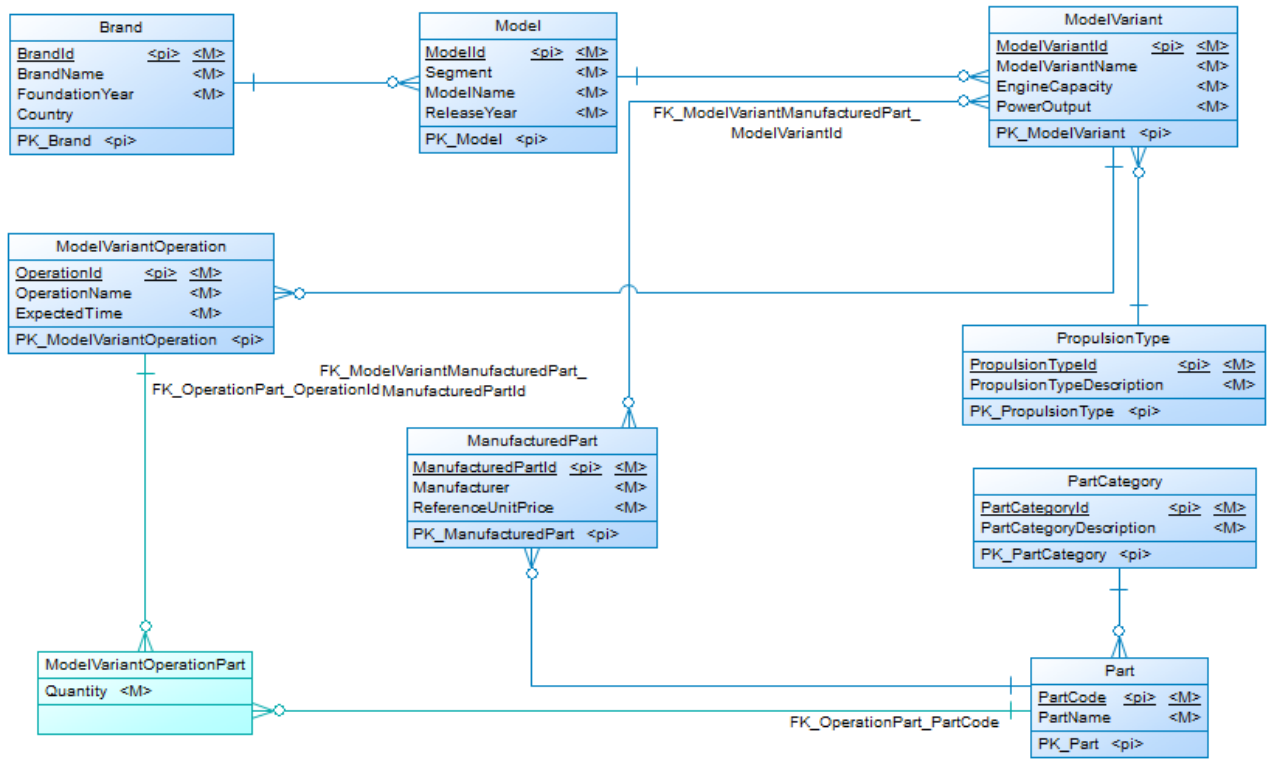

Figura 19 – Modelo lógico de dados

4.3. DESENVOLVIMENTO DO SISTEMA

4.3.1. Arquitetura

Nesta secção é apresentado a arquitetura do projeto. Tal como poder ser observado na Figura 20 o sistema encontra-se dividido em três camadas, *Interaction, Business* e *Data Layers*, pelo que um utilizador apenas interage com a primeira.

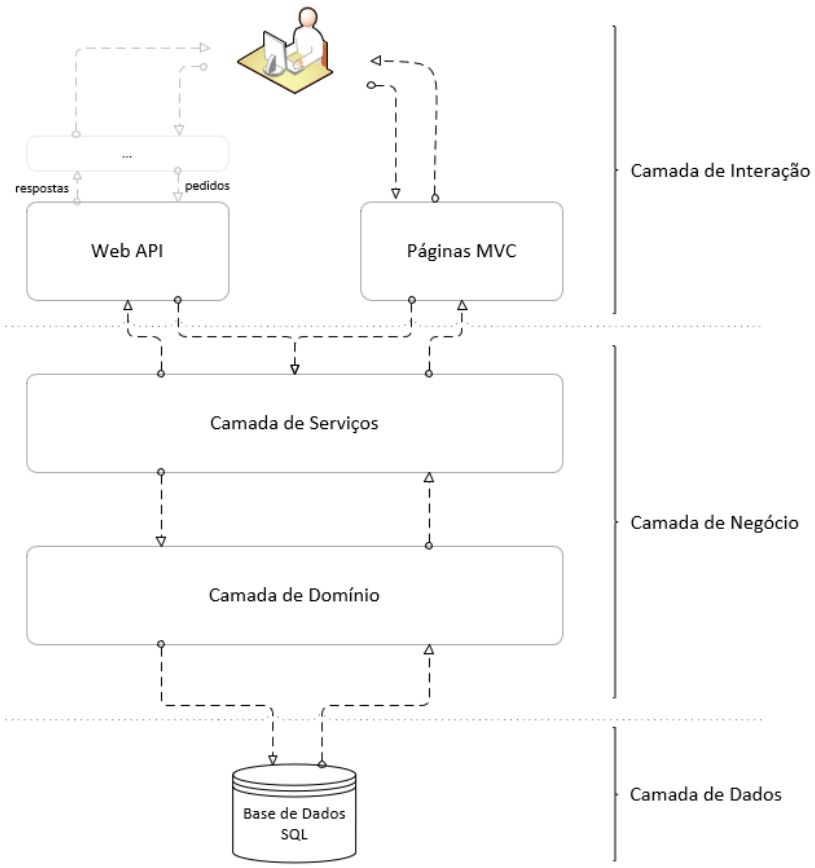

Figura 20 – Arquitetura do sistema

Cabe à *Interaction Layer*, tal como o nome indica, fazer a interação com o utilizador. É nesta camada que se encontra o *controller* da WebAPI, bem como as páginas MVC. Estes dois componentes oferecem ao utilizador duas formas distintas de interagir com o sistema. Na página desenvolvida é disponibilizada uma interface gráfica de interação simples que permite executar e visualizar todas as etapas do processo, desde que é selecionado o veículo até à produção do orçamento. Ao ser também disponibilizada um WebAPI, é dada a possibilidade de comunicar diretamente com o sistema por via

de *endpoints* REST que possibilitam também que sejam desenvolvidas outras interfaces para interagir com o sistema.

Abaixo da camada de interação está implementada a camada de negócio. É nesta camada que se definem e executam/fazem cumprir as regras de negócio do sistema. Esta camada é também dividida em duas subcamadas: uma de Domínio, responsável por todas as operações que envolvam entidades da base de dados, sendo o único ponto de interação com esta, e também uma camada de serviços que faz a ponte entre a *Domain Layer* e aquelas de mais alto nível.

Por fim, existe a camada de dados, composta por uma base de dados SQL onde estão armazenados todos os dados relativos às entidades do sistema.

4.3.2. Modelo Físico de Dados

O Modelo Físico de dados do sistema foi desenvolvido recorrendo à linguagem SQL através da ferramenta SQL Server 2014 Management Studio. Neste encontram-se representadas todas as entidades necessárias à realização do projeto, bem como as interações entre entidades.

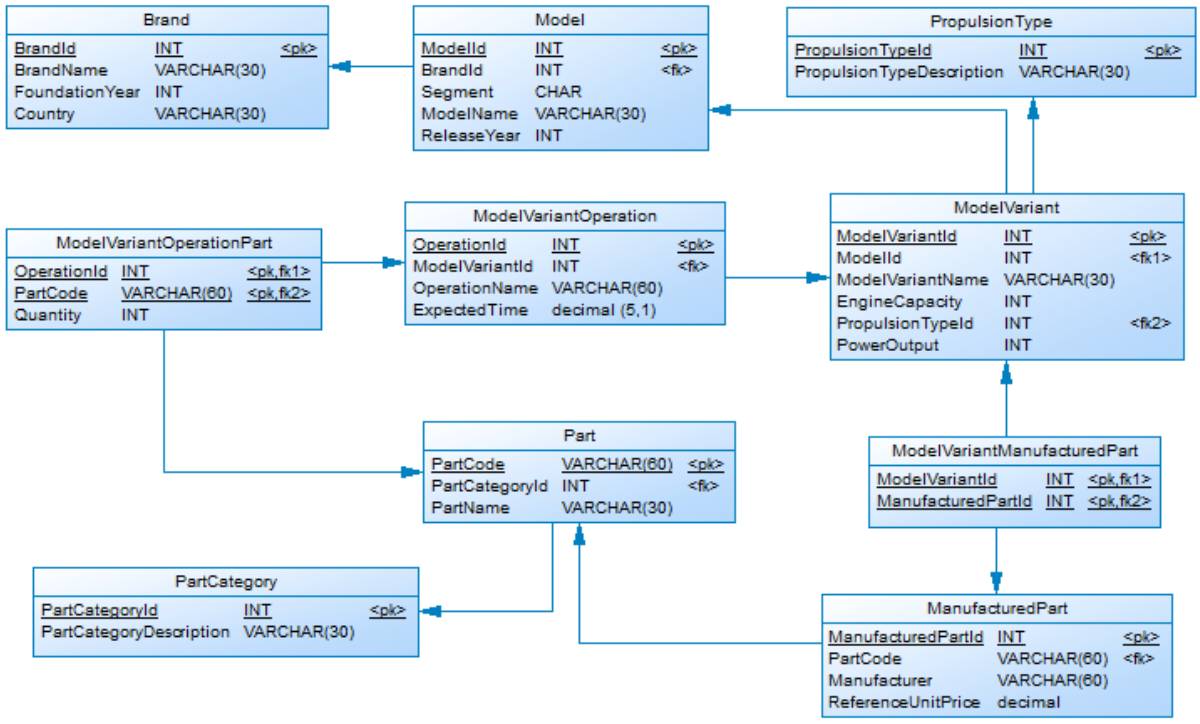

Figura 21 – Modelo físico de dados

4.3.3. Estrutura da Aplicação

A aplicação desenvolvida é uma *Multi-Page Application* (MPA). Esta é considerada a abordagem de desenvolvimento "tradicional" para uma aplicação web. Uma aplicação deste género é composta por múltiplas páginas, sendo possível navegar entre elas. Por norma, sempre que uma página é carregada é também efetuada uma chamada ao servidor.

Foi escolhida esta abordagem para o desenvolvimento da aplicação por ser um modelo intuitivo, mas também com o principal foco do sistema em mente. Conforme o utilizador vai preenchendo e avançando na aplicação este deverá experienciar algo semelhante ao preenchimento de um formulário, uma vez que cada página apresentada corresponde a uma atividade.

Assim, na primeira página da aplicação é feita a identificação do veículo. De seguida o utilizador poderá escolher, de forma não obrigatória, entre múltiplas operações do veículo selecionado, quais deseja incluir na sua simulação de orçamento. Uma vez efetuada ou ignorada esta seleção o utilizador navega para uma página semelhante onde deverá selecionar as peças. Terminada a seleção navega-se para uma página de resumo dos dados até então inseridos. Nesta página é dada a possibilidade de, por cada item do resumo, se navegar para uma outra página de modo a retificar algumas informações das peças, voltando depois à página de resumo e só sendo possível progredir quando todas as peças tiverem os seus dados completos. A terminar está a página do orçamento onde sistema determina de forma otimizada quais as operações a realizar e, tendo como base o tempo de referência de cada uma, calcular um valor final estimado.

4.3.4. Programação do Sistema

Neste capítulo é explicado o desenvolvimento do sistema. Este é composto por duas peças de *software* distintas: uma base de dados relacional SQL e uma solução .NET. A solução .NET está dividida por seis projetos diferentes, cada um com a sua função:

- DBManager: neste projeto estão presentes todos os modelos correspondentes a entidades da base de dados, bem como o *context* a ser usado na solução do sistema;
- Infrastructure: projeto que contem os DTOs para o *frontend* bem como as intefaces para os serviços que serão implementados no Business. Estre projeto serve de apoio entre Business e API/Web;
- Business: projeto com a implementação dos serviços, e aplicação de regras de negócios, sendo este o "core" do projeto geral;

- API: projeto com a API *per si*, que faz todos os acessos e uso dos projetos citados anteriormente e fornece as funcionalidades em *endpoints*;
- Web: projeto WEB que não só faz uso dos serviços citados anteriormente, como também apresenta os dados por via de uma interface *frontend* para um usuário final;
- Tests: contém os testes efetuados.

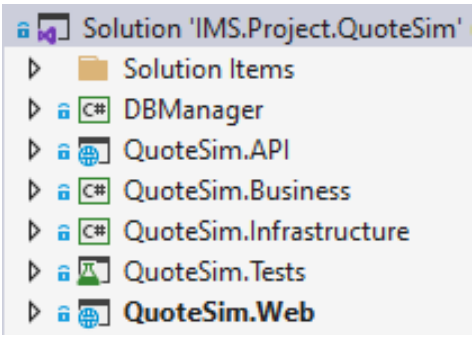

Figura 22 – Estrutura da solução .NET

Primeiro começou por se implementar a base de dados através dos respetivos scripts de criação[1]. Uma vez criada a base de dados e as suas entidades recorreu-se à instrução *Scafford* para gerar as classes e o contexto correspondentes no projeto DBManager.

Já os projetos relacionados com a interação com o utilizador e a respetiva lógica foram desenvolvidos tendo por base uma abordagem *top-down*. Nesta abordagem começaram primeiro a ser desenvolvidos os controladores da API tendo por base os requisitos funcionais definidos. De forma a definir os *endpoints* da API foi necessário definir a informação a ser passada e devolvida em cada pedido. Tal foi definido através de DTOs. Um DTO (*Data Transfer Object*) é uma classe corresponde a uma determinada entidade, apenas com as propriedades necessárias, que permite estabelecer uma abstração no tratamento e manipulação de dados.

Uma vez delimitada a estrutura da API começou por se estabelecer o contrato da camada de serviço. Este contrato é definido através da interface ICarService. Esta é composta por métodos abstratos, correspondentes às diferentes ações do sistema, a serem depois implementados nas classes que a implementem, neste caso a classe CarService. É sobre a camada de serviço e em específico sobre a classe CarService que recaem os testes unitários, assim, e antes de desenvolver a camada de domínio, é desenvolvida uma bateria de testes para cada ação, nos quais é validado se o resultado

[1] Anexo 1

obtido bate com o resultado esperado, permitindo deste modo ir validando ao longo do desenvolvimento e de possíveis iterações do sistema se este se comporta dentro dos parâmetros definidos.

Por fim, foi desenvolvida a camada de domínio, representada na classe CarDomain, que é o único ponto do sistema em que são realizados pedidos/interações com a base de dados.

Com as peças de *software* acima descritas já se teria um sistema completamente funcional, pronto a interagir com outros sistemas por via da API implementada. De modo a cumprir os objetivos estabelecidos para este projeto foi também necessário desenvolver uma camada de interface visual com a qual um utilizador possa interagir.

Cada página corresponde a uma *Razor Page*, divididas em ficheiros *.cs* e *.cshtml*, correspondentes à parte lógica e visual da pagina.

Todos os ficheiros *.cs* têm os métodos *OnGet* e *OnPost*. Estes métodos correspondem a *handlers* de eventos específicos. O método *OnGet* é chamado aquando do carregamento da página. É neste que é efetuada a chamada ao serviço de forma a obter os dados a apresentar em cada página.

O sistema desenvolvido é constituído por 6 páginas que interagem entre si tal como apresentado pela Figura 23.

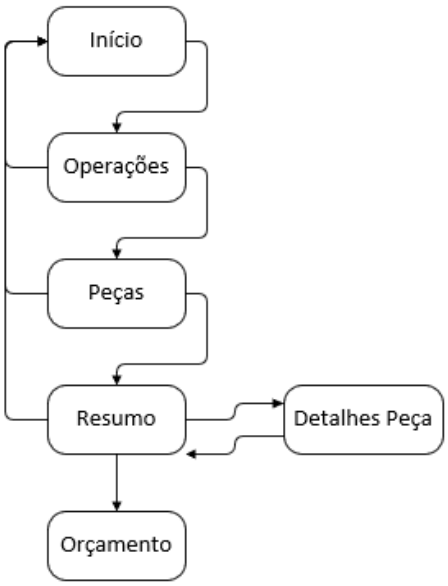

Figura 23 – Mapa de interação entre páginas

De seguida serão apresentadas as páginas do sistema. Para cada página será exposto um breve resumo das suas funcionalidades bem como um *print* da sua interface. Será também, para cada caso, descrito um ou vários *endpoints* da API que suportem as funcionalidades exibidas.

4.3.4.1. Identificação do veículo (página inicial)

Esta página corresponde ao ponto de entrada no sistema. Nesta é apresentada ao utilizador uma *dropdown* na qual ele deverá escolher um veículo. A *dropdown* está organizada em camadas – Marca > Modelo > Variante – sendo apenas possível selecionar as variantes. Uma vez que estas entidades são diretamente dependentes ao selecionar a variante também estará a selecionar as outras duas de forma implícita. Apesar de não ser apresentado ao utilizador, cada variante é caracterizada por um *id* único que é passado como parâmetro aquando da navegação para a página seguinte.

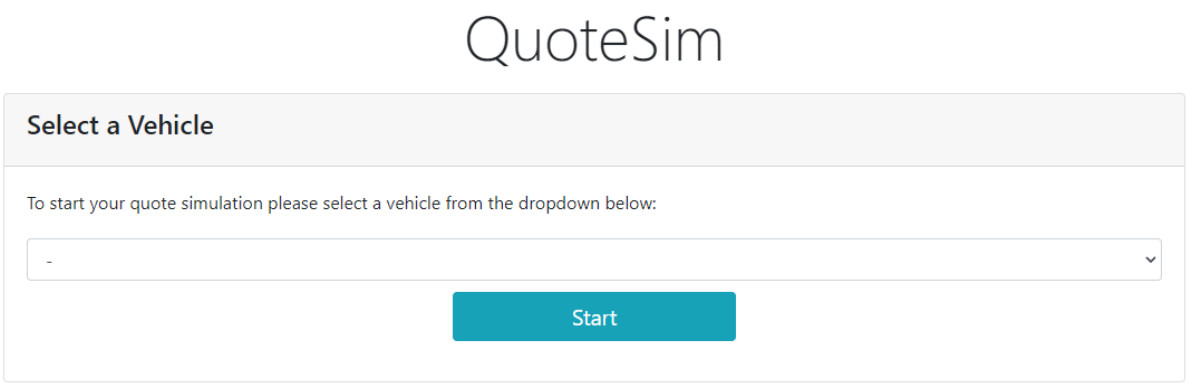

Figura 24 – Página inicial

Do lado da API para obter a mesma informação implementou-se o endpoint GET Car/list como descrito na Tabela 1:

GET /Car/list
Resposta:

[
  {
  "brandId": 0,
  "brandName": "string",
  "models": [
  {
  "modelId": 0,
  "modelName": "string",
  "releaseYear": 0,
  "variants": [

```
          {
            "variantId": 0,
            "variantName": "string",
            "engineCapacity": 0,
            "propulsionType": "string",
            "powerOutput": 0
          }
        ]
      }
    ]
  }
]
```


Tabela 1 – Endpoint de listagem de veículos

4.3.4.2. Lista de operações

Quando se navega para esta página é passado como parâmetro o *id* da variante do veículo selecionada. A partir desse *id* são obtidos os detalhes da mesma bem como a lista de operações possíveis de realizar. Esta informação é apresentada ao utilizador sob a forma de dois cartões como demonstrado na Figura 25. Na lista de operações o utilizador pode selecionar aquelas que deseja contemplar na sua simulação ou simplesmente saltar este passo. Uma vez selecionadas as operações são guardadas sob a forma de *cookie* de sessão de modo a serem também apresentadas nas páginas seguintes.

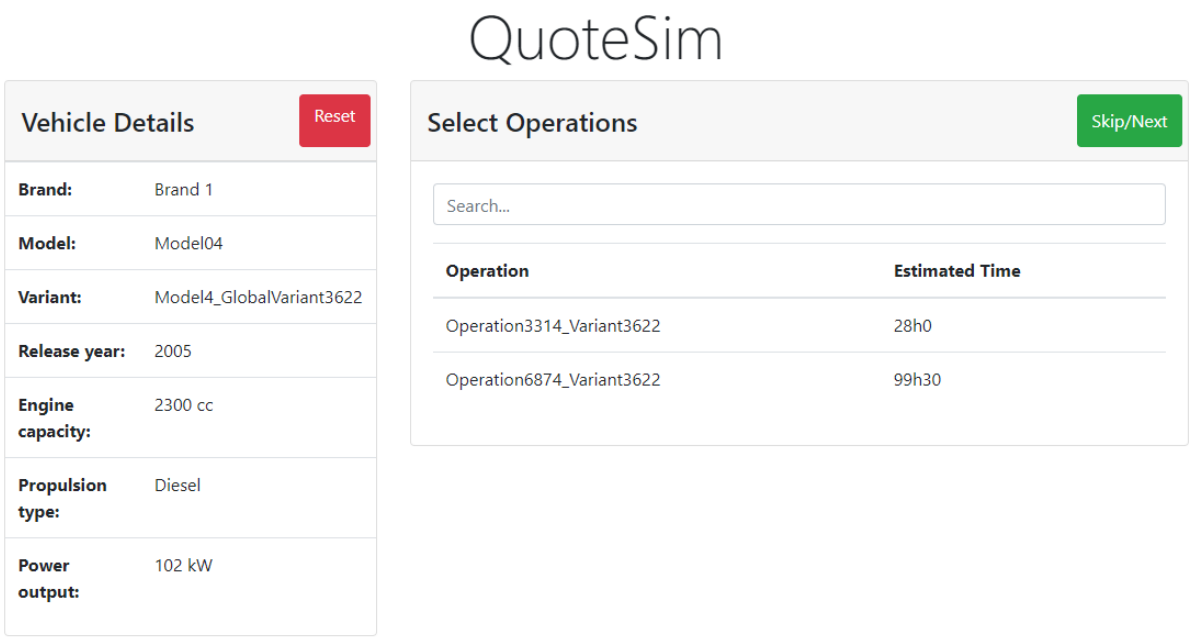

Figura 25 – Página de seleção e operações

Do lado da API para obter estas informações implementou-se os *endpoints* GET Car/{variantId} e GET Car/{variantId}/operation/list como descrito nas Tabela 2 e 3:

GET /Car/{variantId}
Parâmetros:
`"variantId": 0`
Resposta:

`{`
`  "brandId": 0,`
`  "brandName": "string",`
`  "modelId": 0,`
`  "modelName": "string",`
`  "modelReleaseYear": 0,`
`  "modelVariantId": 0,`
`  "modelVariantName": "string",`
`  "engineCapacity": 0,`
`  "engineCapacityUnit": "string",`
`  "powerOutput": 0,`
`  "propulsionType": "string"`
`}`

Tabela 2 – Endpoind de detalhes de veículo

GET /Car/{variantId}/operation/list
Parâmetros:
`"variantId": 0`
Resposta:

`[`
`  {`
`  "operationId": 0,`
`  "operationName": "string",`
`  "expectedTime": 0`
`  }`
`]`

Tabela 3 – Endpoint de listagem de operações

4.3.4.3. Lista de peças

Esta página é, em parte, semelhante à da listagem de operações. Neste caso também é passado como parâmetro o *id* da variante e também são apresentados os detalhes da mesma bem como,

nesta situação, a lista de peças associadas a esta. Para além destes cartões é também apresentado um outro com as peças referentes às operações selecionadas anteriormente. Uma vez selecionadas as peças, neste caso, não são guardadas sob a forma de *cookie* de sessão, mas sim em *Session state* de modo a serem também apresentadas nas páginas seguintes. Foi escolhida esta abordagem uma vez que a lista de peças é potencialmente ilimitada podendo o tamanho de dados a guardar exceder o limite para armazenar em *cookies* (4096 bytes).

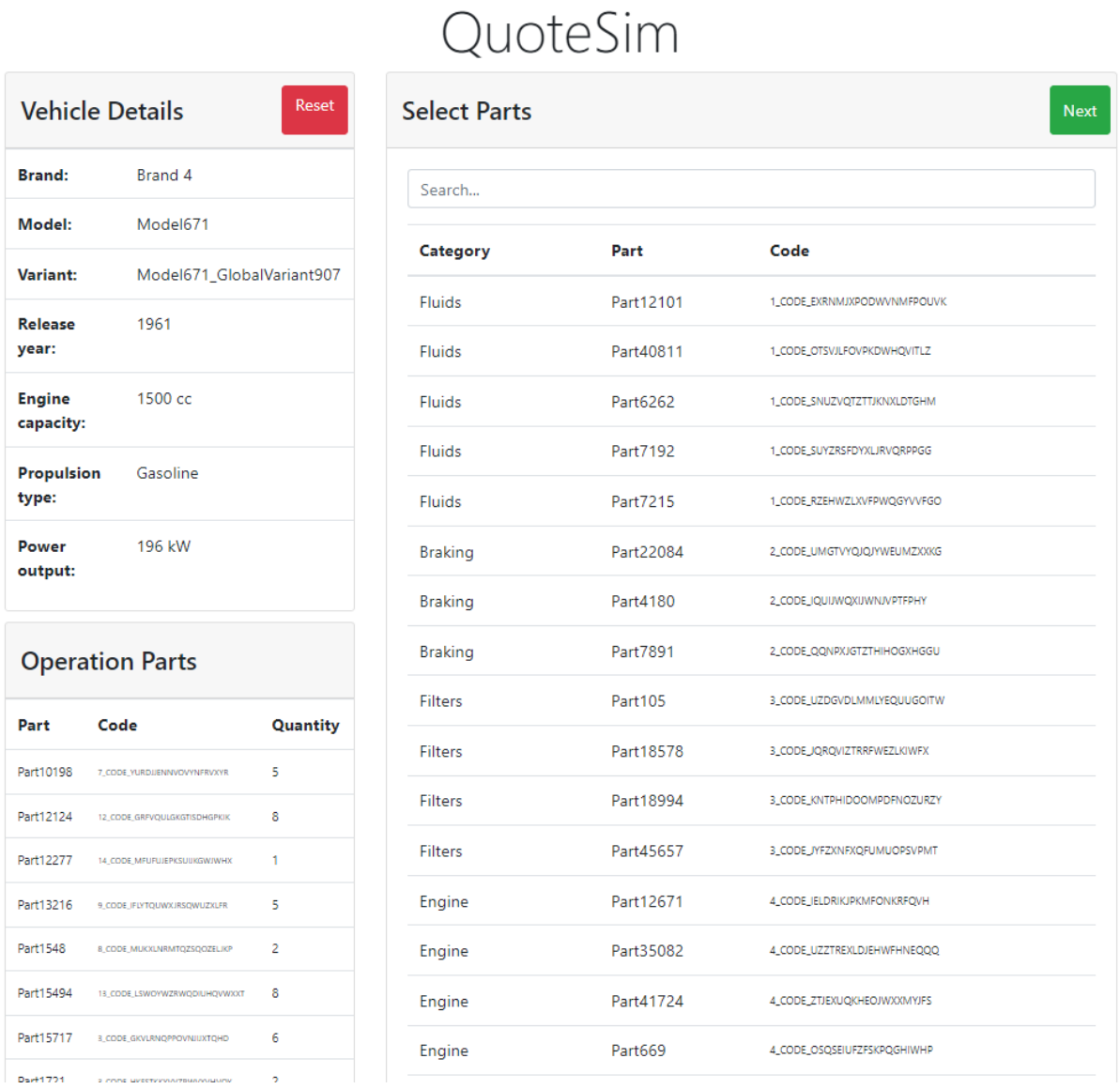

Figura 26 – Página de seleção de peças

Do lado da API para obter estas informações recorre-se ao *endpoint* GET Car/{variantId} já descrito anteriormente e implementou-se os *endpoints* GET Car/{variantId}/part/list e GET /Car/operation /part/list como descrito na Tabela 4 e 5:

GET /Car/{variantId}/part/list
Parâmetros:
`"variantId": 0`
Resposta:
`[ { "partCode": "string", "partName": "string", "partCategoryId": 0, "partCategoryDescription": "string", "quantity": 0 } ]`

Tabela 4 – Endpoint de listagem de peças

GET /Car/operation/part/list
Parâmetros:
`"operationIds": [0]`
Resposta:
`[ { "partCode": "string", "partName": "string", "partCategoryId": 0, "partCategoryDescription": "string", "quantity": 0 } ]`

Tabela 5 – Endpoint de listagem de peças para determinadas operações

4.3.4.4. Resumo

Esta página corresponde a resumo dos dados inseridos até ao momento. Quando se navega para esta página são carregados os dados guardados em *Session state* e a partir daí são construídos dois

cartões, um de detalhes do veículo e outro correspondente à lista de peças selecionadas. Por cada entrada da lista de operações é possível navegar para outra página, onde são editados os detalhes da mesma para o efeito da simulação, passando como parâmetro o código da peça.

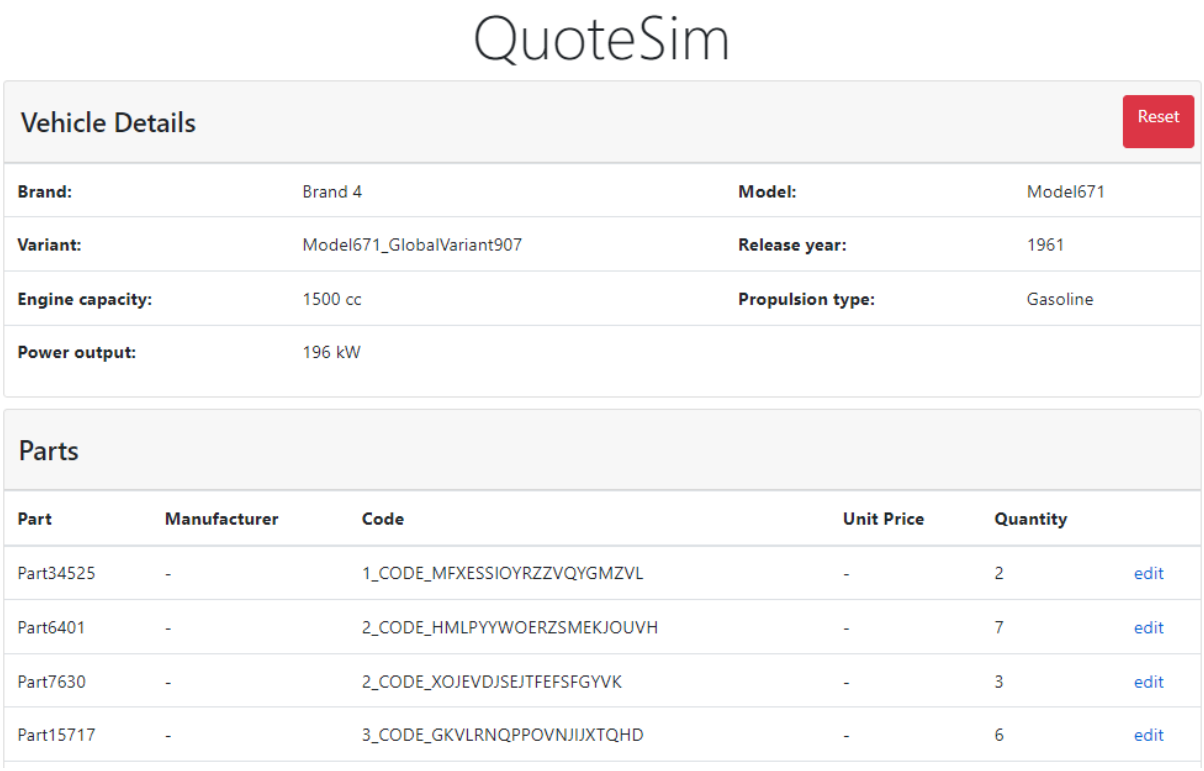

Figura 27 – Página de resumo

Neste caso, do lado da API, e sendo esta uma página de resumo, os *endpoints* correspondem aos já anteriormente descritos.

4.3.4.5. Opções de peça

Nesta página o utilizador escolhe as opções para determinada peça, mais em concreto a quantidade bem como o fabricante (e consequentemente o preço). Para apresentar estes dados é carregada a informação armazenada, até então, em *Session state*, mas também o código da peça que é passado como parâmetro aquando da navegação para esta página. Uma vez atualizada a peça, os dados são novamente guardados na sessão e feita a navegação para a página de resumo.

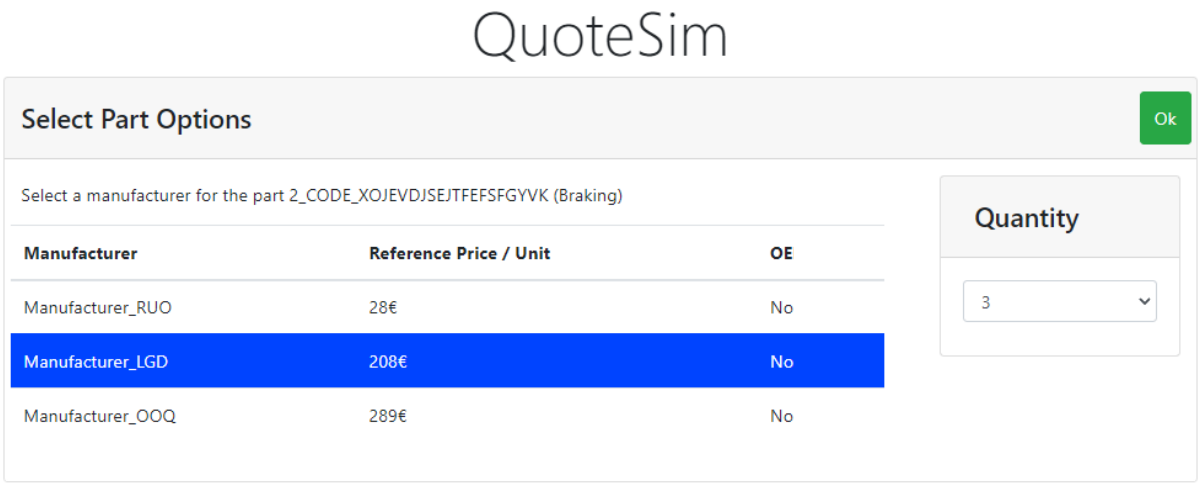

Figura 28 – Página de opções de peça

Do lado da API para obter a a informação relativa às peças de diversos fabricantes para determidado código de peça implementou-se o endpoint GET Car/part/{partCode}/list como descrito na Tabela 6:

GET / Car/part/{partCode}/list
Parâmetros:
`"partCode": "string"`
Resposta:
`[ { "manufacturedPartId": 0, "manufacturer": "string", "referenceUnitPrice": 0 } ]`

Tabela 6 – Endpoint de listagem de peças por código

4.3.4.6. Orçamento

Através dos dados validados na página de resumo e passados para esta através do seu armazenamento em *Session state,* nesta página podemos observar novamente os detalhes do veículo seguidos da simulação de orçamento subdividida em operações, tal como está representado na Figura 29. As operações a apresentar são determinadas tendo em conta as peças selecionadas de forma a gerar o orçamento com o custo mais baixo possível, de acordo com os seguintes passos:

1. São obtidas todas as operações apenas constituídas pelas peças selecionadas;

2. As operações são ordenas do maior para o menor número de peças que as compõem e da mais barata para a mais cara;

3. Guarda-se a primeira operação da lista ordenada;

4. Repetem-se os passos 1 a 3 com as peças não incluídas na operação selecionada até que não seja possível obter nenhuma operação com as peças restantes;

5. Soma-se o custo das operações obtidas;

6. Repetem-se os passos 1 a 5 ignorando a operação já selecionada no ponto 3;

7. Caso o valor seja superior ao obtido no ponto 4, é devolvido o resultado calculado. Caso contrário, repete-se novamente o processo com a operação seguinte até um valor de peças por operação igual a metade mais um do valor máximo, obtendo-se o valor mais baixo possível.

Caso por alguma razão determinada peça não se inclua em nenhuma das operações, é dada a possibilidade ao utilizador de providenciar um tempo de montagem para esta e submeter novamente a simulação.

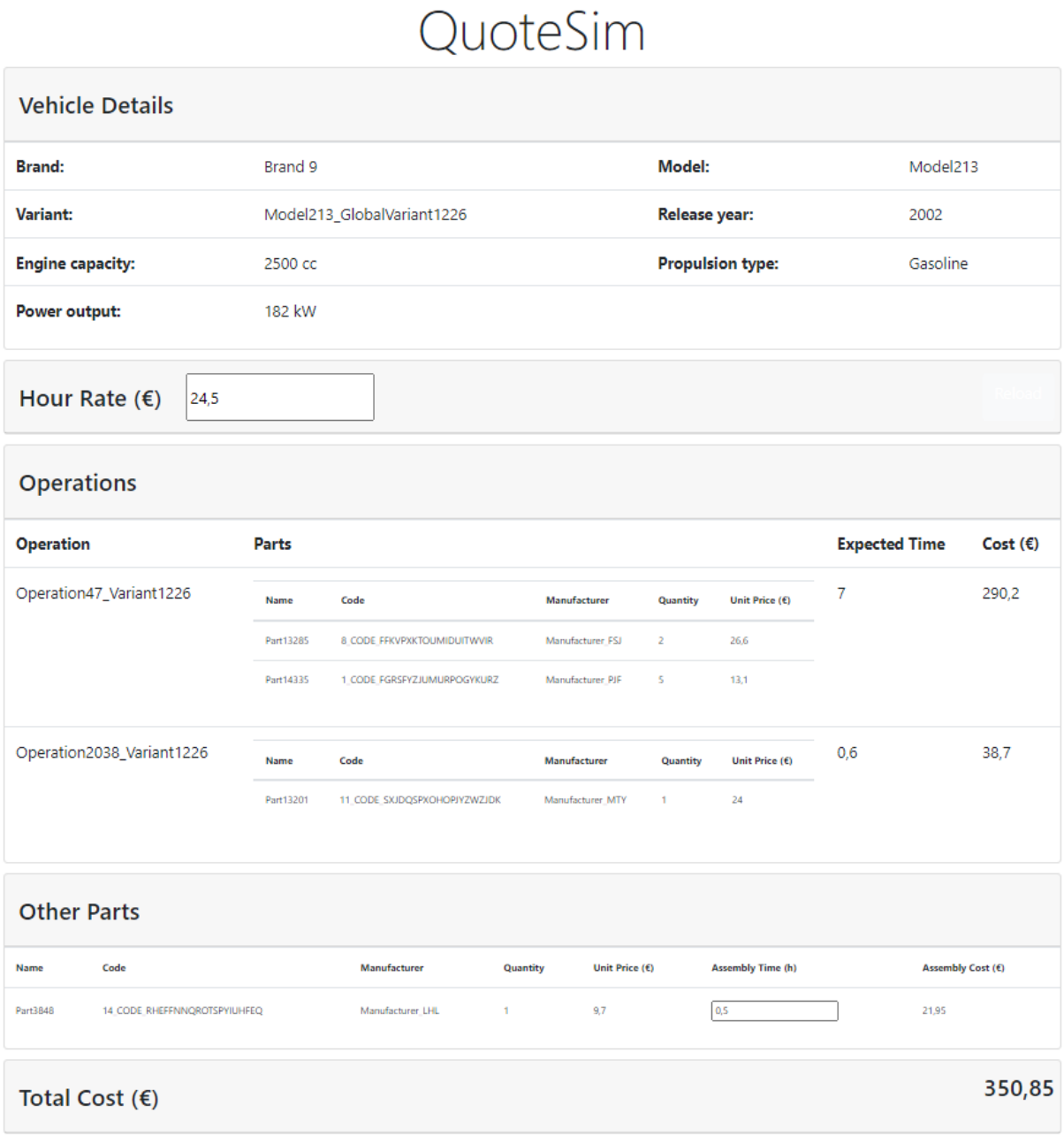

Figura 29 – Página de resultado da simulação de orçamento

Esta distribuição sequencial de elementos permite ao utilizador, caso assim deseje, imprimir a simulação gerada.

Do lado da API para obter a a informação relativa às peças de diversos fabricantes para determidado código de peça, implementou-se o endpoint GET Car/maintenanceQuote, como descrito na Tabela 7, que recebe um modelo com o valor de referência para a mão de obra/hora em conjunto com uma lista de todas as peças a considerar incluindo a sua quantidade e, se necessário, o tempo necessário para a sua aplicação no veículo:

GET / Car/maintenanceQuote

Parâmetros:

```
{
  "hourRate": 0.0,
  "parts": [
    {
      "manufacturedPartId": 0,
      "quantity": 0,
      "assemblyTime": null
    }
  ]
}
```

Resposta:

```
{
  "hourRate": 0,
  "operations": [
    {
      "operationId": 0,
      "operationName": "string",
      "expectedTime": 0,
      "parts": [
        {
          "partCode": "string",
          "partName": "string",
          "partCategoryId": 0,
          "partCategoryDescription": "string",
          "manufacturer": "string",
          "referenceUnitPrice": 0,
          "quantity": 0
        }
      ],
      "totalCost": 0
    }
  ],
  "otherParts": [
    {
      "partCode": "string",
      "partName": "string",
      "partCategoryId": 0,
      "partCategoryDescription": "string",
      "manufacturer": "string",
      "referenceUnitPrice": 0,
      "quantity": 0,
      "assemblyTime": 0,
      "totalCost": 0
    }
  ],
  "totalCost": 0
}
```

Tabela 7 – Endpoint de orçamento

4.3.5. Testes

Para efeitos de testes e, uma vez que não foi possível obter os dados de forma gratuita, a base de dados foi preenchida com valores *dummy* gerados aleatoriamente.

Na Tabela 8 podem-se observar os testes realizados, mais especificamente sobre que método da camada de serviço é que estes incidem e de que forma é que estes são testados.

Método da camada de serviço	**Descrição do teste**	**Resultado esperado**
`GetDetailedBrandList`	Validar que a lista a retornar contém elementos	Lista retornada deverá estar inicializada e populada
`GetVariantDetails`	Validar retorno para um id de variante inválido	Modelo de retorno vazio (*null*)
`GetVariantDetails`	Validar retorno para um id de variante válido	Modelo de retorno válido
`GetAllVariantOperations`	Validar retorno para um id de variante inválido	Modelo de retorno vazio (*null*)
`GetAllVariantOperations`	Validar retorno para um id de variante válido	Modelo de retorno válido
`GetPartsByOperations`	Validar retorno para um id de operação inválido	Modelo de retorno vazio (*null*)
`GetPartsByOperations`	Validar retorno para um id de operação válido	Modelo de retorno válido
`GetAllVariantParts`	Validar retorno para um id de variante inválido	Modelo de retorno vazio (*null*)
`GetAllVariantParts`	Validar retorno para um id de variante válido	Modelo de retorno válido

`GetSimilarParts`	Validar retorno para um código de peça sem valor (*null*)	Modelo de retorno vazio (*null*)
`GetSimilarParts`	Validar retorno para um código de peça inválido	Modelo de retorno vazio (*null*)
`GetSimilarParts`	Validar retorno para um código de peça válido	Modelo de retorno válido
`GetSimilarParts`	Validar retorno para um código de peça e id de variante válidos	Modelo de retorno válido
`GetSimilarParts`	Valid part code and invalid variantId	Modelo de retorno vazio (*null*)
`GetQuote`	Validar retorno para um modelo de pedido sem valor (*null*)	Modelo de retorno vazio (*null*)
`GetQuote`	Validar retorno para um modelo de pedido sem nenhuma peça	Modelo de retorno vazio (*null*)
`GetQuote`	Validar retorno para um modelo de pedido com o valor de mão de obra/hora igual a zero	Modelo de retorno vazio (*null*)
`GetQuote`	Validar retorno para um modelo de pedido com o valor de mão de obra/hora diferente de zero e com múltiplas peças	Modelo de retorno válido

Tabela 8 – Testes da camada de serviço

5. CONSIDERAÇÕES FINAIS

5.1. SÍNTESE

Este projeto surgiu a partir de uma necessidade existente por parte de diversos proprietários de automóveis, sem conhecimentos de mecânica, em conseguirem avaliar os custos associados às diversas operações de manutenção e reparação das suas viaturas. Para tal foi feito um estudo teórico aprofundado das áreas associadas ao projeto e ao problema a abordar, permitindo assim consolidar bases para o desenvolvimento do mesmo.

O referido estudo permitiu adquirir conhecimentos teóricos chave para o desenvolvimento prático do projeto, possibilitando, deste modo, uma melhor tomada de decisões. A componente prática do projeto começou por um levantamento e análise dos requisitos funcionais. Posteriormente, foi realizado um plano do sistema proposto com a respetiva arquitetura. Neste plano foi detalhada a composição do sistema, bem como as interações entre os diversos componentes do mesmo.

Ao longo da fase prática foi fundamental estudar metodologias de desenvolvimento de *software*, de modo a adotar boas práticas durante a implementação do projeto. Foi também importante rever conceitos de arquitetura de *software* garantindo, assim, uma boa estrutura do código implementado. Foi com suporte nestes conhecimentos que se desenvolveu todo o sistema, começando pela base de dados, passando para a API e páginas web, com todas as camadas intermédias que as suportam. Conforme o sistema ia sendo desenvolvido, garantiu-se também que este estava de acordo com o especificado e em correto funcionamento através de uma bateria de testes. No fim, obteve-se um sistema que permite gerar de forma simplificada uma simulação de custos de manutenção automóvel, colmatando, deste modo, a necessidade inicialmente identificada.

5.2. CONTRIBUIÇÕES DO PROJETO

O projeto foi desenvolvido com o objetivo de permitir dar ao utilizador, independentemente do seu conhecimento de mecânica automóvel, a possibilidade de avaliar os custos associados às diversas operações de manutenção e reparação das suas viaturas e de fornecer uma ferramenta que permita, a oficinas e mecânicos, ter uma referência no momento de realizar orçamentos. Assim, este projeto poderá ter um contributo não só para o consumidor, mas também para os prestadores de serviços.

5.3. LIMITAÇÕES E TRABALHO FUTURO

Este projeto encontra-se limitado pela falta de dados reais de acesso gratuito. A estrutura de dados desenvolvida foi preenchida com dados *dummy* que permitiram testar a funcionalidade pretendida, mas sem ser possível avaliar um caso real. De modo a colmatar esta limitação seria necessário o investimento monetário num ou vários *datasets* com os dados, ou então efetuar um estudo intensivo do mercado.

A própria base de dados desenvolvida também limita o sistema no sentido em que esta apenas contempla a existência de uma moeda e não ter em consideração os valores de impostos cobrados. Desta forma o sistema apenas se adaptaria à realidade de um país, sendo necessário alterar o modelo de dados para que esta fosse uma solução universal.

Por fim, sendo os valores de custo das peças dinâmicos seria necessária uma constante monotorização e manutenção dos dados de modo a providenciar sempre cálculos de orçamento ideias.

BIBLIOGRAFIA

ACP. (n.d.). *GARANTIAS NOS CARROS NOVOS E USADOS*. Retrieved June 5, 2020, from https://www.acp.pt/automovel/manutencao-automovel/cuidados-com-o-carro/garantias-nos-carros-novos-e-usados

Afsahi, M., & Shafiee, M. (2020). A stochastic simulation-optimization model for base-warranty and extended-warranty decision-making of under- and out-of-warranty products. *Reliability Engineering and System Safety*.

ALLDATA. (n.d.). *About Us | ALLDATA*. Retrieved January 27, 2021, from https://www.alldata.com/us/en/aboutus

Andaleeb, S. S., & Basu, A. K. (1994). Technical complexity and consumer knowledge as moderators of service quality evaluation in the automobile service industry. *Journal of Retailing*.

Autotrader. (2015). *Survey Shows 72 Percent of Car Owners Perform Maintenance on Their Own*. http://press.autotrader.com/2015-07-01-Survey-Shows-72-Percent-of-Car-Owners-Perform-Maintenance-on-Their-Own

Avison, D. E., & Fitzgerald, G. (2003). Where now for development methodologies? *Communications of the ACM*.

Awad, M. A. (2005). *A Comparison between Agile and Traditional Software Development Methodologies*.

Baltusis, P. (2004). On board vehicle diagnostics. *SAE Technical Paper*.

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Principles behind the Agile Manifesto*. https://agilemanifesto.org/iso/en/principles.html

Bureau of Transportation Statistics. (2019). *World Motor Vehicle Production, Selected Countries (Thousands of vehicles)*. United States Department of Transportation. https://www.bts.gov/bts/archive/publications/national_transportation_statistics/table_01_23

Castrol. (n.d.). *ENGINE OIL VISCOSITY GRADES*. Retrieved January 26, 2021, from https://www.castrol.com/en_gb/united-kingdom/home/car-engine-oil-and-fluids/engine-oils/engine-oil-viscosity-grades.html

Côrtes, P. L. (2008). *Administração de Sistemas de Informação*. Editora Saraiva.

Daimler. (2019). *Tradition: Founders and Pioneers*. https://www.daimler.com/company/tradition/founders-pioneers/carl-benz.html

Darley, W. K., & Luethge, D. J. (2019). Service value and retention: Does gender matter? *Journal of Retailing and Consumer Services*.

Date, C. (1987). *A guide to the SQL Standard: a user's guide to the standard relational language SQL*.

Denton, T. (2008). *ATT Automotive Level 1*.

Federal Trade Commission. (n.d.). *Auto Warranties & Routine Maintenance*. https://www.consumer.ftc.gov/articles/0138-auto-warranties-routine-maintenance

Goldstein, H., Poole, C., Safko, J., & Addison, S. R. (2002). Classical Mechanics, 3rd ed. *American Journal of Physics*.

History.com Editors. (2010). *Automobile History*. A&E Television Networks. https://www.history.com/topics/inventions/automobiles

HOFFER, G. E., PRUITT, S. W., & REILLY, R. J. (1994). When Recalls Matter: Factors Affecting Owner Response to Automotive Recalls. *Journal of Consumer Affairs*.

IATE. (2018). *European Union Terminology - frontend & backend*. https://iate.europa.eu/entry/result/1696084

IBM Research. (2014). *Software Development*. https://researcher.watson.ibm.com/researcher/view_group.php?id=5227

Impostosobreveiculos.info. (2019). *Imposto Único Circulação (IUC) 2020*. https://impostosobreveiculos.info/iuc/imposto-unico-circulacao-iuc-2020/

Insurance Information Institute. (n.d.). *FAQs About Direct Repair Programs and Generic Auto Parts*. Retrieved January 26, 2021, from https://www.iii.org/article/faqs-about-direct-repair-programs-and-generic-auto-parts

Kachadourian, G. (2005). Car makers fight for service customers. *Tire Business*.

Kennedy, A. B. W. (1886). The Mechanics of Machinery. In *Nature*. Macmillan.

Legislação. (2008). *Venda de Bens de Consumo e das Garantias a Ela Relativas - DL 67/2003*.

Mercedes-Benz. (n.d.). *Inovação. Para uma mobilidade livre de emissões.* Retrieved June 5, 2020, from https://www.mercedes-benz.pt/passengercars/the-brand/innovation/concept-eqa.module.html

Microsoft. (n.d.). *Visual Studio: IDE and Code Editor for Software Developers and Teams*. Retrieved September 20, 2021, from https://visualstudio.microsoft.com/

Microsoft. (2021). *What's new in .NET 5*. https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-5

Mobil. (n.d.). *Vehicle maintenance – How to change oil and more*. Retrieved January 26, 2021, from https://www.mobil.com/en/lubricants/for-personal-vehicles/auto-care/vehicle-maintenance/

OICA. (2019). *New cv registrations or sales*. http://www.oica.net/wp-content/uploads/pc_sales_2019.pdf

Parahar, M. (2020). *Difference between System software and Application software.* Tutorialspoint. https://www.tutorialspoint.com/difference-between-system-software-and-application-software

Pierce, B. C. (2002). Types and Programming Languages. *Notes*, *2002*.

Pressman, R. S. (2014). Software Quality Engineering: A Practitioner's Approach. In *Software Quality Engineering: A Practitioner's Approach*.

RedHat. (n.d.). *What is an API?* Retrieved January 27, 2021, from https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Renault. (n.d.). *Renault Sport: a excelência tecnológica*. Retrieved June 5, 2020, from https://www.renault.pt/desporto-automovel/excelencia-tecnologica.html

Rocha, J. (2017). *Carros com garantia maior: o que deve saber*. https://www.e-konomista.pt/carros-com-garantia-maior/

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*.

Santos, V. (2019). *Information Systems Development - Class Slides: Part 6 - Models of software development processes* (p. 79). Universidade Nova de Lisboa - Nova IMS.

Shyun, S. (2017). *How Auto Repair Quotes Are Calculated*. MechanicLot. https://medium.com/mechaniclot/how-auto-repair-quotes-are-calculated-714f6638403e

Squareboat. (n.d.). *Different Types of Software with Examples*. Retrieved January 27, 2021, from https://squareboat.com/blog/different-types-of-software-with-examples

Stjepandić, J., Wognum, N., & Verhagen, W. J. C. (2015). Concurrent engineering in the 21st century: Foundations, developments and challenges. In *Concurrent Engineering in the 21st Century: Foundations, Developments and Challenges*.

Thomas, V. J., & Maine, E. (2019). Market entry strategies for electric vehicle start-ups in the automotive industry – Lessons from Tesla Motors. *Journal of Cleaner Production*.

Total. (n.d.). *Classes of brake fluids*. Retrieved January 26, 2021, from https://www.lubricants.total.com/help-support?keys=&field_topics_tid=18#

UML. (2005). *INTRODUCTION TO OMG'S UNIFIED MODELING LANGUAGE*. https://www.uml.org/what-is-uml.htm

Weldon, P., Morrissey, P., & O'Mahony, M. (2018). Long-term cost of ownership comparative analysis between electric vehicles and internal combustion engine vehicles. *Sustainable Cities and Society*.

White, S. A., & Miers, D. (2008). *BPMN Modeling and Reference Guide*.

Xoneca. (2015). *Shape of a female OBD-II type A connector.* Wikimedia. https://commons.wikimedia.org/wiki/File:OBD-II_type_A_female_connector_shape.svg

ANEXOS

Anexo 1 - Scripts de criação da base de dados

```
CREATE TABLE Brand
(
	BrandId INT IDENTITY(1,1),
	BrandName VARCHAR(30) NOT NULL,
	FoundationYear INT NOT NULL,
	Country VARCHAR(30),

	CONSTRAINT PK_Brand PRIMARY KEY (BrandId)
)
GO

CREATE TABLE PropulsionType
(
	PropulsionTypeId INT IDENTITY(1,1),
	PropulsionTypeDescription VARCHAR(30) NOT NULL,

	CONSTRAINT PK_PropulsionType PRIMARY KEY (PropulsionTypeId)
)
GO

CREATE TABLE Model
(
	ModelId INT IDENTITY(1,1),
	BrandId INT NOT NULL,
	Segment CHAR NOT NULL,
	ModelName VARCHAR(30) NOT NULL,
	ReleaseYear INT NOT NULL,

	CONSTRAINT PK_Model PRIMARY KEY (ModelId),
	CONSTRAINT FK_Model_BrandId FOREIGN KEY (BrandId) REFERENCES Brand (BrandId)
)
GO

CREATE TABLE ModelVariant
(
	ModelVariantId INT IDENTITY(1,1),
	ModelId INT NOT NULL,
	ModelVariantName VARCHAR(30) NOT NULL,
	EngineCapacity INT NOT NULL,
	PropulsionTypeId INT NOT NULL,
	PowerOutput INT NOT NULL,

	CONSTRAINT PK_ModelVariant PRIMARY KEY (ModelVariantId),
	CONSTRAINT FK_ModelVariant_ModelId FOREIGN KEY (ModelId) REFERENCES Model
(ModelId),
	CONSTRAINT FK_ModelVariant_PropulsionTypeId FOREIGN KEY (PropulsionTypeId)
REFERENCES PropulsionType (PropulsionTypeId)
)
GO

CREATE TABLE PartCategory
(
	PartCategoryId INT IDENTITY(1,1),
	PartCategoryDescription VARCHAR(30) NOT NULL,

	CONSTRAINT PK_PartCategory PRIMARY KEY (PartCategoryId)
```

```
)
GO

CREATE TABLE Part
(
	PartCode VARCHAR(60) NOT NULL,
	PartCategoryId INT NOT NULL,
	PartName VARCHAR(30) NOT NULL,

	CONSTRAINT PK_Part PRIMARY KEY (PartCode),
	CONSTRAINT FK_Part_PartCategoryId FOREIGN KEY (PartCategoryId) REFERENCES
PartCategory (PartCategoryId)
)
GO

CREATE TABLE ManufacturedPart
(
	ManufacturedPartId INT IDENTITY(1,1),
	PartCode VARCHAR(60) NOT NULL,
	Manufacturer VARCHAR(60) NOT NULL,
	ReferenceUnitPrice decimal NOT NULL,

	CONSTRAINT PK_ManufacturedPart PRIMARY KEY (ManufacturedPartId),
	CONSTRAINT FK_ManufacturedPart_PartCode FOREIGN KEY (PartCode) REFERENCES Part
(PartCode)
)
GO

CREATE TABLE ModelVariantManufacturedPart
(
	ModelVariantId INT NOT NULL,
	ManufacturedPartId INT NOT NULL,

	CONSTRAINT PK_ModelVariantManufacturedPart PRIMARY KEY (ModelVariantId,
ManufacturedPartId),
	CONSTRAINT FK_ModelVariantManufacturedPart_ModelVariantId FOREIGN KEY
(ModelVariantId) REFERENCES ModelVariant (ModelVariantId),
	CONSTRAINT FK_ModelVariantManufacturedPart_ManufacturedPartId FOREIGN KEY
(ManufacturedPartId) REFERENCES ManufacturedPart (ManufacturedPartId)
)
GO

CREATE TABLE ModelVariantOperation
(
	OperationId INT IDENTITY(1,1),
	ModelVariantId INT NOT NULL,
	OperationName VARCHAR(60) NOT NULL,
	ExpectedTime decimal (5,1) NOT NULL,

	CONSTRAINT PK_ModelVariantOperation PRIMARY KEY (OperationId),
	CONSTRAINT FK_ModelVariantOperation FOREIGN KEY (ModelVariantId) REFERENCES
ModelVariant (ModelVariantId)
)
GO

CREATE TABLE ModelVariantOperationPart
(
	OperationId INT NOT NULL,
	PartCode VARCHAR(60) NOT NULL,
	Quantity INT NOT NULL,

	CONSTRAINT PK_OperationPart PRIMARY KEY (OperationId, PartCode),
```

```
	CONSTRAINT FK_OperationPart_OperationId FOREIGN KEY (OperationId) REFERENCES
ModelVariantOperation (OperationId),
	CONSTRAINT FK_OperationPart_PartCode FOREIGN KEY (PartCode) REFERENCES Part
(PartCode)
)
GO
```

Anexo 2 – API


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "QuoteSim.API",
    "version": "v1"
  },
  "paths": {
    "/Car/list": {
      "get": {
        "tags": [
          "Car"
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/BrandDetailedListDto"
                  }
                }
              },
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/BrandDetailedListDto"
                  }
                }
              },
              "text/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/BrandDetailedListDto"
                  }
                }
              }
            }
          }
        }
      }
    },
    "/Car/{variantId}": {
      "get": {
        "tags": [
          "Car"
        ],
        "parameters": [
          {
            "name": "variantId",
```

```
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer",
          "format": "int32"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "content": {
          "text/plain": {
            "schema": {
              "$ref": "#/components/schemas/VariantDetailsDto"
            }
          },
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/VariantDetailsDto"
            }
          },
          "text/json": {
            "schema": {
              "$ref": "#/components/schemas/VariantDetailsDto"
            }
          }
        }
      }
    }
  },
  "/Car/{variantId}/operation/list": {
    "get": {
      "tags": [
        "Car"
      ],
      "parameters": [
        {
          "name": "variantId",
          "in": "path",
          "required": true,
          "schema": {
            "type": "integer",
            "format": "int32"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "text/plain": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/OperationListDto"
                }
              }
            },
            "application/json": {
```

```
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/OperationListDto"
              }
            }
          },
          "text/json": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/OperationListDto"
              }
            }
          }
        }
      }
    }
  },
  "/Car/{variantId}/part/list": {
    "get": {
      "tags": [
        "Car"
      ],
      "parameters": [
        {
          "name": "variantId",
          "in": "path",
          "required": true,
          "schema": {
            "type": "integer",
            "format": "int32"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "text/plain": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
                }
              }
            },
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
                }
              }
            },
            "text/json": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
```

```
              }
            }
          }
        }
      }
    }
  },
  "/Car/operation/part/list": {
    "get": {
      "tags": [
        "Car"
      ],
      "parameters": [
        {
          "name": "operationIds",
          "in": "query",
          "schema": {
            "type": "array",
            "items": {
              "type": "integer",
              "format": "int32"
            }
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "text/plain": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
                }
              }
            },
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
                }
              }
            },
            "text/json": {
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/components/schemas/PartListDto"
                }
              }
            }
          }
        }
      }
    }
  },
  "/Car/part/{partCode}/list": {
    "get": {
```

```
    "tags": [
      "Car"
    ],
    "parameters": [
      {
        "name": "partCode",
        "in": "path",
        "required": true,
        "schema": {
          "type": "string"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "content": {
          "text/plain": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/ManufacturedPartDto"
              }
            }
          },
          "application/json": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/ManufacturedPartDto"
              }
            }
          },
          "text/json": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/components/schemas/ManufacturedPartDto"
              }
            }
          }
        }
      }
    }
  },
  "/Car/maintenanceQuote": {
    "get": {
      "tags": [
        "Car"
      ],
      "parameters": [
        {
          "name": "HourRate",
          "in": "query",
          "schema": {
            "type": "number",
            "format": "double"
          }
        },
        {
```

```
          "name": "Parts",
          "in": "query",
          "schema": {
            "type": "array",
            "items": {
              "$ref": "#/components/schemas/QuoteRequestPartModel"
            }
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "text/plain": {
              "schema": {
                "$ref": "#/components/schemas/QuoteResponseModel"
              }
            },
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/QuoteResponseModel"
              }
            },
            "text/json": {
              "schema": {
                "$ref": "#/components/schemas/QuoteResponseModel"
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "BrandDetailedListDto": {
        "type": "object",
        "properties": {
          "brandId": {
            "type": "integer",
            "format": "int32"
          },
          "brandName": {
            "type": "string",
            "nullable": true
          },
          "models": {
            "type": "array",
            "items": {
              "$ref": "#/components/schemas/ModelListDto"
            },
            "nullable": true
          }
        },
        "additionalProperties": false
      },
      "ManufacturedPartDto": {
        "type": "object",
        "properties": {
```

```
      "manufacturedPartId": {
        "type": "integer",
        "format": "int32"
      },
      "manufacturer": {
        "type": "string",
        "nullable": true
      },
      "referenceUnitPrice": {
        "type": "number",
        "format": "double"
      }
    },
    "additionalProperties": false
  },
  "ModelListDto": {
    "type": "object",
    "properties": {
      "modelId": {
        "type": "integer",
        "format": "int32"
      },
      "modelName": {
        "type": "string",
        "nullable": true
      },
      "releaseYear": {
        "type": "integer",
        "format": "int32"
      },
      "variants": {
        "type": "array",
        "items": {
          "$ref": "#/components/schemas/ModelVariantDto"
        },
        "nullable": true
      }
    },
    "additionalProperties": false
  },
  "ModelVariantDto": {
    "type": "object",
    "properties": {
      "variantId": {
        "type": "integer",
        "format": "int32"
      },
      "variantName": {
        "type": "string",
        "nullable": true
      },
      "engineCapacity": {
        "type": "integer",
        "format": "int32"
      },
      "propulsionType": {
        "type": "string",
        "nullable": true
      },
      "powerOutput": {
        "type": "integer",
```

```
      "format": "int32"
    }
  },
  "additionalProperties": false
},
"OperationListDto": {
  "type": "object",
  "properties": {
    "operationId": {
      "type": "integer",
      "format": "int32"
    },
    "operationName": {
      "type": "string",
      "nullable": true
    },
    "expectedTime": {
      "type": "number",
      "format": "double"
    }
  },
  "additionalProperties": false
},
"PartListDto": {
  "type": "object",
  "properties": {
    "partCode": {
      "type": "string",
      "nullable": true
    },
    "partName": {
      "type": "string",
      "nullable": true
    },
    "partCategoryId": {
      "type": "integer",
      "format": "int32"
    },
    "partCategoryDescription": {
      "type": "string",
      "nullable": true
    },
    "quantity": {
      "type": "integer",
      "format": "int32"
    }
  },
  "additionalProperties": false
},
"QuoteOperationModel": {
  "type": "object",
  "properties": {
    "operationId": {
      "type": "integer",
      "format": "int32"
    },
    "operationName": {
      "type": "string",
      "nullable": true
    },
    "expectedTime": {
```

```
      "type": "number",
      "format": "double"
    },
    "parts": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/QuoteOperationPartModel"
      },
      "nullable": true
    },
    "totalCost": {
      "type": "number",
      "format": "double"
    }
  },
  "additionalProperties": false
},
"QuoteOperationPartModel": {
  "type": "object",
  "properties": {
    "partCode": {
      "type": "string",
      "nullable": true
    },
    "partName": {
      "type": "string",
      "nullable": true
    },
    "partCategoryId": {
      "type": "integer",
      "format": "int32"
    },
    "partCategoryDescription": {
      "type": "string",
      "nullable": true
    },
    "manufacturer": {
      "type": "string",
      "nullable": true
    },
    "referenceUnitPrice": {
      "type": "number",
      "format": "double"
    },
    "quantity": {
      "type": "integer",
      "format": "int32"
    }
  },
  "additionalProperties": false
},
"QuotePartModel": {
  "type": "object",
  "properties": {
    "partCode": {
      "type": "string",
      "nullable": true
    },
    "partName": {
      "type": "string",
      "nullable": true
```

```
    },
    "partCategoryId": {
      "type": "integer",
      "format": "int32"
    },
    "partCategoryDescription": {
      "type": "string",
      "nullable": true
    },
    "manufacturer": {
      "type": "string",
      "nullable": true
    },
    "referenceUnitPrice": {
      "type": "number",
      "format": "double"
    },
    "quantity": {
      "type": "integer",
      "format": "int32"
    },
    "assemblyTime": {
      "type": "number",
      "format": "double",
      "nullable": true
    },
    "totalCost": {
      "type": "number",
      "format": "double",
      "nullable": true
    }
  },
  "additionalProperties": false
},
"QuoteRequestPartModel": {
  "type": "object",
  "properties": {
    "manufacturedPartId": {
      "type": "integer",
      "format": "int32"
    },
    "quantity": {
      "type": "integer",
      "format": "int32"
    },
    "assemblyTime": {
      "type": "number",
      "format": "double",
      "nullable": true
    }
  },
  "additionalProperties": false
},
"QuoteResponseModel": {
  "type": "object",
  "properties": {
    "hourRate": {
      "type": "number",
      "format": "double"
    },
    "operations": {
```

```
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/QuoteOperationModel"
      },
      "nullable": true
    },
    "otherParts": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/QuotePartModel"
      },
      "nullable": true
    },
    "totalCost": {
      "type": "number",
      "format": "double"
    }
  },
  "additionalProperties": false
},
"VariantDetailsDto": {
  "type": "object",
  "properties": {
    "brandId": {
      "type": "integer",
      "format": "int32"
    },
    "brandName": {
      "type": "string",
      "nullable": true
    },
    "modelId": {
      "type": "integer",
      "format": "int32"
    },
    "modelName": {
      "type": "string",
      "nullable": true
    },
    "modelReleaseYear": {
      "type": "integer",
      "format": "int32"
    },
    "modelVariantId": {
      "type": "integer",
      "format": "int32"
    },
    "modelVariantName": {
      "type": "string",
      "nullable": true
    },
    "engineCapacity": {
      "type": "integer",
      "format": "int32"
    },
    "engineCapacityUnit": {
      "type": "string",
      "nullable": true
    },
    "powerOutput": {
      "type": "integer",
```

```
            "format": "int32"
          },
          "propulsionType": {
            "type": "string",
            "nullable": true
          }
        },
        "additionalProperties": false
      }
    }
  }
}
```