Capítulo 1

Introdução

No contexto do ensino superior, tem-se dado cada vez mais importância à integração de ferramentas tecnológicas que tornem o processo de aprendizagem mais prático, como é o caso do simulador utilizado pelos alunos do ISCAL, *International Corporate Management* [24] (que iremos referir como plataforma de simulação). Esta ferramenta permite que os estudantes apliquem, de forma interativa, os conhecimentos adquiridos ao longo do curso, colocando-os em situações de tomada de decisão semelhantes às que enfrentariam num ambiente real. Deste modo, o simulador não só reforça os conteúdos teóricos, como também reforça o desenvolvimento das competências aprendidas.

Neste contexto, a análise eficiente de dados torna-se essencial para a tomada de decisões e tem impacto na avaliação final dos alunos, no entanto, a complexidade e a falta de uma ferramenta visual que ajude a perceber as informações apresentadas pode representar um desafio grande para os estudantes.

1.1 Motivação

O presente projeto surgiu da necessidade dos alunos do ISCAL que utilizam a plataforma *International Corporate Management* da empresa *Marketplace Simulations*, que é um simulador de negócios internacionais e que iremos descrever no capítulo 3.3.1. Embora a plataforma apresente toda a informação necessária à tomada de decisões, esses dados estão espalhados em múltiplas

páginas e apresentados em tabelas, com poucas funcionalidades de visualização. Esta limitação obriga os alunos a alternar entre páginas, copiar dados manualmente ou criar folhas de cálculo externas, comprometendo a eficiência e a análise da informação.

1.2 Objetivos

A aplicação proposta neste relatório pretende ajudar nesse sentido, disponibilizando uma interface que permite aos utilizadores carregar dados retirados da plataforma de simulação e tornar esses ficheiros em visualizações que podem ser consultadas e manipuladas. A aplicação deve permitir aos utilizadores:

- Criar uma conta na plataforma que suporte a persistência da informação carregada.
- Carregar ficheiros exportados da plataforma de simulação.
- Visualizar os dados em gráficos interativos.
- Conseguir ter uma experiência de utilização intuitiva e fácil.

Pretende-se também que a aplicação adote uma arquitetura simples de manter e que utilize os mesmos conceitos que a plataforma de simulação, dando importância aos seguintes itens:

- Normalização e transformação automática de dados carregados.
- Facilidade na gestão de ficheiros, com o objetivo de oferecer uma interface intuitiva para os utilizadores finais.
- Organizar a informação por utilizador, garantindo que cada utilizador apenas consegue consultar a sua informação carregada.
- Adotar um modelo de funcionamento semelhante à plataforma de simulação, de modo a tornar a utilização mais fácil e garantir que a nossa aplicação tenha fronteiras claras de utilização.

Capítulo 2

Trabalho Relacionado

O projeto que desenvolvemos insere-se num contexto mais geral de ferramentas pedagógicas e de Sistemas de Apoio à Decisão (SAD), ainda que neste caso concreto, em ambientes simulados no ensino superior. No âmbito do ISCAL, a utilização da plataforma *International Corporate Management* [24] permite aos estudantes desenvolver competências práticas em ambientes virtuais de negócios, simulando o funcionamento de mercados reais. A necessidade de suporte à análise e simulação motivou o desenvolvimento de outras ferramentas auxiliares, com destaque para um projeto também realizado em parceria com o ISCAL, focado em simulações de modelos económicos.

Esse projeto, embora partilhe uma motivação semelhante, segue uma abordagem diferente. Em particular, a aplicação referida acima permite simular cenários específicos com base em *inputs* manuais, o que pode ser útil para quando se procura prever resultados com foco muito concreto (por exemplo, simular o impacto de uma única variável nos resultados). A nossa plataforma foca-se numa outra vertente, pois pretende ser uma ferramenta para auxiliar o ensino de forma prática.

O projeto recorre à extração de dados diretamente da plataforma de simulação, e à apresentação dos mesmos. Não será uma plataforma de simulação, mas sim uma ferramenta para auxiliar a tomada de decisão.

Sistemas de apoio à decisão

Um SAD é uma aplicação, ou um conjunto de aplicações, pensadas para ajudar utilizadores a tomar decisões mais informadas. Ao contrário de siste-

mas que tomam decisões de forma autónoma, um SAD funciona como uma ferramenta que mostra aos utilizadores os dados e análises para que estes possam avaliar alternativas, prever resultados e escolher a melhor ação possível.

Na prática, um SAD é responsável por recolher, organizar e processar grandes volumes de dados, muitas vezes com origem em várias fontes, e transformá-los em informação relevante. Esta informação é normalmente apresentada através de tabelas, indicadores ou representações visuais como gráficos de barras. Um aspeto importante é a capacidade de aplicar filtros e explorar diferentes cenários, o que permite ao utilizador testar hipóteses, identificar padrões e comparar dados.

Estes sistemas são particularmente úteis em contextos onde há grande complexidade como por exemplo, na gestão empresarial, em análises financeiras, ou na análise de mercados. Num ambiente académico, como é o caso da plataforma usada neste projeto, um SAD ajuda os estudantes a perceber que decisões tomar, mostrando dados históricos para que possam ser identificadas tendências ou padrões.

É importante reforçar que o SAD não substitui o utilizador. Em vez disso, aumenta a capacidade do utilizador de interpretar a informação disponível e tomar decisões, reduzindo o risco de erro e aumentando a confiança. Estes sistemas podem ser plataformas criadas especificamente para o efeito, ou podem ser derivadas de outras plataformas como *PowerBI*[21], *Tableau*[27], *Grafana*[18], ou outras ferramentas de visualização de dados.

Capítulo 3

Modelo Proposto

Neste capítulo vamos descrever o modelo que serviu de base à implementação da aplicação, com foco na forma como os dados são organizados, processados e apresentados ao utilizador. A estrutura proposta resulta das necessidades identificadas durante a análise dos ficheiros obtidos da plataforma de simulação, bem como dos requisitos definidos com os orientadores do projeto.

O modelo tem como objetivo garantir que a aplicação desenvolvida corresponde as necessidades identificadas. Para isso, foram definidos vários requisitos, e foi estudado como iriamos apresentar os dados tendo em conta os ficheiros recebidos, entre outras decisões que foram tomadas durante o desenvolvimento do projeto.

Nos próximos parágrafos vamos então detalhar cada uma das partes do modelo proposto, começando pelos requisitos funcionais e não funcionais que foram definidos.

3.1 Requisitos

Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades obrigatórias que a aplicação deve oferecer aos utilizadores. No contexto deste projeto, definem as ações que a aplicação deve ser capaz de executar.

Para o projeto, identificamos os seguintes requisitos, ordenados por prioridade. Os tabelas de requisitos por inteiro, incluindo algumas definições adicionais, estão incluídas em apêndice A.

- **Visualização de dados:** O utilizador deve poder visualizar gráficos interativos baseados nos dados carregados, com a possibilidade de aplicar filtros como país ou *quarter* selecionado.
- **Gestão de ficheiros:** O utilizador deve poder carregar ficheiros, associá-los a *quarters* e eliminá-los quando necessário. A aplicação valida os formatos e garante que apenas ficheiros válidos são tratados.
- **Gestão de *quarters*:** O utilizador deve poder criar períodos, identificados como *Quarter N*, para organizar os ficheiros carregados. Tem também de ser possível visualizar a lista de *quarters* disponíveis.
- **Autenticação:** O utilizador deve poder criar uma conta e usar autenticar com a conta criada. Os dados apresentados devem ser apenas os do utilizador autenticado.
- **Normalização de dados:** Os dados extraídos dos ficheiros devem ser automaticamente normalizados para garantir consistência e compatibilidade com a aplicação.

Requisitos não funcionais

Alguns requisitos não funcionais também foram importantes para garantir a estabilidade e a capacidade de interação da aplicação. Os requisitos identificados foram os seguintes:

- **Usabilidade:** A aplicação deve ser intuitiva, suportar múltiplos *browsers* e permitir o carregamento progressivo de gráficos sem bloquear a interface visual (*lazy load*).
- **Performance:** A aplicação deve ser capaz de suportar múltiplos utilizadores em simultâneo, mantendo uma resposta rápida às interações.
- **Acessibilidade:** A aplicação deve ser acessível por teclado e compatível com leitores de ecrã (*screen-reader friendly*).

3.2 Casos de Utilização

Com base nos requisitos funcionais e não funcionais identificados na secção 3.1, foram definidos os casos de utilização que iremos suportar. A figura 3.1 apresenta o diagrama UML dos casos de utilização da aplicação. Este diagrama ilustra as interações entre os vários atores que fazem parte da aplicação.

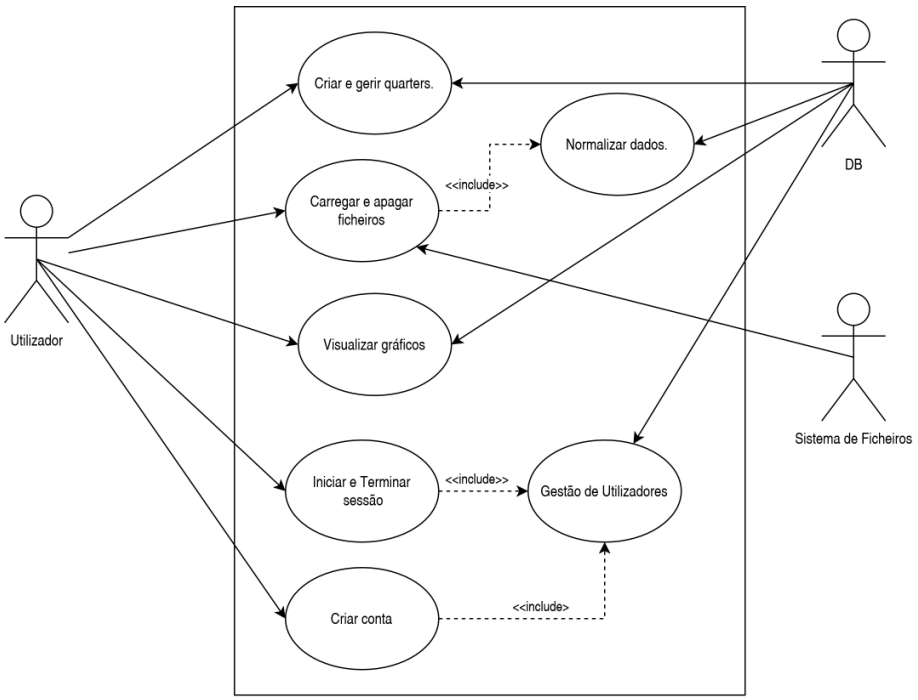

Figura 3.1: UML dos Casos de Utilização

Na figura 3.1 podemos ver os atores e os casos de utilização da aplicação, que são os seguintes:

- **Utilizador:** É o ator principal, responsável por interagir com a aplicação. Pode criar conta, iniciar sessão, carregar ficheiros, visualizar gráficos, entre outras ações.

- **Base de Dados (DB):** Responsável por armazenar dados dos ficheiros carregados e metadados associados, e os *quarters* criados pelo utilizador.

- **Sistema de Ficheiros:** Responsável pelo armazenamento físico dos ficheiros carregados.

Temos também definidos os seguintes casos de utilização, que descrevemos com algum detalhe:

- **Criar e gerir *quarters*:** Permite ao utilizador, após autenticação, criar *quarters*. Cada *quarter* é identificado por um número (único por utilizador) e guardado na base de dados, sendo associado a um *Universally Unique Identifier* (UUID) para efeitos internos.
- **Carregar ficheiros:** O utilizador seleciona um *quarter* e carrega ficheiros exportados da plataforma de simulação. A aplicação valida os ficheiros, extrai e processa cada folha de cálculo e aplica a *pipeline* de normalização. Os dados normalizados são armazenados e associados ao *quarter* correspondente.
- **Eliminar ficheiros:** Permite ao utilizador remover ficheiros que foram carregados. A aplicação atualiza os registos no *backend*, marca os dados antigos como não correntes e evita que sejam considerados nos gráficos, garantindo que apenas uma versão está ativa por ficheiro.
- **Normalizar dados:** Sub-processo automático executado durante o carregamento de ficheiros. Converte os dados para um formato estruturado através de uma *pipeline* de normalização. Comunica com a base de dados e a aplicação de ficheiros.
- **Consultar gráficos:** Após o carregamento dos ficheiros, o utilizador pode consultar gráficos com base nos ficheiros carregados. A aplicação permite aplicar filtros, navegar entre *quarters* e ver gráficos.
- **Criar conta:** Permite o registo de novos utilizadores. A aplicação valida os dados inseridos, verifica se o utilizador já existe e cria a conta, autenticando automaticamente o utilizador após sucesso.
- **Iniciar e Terminar sessão:** O utilizador introduz credenciais válidas (nome de utilizador e palavra-passe) e, caso sejam corretas, é autenticado e redirecionado para a interface principal. Este processo inclui a gestão da sessão.

3.3 Fundamentos

Neste capitulo iremos descrever as bases nas quais fundamentamos o projeto, e algumas noções necessárias para conseguir contextualizar as decisões tomadas.

3.3.1 *Marketplace Simulations*

A *Marketplace Simulations* [24] é uma empresa que desenvolve plataformas de simulação para fins educativos, ou seja, ferramentas que colocam os estudantes numa espécie de jogo onde cada equipa gere a sua própria empresa e compete com os colegas em cenários simulados. Acabam então por funcionar como laboratórios virtuais que podem ser usados no ensino superior, onde os alunos aplicam os conceitos aprendidos numa experiência em contexto educativo.

No caso concreto do projeto, a plataforma de simulação é utilizada tipicamente no último semestre, na cadeira Projeto de Simulação em Negócios Internacionais da Licenciatura de Comércio e Negócios Internacionais [9], que tem a interface apresentada na figura 3.2.

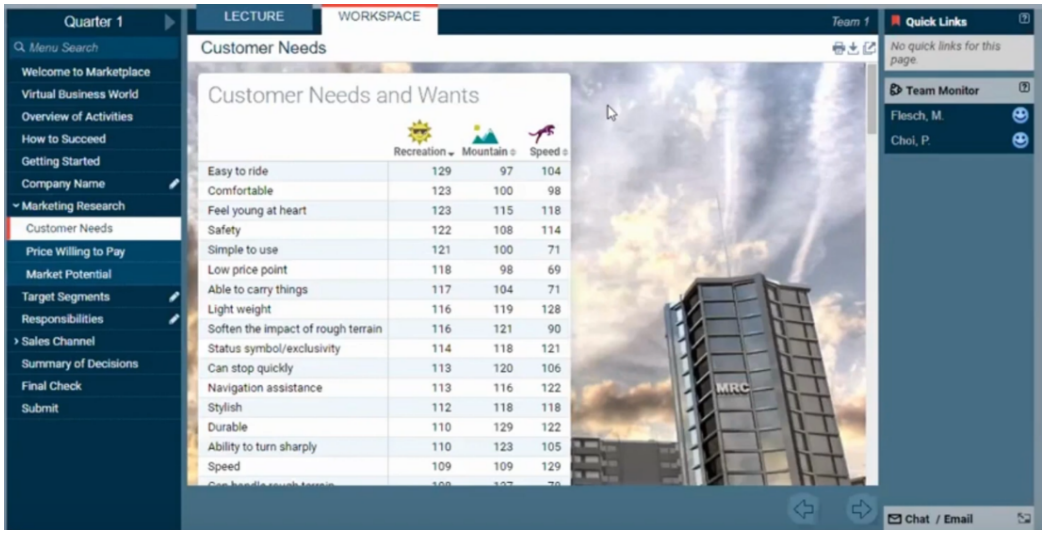

Figura 3.2: Captura de ecrã da aplicação de simulação

Nesta plataforma, cada grupo de alunos representa uma empresa que

tem de atuar num mercado internacional, tomando decisões sobre o posicionamento de produto, investimento, preços, distribuição, entre outras opções. Essas decisões são processadas pela plataforma, que simula o comportamento do mercado com base num algoritmo interno.

A evolução temporal da simulação é dada por *quarters*, que representam uma semana simulada. No final de cada *quarter*, os alunos recebem os dados com os resultados das decisões anteriores, o que obriga a uma análise comparativa entre períodos. É este ciclo (decidir, analisar, ajustar, repetir) que dá sentido à simulação e aproxima o exercício a uma situação real.

A simulação está dividida em secções, cada uma representando uma área distinta do negócio gerido pelos alunos. Estas secções agrupam métricas, decisões ou outras informações relacionadas com aspetos específicos da empresa simulada, como por exemplo a procura por segmento, o desempenho financeiro, a perceção da marca ou a eficácia da equipa de vendas.

Cada secção representa também uma área onde os alunos têm de tomar decisões. As decisões variam consoante o tipo de secção, como por exemplo, na secção *Customer Needs*, os alunos devem decidir que segmentos pretendem servir e com que marcas. Estas decisões são submetidas no final de cada *quarter*, influenciando os resultados do seguinte periodo, que por sua vez geram novos dados para análise.

A plataforma de simulação permite exportar os dados de cada secção em ficheiros *Excel*, sendo que cada ficheiro está associado a uma destas secções como por exemplo a secção *Customer Needs*, entre outras.

Para a análise, os alunos têm à disposição os dados, na plataforma, nas várias secções disponiveis que na sua maioria são apresentados em tabelas, o que faz com que os alunos saltem entre secções, ou tenham de fazer gráficos à mão ou em último caso, extrair a informação da plataforma, não sendo então prático analisar a informação só pela plataforma de simulação.

Esta plataforma levanta outros desafios. Como não é de código aberto, não é possível perceber como funcionam os seus algoritmos internos, e o seu propósito só se torna evidente no contexto da unidade curricular, o que dificulta bastante explicar o seu funcionamento fora desse contexto.

3.3.2 *Extract, Transform Load*

Uma *pipeline Extract, Transform, Load* (ETL) é um padrão utilizado em sistemas de tratamento e integração de dados, com o objetivo de mover dados de uma ou mais fontes para um destino, geralmente uma base de dados ou plataformas específicos para a análise de dados [29]. Uma *pipeline* ETL é composta por três fases principais:

- *Extract* (Extrair): Consiste em recolher os dados das fontes de informação, que podem incluir bases de dados relacionais, APIs externas, sensores, entre outros. Esta fase preocupa-se com a capacidade de ler dados e de confirmar que são possíveis de extrair. Exemplos de ferramentas que podem ser utilizadas nesta fase incluem o *Apache NiFi* [3], *Fivetran* [14], *Airbyte* [1] e o *Google Cloud Dataflow* [17].

- *Transform* (Transformar): Nesta fase os dados extraídos são processados e convertidos num formato mais adequado ao destino. Isso pode incluir tarefas como limpeza de dados (remoção de valores nulos ou duplicados), normalização, mudanças de tipo, ou cálculos. É nesta fase em que se garante a consistência da informação. Ferramentas populares para transformação incluem o *dbt (data build tool)* [8], o *Apache Beam* [2] (usado com *Dataflow*) e o *Apache Spark* [4].

- *Load* (Carregar): Por fim, os dados transformados são inseridos na aplicação de destino, normalmente um *data warehouse*, O carregamento é feito dependendo dos requisitos da aplicação de destino. Exemplos de produtos que podem ser utilizados como destino são bases de dados como *BigQuery* [16], *Snowflake* [25], *Clickhouse* [5], entre outros.

As *pipelines* ETL são muito utilizadas em contextos onde há necessidade de consolidar dados de várias origens, permitindo análises mais completas. São fundamentais em áreas como *business intelligence*, ciência de dados e integração de sistemas.

3.3.3 Padrão Cliente-Servidor

O projeto desenvolvido segue uma arquitetura de aplicações cliente-servidor, dividida em duas grandes camadas: *backend* e *frontend*.

Backend

O *backend* corresponde a "'parte invisível" da aplicação, ou seja, tudo o que acontece do lado do servidor. É a camada responsável por tratar os dados, executar a lógica de negócio e responder aos pedidos efetuados pelos utilizadores. No contexto específico deste projeto, o *backend* é responsável por:

- carregamento de ficheiros;
- processamento e normalização dos dados;
- autenticação e gestão de sessões;
- disponibilização de uma *Application Programming Interface* (API) para o *frontend* que permita acesso aos dados.

Frontend

O *frontend* corresponde à "parte visível" da aplicação, ou seja, aquilo que o utilizador vê e com que interage no *browser*. É a camada responsável por apresentar a informação de forma clara e permitir a interação com as funcionalidades disponibilizadas pela aplicação. No contexto deste projeto, o *frontend* é responsável por:

- os formulários, utilizados por exemplo para autenticação, criação de trimestres e envio de ficheiros;
- os gráficos que são apresentados ao utilizador, com os dados processados pelo *backend*;
- e outros elementos visuais criados a partir dos dados processados, como mensagens de erro, modais, entre outros.

Esta separação facilita a manutenção da aplicação e permite que ambas as partes sejam desenvolvidas de forma independente, podendo até ser substituídas sem necessidade de reescrever a aplicação completo. As tecnologias utilizadas para o desenvolvimento serão descritas no capítulo 3.5.

3.3.4 Tipos de gráficos

Durante a análise efetuamos foram estudados diferentes tipos de visualizações que podiam ser aplicadas, de acordo com os dados que tínhamos disponíveis. Cada tipo de visualização foi escolhido com base na sua capacidade de representar visualmente os dados e a sua facilidade de interpretação. Consideramos então vários tipos de visualizações gráficas.

Importa salientar que as imagens que acompanham esta secção foram retiradas diretamente da aplicação desenvolvida, já tendo em conta as limitações e as transformações realizadas nos dados. Estas transformações são discutidas mais a fundo no capítulo 4.1.1.

Gráfico de Barras (*Bar Chart*):

Este tipo de visualização foi utilizado para comparar valores entre diferentes categorias, como marcas, cidades ou segmentos de mercado. A disposição visual das barras permite uma leitura rápida das diferenças de desempenho entre categorias, sendo útil para dados que não são temporais. Foi também utilizado a variante barras agrupadas, dependendo se havia necessidade de comparar valores entre categorias.

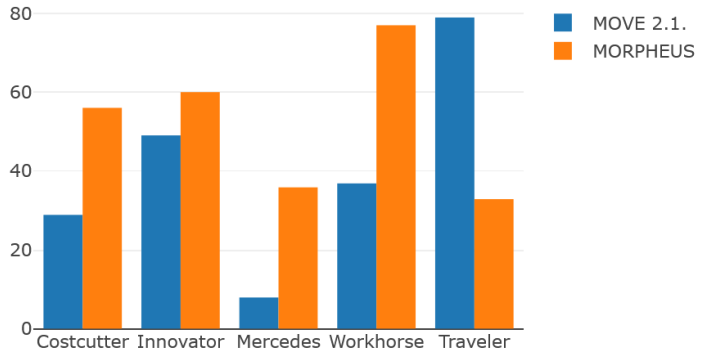

Figura 3.3: Exemplo de gráfico de barras agrupadas

Gráfico de Sectores (*Pie Chart*):

Utilizado para representar distribuições percentuais, como partições de dados por segmento. Este tipo é de gráfico é intuitivo para visualizar como uma totalidade se divide entre diferentes partes, sendo apropriado para dados onde se pretendia mostrar proporções relativas (como por exemplo dados categorizados por segmentos)

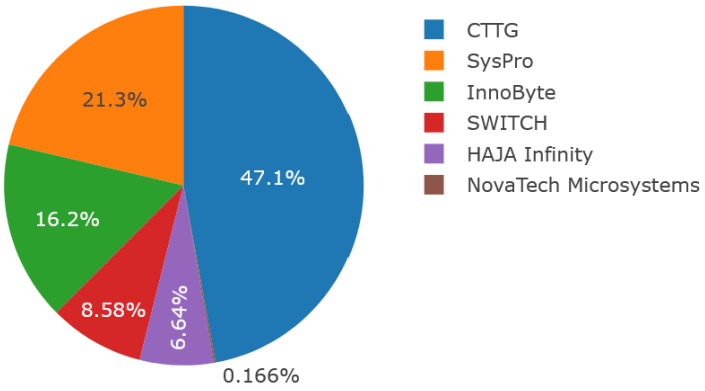

Figura 3.4: Exemplo de gráfico de sectores

Gráfico Financeiro (*Waterfall Chart*):

Utilizado especificamente para representar balanços financeiros, como valores acumulados de receitas e despesas. Permite também visualizar como diferentes contribuições individuais afetam um valor final, facilitando a análise de ganhos e perdas.

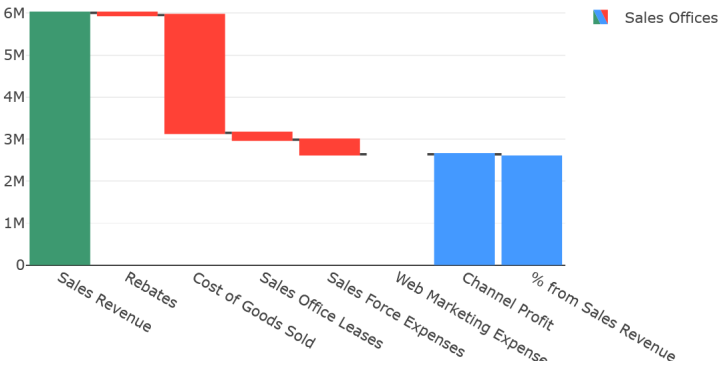

Figura 3.5: Exemplo de gráfico financeiro

Diagrama de Quartis (*Box Plot*):

Os diagramas de quartis foram utilizados para representar dados que eram distribuições com mínimos e máximos, permitindo visualizar de forma prática a mediana, os quartis e valores extremos. A interpretação deste tipo de visualização exige um conhecimento prévio de conceitos estatísticos como mediana e quartis, o que pode dificultar a leitura, mas que simplifica a representação dos dados.

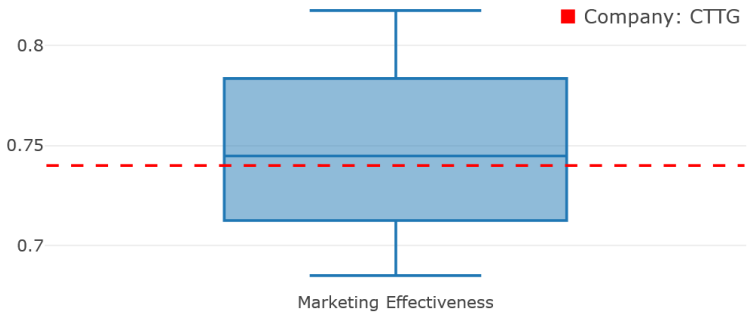

Figura 3.6: Exemplo de diagrama de quartis

Diagrama de Sankey (*Sankey Diagram*):

Os diagramas de *Sankey* são utilizados para representar fluxos entre diferentes categorias, sublinhando a quantidade transferida de uma categoria para outra. Cada fluxo é representado por uma linha cuja largura é proporcional à quantidade movida, permitindo uma visualização intuitiva da importância dos fluxos. Para evitar representações complexas, decidimos apenas recorrer a este tipo apenas quando os dados não poderiam ser representados de outra forma, ou tratados de forma a simplificar a representação.

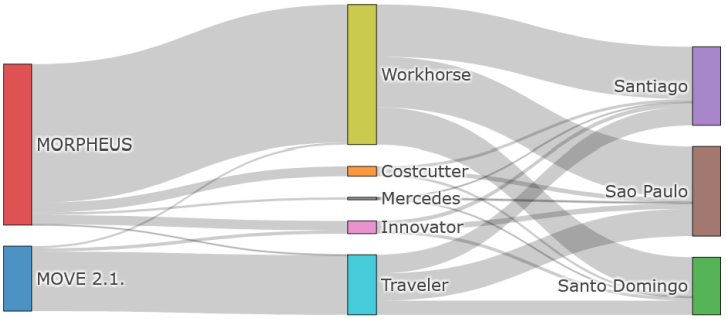

Figura 3.7: Exemplo de gráfico de *Sankey*

Outros tipos de gráficos relevantes para o projeto, mas que não foram utilizados são:

Gráfico de Linhas (*Line Chart*):

Escolhido para representar séries temporais, como a evolução de uma variável ao longo do tempo. As linhas permitem identificar tendências e variações. No projeto, não tivemos necessidade de usar este tipo uma vez que nenhum dos dados recebidos era relativos a séries temporais, pelo que a sua utilização não era intuitiva para os utilizadores nem resultava numa visualização que fizesse sentido para os dados recebidos.

Gráfico de Radar (*Radar Chart*):

Utilizado para comparar múltiplas variáveis em relação a um valor comum, como no caso da avaliação de diferentes necessidades dos clientes em simultâneo. Este tipo de gráfico permite identificar rapidamente pontos fortes e fracos em várias dimensões de análise. Apesar destas vantagens, nos dados que recebemos não conseguimos identificar utilizações onde este tipo de gráfico beneficiasse os utilizadores finais.

Gráfico de Dispersão (*Scatter Plot*):

Os gráficos de dispersão foram considerados para representar relações entre duas variáveis. Cada ponto no gráfico representa uma observação individual, permitindo identificar padrões de correlação (positiva, negativa ou inexistente) entre variáveis. No nosso caso, consideramos utilizar um tipo específico de gráfico de dispersão (gráfico de dispersão num mapa) mas a leitura tornou-se confusa, uma vez que era pouco legível para alunos, e não

considerava localizações fictícias, fazendo com que a leitura fosse difícil.

A escolha dos tipos de gráficos teve como objetivo manter a clareza e facilidade da informação e permitir diferentes leituras sobre os dados extraídos. Cada gráfico foi pensado para às características dDOS dados disponiveis, considerando o formato dos dados (quantitativa ou categórica) e o objetivo da análise (comparação, distribuição, evolução ou composição).

3.4 Abordagem

O projeto proposto pretende então conseguir transformar dados desta plataforma em algo mais fácil e rápido de analisar. Tendo o contexto da plataforma, iremos então descrever duas abordagens que considerámos.

3.4.1 *Web scraping*

A primeira abordagem que consideramos foi a hipótese de automatizar a extração dos dados diretamente da plataforma do *Marketplace Simulations*, através de técnicas de *web scraping*. A ideia parecia interessante numa fase inicial, já que permitiria reduzir a dependência do utilizador no processo de exportação dos dados. No entanto, rapidamente percebemos que esta abordagem trazia vários desafios que, na prática, a tornavam pouco viável, ou mesmo arriscada.

Primeiro, cada conta na plataforma está associada a um grupo de alunos, ou seja, é uma conta ativa e única, usada diretamente durante a simulação. Isto significa que qualquer processo automático que iniciasse sessão, mesmo que fosse só para leitura, poderia de forma acidental interagir com a interface e acabar por alterar alguma opção, o que poderia comprometer a avaliação dos alunos porque podia afetar a sua avaliação final. Além disso, como o acesso à plataforma é feito por licenças pagas, não existe qualquer possibilidade de criar uma conta de serviço ou utilizador apenas para leitura. Ou seja, qualquer tentativa de *web scraping* teria de reutilizar credenciais reais, o que levanta não só questões de segurança, mas também (possivelmente) legais, uma vez que a plataforma pode não permitir o *web scraping*.

Outro fator que influenciou a nossa decisão foi o próprio risco técnico do *web scraping* uma vez que plataformas deste tipo estão muitas vezes protegidas com mecanismos contra automação (como por exemplo ecrãs CAPTCHA), e não conhecendo em detalhe a aplicação, poderíamos facilmente encontrar esse tipo de proteções, que são difíceis de automatizar.

Pelos motivos acima, optámos por não seguir esta abordagem. Em vez disso, definimos como parte da utilização da nossa aplicação que os próprios alunos devem exportar dados a partir da plataforma de simulação e carregar para a nossa aplicação. Esta solução, embora mais manual, garante segurança, respeita a integridade das contas dos utilizadores, e evita problemas legais ou técnicos com a aplicação de simulação.

3.4.2 Exportação e carregamento manual de ficheiros

A abordagem que acabamos por usar foi os alunos exportam manualmente os dados diretamente da plataforma de simulação e carregarem esses ficheiros na nossa plataforma. A partir daí, a aplicação processa e normaliza os dados recebidos, e mostra gráficos com base nesses dados. Esta abordagem, apesar de requerer uma ação manual do lado do utilizador, é mais segura que a alternativa de *web web scraping*, pelos os motivos apresentados acima. Deste modo, garantimos um equilíbrio entre usabilidade, segurança e fiabilidade da aplicação. Acabamos então por criar um SAD (cf., capítulo 2) que permita os estudantes tomarem melhores decisões.

Com esse objetivo, procurámos que a nossa aplicação refletisse a estrutura da própria simulação. Como tal, os dados são organizados por *quarters*, como acontece na plataforma de simulação e permitir o carregamento dos dados exportados. Estes ficheiros são depois processados o que nos permite trabalhar com dados mais consistentes.

Esta estrutura base implica a existência de três entidades principais na nossa aplicação (que iremos descrever nos capítulos seguintes) e cuja interação define o funcionamento da aplicação.

Quarters

Os *quarters* funcionam como *buckets* para organizar os ficheiros carregados pelos utilizadores. Cada utilizador pode criar múltiplos *quarters*, identificados de forma única por um número. O propósito em incluir este conceito é para que a aplicação consiga associar os dados extraídos dos ficheiros carregados ao *quarter* correspondente sem depender de manipulações do nome do ficheiro carregado, e desde modo assegurar que a plataforma consegue identificar corretamente os *quarters*.

A desvantagem é que requer *input* do utilizador para que seja criado ou atribuído o *quarter* correto. De modo a facilitar a criação de *quarters*, tentamos que experiência de utilização fosse centrada no carregamento de ficheiros, limitando os *quarters* a um campo obrigatório no formulário de carregamento auto-preenchido, ou seja, tornando implícita a criação de *quarters* no momento de carregamento de ficheiros.

Ficheiros

Os ficheiros são inicialmente carregados no formato *Excel Spreadsheet Format* (XLSX) (que é o formato que a plataforma de simulação exporta), contendo uma ou várias folhas de cálculo. Cada folha é tratada como uma fonte de dados individual e é de onde são extraídos os dados. O ficheiro XLSX é guardado como referência, mas não é diretamente utilizado para visualização, ficando apenas como artefacto interno da aplicação. O modelo proposto apenas considera um gráfico por folha de cálculo.

O processamento que é aplicado é uma *pipeline* de transformação de dados (conhecido na indústria como ETL (cf., capítulo 3.3.2)), que aplica regras para que os gráficos possam ser mostrados de forma consistente. Os dados resultantes desta transformação estão associados ao ficheiro XLSX original, sendo até possível voltar a processar ficheiros de forma a recriar informação (esta funcionalidade não é disponibilizada aos utilizadores finais e não foi desenvolvida de forma explícita).

A aplicação garante que só existe uma versão ativa de cada ficheiro. Caso o utilizador carregue novamente um ficheiro com o mesmo nome, o ante-

rior será marcado como não ativo, evitando duplicações e garantindo que os gráficos usam apenas dados mais recentes, e em caso de remoção, garante que conseguimos reverter para a versão anterior.

Pipeline de Transformação de Dados

Para conseguirmos então garantir uma experiência de utilização consistente, desenvolvemos uma *pipeline* transformação de dados, semelhante a uma *pipeline* ETL (cf., capítulo 3.3.2), ainda que neste projeto tenha sido desenvolvido com uma escala mais pequena, e com recurso a bibliotecas de processamento de dados.

O objetivo é garantir que os ficheiros carregados, que muitas vezes contêm nomes de colunas inconsistentes, quebras de linha, espaços em excesso ou colunas irrelevantes, sejam adaptados para serem visualizados, e que o processamento seja determinístico.

Como podemos receber muitos ficheiros, a variabilidade entre os dados recebidos é muito alta, pelo que alguns dados passam por mais do que uma fase de transformação. Esta decisão foi tomada com base numa análise manual, em que identificámos possíveis fontes de dados que precisam de mais do que uma fase de transformação. As várias fases de transformação alteram os dados de modo a facilitar a representação visual dos mesmos e é um passo essencial no projeto, porque garante que a aplicação trabalha com formatos e regras conhecidas, e remove a variabilidade dos ficheiros importados. Para proteger a aplicação, decidimos apenas mostrar os dados que foram extraídos de ficheiros conhecidos, de modo a garantir a consistência dos dados mostrados.

As fases de transformação irão ser descritas em mais detalhe nos capítulos seguintes, uma vez que a implementação destas pipeline estão relacionadas à tecnologia escolhida, mas o desenvolvimento desta *pipeline* é um fator diferenciador deste projeto, uma vez que tem de lidar com dados que não estão estruturados de forma a facilitar representações visuais.

Utilizadores

A aplicação foi projetada para funcionar com utilizadores. Cada utilizador tem a sua conta, e pode criar *quarters*, carregar ou alterar ficheiros, e ver aos gráficos criados a partir desses ficheiros.

Apesar da aplicação não suportar explicitamente grupos, assume-se que alunos do mesmo grupo podem carregar ficheiros semelhantes, mas a aplicação trata-os como ficheiros diferentes. Assim, evita-se a complexidade adicional de gerir permissões ou partilha de dados entre contas. Também se assume que as contas podem ser criadas ao nível do grupo, pelo que para a aplicação, é indiferente se a conta é individual ou partilhada entre membros desse grupo.

Cada utilizador tem acesso apenas aos seus próprios dados, garantindo o isolamento da informação. Esta separação é feita a nível da base de dados, através da associação de cada entidade (ficheiro ou *quarter*) ao utilizador que criou.

No futuro, pode ser considerada a funcionalidade de desativação automática de contas (por exemplo, após o final do semestre) ou a gestão das contas por um docente da unidade curricular.

No próximo capitulo iremos então descrever as escolhas tomadas de forma a concretizar este modelo.

3.5 Arquitetura e Tecnologias Adotadas

A arquitetura final da aplicação pode ser observada na Figura 3.8. Esta segue o modelo cliente-servidor, onde os utilizadores interagem com a aplicação através de um *browser*. A escolha das tecnologias baseou-se na familiaridade prévia com cada uma, na maturidade da mesma, as respetivas comunidades, bem como o contexto e requisitos do projeto. Nas seguintes secções especificamos que tecnologias escolhemos e que funções desempenham na aplicação.

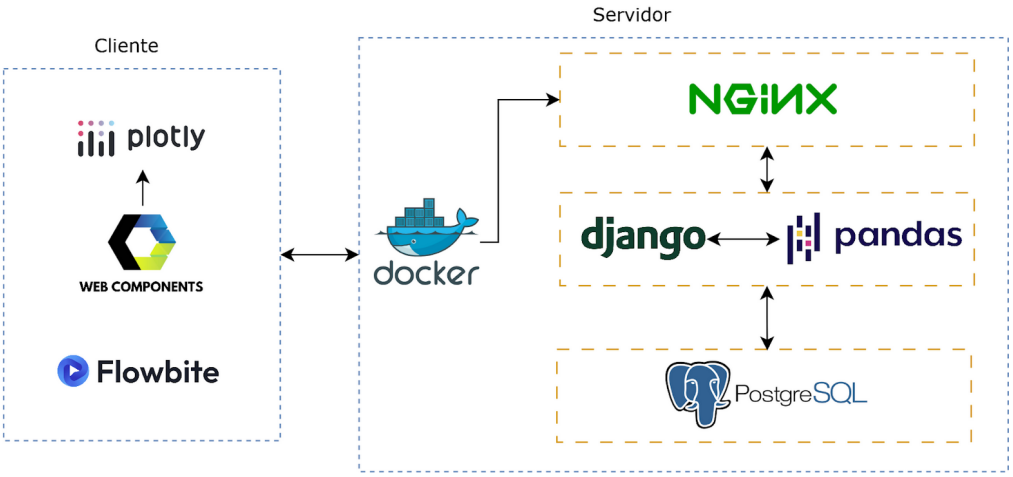

Figura 3.8: Arquitetura da aplicação

3.5.1 Django

Para o desenvolvimento do *backend* da aplicação, optámos por usar a *framework Django*. A escolha deveu-se ao facto de o *Django* ser uma *framework* muito usada, com uma arquitetura conhecida e uma comunidade muito grande.

Uma das principais vantagens do *Django*[12] é o facto de seguir uma variante do padrão *Model-View-Controller* (MVC), conhecida como *Model-Template-View* (MTV), o que facilita bastante a organização da aplicação e a separação de responsabilidades.

- o *Model* representa a camada de dados e corresponde ao modelo do domínio;
- o *Template* representa a camada de apresentação (interface com o utilizador);
- a *View* é a camada intermediária que processa a lógica da aplicação e interage com as outras camadas.

Para além disso, o *Django* utiliza um *Object-Relational Mapping* (ORM), que permite interagir com bases de dados sem necessidade de escrever *Structured Query Language* (SQL) manualmente. Este ORM mapeia os modelos

para entidades na base de dados e suporta operações como filtros, agregações e relações entre tabelas [12], e ao mesmo tempo previne falhas de segurança, como *SQL injection*, através do uso de consultas parametrizadas [20].

Outra vantagem é a as funcionalidades gestão de utilizadores e sessões já estar incluído no *framework*. Isto permite ao programador focar-se no desenvolvimento das funcionalidades específicas da aplicação sem ter de construir esse suporte de raiz.

Outro fator que consideramos positivo foi a integração com outras bibliotecas *Python*, como o *Pandas*, que usamos para o processamento de dados. Esta compatibilidade torna o desenvolvimento mais rápido e flexível, reduzindo o esforço necessário (e código) para ligar diferentes tecnologias.

Em alternativa, podíamos ter escolhido bibliotecas como *Flask* [19] ou *FastAPI* [23]. Apesar de *FastAPI* ser mais recente e mais rápido para API *Representational State Transfer* (REST), o nossa aplicação, focado na transformação de dados e visualização, não necessitava de uma abordagem completamente assíncrona e não funciona totalmente com API REST. Já o *Flask*, apesar de ser mais leve, não vem com funcionalidades como gestão de utilizadores e criação de contas, pelo que teríamos de desenvolver esses mecanismos de raiz.

Relativamente a outras opções como *WordPress*, excluímos essa hipótese. O *WordPress*, que é uma plataforma que suporta sites de conteúdo como blogues ou sites de notícias, não é adequado para projetos que exigem muito controlo sobre a estrutura dos dados e não integra bem com as outras tecnologias escolhidas.

3.5.2 Pandas

Para o processamento de dados, a biblioteca que decidimos usar foi o *Pandas*. Esta escolha foi motivada pelo facto de ser muito utilizada em projetos de engenharia de dados, e por ser escrita em *Python*.

O *Pandas* permite ler e transformar ficheiros XLSX. A sintaxe e as funcionalidades disponibilizadas para operações como normalização, filtragem, agrupamento e ordenação tornaram esta biblioteca adequada às necessidades

do projeto. Uma vantagem adicional é a capacidade de converter os dados para formatos como *JavaScript Object Notation* (JSON), o que facilitou a sua integração com o restante da aplicação.

Apesar destas vantagens, o *Pandas* não é a melhor opção se considerarmos *datasets* muito grandes, uma vez que funciona inteiramente em memória. Em contextos com maiores volumes de dados, poderiam ser consideradas bibliotecas como *Dask* ou *Polars* por terem capacidades de paralelismo para lidar com um maior conjunto de dados (acabam por ser mais adequadas para *BigData*). No entanto, para os objetivos e escala deste projeto, o *Pandas* é a escolha mais equilibrada considerando o *scope* do projeto que desenvolvemos.

3.5.3 Plotly

Para mostrar os gráficos da aplicação, optámos por utilizar o *Plotly*, que é uma biblioteca que permite criar visualizações, e que está disponível em várias linguagens como *Python* e *Javascript*. Esta escolha foi motivada pelo facto de o *Plotly* suportar suportar uma *API* declarativa, o que torna mais prática a integração com o *frontend* e *backend*, porque assim a configuração dos gráficos pode ser retornada pelo servidor, permitindo uma configuração mais estável e garantindo que todos os utilizadores vêm o mesmo tipo de gráfico. Internamente, o *Plotly* utiliza uma outra biblioteca, *D3.js*, que é uma biblioteca com capacidade de criar gráficos personalizados.

Além disso, a biblioteca oferece suporte a uma grande variedade de gráficos, desde gráficos de barras até formatos mais especializados como mapas de calor, o que foi importante para explorar os vários tipos de visualização poderíamos usar. Em comparação, bibliotecas como *Chart.js*, embora sejam mais leves, não oferecem a mesma flexibilidade e variedade de visualizações. Por este motivo, *Plotly* era a solução mais adequada para a flexibilidade e diversidade de visualizações que queríamos. O *Plotly* permite criar versões da biblioteca com gráficos específicos, reduzindo bastante o tamanho do ficheiro *Javascript*. Uma lista completa das diferenças pode ser consultada no capítulo B.

A versão *Python* desta biblioteca foi também importante no desenvolvimento dos gráficos, uma vez que nos permitiu explorar os dados fora da

aplicação, de modo a escolher que tipo de gráfico seria o mais adequado aos dados que estávamos a analisar e que transformações eram necessárias. Este processo de exploração permitiu-nos perceber o que era possível fazer, e com a ajuda dos orientadores, definir estratégias alternativas para os conjuntos de informação que eram mais difíceis de mostrar.

3.5.4 *WebComponents*

Para conseguirmos isolar a implementação dos gráficos decidimos utilizar *WebComponents* [31]. Esta decisão teve como base a necessidade de manter uma boa experiência e criar componentes que podiam ser reutilizados para os vários tipos de visualização.

Os *WebComponents* [31] são uma especificação nativa dos *browsers*, que permite definir componentes reutilizáveis com encapsulamento de estilo e comportamento. Ao evitarmos dependências pesadas como o *React*, conseguimos reduzir a complexidade técnica da aplicação, mantendo ao mesmo tempo a flexibilidade e capacidade de reutilização dos componentes desenvolvidos.

Apesar de *React* ser uma biblioteca muito utilizada para aplicações *web*, considerámos que para este projeto, a sua introdução seria aumentar as dependências e complexidade. A alternativa escolhida é uma solução mais leve e mais fácil de manter.

Outra razão que nos levou a esta escolha foi o facto do *Django* já trazer consigo as funcionalidades de criação e gestão de contas de utilizador1, pelo que se a aplicação *web* fosse escrita totalmente em *React*, teríamos de usar o *Django* apenas como uma API REST, e utilizar métodos alternativos de criação e gestão de contas de utilizador1 como *OAuth2*.

3.5.5 *PostgreSQL*

Para a persistência de dados, utilizámos a aplicação de gestão de bases de dados *PostgreSQL* que é uma das soluções *open-source* mais completas atualmente. Inicialmente foi considerada a utilização de *SQLite* por simplicidade durante o desenvolvimento, mas como iriamos usar JSON para armazenar

dados, decidimos que o *PostgreSQL* era a melhor opção. O *PostgreSQL* têm um bom desempenho, suporte a consultas mais complexas e é compatível com o ORM do *Django*.

3.5.6 Docker

O *Docker*[13] foi utilizado neste projeto, tanto para desenvolvimento como para ambientes de produção. A principal vantagem é a criação de ambientes de execução consistentes e isolados, garantindo que a aplicação corre da mesma forma em diferentes máquinas, sem conflitos de dependências ou configurações, e consistente em vários sistemas operativos.

Existem vários *runtimes* compatíveis com *Docker*[13], e devido ao modelo de negócio da empresa *Docker*[13] (que adotou um modelo pago), optámos por usar o *Rancher Desktop*[26] (uma aplicação com uma API compatível com *Docker*) para o desenvolvimento, e o *Docker Engine*[13] para ambientes de produção.

Durante o desenvolvimento, o *Docker* facilitou a orquestração dos vários serviços usados (a aplicação *Django*, a base de dados, e o servidor *Hypertext Transfer Protocol* (HTTP) *Nginx* configurado como *reverse proxy*), através de ficheiros `Dockerfile` e `docker-compose.yml`. Isto permitiu-nos montar o ambiente da aplicação, sem necessidade de instalações manuais.

Para produção, o *Docker* permite recriar um ambiente de produção sempre com a mesma configuração e gerir os vários serviços que a aplicação utiliza, o que facilita o processo de *deploy*, e escalabilidade futura. Esta abordagem garante que o código testado é exatamente o que será executado em produção, reduzindo erros e aumentando a estabilidade da aplicação.

O *Docker*[13] foi também utilizado durante o desenvolvimento deste relatório, porque permitiu-nos ter um ambiente *LaTex* configurado com todas as extensões apenas com um único comando no terminal sem instalar aplicações.

3.6 Outras ferramentas utilizadas

Além das tecnologias centrais utilizadas no projeto, foi também fundamental a utilização de um conjunto de ferramentas de suporte que facilitaram o trabalho, a organização e o desenvolvimento do projeto. As principais ferramentas utilizadas foram:

- **Git [6] e Github [15]**: Para assegurar o controlo de versões, foi utilizado a aplicação *Git*, em conjunto com a plataforma *Github*. Esta combinação permitiu não só manter um histórico das alterações feitas e reverter alterações que podiam ter introduzido defeitos na aplicação
- **Notion [22]**: A organização do trabalho foi feita na plataforma *Notion*. Esta aplicação é prática de fácil de usar e permitiu-nos para planear as tarefas e acompanhar o progresso das diferentes fases do projeto e também tirar notas durante o desenvolvimento, que foram importantes para o desenvolvimento do relatório.
- ***Visual Studio Code* (VSCode)[7]**: Como editor, foi utilizado o editor VSCode. Através da instalação de extensões, foi possível adaptar o VSCode às necessidades do projeto, melhorando a experiência de programação.

Estas ferramentas foram importantes no desenvolvimento do projeto, não só do ponto de vista técnico, mas também no que respeita à organização e planeamento do mesmo.

Capítulo 4

Implementação do Modelo

Este capítulo descreve como se passou do modelo conceptual para a implementação da aplicação. O desenvolvimento foi feito de forma iterativa, com testes e validações com base nos objetivos do projeto e nos dados recebidos. Primeiro foi feita a análise e classificação dos dados, que ajudou a definir como seriam processados e que gráficos íamos usar. Depois, vamos detalhar o desenvolvimento da *pipeline* de transformação e a lógica de organização de ficheiros. Por fim, é explicado como as tecnologias escolhidas foram usadas para criar a aplicação.

4.1 Análise e Transformação dos dados

O processo de análise começou após a reunião inicial com os orientadores do projeto, onde recebemos os ficheiros que iriam ser trabalhados. Tratar estes ficheiros implicou várias fases de análise, onde começamos a pensar em estratégias para conseguir extrair dados e possíveis gráficos, e tornar este processo modular.

4.1.1 Análise inicial dos dados

Após termos recebido os dados, fizemos uma primeira análise com o objetivo de perceber a informação recebida. No total, recebemos 30 ficheiros XLSX que foram exportados da plataforma, correspondentes às diferentes secções da

plataforma de simulação. Cada ficheiro pode conter várias folhas de cálculo, e no total contámos 101 folhas de cálculo nos ficheiros que recebemos.

Durante esta análise, percebemos que:

- Os nomes dos ficheiros e das folhas variam, mas seguem uma estrutura relativamente consistente;
- Alguns ficheiros apresentam estruturas de dados semelhantes, mas com diferenças no nível de detalhe ou na organização das colunas e linhas;
- Existiam ficheiros com dados que iriam precisar de mais transformação uma vez que representavam vários tipos de unidades (como por exemplo percentagens, valores monetários, valores relativos, entre outros) na mesma folha de cálculo;
- Dados com muito detalhe, e com várias colunas sem representação (células marcadas com "X"ou com valores nulos e fundo amarelo).
- Nenhuma das folhas de cálculo indica unidades (como por exemplo percentagens, euros, entre outros), o que significa que não é possível assumir a unidade dos valores que estão a ser representados. Os ficheiros exportados são exportados exatamente como estão na plataforma de simulação, que não usa unidades numéricas.

Com base nesta primeira análise, concluímos que seria necessário:

- Fazer uma normalização da informação recebida para garantir uma utilização consistente da aplicação;
- Separar os vários ficheiros possíveis em grupos, de forma a identificar dados em comum que pudéssemos aplicar *templates* de gráfico;
- Guardar os dados extraídos num formato mais prático, e que permita uma filtragem dinâmica da informação carregada (de forma a facilitar a visualização dos dados).
- Em algumas séries de dados, identificamos linhas e colunas que se podiam remover devido a serem redundantes (ou por representarem informação que já é representada na mesma folha, como por exemplo

colunas de valores totais, linhas que repetiam as marcas, ou outras discutidas com os orientadores);

- Relativamente à falta de unidades, para manter a consistência com a plataforma de simulação, optámos por não assumir nenhuma unidade para os valores que estão a ser representados, uma vez que não é possível inferir a unidade dos valores a partir dos dados recebidos, a plataforma de simulação não usa unidades numéricas, e para não induzir os alunos em erro com a interpretação dos dados.

Esta primeira análise ajudou-nos a perceber como estruturar o modo como iríamos tratar os diferentes ficheiros que os utilizadores podem carregar, ou seja, deu-nos uma base para sistematizar a transformação da informação de forma consistente. Com isso em mente, optámos por agrupar os dados em várias categorias, tendo por base a estrutura de cada folha de cálculo.

4.1.2 Classificação dos dados

Os grupos definidos foram os seguintes: **simples**, **duplo**, **balanços**, **setores** e **análise específica**. Vamos então descrever o significado de cada grupo e identificar o que é comum entre os gráficos pertencentes a cada um deles.

O grupo **simples** inclui ficheiros em que a estrutura é mais direta, geralmente com apenas duas colunas: uma coluna que representa uma categoria (como por exemplo empresas ou segmentos de mercado) e uma coluna numérica. Nestes casos, optámos por gráficos de barras, uma vez que o objetivo principal é comparar rapidamente valores individuais entre categorias. Nos ficheiros que avaliamos, não encontramos séries temporais, pelo que não justificou gráficos de linhas. Também nesta categoria, alguns dados eram séries mais específicas (como dados relativos a médias e medianas) mas mantinham a mesma estrutura de dados, pelo que foram classificados como **simples** mas utilizaram gráficos diferentes como gráficos de diagramas e quartis e gráficos de setores (também conhecidos como *pie charts*).

O grupo **duplo** refere-se a ficheiros onde existem múltiplas séries de dados associadas à mesma categoria. Ou seja, para cada categoria, existem vários valores (ou várias séries) que precisam de ser representados lado a lado. Para

estes casos, a escolha que fizemos foi apresentar gráficos de barras agrupadas e barras empilhadas, permitindo uma comparação direta entre diferentes séries de dados para a mesma categoria, e permite visualizar todos os dados relacionados com essa categoria.

O grupo **balanços** abrange ficheiros associados a séries de dados financeiros, onde faz sentido representar aumentos e diminuições de valores ao longo de um processo ou período. Para estes ficheiros, o *template* selecionado foi o gráfico de cascata (ou um gráfico *financial waterfall*), dado que este tipo de visualização representa as várias componentes do valor total numa forma fácil de perceber.

O grupo **agrupados** inclui ficheiros onde conseguem ser apresentados grupos de dados associados a uma categoria (como por exemplo, para uma empresa, mostrar a presença nos segmentos de mercado por marca). Nestes casos, faz sentido usar gráficos de setores e barras agrupadas, dependendo da necessidade de comparar proporções entre segmentos.

Finalmente, o grupo de **análise específica** são ficheiros que não se encaixam diretamente nos formatos anteriores, por não partilharem de uma estrutura comum ou quando os dados incluem várias unidades na mesma folha (percentagens com euros na mesma folha de cálculo) e que precisavam de uma análise mais detalhada. Para estes casos, o processo passou por simplificar a informação de modo a encaixar num dos grupos acima ou excecionalmente aplicar um novo tipo de gráfico. Na figura 4.1 é possível ver um exemplo de um ficheiro que foi classificado como **análise específica**.

Income Statement					
Report Item	**Quarter 1**	**Quarter 2**	**Quarter 3**	**Quarter 4**	**Quarter 5**
Gross Profit					
Revenues	0	0	975,210	4,541,830	11,647,298
- Rebates	0	0	9,200	45,200	128,260
- Cost of Goods Sold	0	0	617,654	2,100,222	5,341,414
= Gross Profit	**0**	**0**	**348,356**	**2,396,408**	**6,177,624**
Expenses					
Research and Development	0	120,000	480,000	400,000	6,138,010
+ Quality Costs	0	0	17,733	316,435	440,318
+ Advertising	0	0	265,493	419,932	600,077
+ Sales Force Expense	0	0	118,974	337,855	809,013
+ Sales Office and Web Sales Center Expenses	0	170,000	410,240	771,519	1,164,091
+ Web Marketing Expenses	0	0	0	0	393,220
+ Marketing Research	88,000	0	115,000	115,000	115,000
+ Shipping	0	0	11,845	64,494	194,337
+ Inventory Holding Cost	0	0	11,372	7,174	34,041
+ Excess Capacity Cost	0	0	0	0	0
+ Depreciation	0	0	45,833	45,833	91,736
= Total Expenses	**88,000**	**290,000**	**1,476,490**	**2,478,243**	**9,979,842**
Operating Profit	**-88,000**	**-290,000**	**-1,128,134**	**-81,835**	**-3,802,219**
Miscellaneous Income and Expenses					

Figura 4.1: Excerto de uma folha de cálculo recebida

Para casos em que a não era possível aplicar um *template* de gráfico, não foi possível simplificar a informação e não encontrámos uma representação gráfica que fosse fácil de interpretar, optámos por não aplicar nenhum gráfico, e apenas apenas mostrar uma tabela interativa com os dados onde o utilizador podia filtrar os dados por diferentes colunas.

No final, dos 30 ficheiros recebidos, a classificação foi a seguinte:

Em termos de representações utilizadas, a distribuição é a seguinte:

A cada grupo acima mencionado, associámos um gráfico específico, criando assim um conjunto de *templates* que nos permite criar visualizações de forma programática.

4.1.3 Transformação dos dados

Com a classificação de dados feita, foi necessário desenvolver uma *pipeline* de processamento que permitisse normalizar e transformar os dados dos ficheiros para um formato mais fácil de representar e para guardar na base de dados. Este processo foi desenvolvido com recurso a biblioteca *Pandas*, que oferece métodos para manipulação e transformação de dados. Esta *pipeline* funciona de forma assíncrona, e é chamada pelo *backend* quando um ficheiro é carregado, de forma a não bloquear a interface do utilizador.

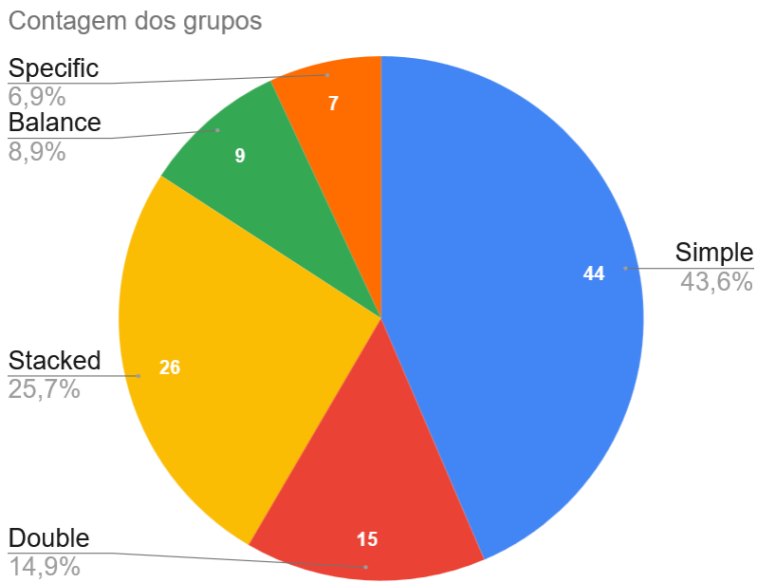

Figura 4.2: Classificação dos ficheiros - contagem final dos grupos

A *pipeline* de processamento foi dividida em duas fases, cada uma com um objetivo específico na transformação dos dados.

4.1.4 Extração e Normalização Inicial

Nesta fase, os dados são extraídos dos ficheiros carregados e normalizados para um formato consistente:

- Extração do título do gráfico a partir da primeira linha de cada folha.
- Extração e normalização (remover expressões como "Q5" do nome) da secção através do nome da folha de cálculo.
- Identificação de cada folha por num nome único (derivado do nome da folha de cálculo), para que possamos identificar os dados de forma única na aplicação (ou seja, um *slug* que mapeia exclusivamente a uma série de dados). Este passo é importante para garantir que cada série de dados é tratada de forma única e consistente, e para conseguirmos manter uma configuração para cada série de dados.

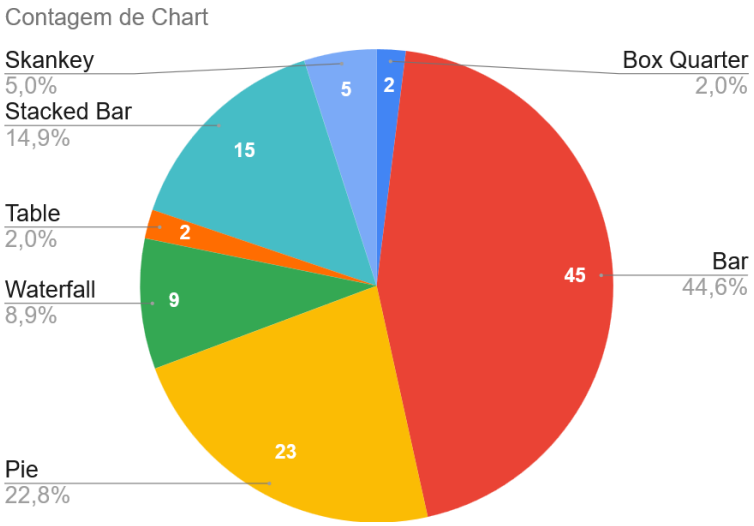

Figura 4.3: Classificação dos ficheiros - contagem final de representações utilizadas

- Normalização dos cabeçalhos das colunas (remoção de quebras de linha, espaços extras).
- Remoção de colunas baseadas na análise feita na secção 4.1.1.
- Normalização dos dados nas células (remoção de quebras de linha, espaços duplos).
- Tratamento de valores nulos e vazios (transformação para tipos compatíveis com *Python*)
- Manutenção da ordem original das colunas para preservar a estrutura dos dados. A biblioteca *Pandas*. automaticamente ordena as colunas, pelo que é necessário manter a ordem original das colunas para preservar a estrutura dos dados.
- Transformação de dados marcados como "X"em formato binário (0/1) para indicar presença/ausência de valores.
- Normalização de valores decimais num valor escolhido pelo utilizador (com o valor por omissão a 9).

4.1.5 Processamento Específico por Tipo

Após a normalização inicial, os dados passam por transformações específicas baseadas no seu tipo, e no tipo de gráfico que pretendemos representar:

- Aplicação de transformações específicas baseadas no tipo de gráfico identificado (financeiro, percentagens, valores relativos entre outros).
- Processamento para dados financeiros, incluindo cálculos de percentagens e valores relativos de outros valores (aplicar somas e subtrações como descrito na folha de cálculo).
- Transformação dos dados para conseguir representar categorias e a relações entre elas (como por exemplo a relação entre marcas e segmentos de mercado e cidades).
- Para folhas que continham múltiplos *quarters*, optámos por escolher o *quarter* para o qual a folha foi carregada, sendo que essa configuração pode ser trocada em código.

De forma a tornar modular toda a lógica de normalização de dados, decidimos criar configurações na aplicação que mapeiam ficheiros a gráficos e as suas transformações e outras configurações associadas a colunas que são usadas para representar os dados, sendo que cada série de dados é identificada unicamente pelo seu *slug*. Esta configuração não é guardada na base de dados, e só pode ser editada em código, porque para os dados extraídos serem mostrados, precisam de ser processados de forma especifica de modo a que consigam ser representados, e a sua visualização foi pensada para ser um gráfico especifico, pelo que não era possível suportar várias representações para o mesmo conjunto de dados.

Após os dados serem processados, já não é possível alterar a visualização dos dados, porque o processamento adapta a informação que recebe para a visualização pretendida.

4.1.6 Conversão e Armazenamento dos dados

Após o processamento dos dados ser feito, os mesmos são guardados como JSON num campo *JSONField* da base de dados *PostgreSQL*. Estes dados guardados tentam ser os mais próximos ao dados mostrados ao utilizador, mas sem perder informação quando contem várias séries dentro do mesmo conjunto de dados (como por exemplo dados de várias marcas).

Para garantir a consistência dos dados e evitar conflitos de ficheiros com o mesmo nome a serem carregados para o mesmo *quarter*, foi implementado um mecanismo que permite marcar ficheiros como processados e não processados e como correntes e não correntes. Quando um utilizador carrega um ficheiro, este é marcado como processado e como corrente, e os dados são guardados na base de dados. Quando um utilizador carrega um novo ficheiro com o mesmo nome, o novo é marcado como corrente, e o antigo é marcado como não corrente. Em nenhum momento são apagados os ficheiros nao correntes que estão guardados em disco. A aplicação tem um mecanismo em que só considera ficheiros marcados como correntes para serem mostrados ao utilizador.

Isto permite recuperar versões anteriores dos dados quando o utilizador apaga um ficheiro ou quando existe algum erro no processamento, uma vez que os dados resultantes só são marcados como correntes quando são extraídos e processados com sucessos. Esta funcionalidade não foi uma funcionalidade pensada para ser exposta diretamente ao utilizador, e não faz parte do uso normal, mas surgiu como consequência da arquitetura escolhida (ficheiros correntes e não correntes).

Este mecanismo de processamento permitiu transformar os dados extraídos dos ficheiros XLSX em um formato estruturado e normalizado, facilitando a visualização e análise dos dados através da aplicação *web*. A modularidade da *pipeline* também permitiu adicionar passos de normalização conforme necessário.

No final, obtemos dados que são práticos de representar, e que podem ser facilmente utilizados pela aplicação para mostrar os gráficos como é representado na figura 4.4.

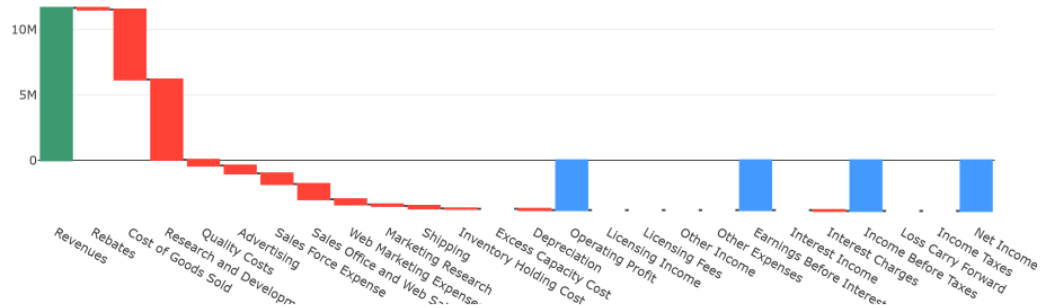

Figura 4.4: Gráfico extraído da figura 4.1 com filtro aplicado para o *Quarter* 5

4.2 Desenvolvimento da Aplicação *web*

Esta secção foca-se na estrutura e funcionamento da aplicação *web*, e como foi desenvolvemos as funcionalidades da aplicação.

4.2.1 Arquitetura do Servidor

A aplicação *web* foi desenvolvida utilizando a *framework Django* como já descrito na secção 3.3. A arquitetura foi pensada para garantir o isolamento dos dados por utilizador e uma gestão dos ficheiros e dados processados.

Modelos de Dados

A aplicação utiliza três modelos, cada um com um papel específico na gestão e organização da informação. Estes modelos são a base da aplicação, sendo que suportam todas as funcionalidades da interface gráfica. Como o *Django* utiliza um ORM, cada classe definida nos modelos representa também uma tabela na base de dados, como ilustrado na Figura 4.5. A classe *User* é uma classe que já vem incluída com o *Django*, e é utilizada para gerir os utilizadores e sessões da aplicação.

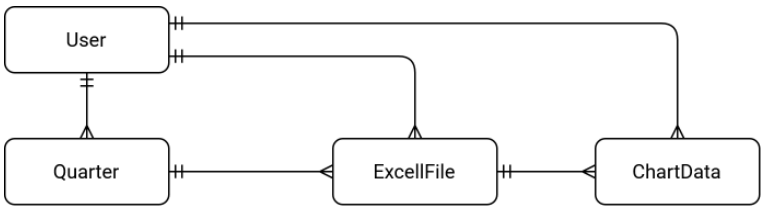

Figura 4.5: Diagrama de entidade-relação da aplicação

O nosso modelo de dados é composto por três modelos principais: *Quarter*, *ExcelFile* e *ChartData*, cada um com um papel específico no contexto da aplicação.

Quarter: Representa um trimestre específico para um utilizador. Cada *quarter* tem:

- Um número único por utilizador.
- Uma precisão configurável para valores numéricos (por omissão 9 casas decimais).
- Um UUID para identificação. Este UUID é único na aplicação toda.
- Relação com o utilizador que o criou.

ExcelFile: Representa um ficheiro *Excel* carregado para um *quarter* específico:

- Armazena o ficheiro físico em pastas organizadas por UUID.
- Mantém o estado de processamento (processado ou não processado).
- Guarda metadados como a secção do ficheiro.
- Relaciona-se com o *quarter* e o utilizador.
- Controla qual versão dos dados está ativa (corrente ou não corrente).

ChartData: Armazena os dados processados de cada folha do *Excel*:

- Guarda os dados em formato JSON.

- Mantém a ordem original das colunas.
- Guarda metadados como o nome da folha e o *slug* para identificar os dados unicamente.
- Relaciona-se com o ficheiro *Excel* de origem e o utilizador.

A utilização de identificadores UUID foi importante uma vez que, para além de identificar unicamente cada instância da entidade, faz com que não seja possível aceder aos dados só a aumentar o identificador, como se fosse o caso se o identificador fosse um número inteiro sequencial. O *Django* já disponibiliza uma chave primária em cada modelo por omissão, que é um número sequencial inteiro.

Gestão de Utilizadores

A aplicação utiliza o mecanismo de utilizadores que já vem incluído com o *Django*. Este mecanismo garante que apenas utilizadores autenticados possam aceder aos seus dados. A autenticação na aplicação é feita através de nome de utilizador e palavra-passe e as sessões são geridas pelo *Django*.

Para garantir a segurança dos dados, todas as rotas que requerem autorização estão marcadas com o decorador `@login_required`. Além disso, a aplicação isola dados dados por utilizador, garantindo que cada utilizador só possa ver os dados que carregou. Este isolamento é implementado nos modelos e pesquisas, que associa ou aplica um filtro com base no utilizador autenticado nas operações de leitura e escrita.

Os utilizadores podem criar conta na aplicação, sendo que podem criar conta para si ou para um grupo de utilizadores. A aplicação não faz distinção entre uma conta para um utilizador ou para uma conta para múltiplos utilizadores.

Gestão de Ficheiros e Processamento de dados

Para manter ficheiros carregados organizados, desenvolvemos um mecanismo que garante que os dados carregados pelos utilizadores ficam organizados em pastas. Em vez de depender apenas do nome do ficheiro (o que podia

facilmente gerar conflitos ou sobreposições de nomes), a aplicação cria uma pasta com UUID do *quarter* onde foram carregados, o que garante que cada ficheiro fica junto aos ficheiros do mesmo *quarter*. Para ficheiros com o mesmo nome, o *Django* automaticamente acrescenta letras ao final do ficheiro de forma a garantir que o nome é único dentro de cada pasta.

Assim que um ficheiro é carregado, o processamento é iniciado recorrendo a *signals*, que são eventos que são lançados quando uma instância do modelo é alterado ou criado.

Para além disso, a aplicação mantém um registo de todos os ficheiros carregados. Quando se carrega um novo ficheiro que resulta nos mesmos dados, a aplicação não apaga o anterior, como já foi descrito no capítulo 4.1.6.

***Endpoints* para comunicação com o cliente**

A aplicação disponibiliza um conjunto de *endpoints* REST que viabilizam a interação com a interface gráfica. A comunicação com o backend é efetuada através de chamadas assíncronas aos *endpoints*, sendo possível passar parâmetros como o *quarter* selecionado e um filtro escolhido pelo utilizador. Em resposta, o backend devolve a estrutura de dados necessária para a renderização, assegurando que cada gráfico é gerado com os dados e configurações apropriados. Estes *endpoints* são rotas específicas que a interface gráfica utiliza para algumas funcionalidades:

- `quarters/new/`: Permite a criação de um novo *quarter*.
- `quarters/edit/<uuid:uuid>/`: Permite editar os detalhes de um *quarter* já existente, identificado pelo seu UUID.
- `quarters/delete/<uuid:uuid>/`: Permite apagar um *quarter* específico.
- `quarters/`: Permite listar todos os *quarters* do utilizador.
- `quarters/files/delete/<uuid:uuid>/`: Permite remover um ficheiro associado a um *quarter*, usando o UUID do ficheiro.
- `api/chart/`: Retorna os dados para os gráficos

O *endpoint* para visualização de gráficos é o mais importante, uma vez que é utilizado para mostrar os gráficos na aplicação *web*, e aceita parâmetros para filtrar e mostrar gráficos de diferentes tipos. Estes *endpoints* são usados com recurso a pedidos HTTP (utilizando `fetch`) e também como páginas que aceitam parâmetros por *Uniform Resource Locator* (URL) (como por exemplo em formulários através do atributo `action`).

```
1    <form method="post" action="/quarters/edit/b12fdf1f-8ce3-4050-
     a28f-07e444e15042/" id="edit-quarter-form" class="upload-form-
     wrapper" enctype="multipart/form-data">
2      {% csrf_token %}
3      <div class="flex justify-end">
4        <button type="submit" class="upload-modal-submit-btn">Save<
     /button>
5      </div>
6    </form>
7
```

Código 1: Excerto do código HTML do formulário de edição de *quarter*

Quando a aplicação carrega, não mostra logo todos os gráficos, mas sim um *skeleton* que depois é substituído pelo gráfico, melhorando a experiência do utilizador. Esse carregamento utiliza o *endpoint* para visualização de gráficos para obter a configuração final de um gráfico especifico a ser mostrado. A configuração retornada é especifica para a biblioteca *Plotly*, que depois, juntamente com o restante código *Javascript*, consegue mostrar o gráfico e aplicar filtros com base nos parâmetros passados.

4.2.2 Interface Gráfica

A interface da aplicação foi desenhada com foco na consistência visual e na usabilidade, utilizando um *design system*, *Flowbite*. O objetivo é garantir uma aplicação responsiva, adequada para todos os ecrãs de computador e monitores e que suportasse minimamente dispositivos móveis, sem comprometer a performance nem a clareza na apresentação dos dados.

Layout **e Linguagem visual**

Um *design system* é uma biblioteca que oferece um conjunto de componentes reutilizáveis que podem ser usados em interfaces visuais. A escolha do *Flowbite* acelerou o desenvolvimento e simplificou a criação da interface visual, e de elementos como modais, formulários e botões.

A navegação principal é feita através de uma barra horizontal no topo da interface, que lista todas as páginas que se podem consultar e que permite os utilizadores aceder à pagina de carregamento de ficheiros. As modais são usados para operações carregamento de ficheiros. Para o desenvolvimento do *Cascading Style Sheets* (CSS) foi utilizada a linguagem , que permite desenvolver estilos de forma mais rápida e mais organizada, que depois é transpilada (através de um *build system* como o *ESBuild*[32]) para CSS normal.

Quando não existe gráficos a mostrar, é mostrada ao utilizador uma mensagem que indica que não existe ficheiros carregados, com uma hiperligação a redirecionar para a página de carregamento de ficheiros. Quando já existe gráficos a mostrar, é mostrada uma barra lateral com as secções e gráficos disponiveis, onde também é possível fazer uma pesquisa por texto de forma a encontrar uma secção ou gráfico específico.

WebComponents

A camada de visualização de dados é composta por componentes *web*, cada um encapsulando toda a sua a lógica, incluindo integrações com as bibliotecas *Plotly* e *Datatables* . Esta abordagem garante isolamento entre componentes e torna possível reutilizar o código *Javascript* com diferentes tipos de gráficos.

A especificação *WebComponents*[31] permite criar os nossos nós *Document Object Model* (DOM), e estende as interfaces DOM já existentes. Esta funcionalidade permite criar componentes que se comportam como elementos HTML, e que podem ser utilizados como tal, e que também podem receber estilos e outras propriedades e atributos e é suportada já pelos os *browsers* mais usados (como pode ser consultado no site *Can I Use*[28] que é um site que indica o suporte de uma funcionalidade em diferentes *browsers*).

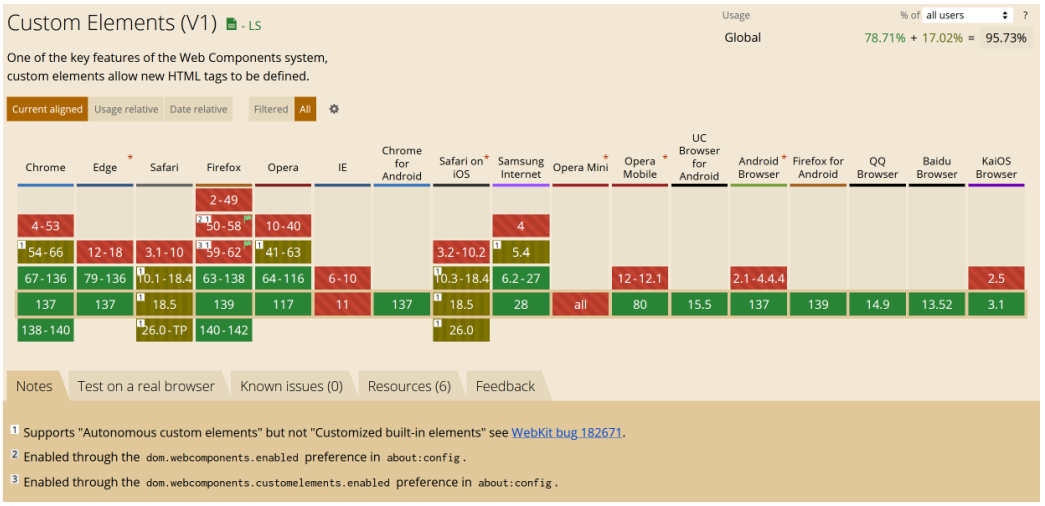

Figura 4.6: Tabela de suporte - *WebComponents*

Na nossa aplicação, os *WebComponents* são usados como elementos HTML normais, que recebem apenas o gráfico que é para mostrar. Ao carregar, fazem uma chamada a um *endpoint* do *backend*, que retorna a configuração do gráfico, que é utilizada para mostrar o gráfico.

```
<div class="grid grid-cols-8 gap-4"> grid
  <plotly-chart id="chart_ad-judgment-apac" chart_slug="ad-judgment-apac" q="1">…</plotly-chart> custom…
  <plotly-chart id="chart_ad-judgment-europe" chart_slug="ad-judgment-europe" q="1">…</plotly-chart> custom…
  <plotly-chart id="chart_ad-judgment-latam" chart_slug="ad-judgment-latam" q="1">…</plotly-chart> custom…
  <plotly-chart id="chart_ad-judgment-mea" chart_slug="ad-judgment-mea" q="1"></plotly-chart> custom…
  <plotly-chart id="chart_ad-judgment-noram" chart_slug="ad-judgment-noram" q="1"></plotly-chart> custom…
</div>
```

Figura 4.7: Utilização de *WebComponents* na aplicação

A comunicação com o *backend* é feita através de chamadas assíncronas aos *endpoints* desenvolvidos, passando parâmetros como o *quarter* selecionado e os filtros ativos. O *backend* devolve a configuração do gráfico para renderização com *Plotly*, assegurando que cada gráfico é gerado com os dados e configurações corretas, como pode ser observado no diagrama de sequência 4.8.

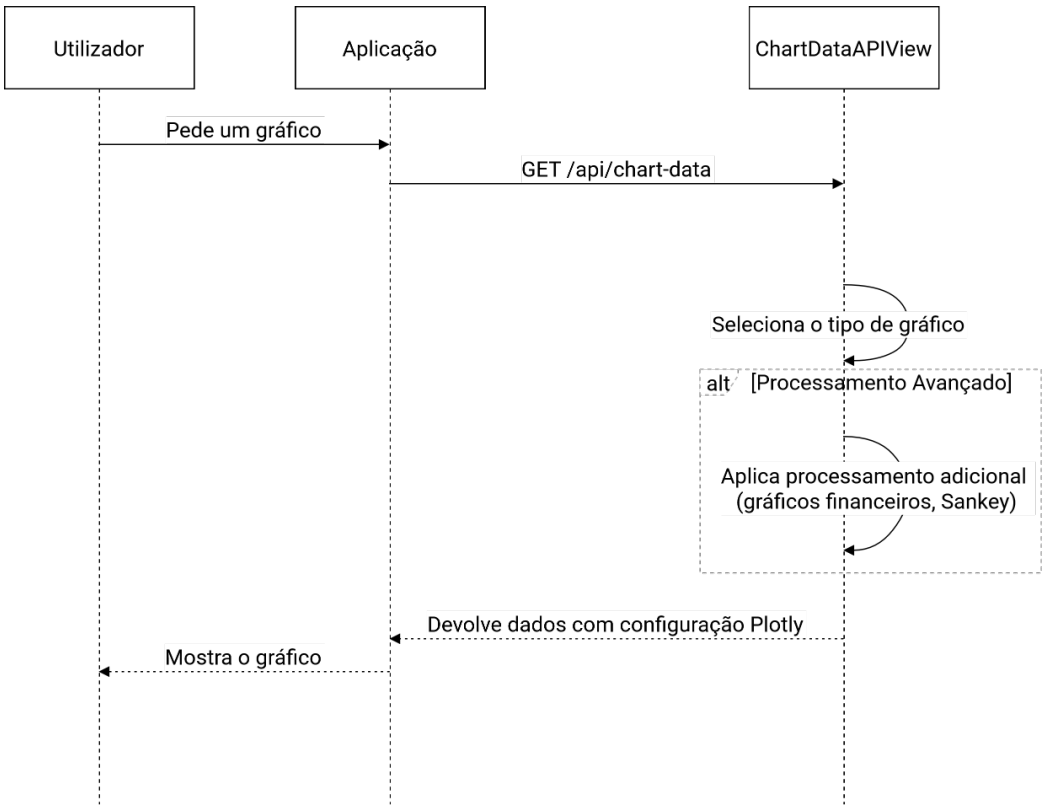

Figura 4.8: Diagrama de sequência da comunicação com o *backend*

A biblioteca *Plotly* suporta uma configuração declarativa, ou seja, apenas precisamos de retornar um objeto estático, e através dessa configuração, a biblioteca consegue criar a visualização

```
var trace1 = {
  x: ['giraffes', 'orangutans', 'monkeys'],
  y: [20, 14, 23],
  name: 'SF Zoo',
  type: 'bar'
};

var data = [trace1];
var layout = {barmode: 'stack'};
Plotly.newPlot('myDiv', data, layout);
```

Código 2: Excerto de uma configuração para um gráfico com a utilização da biblioteca *Plotly*

Os gráficos são carregados de forma progressiva (*lazy load*). Apenas são mostrados quando entra dentro do *viewport* do *browser* do utilizador. O *lazy load* permite que os servidor não fique sobre-carregado com pedidos, e que no lado do cliente, a memória utilizada seja minimizada visto que não são carregados todos os gráficos de uma vez.

As bibliotecas *Plotly* e *Datatables* foram integradas com os *WebComponents*[31], fazendo com que o *WebComponent* consiga mostrar gráficos e tabelas de forma isolada. Esta integração permite que, quando o carregamento de um gráfico é iniciado, que seja apresentado um *skeleton* (uma representação visual com um *spinner* e com a largura e altura aproximada de um gráfico) que é trocada pelo o gráfico em si quando os dados ficam disponíveis. A ideia deste *skeleton* é que o utilizador veja que o gráfico está a ser carregado, e que não cause um grande deslocamento do gráfico na interface (*layout shifting*) quando os dados estão prontos.

A navegação entre *quarters* é facilitada por setas em cada gráfico apresentado. A criação e remoção de *quarters*, bem como o carregamento de ficheiros, é acompanhada por *feedback* visual no momento da ação.

Responsividade em dispositivos móveis

Toda a interface foi pensada para ser utilizada em computadores devido à quantidade de gráficos que são apresentados. A interface adapta-se bem a dispositivos moveis, mas devido ao espaço disponível, não é possível garantir uma boa experiencia de utilização em dispositivos moveis. A aplicação tem, essencialmente, duas *media queries*. Uma *media query* que até aos 1023px (figura 4.9) que inclui *smartphones* e *tablets*, e uma *media query* a partir dos 1024px que inclui computadores e ecrãs maiores (figura 4.10).

Estes *media queries* são as mais comuns, garantindo uma boa experiência de utilização em todos os ecrãs, sendo que não tivemos de fazer código explicitamente para a aplicação se adaptar para dispositivos móveis. Os filtros permitem alterar os gráficos em tempo real, com *feedback* visual imediato não sendo necessário recarregar a página. No momento em que o filtro troca, é feito um pedido a API de gráficos que retorna um novo conjunto de dados para mostrar, com a configuração do gráfico atualizada.

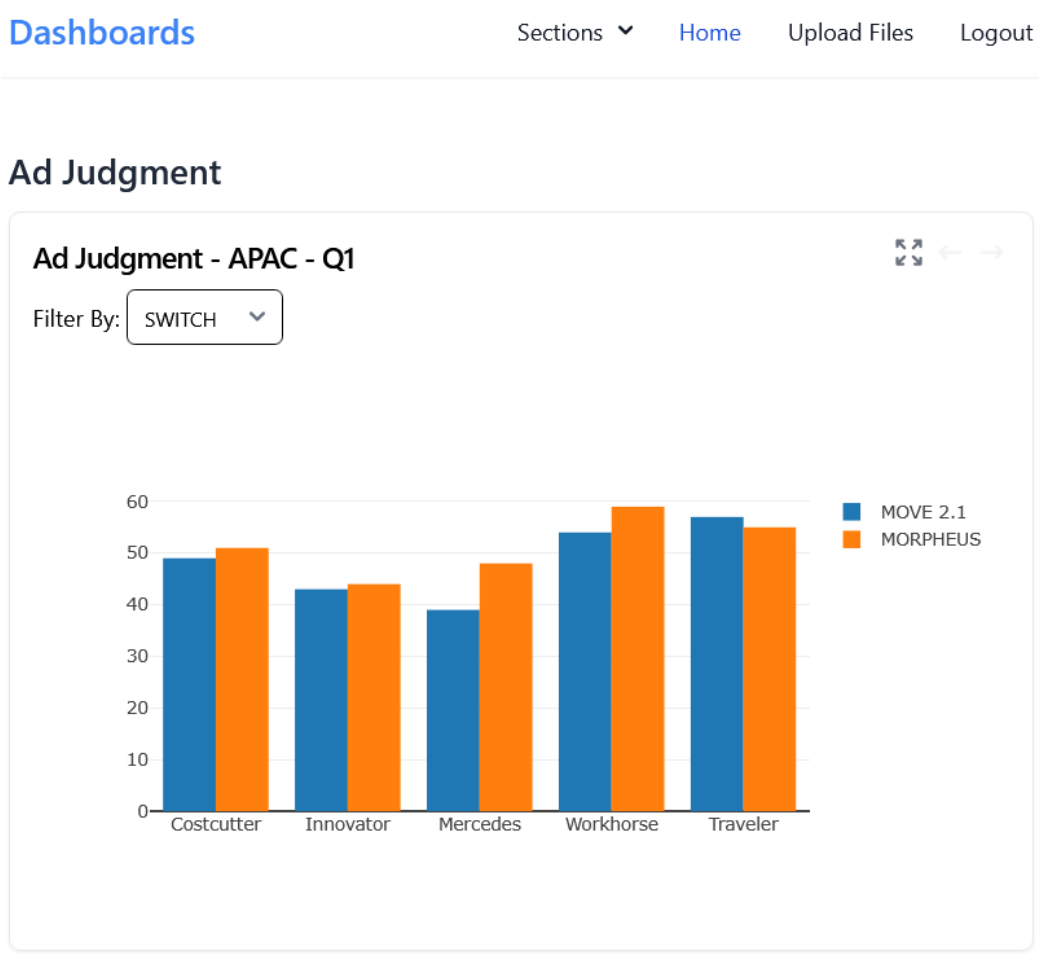

Figura 4.9: Media query até 1023px

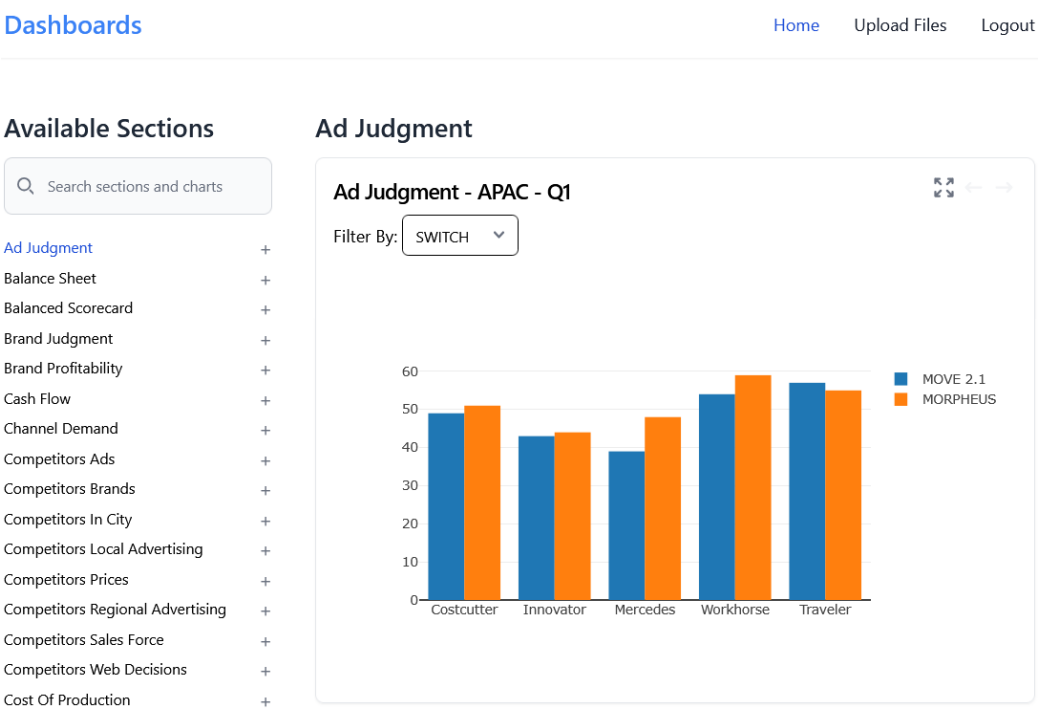

Figura 4.10: Media query a partir dos 1024px

Carregamento de ficheiros

A organização dos dados foi pensada para refletir a lógica do projeto Marketplace Simulations. Cada utilizador pode criar múltiplos *quarters* no momento em que carrega um ficheiro.

Para verificar se os ficheiros são do tipo correto, decidimos validar através do *MIME type*, que é um identificador utilizado para descrever o tipo de conteúdo de um ficheiro. Os *MIME types* seguem o formato `tipo/subtipo`, como por exemplo `application/vnd.openxmlV formats-officedocument.spreadsheetml.sheet` para ficheiros *Excel*. Esta abordagem permite garantir que os ficheiros carregados correspondem realmente ao formato esperado, independentemente da extensão do nome do ficheiro que pode ser facilmente manipulada.

Ao validar o *MIME type* do ficheiro, conseguimos rejeitar ficheiros com o tipo errado. No contexto deste projeto, esta verificação é importante, uma vez que apenas ficheiros XLSX válidos devem ser processados. Durante o

carregamento, a interface valida se os ficheiros são do tipo correto. Apenas ficheiros dos *MIME type* `application/vnd.openxmlformatsofficedocument.spreadsheetml.sheet` e `application/vnd.ms`*`Excel`* são permitidos. Assim, a validação por *MIME type* contribui para a segurança e integridade da aplicação.

Capítulo 5

Validação e Testes

A validação e testes da aplicação foram realizados através de múltiplas abordagens, garantindo uma cobertura de testes da funcionalidade como da usabilidade da aplicação. Este capítulo descreve as diferentes estratégias de teste implementadas e os resultados obtidos.

5.1 Testes de Acessibilidade

A acessibilidade foi uma preocupação no desenvolvimento da aplicação, e para isso foram seguidas as diretrizes *Web Content Accessibility Guidelines* (WCAG) 2.1[30], que definem critérios técnicos para tornar os conteúdos *web* mais acessíveis a todos os utilizadores. Para garantir a conformidade, recorremos a uma abordagem combinada de testes automáticos e validação manual. Entre as ferramentas utilizadas destacam-se duas principais: o *Lighthouse* e o *Axe*.

O *Lighthouse* [11], desenvolvido pela Google, é uma ferramenta *open source* que é lançada a partir das ferramentas de desenvolvimento do Chrome ou como *Command Line Interface* (CLI). Permite avaliar uma página *web* em várias categorias, como por exemplo Performance, Acessibilidade, entre outras. A secção de acessibilidade do *Lighthouse* valida, por exemplo, se os elementos têm contrastes suficientes, se os formulários estão corretamente identificadas com atributos *aria-label*, se os títulos estão estruturados numa hierarquia e se os elementos interativos são acessíveis por teclado. O

Lighthouse classifica numa pontuação de 0 a 100 e destaca problemas com sugestões de como resolver.

Esta ferramenta sinalizou alguns problemas, como a falta de compressão *GNU Zip* (GZIP) e alguns erros de acessibilidade e *Search Engine Optimization* (SEO).

Figura 5.1: Testes de acessibilidade com a ferramenta *Lighthouse*

Estes problemas foram resolvidos (fazendo alterações na configuração do servidor *Nginx* e no código da aplicação), e a ferramenta *Lighthouse* passou a dar uma pontuação melhor nos vários critérios que avalia

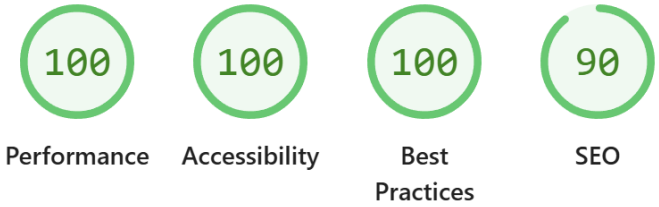

Figura 5.2: Testes de acessibilidade com a ferramenta *Lighthouse* após as correções

Já o *Axe*[10], da empresa *Deque*, é uma biblioteca de testes de acessibilidade, muito utilizada em ambientes profissionais. Pode ser usada como extensão de *browser* ou integrada com ferramentas de testes automatizados. O *Axe* avalia especificamente nos critérios definitos pela WCAG e reporta problemas como a ausência de nomes acessíveis, uso incorreto de *landmarks* *Accessible Rich Internet Applications* (ARIA), falhas de foco, ou elementos visuais sem representações textuais.

Nos primeiros testes feitos com a ferramenta *Axe*, as ferramentas sinalizaram alguns problemas relativos a acessibilidade, como a falta de textos alternativos para alguns elementos visuais, e os níveis dos títulos não estavam a ser usados de forma adequada.

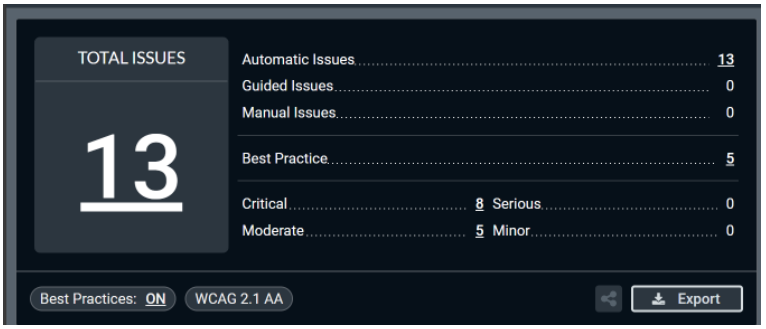

Figura 5.3: Testes de acessibilidade com a ferramenta *Axe*

Após resolver os problemas indicados pela ferramenta *Axe*, conseguimos então melhorar a acessibilidade da aplicação, como se pode ver na imagem abaixo.

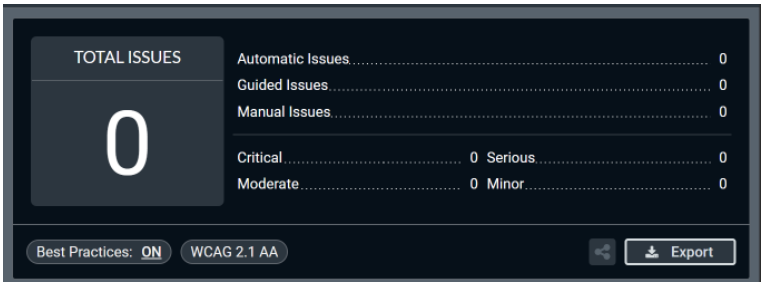

Figura 5.4: Testes de acessibilidade com a ferramenta *Axe* após as correções

No nosso caso, ambas as ferramentas foram utilizadas para identificar problemas diferentes, o *Lighthouse* ajudou-nos a identificar problemas mais relativos a velocidade e performance e dar uma visão rápida do estado da acessibilidade da aplicação, enquanto o *Axe* validou critérios mais técnicos e detetou problemas complexos de acessibilidade. Os testes foram feitos tanto em páginas com dados carregados como em estados vazios, e procurámos garantir que todas as interações essenciais da plataforma fossem acessíveis para

leitores de ecrã e através de navegação apenas por teclado. Desta forma, procurámos alinhar a aplicação com boas práticas de desenvolvimento inclusivo, respeitando não só normas técnicas mas também a responsabilidade de criar interfaces acessíveis a todos.

5.2 Testes Manuais

Para complementar os testes das ferramentas acima, foi desenvolvido um conjunto de testes manuais utilizando a linguagem *Gherkin* para descrever os cenários de teste. *Gherkin* é uma linguagem estruturada, mas legível por humanos, que permite descrever comportamentos esperados da aplicação em forma de cenários do tipo Dado-Quando-Então. Esta abordagem facilita a comunicação entre equipas de desenvolvimento e de testes, garantindo que todos compreendam os objetivos de cada teste.

Foi pedido a cinco pessoas para executarem estes testes manualmente, com base nos cenários escritos, e no final preencherem um formulário com os resultados de cada execução, incluindo observações e eventuais desvios face ao comportamento esperado. Os cenários de teste desenvolvidos com base nos casos de utilização e requisitos da aplicação. Cada cenário foi escrito como exemplificado abaixo. Os restantes cenários foram incluídos no apêndice D.

```
1  Scenario: Creating an account
2    Given I access the page
3    And I don't have an account or logged in
4    Then I should see the "Create Account" link
5    When I click the "Create Account" link
6    Then I should be redirected to the "Create Account" form
7    And I should see the username field
8    And I should see the password filed
9    And I should see the confirm password field
10   When I fill that form
11   And I click "Save"
12   Then I should be redirected to the "Login" page
```

Código 3: Excerto do código *Gherkin* do cenário de teste para a criação de uma conta

Apesar desta amostra não ser muito representativa, foi possível encontrar

bugs na aplicação, como por exemplo, a falta de alguns dados em alguns gráficos, e alguns gráficos não serem coerentes com a informação que estavam a mostrar. Estes *bugs* foram corrigidos assim que foram reportados pelas pessoas que testaram a aplicação.

Juntamente com os casos de teste, foi dado um formulário aos utilizadores para que pudessem partilhar comentários e classificar a experiencia de utilização da aplicação. Para cada cenário de teste que foi dado, foi pedido então que os utilizadores respondessem as seguintes perguntas:

- Consegui completar a tarefa (Sim / Não)
- Precisei de ajuda a completar a tarefa (Sim / Não)
- Achei a interface intuitiva de utiliza (classificação de 1 a 5)
- Comentários

No geral, os utilizadores que testaram a aplicação não encontraram problemas significativos, e a aplicação foi considerada fácil de usar e intuitiva. Em todas as secções do formulário, os participantes atribuíram classificações elevadas à facilidade de uso. Estes resultados sugerem uma curva de aprendizagem baixa e uma interação fluida com a interface.

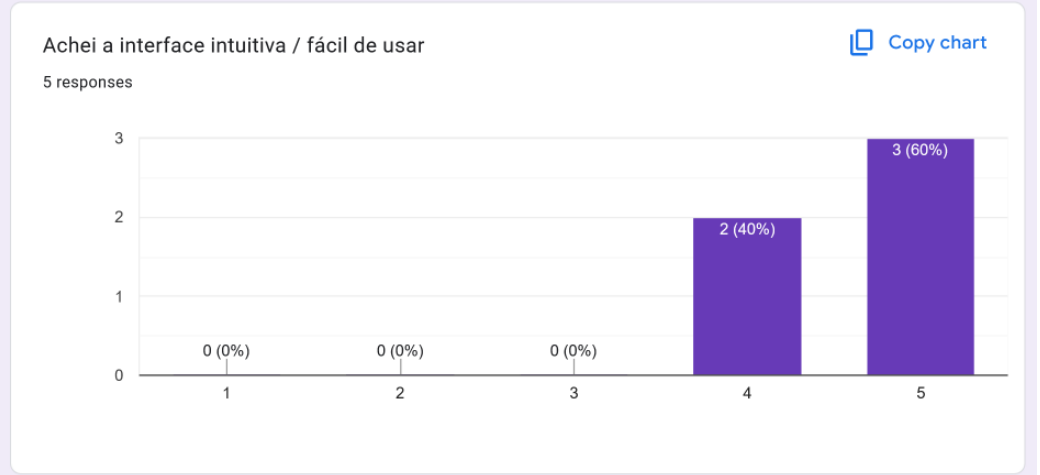

Figura 5.5: Captura de ecrã do formulário de testes manuais - Tarefa "Carregar ficheiros"

Alguns comentários recolhidos destacam aspetos pontuais a melhorar, como a clareza de alguns dos passos dos testes e nomenclaturas utilizadas nos casos de testes, mas nenhum dos participantes reportou bloqueios ou frustrações relevantes na aplicação. Estes dados reforçam que a interface cumpre os princípios de usabilidade, sendo funcional, acessível e compreensível por parte dos utilizadores.

5.3 Compatibilidade com Navegadores

A compatibilidade com diferentes *browsers* foi testada utilizando a plataforma *BrowserStack*, que é uma ferramenta que permite testar a aplicação em múltiplos ambientes. Os testes foram realizados nas versões mais recentes dos principais *browsers*:

- Google Chrome (versões acima da 90)
- Mozilla Firefox (versões acima da 88)
- Microsoft Edge (versões acima da 90)
- Safari (versões acima da 18)

A aplicação é compatível com os *browsers* testados nas versões em que validámos o seu funcionamento.

Capítulo 6

Conclusões e Trabalho Futuro

O desenvolvimento deste projeto resultou numa aplicação que responde às necessidades dos estudantes do ISCAL. Desde o início, o objetivo foi construir uma solução prática, que facilitasse a leitura e exploração da informação, e ao mesmo tempo fosse escalável para crescer com os utilizadores e os dados.

6.1 Conclusões

Um dos pontos fortes deste projeto foi a implementação de um mecanismo para gerir ficheiros e transformar dados em gráficos. A funcionalidade de os utilizadores poderem carregar ficheiros diretamente exportados do simulador e, a partir daí, obterem gráficos, sem qualquer necessidade de formatação manual representa uma melhoria relação ao processo anterior, que era manual, demorado e propenso a erros.

A decisão de usar *Django* como base para o *backend* revelou-se acertada. Além de facilitar a gestão de utilizadores, a *framework* permitiu implementar arquitetura sólida, com separação entre as camadas *frontend* e *backend*. Os *endpoints* que foram desenvolvidos permitiram uma comunicação rápida entre as várias partes da aplicação, suportando atualizações dinâmicas dos gráficos sem impacto na performance.

No lado do *frontend*, optou-se por uma abordagem prática, usando bibliotecas para garantir a responsabilidade e usabilidade. A biblioteca *Plotly* foi importante para criar gráficos interativos que ajudam os estudantes a

interpretar os dados.

Durante o desenvolvimento surgiram alguns desafios técnicos. Um dos principais foi lidar com os dados em cada um dos ficheiros. Alguns traziam folhas com formatação irregular, nomes de colunas inconsistentes ou dados difíceis de perceber. Essas dificuldades levaram-nos a investigar a fundo como alguns dos cálculos eram feitos, e perceber as relações entre as várias colunas, e pensar na melhor maneira de conseguir representar essa informação.

Apesar da solução atual funcionar bem no contexto do projeto, poderá ser necessário alterar no futuro caso os ficheiros evoluam ou incluam novos tipos de dados. Outro ponto que pode ser melhorado é a interface de gestão de *quarters*, que embora funcional, ainda exige alguma familiaridade com o conceito por parte do utilizador.

Para além do lado técnico, este projeto foi também uma excelente oportunidade de crescimento pessoal. Antes de começar, não tinha experiência prática com *Django* nem com bibliotecas como *Pandas*, e o contacto com estas tecnologias acabou por ser interessante.

Em suma, este projeto mostrou como a combinação de ferramentas pode resultar numa solução com impacto na experiência de aprendizagem. A aplicação desenvolvida deixa aberta a porta para melhorias futuras e novos casos de utilização. É uma base que pode ser estendida ou adaptada a outros contextos onde a análise de dados seja um desafio.

6.2 Trabalho Futuro

Apesar de a aplicação atual já cumprir os objetivos definidos inicialmente, identificamos oportunidades de expansão que poderão ser exploradas em futuras iterações do projeto.

Interface e Experiência do Utilizador

Uma das melhorias mais imediatas que podiam ser feitas é ao nível da interface. Pode-se alargar a diversidade de visualizações disponíveis, incluindo novos tipos de gráficos, sendo que é sempre importante ter em mente a experiencia de utilização. Para além disso, seria interessante permitir a criação de

gráficos personalizáveis, onde cada utilizador poderia compor visualizações de acordo com os seus objetivos, e até juntar vários tipos de dados na mesma visualização.

Outro aspeto relevante prende-se com a configuração dinâmica dos gráficos. A ideia é deixar de apresentar visualizações estáticas e passar a oferecer ao utilizador a possibilidade de escolher, em tempo real, os tipos de gráfico, métricas e dimensões que pretende analisar. Esta abordagem aumentaria a flexibilidade da ferramenta.

Análise de Dados e Otimização de Performance

Uma outra abordagem que pode ser interessante passa pela incorporação de técnicas de análise preditiva. A utilização de modelos de regressão ou algoritmos de aprendizagem automática poderá ajudar os estudantes a identificar padrões e antecipar tendências de mercado com base nos dados da simulação.

Em paralelo, podia-se melhorar a eficiência da aplicação no processamento de dados de maior volume. Seria relevante avaliar o uso de bibliotecas com capacidade de tratar grandes quantidades de dado, mantendo tempos de resposta aceitáveis e garantindo uma boa experiência de utilização.

Gestão de Utilizadores e Colaboração

Ao nível da gestão da plataforma, podia-se reforçar as funcionalidades de administração de utilizadores. Isto poderia incluir a criação de perfis diferenciados (por exemplo, docente, grupo, estudante) e mecanismos de controlo de permissões.

Adicionalmente, seria interessante introduzir funcionalidades de colaboração entre utilizadores, nomeadamente a partilha de visualizações e filtros de dados. Este tipo de funcionalidade é particularmente útil em contextos académicos, onde os trabalhos são frequentemente realizados em grupo ao longo do semestre.

Integração com *Marketplace Simulations*

Por fim, um caminho a explorar passa pela integração direta com a plataforma *Marketplace Simulations*, de onde os dados são originalmente extraídos. Uma abordagem possível seria o desenvolvimento de extensões de

browser (para *Chrome*, *Firefox* ou *Edge*) que permitam importar automaticamente os dados para a aplicação.

De forma geral, estas propostas de melhoria mostram que o projeto pode continuar a evoluir significativamente, servindo como base para iterações futuras.