UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e de Engenharia

Curso: Engenharia Informática

Cadeira: Projeto/Estágio

Orientado por: Professor Filipe Quintal

Elaborado por: Rúben Rodrigues

Data: 26/06/2021

Índice

1. Introdução

Neste capítulo introdutório, o âmbito e objetivo do projeto, junto com a empresa e o projeto proposto pela mesma, serão introduzidos.

1.1. Motivação e Objetivo

Como aluno da Licenciatura em Engenharia Informática, e muito interessado na área, estou sempre disposto a adquirir novos conhecimentos nesta área, especialmente a nível prático.

De forma a adquirir mais conhecimento, aplicando os conhecimentos adquiridos ao longo do percurso académico, optei por realizar um projeto proposto pelo professor Filipe Quintal e, ao mesmo tempo, ajudar um aluno a fazer o seu mestrado no mesmo campo que eu no segundo semestre do 3º e último ano da licenciatura.

1.2. O problema

1.2.1. Introdução do problema

O aquecimento global é um dos maiores desafios que a humanidade enfrenta atualmente. Um dos fatores mais notáveis que contribuem para o dito desafio é o dióxido de carbono emitido pela maioria dos veículos nas estradas atualmente. Uma das soluções para reduzir estas emissões é tornar o setor de transportes menos poluente, sendo que uma das resoluções passa por adotar os veículos elétricos (adesão mundial na figura abaixo)

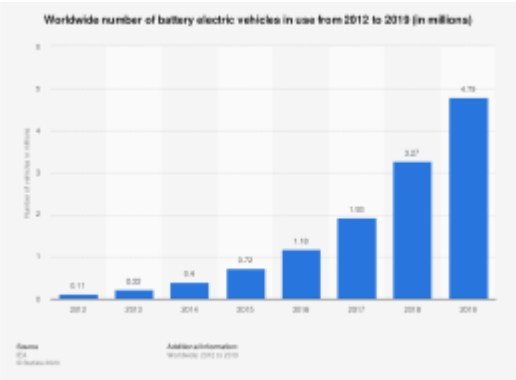

Figura 1 Adesão mundial de veículos elétricos

No entanto, um dos principais fatores que limita a adesão do público aos veículos elétricos é a falta de infraestrutura adequada para os carregar quando comparado com a infraestrutura disponível para reabastecer carros movidos a gás. A infraestrutura elétrica atual está desatualizada e não conseguiria lidar com a procura no evento de o público geral decidir adquirir um veículo elétrico. Portanto, há uma necessidade de encontrar uma maneira de carregar as baterias destes veículos sem sobrecarregar a infraestrutura elétrica.

De momento, o método mais comum de carregar os veículos elétricos, ou qualquer dispositivo com uma bateria, é o *plug-and-charge*, onde o veículo é carregado de forma descontrolada quando ligado a um posto de carregamento. Este método é a causa dos problemas hipotéticos com a infraestrutura atual, lidar com a procura do público sem ficar sobrecarregada, discutido no parágrafo anterior.

Para resolver este problema, o *Smart-Charging* emergiu, tornando o processo de carregar o veículo mais "inteligente" e mais coordenado tomando em conta diversos fatores como o custo da energia e a disponibilidade da infraestrutura elétrica, com base em vários algoritmos. Assim, este tipo de métodos permitem balancear o peso na infraestrutura e reduzir os valores de uso de pico, permitindo assim ao público geral aderir seguramente ao uso de um veículo elétrico.

No entanto, do ponto de vista prático, estudos que envolvem algoritmos de *smart-charging* são difíceis, caros e/ou perigosos, visto que requerem coordenação e gestão complexa de energia elétrica e sistemas de transporte. Assim estes estudos deverão ser conduzidos, pelo menos em primeira instância, com recurso a simulações. E é aqui que entra a solução desenvolvida pelo meu colega.

A solução desenvolvida pelo meu colega surgiu como um estudo de caso para o projeto *SMILE* (*Smart Islands Energy System*), fundado pela União Europeia, que tinha como objetivo introduzir no mercado várias tecnologias relacionadas com o uso de energias renováveis, entre elas novas tecnologias de baterias, veículos elétricos e algoritmos preditivos.

O estudo de caso da solução do meu colega recaiu sobre a implementação de algoritmos *smart-charging* em veículos mais pequenos e com pouco uso de energia, como *scooters*. Por isso, o sistema implementado foi testado com uma companhia na Madeira que oferece *tours* do Funchal, os *Tukxis*, que usam *scooters* tanto movidas a gasolina como elétricas, sendo que as elétricas seriam os veículos mais importantes para este estudo.

Na sua revisão de literatura, ele descobriu que maior parte dos sistemas que permitem este tipo de simulações são soluções para fins académicos, soluções proprietárias e/ou muito limitadas a um certo contexto e também reparou que maior parte destas soluções foram desenvolvidas de uma forma monolítica, isto é, consistem num só repositório de código, o que tornam as soluções mais rígidas e dificultam as alterações que um utilizador gostaria de realizar.

Portanto a solução por ele desenvolvida seria *open-source*, com o objetivo de simular algoritmos de *smart-charging* tendo em conta qualquer contexto e variável conjunto de modelos de dados, sendo facilmente extensível a qualquer contexto de simulação envolvendo energia. E para tornar esta solução de fácil extensão, ele optou por uma arquitetura baseada nos micro serviços, como se vê na figura abaixo.

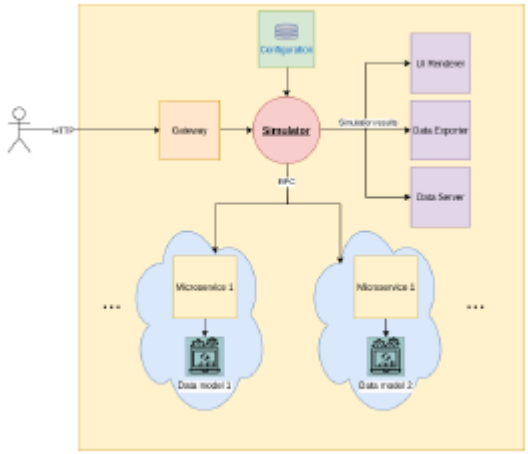

Figura 2 Arquitetura da solução do meu colega

1.2.2. O problema

Do meu ponto de vista, um engenheiro informático inexperiente, testarei o quão extensível a sua solução é, partindo da sua implementação, verificando se esta também é fácil de efetuar, sendo que a implementação será local, e de seguida gerar um modelo de afluência e verificar o quão fácil seria efetuar uma simulação com o modelo de afluência por mim desenvolvido.

Se eu conseguir implementar a sua solução e alterar um dos modelos usados na simulação com facilidade, então confirmarei o objetivo da solução desenvolvida pelo meu colega: a solução realmente é de fácil extensão e “aberta” de um ponto de vista técnico.

2. Solução

Como mencionado na introdução, a solução que desenvolvemos passará por 3 fases principais:

2.1. Implementação em servidor local

A primeira fase da solução será implementar o modelo de simulação num servidor local e verificar que este funciona corretamente sem alterações ao modelo. Por recomendação do orientador e do meu colega, este servidor será implementado no sistema operativo *Linux*, mais especificamente a distribuição *Ubuntu*, assim aproximando-se do que seria uma implementação no mundo real, num servidor remoto, visto que maior parte dos servidores serem executados no sistema operativo *Linux*.

Para implementar o servidor em *Linux* será necessário instalar vários componentes, nomeadamente: *Docker* visto que a solução desenvolvida pelo meu colega usa *containerization*, onde cada serviço tem o seu próprio recipiente, de modo a garantir a flexibilidade da solução e *Pack* que permite instalar outros requisitos necessários para executar o código como *Tensorflow*, *Flask*, entre outros, facilitando o processo de compilação dos diferentes serviços.

Portanto, como engenheiro inexperiente, avaliarei este processo de implementação de acordo com a sua facilidade visto que se eu considerar um processo relativamente fácil, engenheiros mais experientes acharão o processo de implementação ainda mais fácil, garantindo assim o objetivo de portabilidade que o meu colega pretende atingir.

2.2. Pesquisa da afluência de passageiros ao serviço

Visto o meu colega desenvolveu a sua solução de simulação de algoritmos *smart-charging* para o contexto de avaliar como estes poderiam influenciar o serviço de *Tuks* na Madeira, gerido pela companhia *Tukxis*, que transporta prioritariamente turistas, o professor orientador recomendou que eu efetuasse uma pesquisa sobre a chegada de turistas ao Porto do Funchal, visto que estes *Tuks* trabalham prioritariamente com passageiros que chegam ao Porto, para preparar a última etapa da solução.

Por isso, na segunda fase da solução, procurarei por dados disponíveis *online* e gratuitamente que me permitam determinar em que horas os navios de cruzeiro mais comumente chegam ao Porto do Funchal e, por consequência, em que horas o maior número de passageiros chega ao Porto, de modo a construir um modelo usando o *Tensorflow* com base nestes dados

2.3. Implementação de um novo modelo de afluência

Na última fase da solução, que testará a flexibilidade da solução proposta pelo meu colega, irei ajustar o modelo de afluência usado na simulação, com recurso ao *Tensorflow* para a criação de um novo modelo com base nos dados encontrados na etapa anterior.

Tal como na implementação do servidor, a partir do meu ponto de vista como engenheiro inexperiente, sem experiência com o serviço *Tensorflow*, se eu conseguir implementar, com relativa facilidade, um novo modelo de afluência de passageiros ao serviço dos *Tuks*, então poderei concluir que a solução desenvolvida pelo meu colega realmente cumpre com o objetivo de flexibilidade que pretende atingir, sendo que, se é possível alterar um modelo, podemos alterar qualquer dos modelos usados na simulação.

Aprofundar o que vou fazer, a implementação do modelo num servidor local, a pesquisa de um modelo de afluência e a implementação do modelo recorrendo ao *Tensorflow*

3. Revisão de Literatura

3.1. Soluções semelhantes

Nesta secção serão apresentados simuladores parecidos à solução que foi desenvolvida pelo meu colega e serão discutidos os pontos fortes e fracos destas soluções e como a solução do meu colega pode oferecer algo que estas soluções não oferecem ao utilizador.

3.1.1. PerMod

PerMod (*Performance Simulation Model for PV-Battery Systems*) é um sistema que permite analisar o desempenho de um sistema de armazenamento de energia com recurso a baterias especialmente desenvolvidas (*Battery Energy Storage System - BESS)*. Desenvolvido em MATLAB, tem em conta vários parâmetros de *input* relacionados com a perda de energia, por exemplo por conversão ou em *standby*, que são processados no simulador; a esquemática da estrutura do *PerMod* pode ser observada na figura abaixo. [1]

Uma das limitações desta ferramenta é que a degradação das baterias é ignorada e, visto que foi desenvolvida com recurso ao MATLAB, a solução é mais díficil de alterar.

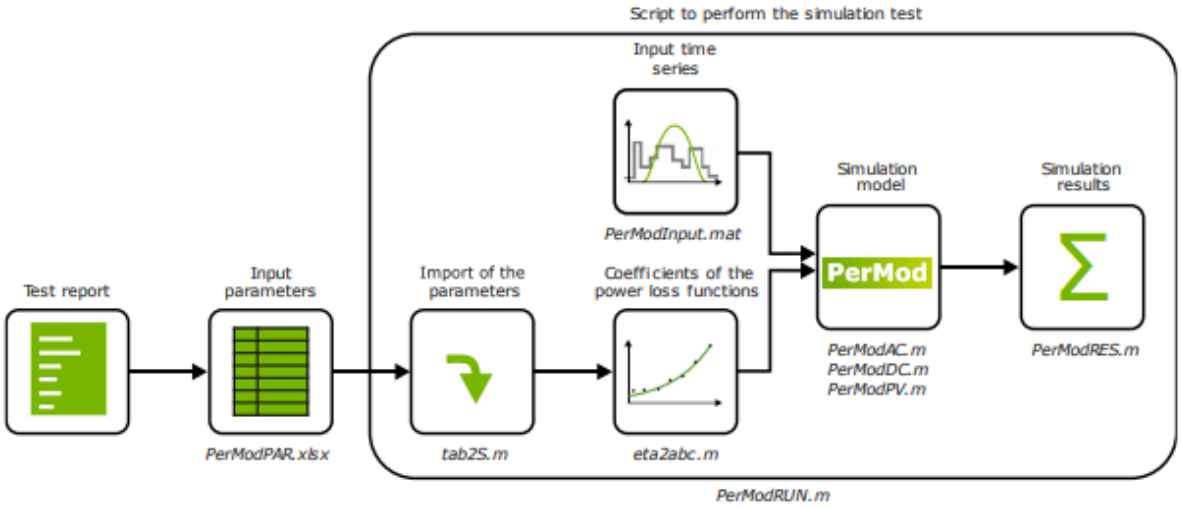

Figura 3 Esquema da estrutura do PerMod

3.1.2. *SimSES*

No mesmo ramo de avaliar *BESSs*, o *SimSES* é um sistema que permite simular o desempenho de sistemas de armazenamento de energia estacionários. Ao contrário da solução anterior, este simulador permite efetuar análises tanto do ponto de vista técnico como do ponto de vista económico. Ou seja, permite analisar a eficiência e o impacto de certos algoritmos de controlo no armazenamento (ponto de vista técnico) e permite ao utilizador comparar diferentes

componentes eletrónicos de modo a maximizar a lucratividade do sistema (ponto de vista económico). [2]

Na fase inicial do sistema, também este tinha sido desenvolvido com recurso ao MATLAB. No entanto, acabou por ser portado para *Python* e tornado *open-source*, tornando a solução mais flexível. A visão global do modelo pode ser observada na figura abaixo [2]

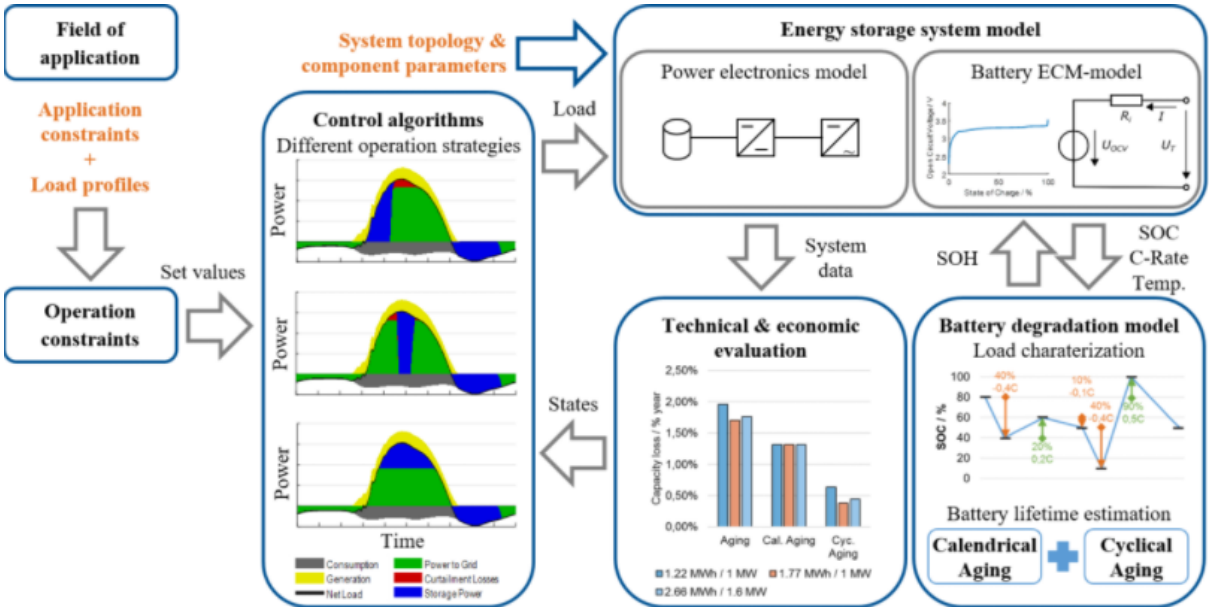

Figura 4 Overview do modelo usado no SimSES

3.1.3. SAM

Um sistema de simulação orientado ao contexto das energias renováveis chamado *System Advisor Model* (*SAM*) foi desenvolvido pelo *National Renewable Energy Laboratory* (*NREL*) dos Estados Unidos da América. Este consiste de uma aplicação para o computador que permite ao utilizador simular projetos de energia renovável e examinar os fatores tecnológicos e económicos, tal como desempenho, métricas financeiras e possibilidade de incentivos normalmente atribuídos a projetos que envolvem energias renováveis. Um exemplo do resultado duma simulação pode ser observado na figura abaixo. [3], [4]

Esta plataforma é *open-source* e foi desenvolvida com recurso ao C e C++, que apresenta tanto positivos, a plataforma está mais otimizada e é mais portável, e negativos, a programação é de mais baixo nível o que implica mais trabalho no evento de alterações e menor capacidade de reutilização. No entanto, também oferece um *Software Development Kit* (*SDK*) que permite criar módulos de simulação adicionais usando outras linguagens de programação, como *Java*, *Python*, MATLAB, *PHP*, entre outras. [4]

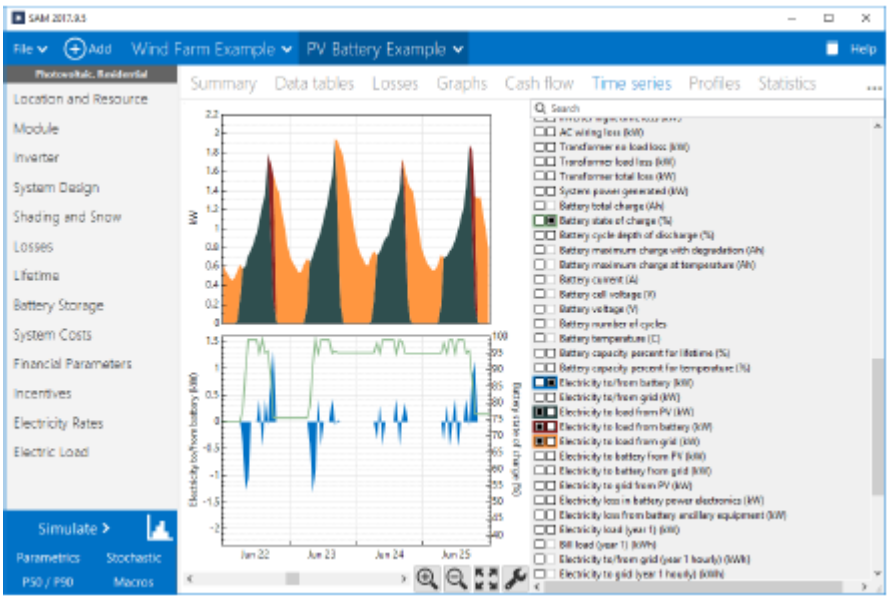

Figura 5 Exemplo de output

3.1.4. V-Elph

O sistema de simulação *V-Elph*, desenvolvido pela Universidade *Texas A&M* com recurso ao *MATLAB* e *Simulink*, permite analisar e comparar diferentes configurações e/ou estratégias de gestão de energia em veículos elétricos e veículos híbridos. O utilizador pode fazer simulações com base num modelo de componentes que tenha selecionada e o sistema gera representações gráficas de modo que o utilizador possa interpretar mais facilmente os resultados. [5]

Tal como o sistema *PerMod*, também desenvolvido com recurso ao *MATLAB*, estes sistema também é muito rígido e limitado. Nas figuras abaixo pode-se ver um exemplo de um modelo de componentes e um exemplo dos resultados gráficos do sistema.

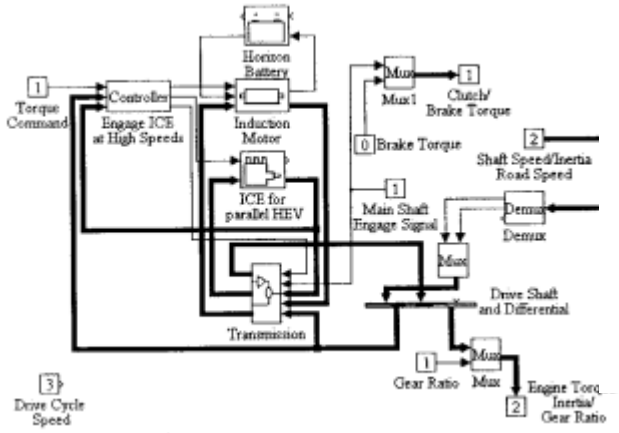

Figura 6 Exemplo de um modelo de componentes de um veículo híbrido

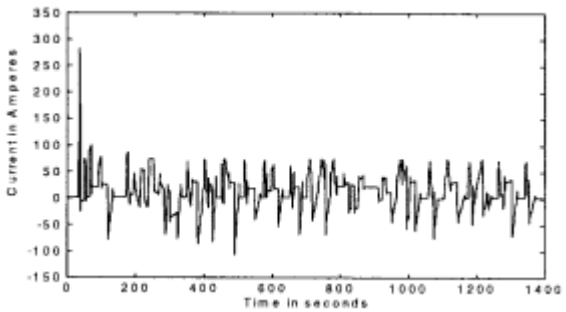

Figura 7 Exemplo de output

3.1.5. BLAST

Por fim, o sistema *Battery Lifetime Analysis and Simulation Tool* (*BLAST*), desenvolvido pelo *NREL*, tal como o *SAM*, apresenta 3 componentes: o *BLAST-V* que permite avaliar veículos elétricos e híbridos; o *BLAST-S* que permite avaliar o armazenamento de energia em aplicações estáticas; e o *BLAST-BTM Lite* que permite avaliar o armazenamento de energia em aplicações *behind-the-meter* (isto é, armazenamento no lado do utilizador, como na sua residência). O sistema *BLAST* como um todo permite prever o comportamento de uma bateria com base nas suas propriedades (como degradação e desempenho térmico), o quanto foi utilizada e dados climatéricos. [6]

Em termos de flexibilidade, é possivelmente a solução mais restrita de todas as discutidas nesta secção, pois o utilizador só tem acesso ao seu instalador (e, por consequência, os ficheiros binários). Uma ilustração da modelação e um exemplo de *output* deste sistema pode ser observado abaixo.

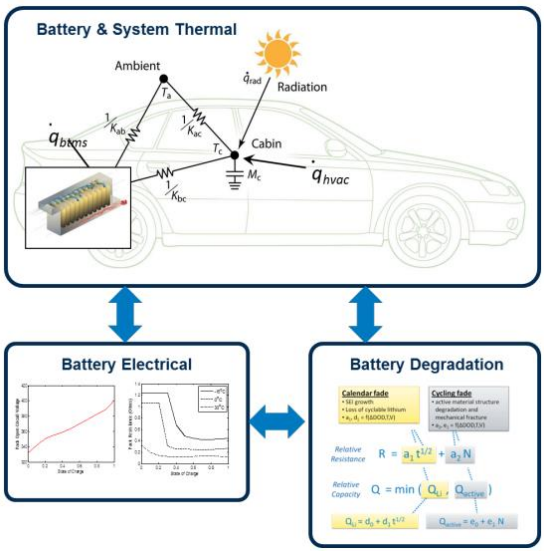

Figura 9 Ilustração do modelo

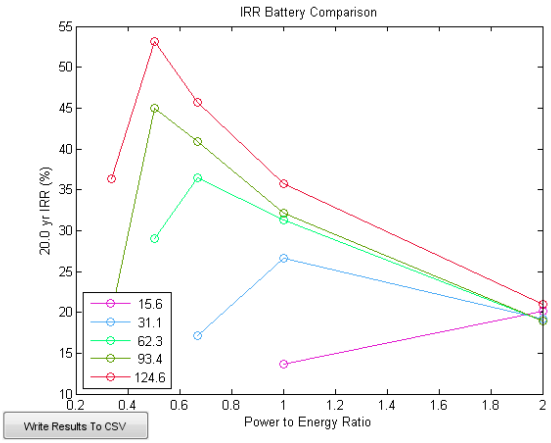

Figura 8 Exemplo de output

3.2. Ferramentas

O processo de desenvolvimento utilizado para desenvolver a solução foi um processo ágil, onde a colaboração entre o aluno e os orientadores foi contínua ao longo do desenvolvimento, permitindo uma maior colaboração com o "cliente".

O processo ágil é uma abordagem iterativa e incremental ao processo de desenvolvimento que divide este em ciclos chamados iterações onde em cada iteração uma pequena parte da implementação do módulo é adicionada. Esta funcionalidade é testada e validada com o cliente de forma a assegurar que foi implementada corretamente e, graças à participação conjunta do cliente com o aluno, permite encontrar problemas e corrigi-los antes que seja tarde demais.

Nas secções abaixo detalha-se as tecnologias usadas e as suas caraterísticas.

3.2.1. Python

Python é uma linguagem de programação de alto nível, orientada a objetos que pode ser usada para escrever *software* para vários tipos de aplicação. Python foi desenhada com a facilidade de ler código em mente, de modo que os programadores pudessem escrever código lógico e limpo em qualquer tipo de projetos, quer pequenos ou grandes. [7]

Python foi concebido no final da década de 1980 por Guido Van Rossum e continua a ser atualizada com novas versões até ao presente dia. [7]

Python é uma das linguagens de programação mais populares atualmente; segundo o *Stack Overflow Developer Survey* de 2020, 44.1% dos programadores indicaram que a usam regularmente e destes 66.7% indicaram que têm interesse em continuar a usá-la.

```
n = int(input('Type a number, and its factorial will be printed: '))

if n < 0:
    raise ValueError('You must enter a non negative integer')

factorial = 1
for i in range(2, n + 1):
    factorial *= i

print(factorial)
```

Figura 10 Excerto de código

3.2.2. Tensorflow

Python tem imensas aplicações, mas uma das suas aplicações principais é no campo de *machine learning* e inteligência artificial graças ao número e qualidade das bibliotecas e *frameworks* disponíveis em *python* para este campo. Uma dessas é o *Tensorflow*.

Tensorflow é uma biblioteca grátis e *open-source* destinada a *machine learning*, mais especificamente, no treino e na inferência de redes neurais artificiais. Foi desenvolvida pela equipa *Google Brain* e lançada sob a licença *Apache* 2.0 em 2015. [8]

Tensorflow facilita a construção de modelos através do uso de *APIs* de alto nível intuitivos e garante a produção de modelos de *machine learning* robustos quer na *cloud*, no *browser* ou no dispositivo. [9]

3.2.3. Nameko, microsserviços e RPCs

Nameko é uma *framework* destinada a construir microsserviços em *Python*. Um serviço construído com base neste *framework* poderá responder a mensagens *RPC* (*Remote Procedural Call*), associar eventos a ações e "ouvir" eventos de outros serviços e suporta métodos simples de *HTTP GET* e *POST*. [10]

Uma arquitetura que recorre a microsserviços constrói uma aplicação como uma coleção de serviços com mínimas ligações entre eles, isto é, eles não têm conhecimento, ou têm pouco conhecimento, das definições de outros componentes. [11]

Uma *Remote Procedural Call* (*RPC*) acontece quando um computador causa uma rotina a ser executada noutro espaço de endereçamento (normalmente outro computador na mesma rede), rotina que está escrita como se fosse executada localmente, isto é, o programador não tem de programar os detalhes da interação remota com o computador que fez a *RPC*. [12]

3.2.4. Flask

Flask é uma micro *framework* escrita em *Python*, e para *Python*. É classificada como uma micro *framework* porque não requer outras bibliotecas ou ferramentas. *Flask* não tem camada de abstração de base de dados, que permitiria unificar a comunicação entre a aplicação e bases de dados, e não tem validação de formulários. No entanto, *Flask* apoia extensões que podem adicionar capacidades à aplicação, como se fossem implementadas diretamente pelo *Flask*. [13]

3.2.5. RabbitMQ

RabbitMQ é um *software open-source* que desempenha a tarefa de *message broker*. [14] Um *message broker* é um módulo intermediário entre duas aplicações que trocam mensagens que traduz uma mensagem do protocolo de mensagem usado pelo remetente para o protocolo de mensagem usado pelo recipiente. O *message broker* trata de validar as mensagens, transformá-las e encaminhá-las, minimizando o conhecimento que as aplicações têm de ter uma da outra, implementando a boa prática no desenho de software conhecida como *decoupling*. [15]

```python
#!/usr/bin/env python3
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(host="localhost"))
channel = connection.channel()
channel.queue_declare(queue="hello")
channel.basic_publish(exchange="", routing_key="hello", body="Hello World!")
print(" [x] Sent 'Hello World!'")
connection.close()
```

Figura 11 Excerto de código que envia mensagem

```python
#!/usr/bin/env python3
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(host="localhost"))
channel = connection.channel()
channel.queue_declare(queue="hello")
print(" [*] Waiting for messages. To exit press Ctrl+C")


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)


channel.basic_consume(queue="hello", on_message_callback=callback)
channel.start_consuming()
```

Figura 12 Excerto de código que recebe mensagem

3.2.6. Docker

Docker é um serviço que permite entregar *software* em pacotes denominados recipientes (*containers*). Estes recipientes estão isolados uns dos outros e agrupam o seu próprio *software*, bibliotecas e ficheiros de configuração, mas podem comunicar entre si através de canais definidos. Estes recipientes são virtuais de modo a poderem correr em qualquer computador, independentemente do sistema operativo do computador. [16]

3.2.7. OpenUI

Para o *front-end* do cliente *Web* do simulador, foi utilizado a *framework OpenUI*. Esta *framework* permite mais estilos e extender as funcionalidades de vários controlos como *dropdowns*, *checkboxes*, entre outros. [17]

3.3. Sumário

A abordagem ágil tomada para o desenvolvimento desta solução deverá ser a correta visto que muitas destas ferramentas serão novas para o aluno por o ciclo de implementar funcionalidade, testar e conferir com o orientador será vantajoso para o desenvolvimento deste módulo.

Relativamente às ferramentas utilizadas, embora estas tenham sido todas especificadas como as ferramentas a utilizar na proposta de projeto, pela pesquisa efetuada, parecem as ferramentas adequadas que tornarão o desenvolvimento deste módulo mais fácil.

4. Implementação

Nesta secção será abordado, de uma forma mais aprofundada, a solução ao problema proposto. Serão explicadas as demais fases do projeto, desde a implementação do projeto desenvolvido pelo meu colega localmente, para verificar a facilidade de implementação, até ao desenvolvimento de outro algoritmo para simular a afluência de pessoas ao serviço.

4.1. Implementação do serviço

Antes de sequer tentar implementar localmente o projeto, foi necessário instalar os serviços necessários para executar o projeto. De acordo com a recomendação do meu colega, para facilitar a implementação do projeto, instalei o sistema operativo Ubuntu, uma distribuição Linux, e de seguida clonei todos os repositórios necessários para implementar o projeto.

Após todos os repositórios estarem disponíveis localmente, no meu computador, procedi à instalação do *Docker*, seguindo as instruções disponibilizadas no tutorial de instalação. Esta instalação, tal como a instalação do sistema operativo, decorreu sem problemas, embora, cada vez que o computador era reiniciado, o *daemon socket* perdia as permissões necessárias para fazer os *makefile* dos repositórios sem problemas.

Com o *Docker* instalado, tentei fazer o *makefile* de um dos repositórios, mas debati-me com um erro. Após conversar com o meu colega ele apontou-me que faltava instalar mais um serviço, o *Pack*, para facilitar a "construção" dos *containers*.

Após conseguir executar o *makefile* dos 3 repositórios que não eram serviços, quando tentei compilar os repositórios dos serviços debati-me com outro problema, desta vez um problema de compatibilidade de versões. A versão do *Python* a ser instalada (3.9.5), atualmente a mais recente, não era compatível com a versão do *Tensorflow* que precisava ser instalada (1.5.0). Outra vez, após reportar isto ao meu colega, ele rapidamente arranjou a solução (alterar a compilação tal que uma versão do *Python* mais antiga, compatível com o *Tensorflow*, fosse instalada) e, de seguida, consegui executar o *makefile* dos repositórios restantes.

Com todos os repositórios a conseguirem ser compilados, após o meu colega informar-me do *URL* onde o site estaria quando todos os *packs* tivessem construídos, acedi ao site, mas ainda não estava funcional, ainda não conseguia fazer a simulação. Depois de falar com o meu colega descobri que já só me faltava um ficheiro de configuração; quando meti o ficheiro de configuração no local certo, a implementação estava completada e podia correr a simulação.

4.2. Pesquisa da afluência de passageiros ao porto do Funchal

Após implementar o projeto e garantir que este já se encontrava funcional, podia avançar para a próxima grande fase do meu projeto, redesenhar o algoritmo de afluência de pessoas ao serviço de *Tuks* que a simulação usava.

Mas, antes de sequer escrever uma linha de código, de acordo com a recomendação do orientador Filipe, decidi pesquisar e ver se conseguia encontrar dados que permitissem melhor estimar a afluência dos passageiros ao porto do Funchal, que, por conseguinte, usariam o serviço de *Tuks*.

Após encontrar muitos sites cujo principal funcionalidade é permitir seguir navios, mas que não apresentavam muito na qualidade de dados sobre a quantidade de navios que tinham chegado ao porto, encontrei o site da APRAM (Administração dos Portos da Região Autónoma da Madeira) que permitia obter dados sobre as horas de chegada de navios ao porto.

Os dados fornecidos pelo site da APRAM eram precisamente o que estava à procura, visto que a afluência de pessoas ao porto na simulação estava dividida pelas 24 horas do dia, isto é, para cada hora podia-se especificar quantos passageiros chegavam.

A questão agora era o quão grande deveria ser a amostra, em termos de quantos dias estariam na amostra de dados, e que período de tempo deveria ser avaliado. Obviamente, o período de tempo não deveria ser muito recente, visto que, devido à situação pandémica, viagens, não só por navio, estão em número muito inferior quando comparado com os números antes da pandemia.

Por isso, para tentar obter uma amostra que representasse a afluência esperada em tempos considerados "normais", isto é, um tempo em que as viagens estivessem a decorrer normalmente, e para obter uma amostra de tamanho considerável, decidi que a amostra deveria conter todos os navios que chegaram ao porto do Funchal no ano 2019.

Com os dados das viagens exportados para uma folha de cálculo, um dos dados importantes sobre as viagens era a companhia que a tinha efetuado. Visto que nem todas as viagens seriam de navios com passageiros, achei adequado retirar dos dados as viagens efetuadas por navios de mercadorias, por isso, após alguma pesquisa no site da APRAM sobre os dados obtidos, identifiquei as companhias cujas viagens eram predominantemente efetuadas por navios de cargo e retirei as viagens efetuadas por estas companhias dos dados.

Agora que os dados eram predominantemente viagens efetuadas por navios de cruzeiro, considerei que a amostra já me permitiria obter uma boa estimação da afluência de passageiros ao porto do Funchal por hora. De modo a obter esta estimação, obtive o número de chegadas ao porto do Funchal por hora do dia e de seguida dividi estes valores por 365, o número de dias no ano, assim obtendo um valor entre 0 e 1 para cada hora do dia que representava a probabilidade de chegar um navio de passageiros nessa dada hora (quanto mais próximo de 1 mais provável seria chegar um navio nessa hora).

Junto com os valores da afluência ao porto por hora, calculei o desvio padrão dos dados de modo a introduzir alguma variação na simulação e não ser só os valores esperados da afluência a ditar quando os passageiros chegariam na simulação, de modo a não ser *hard-coded* como o modelo da afluência estava implementado.

Hora do dia	Número de chegadas	Estimação da probabilidade de chegar um navio
0	42	0.115068493150685
1	4	0.010958904109589
2	8	0.021917808219178
3	1	0.002739726027397
4	8	0.021917808219178
5	4	0.010958904109589
6	47	0.128767123287671
7	110	0.301369863013699
8	53	0.145205479452055
9	24	0.065753424657534
10	142	0.389041095890411
11	8	0.021917808219178
12	27	0.073972602739726
13	19	0.052054794520548
14	15	0.041095890410959
15	5	0.013698630136986
16	13	0.035616438356164
17	9	0.024657534246575
18	3	0.008219178082192
19	276	0.756164383561644
20	127	0.347945205479452
21	114	0.312328767123288
22	15	0.041095890410959
23	13	0.035616438356164

Tabela 1 Tabela resultante da pesquisa de afluência

4.3. Implementação de um novo modelo

Com os dados recolhidos, a próxima fase seria alterar a implementação atual do modelo da afluência, que estava *hard-coded*, de modo a fornecer um pouco mais flexibilidade na definição do modelo da afluência.

Devido à minha relativa inexperiência com *Tensorflow*, decidi mais adequado, após uma reunião com o professor orientador, não tentar construir um novo modelo baseado nos meus dados, mas sim verificar se eu poderia implementar um modelo *Tensorflow* de exemplo de modo a verificar se sempre seria fácil simplesmente alterar o modelo usado no serviço e assim garantir que a solução desenvolvida é mesmo flexível.

A criação deste modelo exemplo não foi difícil, o tutorial que usei era extenso e documentado o suficiente para permitir-me, sem experiência na tecnologia, criar um modelo *Tensorflow*, e após testá-lo, embora não devolvesse *outputs* ou recebesse *inputs* adequados ao problema em questão, era um modelo funcional de *Tensorflow* que poderia ser importado pelo serviço de afluência.

4.4. Alteração do modelo de afluência

Com um modelo *Tensorflow* de exemplo criado, a última etapa seria importar este no serviço de afluência de modo a verificar se o projeto do meu colega realmente é flexível e permite a alteração fácil do modelo *Tensorflow* usado.

Embora não tenha conseguido implementar esta última etapa do projeto, como discutirei nas secções à frente, esta não seria de muita dificuldade para um engenheiro mais experiente na tecnologia *Tensorflow*.

5. Avaliação

O processo, para mim, como engenheiro inexperiente, foi de relativa facilidade, não encontrei nenhum problema demasiado difícil de resolver que um engenheiro mais experiente com as tecnologias utilizadas não conseguisse determinar facilmente, maior parte dos problemas deveram-se até pelo facto de eu não ter as tecnologias requeridas para compilar o projeto e uma simples troca de mensagens com o meu colega apontou-me para a tecnologia que necessitava.

A implementação num servidor local correu relativamente bem, obter os dados relativos à afluência de passageiros ao porto do Funchal também foi de relativa facilidade, dados estes que serviriam como base de um modelo de *Tensorflow*.

A criação do modelo exemplo também não foi de muita dificuldade, mas criar um modelo *Tensorflow* com base nos dados por mim recolhidos provou ser de maior dificuldade e acabei por não conseguir fazer um modelo com base nos meus dados.

No entanto, conseguir carregar um novo modelo, até mesmo de teste, para verificar a flexibilidade da solução, foi a etapa pela qual eu não consegui prosseguir, e discutirei o porquê da falha de implementação desta etapa na secção seguinte.

6. Discussão

Infelizmente, não consegui atingir todos os objetivos definidos para o projeto. A etapa que ficou por implementar foi a etapa de importar um novo modelo para testar a facilidade com que um dos modelos usados na simulação poderia ser trocado, de modo a garantir a flexibilidade da solução desenvolvida pelo meu colega.

Aprendi muito sobre as competências necessárias para implementar um servidor localmente e como pesquisar os dados requeridos para construir um modelo de *Tensorflow*, embora não tenha conseguido fazer dito modelo.

Embora não tenha conseguido completar o projeto, não acredito que isto deveu-se à escolha errónea de tecnologias por parte do meu colega, acredito que ele escolheu as tecnologias adequadas, mas foi a minha inexperiência com as tecnologias, nomeadamente a tecnologia *Tensorflow*, que não me permitiu completar o projeto.

Por isso acredito que um engenheiro mais experiente, em geral e com as tecnologias utilizadas, teria conseguido implementar o projeto desenvolvido pelo meu colega. A não implementação de um novo modelo no serviço de afluência deveu-se a falha minha e não a nenhuma falha no projeto do meu colega.

7. Conclusão

Para concluir, não só este projeto, mas a licenciatura em geral, aprendi muito sobre vários aspetos de Engenharia Informática, desde como certos aspetos de programação em alto nível são convertidos em aspetos de baixo nível e traduzidos para o hardware, até diversas linguagens de programação e tecnologias e diversos conceitos relacionados com a programação.

Não só aprendi diversos conceitos teóricos sobre a área de Engenharia Informática, tal como aprendi como ser autónomo e assim resolver os vários problemas que foram propostos pelos diversos trabalhos práticos que efetuei durante o processo da licenciatura aplicando os diversos conceitos teóricos, incluindo este projeto.

Por fim, acredito que adquiri as competências necessárias para ser um membro valioso do mercado de trabalho graças à licenciatura em Engenharia Informática e que aprofundar alguns dos conceitos da licenciatura no mestrado só seria vantajoso para alargar as minhas oportunidades no mercado de trabalho.

8. Referências

[1] “PerMod: Performance Simulation Model for PV-Battery Systems | pvspeicher.htw-berlin.de.” https://pvspeicher.htw-berlin.de/permod/ (accessed Jun. 02, 2021).

[2] “SimSES - EES.” https://www.ei.tum.de/ees/simses/ (accessed Jun. 02, 2021).

[3] “Home - System Advisor Model (SAM).” https://sam.nrel.gov/ (accessed Jun. 02, 2021).

[4] J. M. Freeman *et al.*, “System Advisor Model (SAM) General Description (Version 2017.9.5),” NREL/TP--6A20-70414, 1440404, May 2018. doi: 10.2172/1440404.

[5] K. L. Butler, M. Ehsani, and P. Kamath, “A Matlab-based modeling and simulation package for electric and hybrid electric vehicle design,” *IEEE Trans. Veh. Technol.*, vol. 48, no. 6, pp. 1770–1778, Nov. 1999, doi: 10.1109/25.806769.

[6] J. Neubauer, “Battery Lifetime Analysis and Simulation Tool (BLAST) Documentation,” NREL/TP--5400-63246, 1167066, Dec. 2014. doi: 10.2172/1167066.

[7] “Python (programming language),” *Wikipedia*. May 12, 2021. Accessed: May 12, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1022766780

[8] “TensorFlow,” *Wikipedia*. May 04, 2021. Accessed: May 12, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=TensorFlow&oldid=1021419449

[9] “TensorFlow,” *TensorFlow*. https://www.tensorflow.org/ (accessed May 12, 2021).

[10] “What is Nameko? — nameko 2.12.0 documentation.” https://nameko.readthedocs.io/en/stable/what_is_nameko.html (accessed May 14, 2021).

[11] “Microservices,” *Wikipedia*. May 06, 2021. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Microservices&oldid=1021710703

[12] “Remote procedure call,” *Wikipedia*. May 01, 2021. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Remote_procedure_call&oldid=1020915529

[13] “Flask (web framework),” *Wikipedia*. May 08, 2021. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1022106487

[14] “RabbitMQ,” *Wikipedia*. Mar. 18, 2021. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=RabbitMQ&oldid=1012832357

[15] “Message broker,” *Wikipedia*. Dec. 16, 2020. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Message_broker&oldid=994556603

[16] “Docker (software),” *Wikipedia*. Apr. 25, 2021. Accessed: May 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=1019840030

[17] “Home.” https://open-ui.org/ (accessed May 14, 2021).