
# Plataforma para o Agregador de Energia

Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e Desenvolvimento (GECAD)

2019 / 2020

**1170527 Carlos Coelho**

# Plataforma para o Agregador de Energia

Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e Desenvolvimento (GECAD)

2019 / 2020

**1170527 Carlos Coelho**


## Licenciatura em Engenharia Informática

Setembro de 2020

Orientador ISEP: **Professor Doutor Carlos Ramos**

Supervisor Externo: **Doutor Pedro Faria**

*«My method is different. I do not rush into actual work. »*

*Nikola Tesla*

# Agradecimentos

À minha família por todo o apoio e formação contínua, que me permitiram estar nesta posição, principalmente, pelo suporte incondicional nesta última etapa.

Ao Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e Desenvolvimento (GECAD), pelo financiamento da bolsa na qual o trabalho foi desenvolvido e enquadrado no projeto Horizon-2020 DOMINOES da União Europeia.

À Professora Doutora Zita Vale, pela oportunidade de participar num projeto com metas tão importantes no panorama de um futuro mais verde.

Ao Professor Doutor Carlos Ramos, pela orientação, disponibilidade e compreensão ao longo do projeto e principalmente na revisão deste documento.

Aos Doutores Luís Gomes e Pedro Faria, pela supervisão, paciência, repreensões e compreensão, que me permitiram evoluir além das competências técnicas.

À Engenheira Brígida Teixeira, pela paciência, preocupação, auxílio e empenho ao longo de todo o projeto e por garantir que o resultado obtido era sempre o melhor possível.

Aos meus amigos com os quais limitei o contacto para a elaboração deste relatório, pela compreensão.

# Resumo

A rede de energia inteligente é inegavelmente o próximo passo na evolução da rede elétrica. A sua natureza distribuída e conectada não só a torna mais resiliente, como facilita a integração de recursos de energia distribuídos. Ao contrário das redes tradicionais, onde a central elétrica varia a sua produção para corresponder à demanda dos consumidores, as redes de energia inteligentes permitem, através de programas de *demand response*, que sejam os consumidores a alterar os seus consumos, de modo a equilibrar o consumo de acordo com as variações da geração. Contudo, estes programas requerem reduções no consumo demasiado elevadas para um utilizador final isolado. A solução consiste na agregação de participantes, permitindo-os participar nos programas referidos como um todo e dividir as suas recompensas. Este processo é realizado pelo Agregador de energia, que possui um variado portefólio de recursos energéticos distribuídos como utilizadores finais e fontes energia renovável.

O presente relatório documenta o processo de desenvolvimento de uma plataforma para o Agregador de energia que permite a monitorização dos seus recursos de energia distribuídos, a agregação desses recursos com diferentes configurações e a avaliação dessas mesmas agregações. Os dados são obtidos através de uma API ligada à rede ou de ficheiros e podem ser alterados na plataforma. Deste modo, o Agregador pode simular vários cenários e avaliar os seus resultados, optando posteriormente pelas configurações com melhor desempenho.

No desenvolvimento desta plataforma foi implementado um sistema de gestão de cenários, para que o Agregador tivesse todos os seus experimentos centralizados. Além disso, foi também desenvolvido um sistema de tarefas que permite a monitorização de ações que, de outro modo, resultariam em longos períodos sem *feedback*.

Com o término do projeto, foi desenvolvida uma plataforma confiável e conectada a uma rede de energia inteligente. A solução desenvolvida permite a análise de vários cenários, reais e fictícios, que poderão ser utilizados como base para novos estudos na área de *demand respons*e e agregação de participantes.

**Palavras-chave (Tema):** Agregação de participantes; Plataforma de simulação; *Demand Response*

**Palavras-chave (Tecnologias):** *Clustering*; Task Queue; *Python*; *R*; *React*;

# Índice

# Índice de Figuras

# Índice de Tabelas

# Índice de Excertos de Código

# Notação e Glossário

| | |
|---|---|
| **AAA** | *Arrange Act Assert* |
| **AMQP** | *Advanced Message Queueing Protocol* |
| **API** | *Application Programming Interface* |
| **AOF** | *Append Only File* |
| **BD** | Base de Dados |
| **DBMS** | *DataBase Management System* |
| **DER** | *Distributed Energy Resources* |
| **DR** | *Demand Response* |
| **DSO** | *Distribution System Operator* |
| **DTO** | *Data Transfer Object* |
| **EDP** | Energias De Portugal |
| **FER** | Fonte de Energia Renovável |
| **FIFO** | *First-In First-Out* |
| **FURPS** | *Functionality, Usability, Reliability, Performance, Supportability* |
| **GECAD** | Grupo de Investigação em Engenharia e Computação Inteligente |
| **GRASP** | *General Responsibility Assignment Software Patterns* |
| **H2020** | Horizonte 2020 |
| **IPP** | Instituto Politécnico do Porto |
| **ISEP** | Instituto Superior de Engenharia do Porto |
| **JSON** | *Javascript Object Notation* |
| **MB** | *Message Broker* |
| **MQ** | *Message Queue* |
| **MQTT** | *Message Queuing Telemetry Transport* |
| **Redis** | *REmote DIctionary Server* |

| | |
|---|---|
| **REST** | *Representational State Transfer* |
| **RDB** | *Redis DataBase file* |
| **RQ** | *Redis Queue* |
| **SG** | *Smart Grid* |
| **SOLID** | *Single-responsibility pinciple, Open-closed pinciple, Liskov substitution pinciple, Interface segregation pinciple e Dependency inversion pinciple* |
| **SS** | *Silhouette Score* |
| **TQ** | *Task Queue* |
| **TSO** | *Transmission System Operator* |
| **UI** | *User Interface* |
| **UML** | *Unified Modeling Language* |
| **URL** | *Uniform Resource Link* |
| **VPP** | *Virtual Power Plant* |
| **VPS** | *Virtual Power Solutions* |
| **W** | *Watts* |
| **WSS** | *Within cluster Sum of Square* |

# 1 Introdução

Neste capítulo é apresentado o enquadramento do tema, a descrição do problema, os objetivos e os contributos do trabalho desenvolvido. Ademais, são descritos a abordagem utilizada e o planeamento do projeto. Por último, é descrita a estrutura do documento.

## 1.1 Contexto

O projeto descrito neste relatório foi desenvolvido no âmbito da unidade curricular de Projeto/Estágio (PESTI) da Licenciatura em Engenharia Informática (LEI) do Instituto Superior de Engenharia do Porto (ISEP) do Instituto Politécnico do Porto (IPP). Foi desenvolvido na sequência de uma bolsa financiada pelo Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e Desenvolvimento (GECAD) e concretizado sob a orientação do Professor Doutor Carlos Ramos e supervisão dos Doutores Luís Gomes e Pedro Faria.

O trabalho desenvolvido na bolsa e investigação está associado ao projeto internacional DOMINOES, que é financiado pelo programa europeu de apoio à investigação e inovação, o Horizonte 2020 (H2020). O DOMINOES (http://dominoesproject.eu/) tem como objetivo mostrar como um distribuidor de energia pode fazer a gestão do balanceamento da rede ativa de modo dinâmico. Com o intuito de cumprir esse objetivo, alicerceia-se em redes elétricas inteligentes (do Inglês, *Smart Grids*, SG) para o desenvolvimento de novas tecnologias de *Demand Response* (DR), agregação, gestão de rede e serviços de negociação *peer-to-peer*. Para a concretização destas metas, o projeto é composto por diversos parceiros internacionais: *Empower* (coordenador – Finlândia), Energias De Portugal (EDP) (Portugal), ISEP (GECAD/IPP), Universidade de Tecnologia de Lappeenranta (Finlândia), *Virtual Power Solutions* (VPS) (Portugal), Universidade de Leicester (Reino Unido), e a Universidade de Sevilha (Espanha) [1].

No entanto, o DOMINOES não é o único projeto europeu que explora o potencial das SG. Na Europa, a transição este tipo de redes está a ser fortemente incentivada não só através de diretivas para a sua implementação, mas também pelo financiamento de projetos de investigação e desenvolvimento [2]. Deste modo, o programa H2020 financiou 173 projetos no valor total de 82,72M€ sob o tópico "*Demonstration of smart grid, storage and system integration technologies with increasing share of renewables: distribution system*" [3], [4].

## 1.2 Enquadramento

Tradicionalmente, devido a limitações tecnológicas, a rede elétrica apenas permitia o fluxo de energia num sentido: das centrais de produção para os consumidores finais [5], [6]. Por outro lado, o fluxo de informação apenas ocorre no sentido inverso [7]. Porém, esta abordagem unidirecional e centralizada está em vias de extinção sendo adaptadas ou substituídas por alternativas mais modernas e versáteis – as Redes Elétricas Inteligentes [8].

Idealmente, uma SG é uma rede que permite o fluxo de energia e informação constante em ambos os sentidos, viabilizando a integração de recursos de energia distribuídos (do Inglês, *Distributed Energy Resource*, DER), como Fontes de Energia Renovável (FER) e flexibilidade de consumo elétrico [9]. Assim, habilita e estimula o utilizador final da rede a exercer um papel ativo na rede, como, por exemplo, ao alterar os seus hábitos de consumo em função de incentivos e desincentivos, sendo uma fonte de flexibilidade elétrica para a rede [10]. Estes programas são denominados de programas de DR e dividem-se em dois tipos – baseados em incentivos e baseados no preço [11].

Muitos dos recursos energéticos distribuídos são caracterizados pela sua pequena capacidade de produção e estão ligados a estruturas de baixa e média tensão. Embora uma SG permita a participação ativa de DER na rede e mercados de energia, a dimensão individual de cada um destes recursos revela-se ser um dos maiores obstáculos à inclusão eficaz dos participantes na rede e na participação competitiva nos mercados de energia [12]. Por um lado, os programas de DR têm requisitos que um utilizador final, individualmente, não consegue atingir [13], [14]. Por outro lado, as FER tendem a ser instáveis, com grande variância e de baixa e média tensão [15].

Deste modo, surge o agregador: uma nova entidade que possui um variado e completo portfólio de DERs e que os agrega de forma a que possam participar nos mercados como um coletivo. Esta abordagem combate os problemas de baixa capacidade de produção e instabilidade, anteriormente colocados através da quantidade. Primeiramente, ao agrupar vários DER, naturalmente a capacidade total é maior. Além disso, à medida que são acrescentados mais DER, a tendência é que estes se equilibrem, criando uma *baseline* mais estável e confiável [16]–[18].

Para a gestão e agregação dos recursos de energia distribuída, o agregador serve-se de uma plataforma web que permite o acompanhamento em tempo real de todos os participantes da rede.

## 1.3 Descrição do Problema

A integração eficiente de DER na rede pode ser dividida em duas partes: a participação ativa dos utilizadores finais através de programas de DR e a incorporação de FER. Idealmente, a primeira será utilizada sobretudo como resposta à segunda.

Contudo, de modo a obter resultados mais previsíveis dentro de cada agregado, é do interesse do agregador agrupar as suas DER consoante os seus padrões de consumo, geração, entre outros. Assim, a agregação deverá ser feita tendo em conta os diversos parâmetros de cada DER, podendo o peso (relevância) de cada parâmetro ser variável na formação de grupos.

Desta forma, os critérios de agregação mais eficazes poderão variar dependendo do *dataset* a ser analisado. Um *dataset* pode ser definido como o conjunto das medições dos diferentes parâmetros de cada participante num intervalo de tempo. Assim, o agregador poderá gerar e editar *datasets*, podendo, então, testar várias configurações em diferentes alturas do ano, por exemplo. Ademais, sendo um projeto em desenvolvimento e com muitos componentes físicos, a existência de medições com falhas é inevitável, o que pode afetar a integridade dos agregados criados.

Por fim, os processos de recolha de dados e de agregação de participantes podem ser morosos, pelo que é do interesse do agregador a monitorização dos mesmos.

### 1.3.1 Objetivos

Este projeto tem como finalidade o desenvolvimento de uma plataforma que permitirá ao agregador fazer a monitorização e agregação das suas DERs. Assim, foram definidos os seguintes objetivos:

- *Estudo da área do projeto* – de forma a compreender o domínio em que o problema se insere, nomeadamente, redes elétricas inteligentes, agregadores de energia e programas de *demand response*;
- *Estudo de algoritmos de clustering* – de modo a compreender e avaliar os resultados gerados na agregação de participantes;
- *Estudo de trabalhos semelhantes* – de modo a perceber os trabalhos existentes sobre este tema e a sua abordagem;
- *Levantamento e análise de requisitos* – que serão utilizados como parâmetros de avaliação da solução desenvolvida;
- *Desenho da solução* – que permita o desenvolvimento rápido e modular da plataforma;

- *Desenvolvimento e adaptação de algoritmos de clustering* – para a agregação de participantes
- *Desenvolvimento do frontend* – flexível e simples que permita a gestão de participantes, agregação e a análises gráficas dos seus resultados
- *Desenvolvimento do backend* – de suporte ao *frontend* que forneça serviços de persistência, agregação e comunicação com outros servidores
- *Testes e Avaliação* – de modo a garantir que a solução desenvolvida é de qualidade e respeita todos os requisitos definidos.

### 1.3.2 Abordagem

Numa primeira fase, os objetivos eram a familiarização com os conceitos do domínio e tecnologias a utilizar. Primeiramente, foram estudados os conceitos de SG, agregadores de energia e programas de DR para que posteriormente os requisitos pudessem ser mais facilmente compreendidos. De seguida, foram explorados os algoritmos de *clustering* utilizados, para futura adaptação e implementação. A última etapa desta fase teve como finalidade a familiarização com as tecnologias a utilizar no projeto, nomeadamente *Python* e *React*.

As fases seguintes seguiram uma abordagem baseada na *framework Scrum*. O principal objetivo do *Scrum* é a construção iterativa de soluções estáveis através de uma abordagem empírica e dinâmica, sendo os principais conceitos:

- *Product Backlog* – Lista dinâmica de prioridades com os requisitos necessários para o produto final;
- *Product Manager* – Representante do cliente e outros *stakeholders*, esclarece o *Scrum Master* em questões que tenham surgido e atualiza o *product backlog*;
- *Scrum Master* – Coordenador da equipa, ajuda a equipa a aplicar as práticas *Scrum* e permite que a equipa se foque no desenvolvimento do *software*, responsabilizando-se pelos assuntos externos; é o intermediário de comunicação com o *Product Owner*;
- *Daily Scrum* – Reunião diária de curta duração para discutir os principais avanços e bloqueios do projeto;
- *Sprint* – É uma etapa de desenvolvimento, é caracterizado por um conjunto de requisitos a desenvolver e pela sua duração;
- *Sprint Review* – Reunião com os *stakeholders* para apresentar o trabalho desenvolvido e receber *feedback*.

Como representado na Figura 1, durante o projeto, alguns destes conceitos foram substituídos ou modificados. Sendo este um trabalho individual, não existiram reuniões diárias. No entanto, o papel de *Scrum Master* foi adotado pelos supervisores, existindo, a cada 3 dias, atualizações dos progressos ou obstáculos do projeto. Os supervisores exerceram ainda o papel de *Product Manager*, avaliando e atualizando a lista de requisitos nas reuniões bimensais. Deste modo, os sprints tinham a duração de 2 semanas.

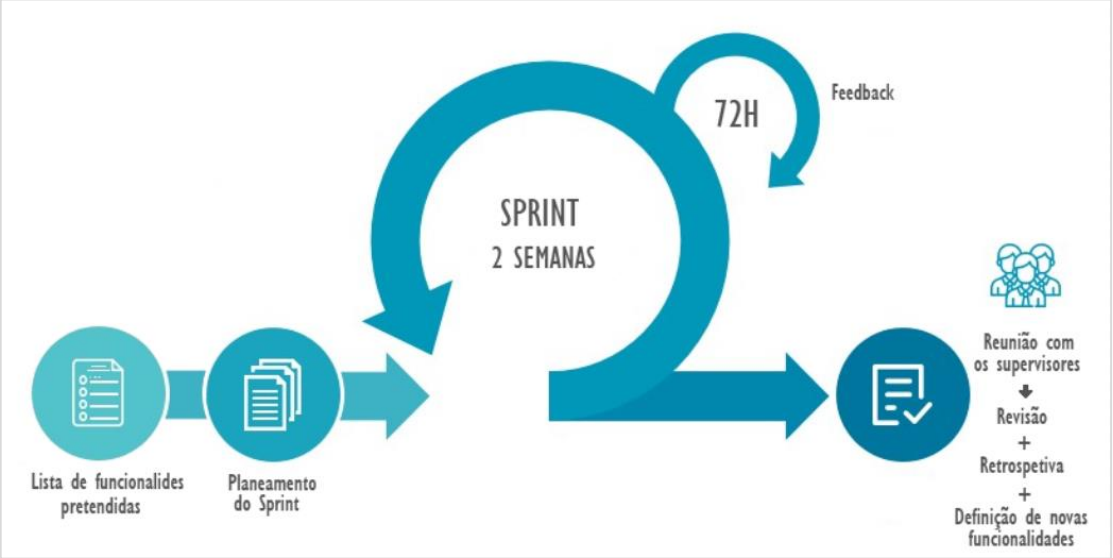



*Figura 1 – Processo de desenvolvimento baseado em Scrum (adaptada de* [19]*)*

### 1.3.3 Contributos

Do trabalho desenvolvido neste projeto resultou uma plataforma para o agregador que será incluída no *Work Package* 3.5 – "*Aggregation based demand response*" do projeto DOMINOES (id 771066) no qual o ISEP é representado pelo GECAD [20].

Especificando, através do sistema de *datasets*, da agregação parametrizada e dos métodos de avaliação dos agregados, a plataforma resultante permite a análise de desempenho de diferentes configurações de parâmetros para diferentes cenários reais ou simulados através de algoritmos e de gráficos dinâmicos.

Além disso, a validação das medições existentes informa o agregador da viabilidade dos dados recolhidos, permitindo uma escolha consciente aquando a importação dos mesmos.

Por fim, a execução em segundo plano e respetivo rastreamento não só dos processos de agregação, mas também de recolha de dados, libertam o agregador para (I) utilizar plataforma normalmente enquanto os processos decorrem e (II) fazer testes a *datasets* e configurações diferentes em simultâneo, sem a necessidade de a aplicação estar aberta.

### 1.3.4 Planeamento do trabalho

Tal como anteriormente referido na Abordagem, o projeto foi dividido em sprints de duas semanas, tendo sido realizados, até ao momento, os descritos na Tabela 1. O cronograma do projeto está dividido por objetivos e pode ser consultado na Figura 2.

*Tabela 1 – Resumo dos Sprints*

| **Sprint** | **Resumo** |
|---|---|
| *00* | Familiarização com o tema, algoritmos de *clustering* e a biblioteca *frontend, React;* Implementação de um mecanismo *Drag'n'Drop* |
| *01* | Pesquisa e implementação de tecnologias de filas de tarefas em *Python*; Conexão entre o *Python* e um ambiente *R*; Adaptação de algoritmos de *clustering* |
| *02* | Rastreamento de tarefas; Pedidos de agregação através da interface gráfica. |
| *03* | Visualização e edição de um *dataset* na interface gráfica |
| *04* | Agregação através da interface gráfica; Gestão de *datasets* |
| *05* | Expansão para várias agregações por pedido; O pedido agora é dependente de um *dataset*; Persistência dos pedidos e tarefas de agregação |
| *06* | Implementação do rastreamento das tarefas de agregação na interface gráfica |
| *07* | Expansão da visualização dos resultados de agregação |
| *08* | *Download* de *datasets* e *download* de resultados de agregação |
| *09* | Preparação da Apresentação para o GECAD |
| *10* | Documentação e Implantação da plataforma |
| *11* | Familiarização com a API do DOMINOES; Testes de eficiência à API; Persistência dos dados da API |
| *12* | Importação de dados da API para a interface gráfica |

É de realçar que o *sprint* 00 e o *sprint* 11 tiveram excecionalmente a duração de 4 semanas. No primeiro caso isso deveu-se ao facto de ser o *sprint* introdutório e de familiarização com a equipa, tema e tecnologias a utilizar. No segundo caso, trata-se de um mês onde existiu pouca disponibilidade e a necessidade de familiarização com a API

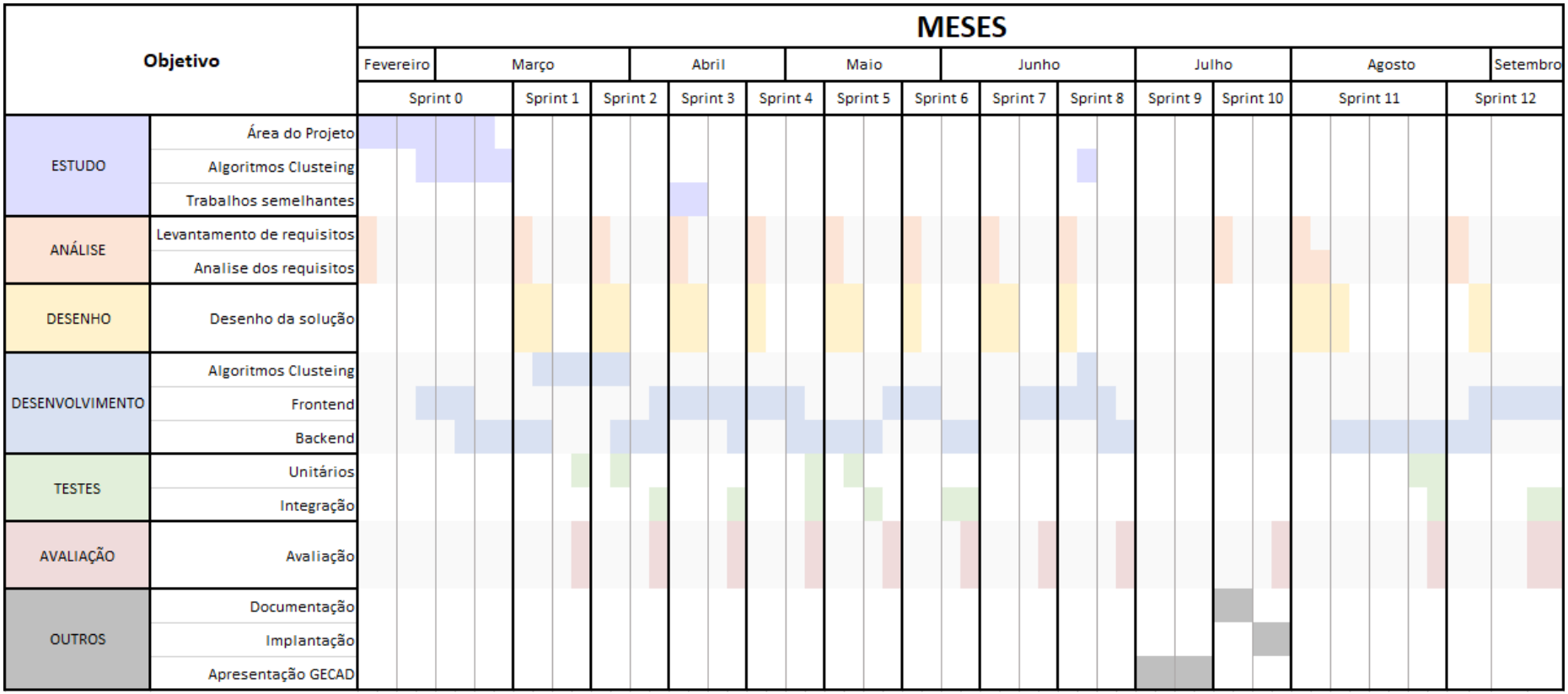



*Figura 2 – Cronograma do planeamento dos objetivos*

## 1.4 Estrutura do relatório

O documento encontra-se subdividido em seis capítulos. Após a presente secção introdutória, no capítulo 2, serão primeiramente explorados os principais conceitos teóricos da área de energia referentes às redes elétricas inteligentes. De seguida são descritas e comparadas diferentes técnicas de *clustering* e implementações de filas de tarefas. Além disso, são expostos conceitos alicerceais à arquitetura *web* moderna. Por fim, são apresentados trabalhos que pretendem resolver problemas semelhantes ao do projeto desenvolvido.

No capítulo 3 o problema é analisado. Par tal, este é detalhado e decomposto de forma a evidenciar os passos necessários à sua resolução. Posteriormente, é apresentado o modelo de domínio criado e realizado o levantamento de requisitos.

O capítulo 4 é referente ao processo de desenho da solução final anteriormente analisada. Deste modo, são propostas e avaliadas diferentes alternativas.

No capítulo 5, é evidenciada a implementação da solução proposta no desenho dividida por etapas de integração. São, ainda, apresentados os testes realizados e a avaliação do trabalho desenvolvido.

No capítulo 6, são listadas as conclusões do projeto: objetivos concretizados, limitações da solução, possíveis desenvolvimentos futuros e uma apreciação pessoal do trabalho realizado.

Por fim, o documento contém as secções de referências e anexos.

# 2 Estado da arte

Este capítulo é dedicado à exploração de conceitos teóricos e tecnologias relacionados com o trabalho desenvolvido. Primeiramente, é realizada uma contextualização referente à área da energia. De seguida, são analisados algoritmos de *clustering* e implementações de filas de tarefas. São ainda apresentados elementos que moldam a arquitetura web moderna e, por fim, trabalhos semelhantes ao desenvolvido.

## 2.1 Redes Elétricas Inteligentes

Desde a sua invenção na segunda metade do século XIX, a principal alteração nos sistemas de energia foi a normalização de grandes centrais de produção de energia face à inicial exclusividade da geração local. Além disso, os sistemas evoluíram nas fontes de energia utilizadas, escala das centrais e meios de distribuição de energia. Porém, foi sempre preservado o caráter unidirecional do paradigma original [21]. Para realçar essa estagnação, a publicação [22] comparou os progressos na telefonia com os das redes de energia:

> *"Se Alexander Graham Bell fosse de alguma forma transportado para o século XXI, não conseguiria reconhecer os componentes da telefonia moderna […] enquanto que Thomas Edison, um dos principais arquitetos da rede [de energia], estaria totalmente familiarizado com a rede."*

O excerto acima apresentado, foi traduzido livremente pelo autor a partir do texto original em língua inglesa:

> *"If Alexander Graham Bell were somehow transported to the 21st century, he would not begin to recognize the components of modern telephony – cell phones, texting, cell towers, PDAs, etc. – while Thomas Edison, one of the grid's key early architects, would be totally familiar with the grid."*

Esta abordagem opera apenas num sentido: das centrais de geração para as linhas de transmissão de energia, ou *Transmission System Operator* (TSO) e do TSO para as redes de distribuição elétrica, ou *Distribution System Operator* (DSO) que, por sua vez, repartem a energia pelos consumidores finais.

Para garantir o bom funcionamento de um sistema de energia, a produção e o consumo de energia devem-se neutralizar, pois tanto a subcarga como a sobrecarga do sistema podem levar à sua falência. Neste modelo, as centrais são capazes de aumentar ou reduzir a sua geração célere e dinamicamente, ao alterar a quantidade de combustível utilizado. Este ajuste é feito dependendo da frequência elétrica da rede, que é o principal meio de comunicação entre o consumidor final e a central [5], [7].

Porém, com a normalização da eletricidade para uso doméstico, estas abordagens centralizadas começaram a revelar vulnerabilidades, nomeadamente a sua ineficiência em horas em que a demanda não está no seu pico. Segundo [23], 20% da capacidade de geração apenas é necessária para dar resposta às horas de ponta, que representam apenas 5% das horas totais. Ou seja, durante 95% dos períodos, as centrais funcionam abaixo de 80% da sua capacidade.

Além disso, na União Europeia, a integração de fontes de energia mais limpas e sustentáveis como as FER tem sido encorajada [24]. No entanto, estas fontes são caracterizadas pela sua geração intermitente devido a fatores climatéricos e pelo seu baixo nível de geração de energia comparativamente às centrais atualmente em vigor [25]. Ambas estas características são incompatíveis com modelo centralizado anteriormente descrito, dificultando a integração das FER no mesmo.

Uma SG é uma rede cujo modelo que se distingue da abordagem clássica por permitir o fluxo bidirecional de energia e, principalmente, informação [9]. Este sistema *duplex* não só habilita a participação ativa de todas entidades, mas também facilita a integração massiva de DER que tornam a rede mais redundante e resistente a falhas ou ataques [6], [8], [26].

Porém, a integração eficiente de FER mantém-se um obstáculo. Continua a ser necessário lidar com a intermitência e aparente aleatoriedade destas fontes de energia. Não só podem existir alturas em que a produção é abaixo do esperado, como momentos em há picos de geração que sobrecarregam a rede [27].

Flexibilidade *e DR*, são conceitos abordados de seguida que podem ser melhor explorados devido à natureza distribuída e conectada das SGs. Procuram solucionar os desafios gerados por alterações no equilíbrio entre a produção e o consumo causados tanto pela intermitência e variância na geração como pelo aumento do consumo em horas de ponta.

### 2.1.1 Programas de *Demand Response*

Segundo [28], a flexibilidade de um sistema pode ser definida como a sua capacidade de modificar a produção ou consumo de eletricidade em resposta a um evento. Nos modelos centralizados, esta flexibilidade era facultada pela própria central, ao alterar a produção consoante a o consumo [5]. Por outro lado, a SG permite que este ajuste seja feito por ambas as partes, envolvendo efetivamente os seus participantes no equilíbrio da rede.

Na literatura, *Demand Response* é apresentada como a alteração dos hábitos de consumo de um utilizador final em resposta a variações no preço da energia. Tipicamente, isto envolve

sacrifícios por parte do utilizador final como um menor nível de conforto ou alteração de rotinas. Um dos principais objetivos dos programas de DR é minimizar o impacto das horas de ponta e estes programas podem-se dividir em dois modelos [11]:

1. Baseados em incentivos – a rede pede aos participantes a alteração do consumo, existindo benefícios ou penalizações resultantes do cumprimento desse pedido
2. Baseados no preço – o preço da eletricidade aumenta consoante a demanda, desencorajando o consumo de energia nesses períodos

Exemplificando, um participante do modelo 1, pode programar os seus dispositivos de aquecimento inteligentes para responderem aos pedidos de redução do consumo da rede, abdicando de algum do seu conforto em troca de benefícios, como pagamentos em função da sua redução. Por outro lado, no modelo 2, o participante poderá mover tarefas domésticas de alto consumo energético como o uso de fornos ou máquinas de lavar para períodos em que o preço da eletricidade seja mais baixo. O utilizador final poderá ainda usufruir de uma combinação de ambos os modelos, uma vez que estes não são mutualmente exclusivos [29], [30].

Por outro lado, em vez de reduzir o consumo, o participante pode gerar localmente uma parte ou a totalidade da energia que consome, efetivamente diminuindo o consumo energético do ponto de vista da rede. Esta é uma opção viável quando o preço da energia ou o incentivo de redução é superior ao custo dessa geração localizada [29]. Do mesmo modo, o participante pode produzir mais energia que o seu consumo, vendendo-a de volta à rede. Estes participantes que podem fazer tanto o papel de consumidor como de produtor designam-se de “prosumidores” [31].

Desta forma, DR pode ser descrita como a flexibilidade da rede derivada dos consumidores.

### 2.1.2 Agregadores de energia

Segundo [32], uma Central Virtual de Energia (do Inglês, *Virtual Power Plant*, VPP) é um conjunto de DERs, isto é, geradores, cargas controláveis e armazenamento de energia que, reunidos, funcionam como uma central de energia tradicional, sendo capaz de responder flexivelmente às necessidades energéticas da rede em que se insere.

De modo semelhante, [15] descreve o agregador de energia como uma entidade que agrupa diferentes DERs de modo a agirem como apenas uma unidade nos mercados de energia. Em [33], a palavra demanda é adicionada ao conceito, formando o agregador de *demanda* de energia. Neste caso, as DERs correspondem maioritariamente a consumidores finais, sendo o

foco do agregador disponibilizar flexibilidade de consumo para a rede recorrendo a programas de DR.

Por outro lado, o conceito de agregador tem sido popularizado como o agente responsável pela gestão de uma VPP, disponibilizando flexibilidade para a rede não só pela aplicação de programas de DR, mas também através da geração de energia como descrito em [34], [35].

No contexto deste documento, um agregador é uma entidade que possui um vasto portefólio DERs como FER, consumidores finais e até mesmo VPPs.

## 2.2 Algoritmos de *Clustering*

O *clustering*, é uma técnica de Mineração de Dados (do Inglês, *Data Mining*) que tem como objetivo a formação de grupos. Um grupo deve conter elementos semelhantes entre si e, simultaneamente, distintos dos elementos dos outros grupos [36].

Porém, existe uma vasta variedade de algoritmos de agregação e métodos de avaliação. Os métodos de avaliação são, geralmente, utilizados juntamente com algoritmos que permitem escolher o número de grupos a utilizar como o de Agregação Baseada em Centroides.

### 2.2.1 Agregação Baseada em Centroides

No contexto deste documento a Agregação Baseada em Centroides (do Inglês, *Centroid-Based Clustering*) refere-se exclusivamente ao conceito apresentado por Steinhaus em [37], intitulado de "*k-means"* por Macqueen em [38]. Este tem como objetivo minimizar as diferenças internas iterativamente, para um número definido de grupos *k*. É um dos algoritmos de agregação mais simples e com complexidade mais baixa, O(n). Para tal, este algoritmo obedece aos seguintes passos, refletidos na Figura 3 para *k*=3:

I. São escolhidos, aleatoriamente, 3 elementos diferentes, cada um representando o valor inicial do centroide de um grupo (na figura, 1.1)
II. Para cada ponto, são medidas as distâncias aos *k* centroides, e é-lhe atribuído o grupo cuja distância medida foi menor (na figura, 1.2)
III. Os valores dos centroides são recalculados, correspondendo às médias dos elementos do respetivo grupo (na figura, 1.3)
IV. Repetem-se os passos II e III até que não tenham existido alterações nos grupos ou seja alcançado um critério de paragem alternativo (na figura, 2.1 a 3)
V. Repetem-se os passos I a IV o número de vezes definidos, *n*, e em cada a diferença entre os elementos dos grupos é avaliada, selecionando a melhor iteração

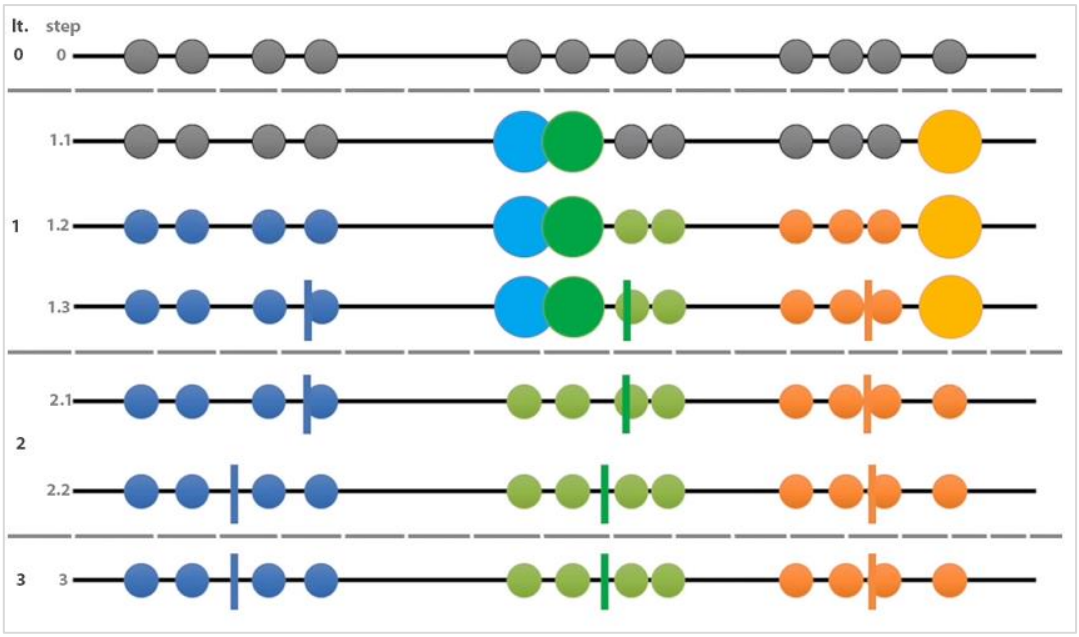



*Figura 3 – Visualização da Agregação baseada em Centroides (adaptado de* [39]*)*

### 2.2.2 Agregação Hierárquica

Na Agregação Hierárquica (do Inglês, *Hierarchical Clustering*), para *n* elementos, são inicialmente formados *n* grupos e, em cada iteração, são fundidos os dois mais semelhantes. Deste modo, o algoritmo apresenta uma complexidade de $O(n^2)$. Este processo repete-se até que exista apenas um grupo ou seja alcançado outro critério de paragem [40]. Por exemplo, na Figura 4, o primeiro grupo a ser gerado é o grupo E, fundindo os grupos A e B. O grupo final, G, contém dois subgrupos, E e F, ordenados hierarquicamente e ordem de formação, sendo G o grupo mais à direita e com maior nível hierárquico. Para obter o número de grupos desejado, *k*, basta consultar a iteração *n*-*k*.

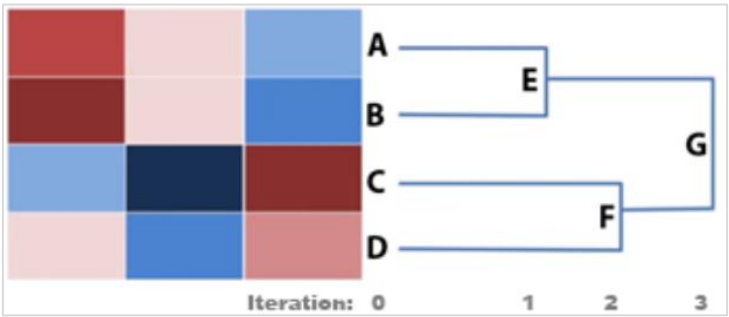



*Figura 4 – Visualização da Agregação Hierárquica (adaptado de* [41]*)*

### 2.2.3 Agregação Baseada em Densidade

A Agregação Baseada em Densidade (do Inglês, *Density-Based Clustering*) distingue-se pela sua capacidade de identificar conjuntos com formas abstratas, como os representados na Figura 5. Outro fator de diferenciação é que este algoritmo não precisa de saber quantos

grupos formar, mas sim qual a distância máxima, Ɛ, entre dois pontos pertencentes ao mesmo grupo. Este algoritmo tem uma complexidade de $O(n^2)$, é iterativo e o seu modelo mais básico, embora apresente algumas limitações, pode ser descrito em 4 passos [42]:

1. É selecionado aleatoriamente um ponto que ainda não tenha um grupo atribuído;
2. São analisados todos os pontos num raio, Ɛ, selecionando o mais próximo sem grupo atribuído e atribuindo-lhe o grupo do anterior; um ponto que não tenha qualquer vizinho nesse raio é considerado um *outlier*;
3. Repete-se o passo 2 até não existir um ponto sem grupo no raio;
4. Repete-se o passo 1-3 até não existirem mais pontos sem grupo atribuído.

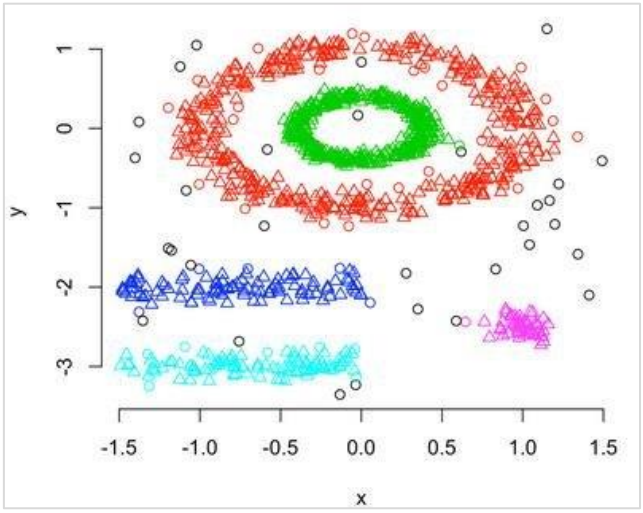



*Figura 5 – Exemplo dos grupos formados pelo Density-based Clusteing (retirado de* [43]*)*

### 2.2.4 Comparação dos Algoritmos de *Clustering*

Embora o algoritmo a utilizar tenha sido imposto pelo GECAD, foram avaliados os seguintes critérios discriminados na Tabela 2: (I) Possibilidade de escolha do número de grupos desejado, (II) Complexidade temporal do algoritmo e (III) Determinismo do algoritmo.

Ambos os algoritmos Baseado em Centroides e Hierárquico permitem a formação de quantidades arbitrárias de grupos. Porém, o segundo tem uma complexidade temporal superior o que o torna menos escalável. Pelo mesmo motivo, o Baseado na Densidade não será a melhor solução a longo prazo e, além disso, não permite diferentes escolher a quantidade de grupos a formar. O único algoritmo totalmente determinístico é o Hierárquico, no entanto, é possível atenuar a natureza semi-aleatória dos algoritmos executando-os várias vezes.

Deste modo, a melhor opção será a Agregação Baseada em Centroides, embora a Agregação Hierárquica também seja uma alternativa a considerar, caso existam flutuações não desejadas nos resultados.

Tabela 2 – Comparação entre vários algoritmos de agregação

| ***Requisito*** | **Baseado em Centroides** | **Baseado em Densidade** | **Hierárquico** |
|---|---|---|---|
| *Agregação por K* | Sim | Não | Sim |
| *Complexidade* | O(n) | $O(n^2)$ | $O(n^2)$ |
| *Determinístico* | Não | Não | Sim |

### 2.2.5 Avaliação dos *Clusters*

Em métodos de *clustering* onde são permitidas diferentes configurações na geração de grupos, é importante que estes sejam avaliados, de modo a saber qual o melhor número de grupos (ou raio da vizinhança, no caso do Baseado em Densidade) para os dados agregados. Por norma, as medidas de avaliação são coeficientes correspondentes à homogeneidade intra-grupo e heterogeneidade inter-grupo. Deste modo, foram analisados os métodos de avaliação *Elbow*, *Silhouette* e *Gap Statistic*.

#### *Elbow*

No método *Ebow*, é calculada, para cada número de grupos a formar (*k)* é computada a soma dos quadrados das distâncias dos elementos de cada grupo ao seu centroide, ou *Within cluster Sum of Square* (WSS). De seguida, é gerado um gráfico, como o da Figura 6, representando no eixo horizontal o *k* e no eixo vertical a WSS.

Este método avalia a redução da variância, sendo considerado como o melhor *k*, o ponto onde essa redução estabiliza,  conhecido como cotovelo [44].

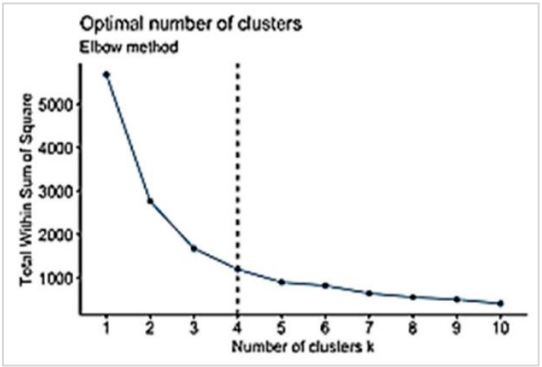



*Figura 6 – Avaliação Elbow (retirado de* [45]*)*

***Silhouette***

O *Silhouette Score* (SS) é uma medida que avalia a coesão dos grupos. Calcula não só a semelhança entre um ponto e os restantes do seu grupo, mas também a dessemelhança entre esse mesmo elemento e os dos outros grupos. Assim, é obtido um valor entre -1 e 1, onde 1 significa que existe uma alta coesão nos grupos e -1 reflete que não existe conformidade entre os elementos do mesmo grupo [46].

Na Figura 7, está representado um gráfico derivado deste método, sendo o eixo horizontal o número de grupos a utilizar, *k*, e no eixo vertical a média dos SS calculados. É considerado o melhor *k*, aquele onde a média dos SS é mais elevada.

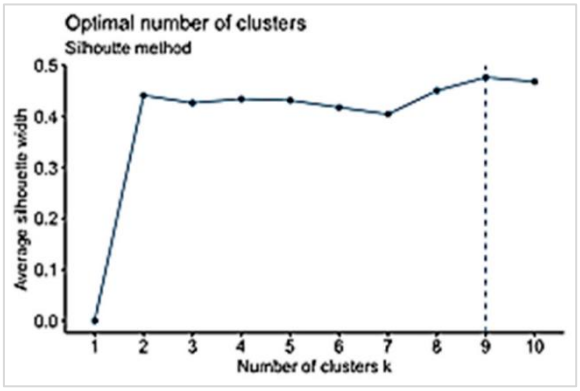



*Figura 7 – Avaliação Silhouette (retirado de* [45]*)*

***Gap Statistic***

O método *Gap Statistic* é uma alternativa que compara o WSS com os valores esperados por uma distribuição aleatória uniforme. Quanto menor for o valor desta estatística, mais semelhante o grupo é à distribuição referida.

A Figura 8 representa a estatística Gap (eixo vertical) gerada para cada número de grupos (eixo horizontal). Neste modelo, é considerado o melhor número de grupos (*k*), o primeiro *k* para o qual $Gap_k \geq Gap_{k+1} - S_{k+1}$ , sendo *S* o desvio padrão [47].

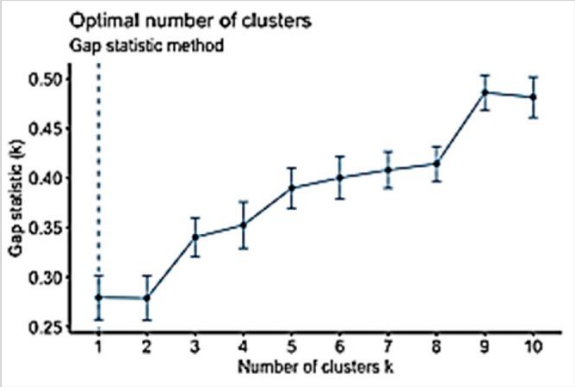



*Figura 8 – Avaliação GAP (retirado de* [45]*)*

## 2.3 *Brokers* e Filas de Tarefas

Uma vez que os algoritmos de agregação podem demorar até 5 minutos a executar (dependendo dos parâmetros), foram procuradas soluções que permitissem que o utilizador recebesse rapidamente uma resposta do servidor garantindo o bom funcionamento da plataforma [48]. Assim, foram exploradas bibliotecas de Filas de Tarefas (do Inglês, *Task Queue*, TQ), em *Python*.

Num sistema de gestão de tarefas são, primeiramente, definidas as tarefas que poderão, posteriormente, ser executadas. As TQ, geralmente, empreendem a metodologia *First In First Out* (FIFO) para gerir as tarefas, embora algumas implementações também integrem prioridades. A estrutura destes sistemas pode ser representada pela Figura 9. Neste exemplo, existe um processo principal que regista no Distribuidor de Mensagens (do Inglês, *Message Broker*, MB) a tarefa a executar e respetivos parâmetros. Ao receber novas tarefas o MB notifica os processos auxiliares, os *workers* e distribui-as ordenadamente pelos *workers* disponíveis. Por fim, as tarefas são executadas, pelos processos auxiliares, em segundo plano.

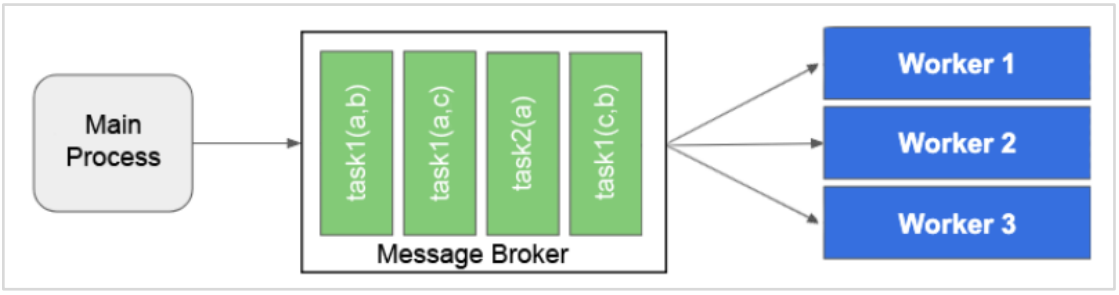



*Figura 9 – Exemplo do funcionamento de uma TQ (adaptado de* [49]*)*

### 2.3.1 Redis

O *Remote Dictionary Server* (Redis) é um sistema *open-source* grátis de gestão de BDs (do inglês, *Database Management System*, DBMS), criado por Salvatore Sanfilipo em 2009 e, desde 2015, oficialmente desenvolvido pela *Redis Labs* [50].

O Redis destaca-se por disponibilizar duas funcionalidades num só serviço: *cache* e Base de Dados (BD). Assim, toda a informação da BD é carregada para a memória do servidor, permitindo tempos de resposta inferiores a 1 milissegundo [51].

Para complementar esta funcionalidade, este sistema oferece duas opções de persistência: a *Redis Database file* (RDB), que guarda cópias exatas do estado atual da BD no disco e a Append Only Files (AOF), que regista os comandos executados num ficheiro de *logs;* possibilitando o uso de ambos os em simultâneo ou mesmo o uso de nenhum [52].

Ademais, pauta-se pela sua compatibilidade: suporta diferentes estruturas de dados como *strings*, listas, mapas e *sets*, além de implementar um modelo chave-valor [53]. Segundo a DB-Engines [54], foi considerado como sendo a melhor BD nesta categoria.

Além das funcionalidades anteriormente referidas, o *Redis* disponibiliza também uma implementação de *Publish/Subscribe,* podendo, então, ser utilizado como MB [55].

Por fim, é compatível com a grande maioria das linguagens de programação mais populares, incluindo *C++*, *Java*, *Perl*, *PHP*, *Python* e R mas apenas possui suporte oficial para sistemas operativos baseados em *Unix* [51], [56].

### 2.3.2 RabbitMQ

O *RabbitMQ* é um projeto *open-source* grátis, para a gestão e alistamento de mensagens (do inglês, *Message Queue, MQ*), lançado em 2007 pela *Rabbit Technologies* e, desde 2013, oficialmente desenvolvido pela *Pivotal Software* [57], [58].

A linguagem de programação utilizada no desenvolvimento do *RabbitMQ*, *Erlang*, foi desenvolvida pela multinacional de telecomunicações *Ericsson* e é a base de aplicações como o *WhatsApp* [59], [60]. Usufruindo das *frameworks* desbloqueadas pela linguagem, nomeadamente a *Open Telecom Platform*, o *RabbitMQ* oferece implementações para protocolos específicos à troca e alistamento de mensagens como o *Advanced Message Queueing Protocol* (AMQP) e o *Message Queuing Telemetry Transport* (MQTT) que garantem a fiabilidade e segurança deste processo [61].

Este MB divide as mensagens em dois tipos: as mensagens persistentes (do Inglês, *persistent messages*) que são escritas no disco quando são recebidas; e as mensagens transitórias (do

Inglês, *transient messages*) que apenas são escritas no disco quando é necessário libertar espaço na memória [62].

Todos estes fatores contribuem para que o *RabbitMQ* se destaque pela sua fiabilidade, escalabilidade e, ainda, capacidade de processar mensagens com maior dimensão [63].

Por fim, é compatível com a grande maioria das linguagens de programação mais populares, como *C++*, *Java*, *Perl*, *PHP* e *Python* [64]. Oferece ainda suporte para diversas plataformas, incluindo Linux, Windows, Windows Server e *macOS* [65].

### 2.3.3 Celery

O *Celery* é uma TQ *open-source* desenvolvida em *Python* e que foi lançada em 2009 por Ask Solem [66].

Esta biblioteca destaca-se pela sua robustez e flexibilidade, suportada por uma documentação bastante extensa e várias possibilidades de configuração [67]. Por estes motivos, a biblioteca é, por vezes, considerada demasiado complexa. É recomendada a utilização do *RabbitMQ* como MB, mas é também compatível com o *Redis*, o *Amazon Simple Queue Service* e o *Zookeper* [68].

Para os MB *RabbitMQ* e *Redis*, o *Celery* habilita a utilização de filas de prioridade (do Inglês*, priority queues*). Assim, ao criar uma tarefa, é possível atribuí-la a uma fila. Ademais, ao inicializar um processo *worker*, é, também, possível explicitar as filas a que está subscrito. É, ainda, recomendada a utilização de um maior número de *workers* para as filas consideradas mais prioritárias [69].

O *Celery* oferece, ainda, outras ferramentas complementares, nomeadamente o *Celery Beat*, que permite o agendamento de tarefas [70]; e o *Flower*, que permite monitorizar as tarefas através de uma interface gráfica [71].

Por fim, o *Celery* oferece suporte oficial para o *Windows* e sistemas *Unix*, embora o primeiro apenas seja garantido até à versão 4.0 [72].

### 2.3.4 Python Redis Queue

A *Python Redis Queue*, ou apenas *Redis Queue* (RQ), é uma biblioteca *open-source* de gestão de tarefas, publicada em 2012 por Vincent Driessen [73].

A RQ utiliza o *Redis* como MB e foi desenvolvida em *Python,* com o intuito de ser uma solução mais leve e fácil de utilizar face às alternativas já existentes e baseadas no protocolo AMQP, como o *Celery* [74].

Esta TQ permite categorizar as tarefas, sendo posteriormente gerada uma fila por cada categoria. Ademais, ao inicializar um processo *worker*, é possível explicitar quais as filas a que está subscrito, ordenando-as por prioridade [75].

Além da TQ, existem ainda outras ferramentas complementares à RQ, nomeadamente o *RQ-Scheduler*, que permite o agendamento de tarefas [76]; e o *RQ-Dashboard*, que permite monitorizar as tarefas recorrendo a uma interface gráfica [77].

No entanto, sendo baseada no Redis, a RQ apenas suporta oficialmente sistemas *Unix* [78].

### 2.3.5 Huey

O *Huey* é um projeto *open-source* publicado em 2011 por Charles Leifer [79]. Esta biblioteca foi escrita em *Python*, com o objetivo de ser uma TQ simples leve e flexível, suportando como MB o *Redis*, o *sqlite* e, ainda, a própria memória da máquina [80].

Ademais, o *Huey* integra diretamente a prioridade das tarefas, sendo possível especificar, na criação da tarefa, a sua prioridade [69]. Esta alternativa implementa dois tipos de tarefas: as normais, que são adicionadas à TQ quando são chamadas e as periódicas, que executam em intervalos previamente definidos [70].

Porém, o projeto apenas suporta oficialmente sistemas baseados em Unix [81].

### 2.3.6 Comparação entre as Filas de Tarefas

Para a seleção da TQ mais adequada, foram considerados os requisitos: (I) Compatibilidade com diferentes MB, (II) Facilidade de utilização, (III) Flexibilidade da biblioteca, (IV) Maturidade da tecnologia, (V) Interoperabilidade entre sistemas operativos, (VI) Qualidade da documentação e (VII) Suporte da Comunidade.

Apesar dos resultados promissores apresentados na Tabela 3, a falta de compatibilidade com plataformas não baseadas em *UNIX* inviabilizam a utilização do *PythonRQ* e do *Huey*. O *Celery*, por sua vez, também já não suporta oficialmente o *Windows,* mas as funcionalidades afetadas não são críticas para o desenvolvimento do projeto e, devido à sua comunidade, existem várias correções e implementações para o suporte multiplataforma. Além disso, é uma tecnologia com mais de uma década de existência, com uma vasta documentação e testada em inúmeros cenários, trazendo assim garantias de fiabilidade robustez.

Desta forma, apenas o *Celery* satisfaz o requisito de multiplataforma, sendo, portanto, a tecnologia mais adequada. Porém, é de salientar o facto que, em plataformas baseadas em *UNIX*, ambas as outras alternativas seriam opções viáveis para aplicações que não necessitassem da complexidade que o *Celery* oferece.

*Tabela 3 – Comparação das várias Task Queues em Python*

| ***Requisitos*** | **Celery** | **PythonRQ** | **Huey** |
|---|---|---|---|
| *Compatibilidade de Brokers* | RabbitMQ / Redis | Redis | Redis / sqlite / memória |
| *Facilidade de Uso* | Média | Alta | Alta |
| *Flexibilidade* | Alta | Médio | Alta |
| *Maturidade da Tecnologia* | Muito Alta | Alta | Alta |
| *Multiplataforma* | Sim | Não | Não |
| *Qualidade da Documentação* | Muito Alta | Média | Média |
| *Suporte da Comunidade* | Alto | Alto | Médio |

## 2.4 Arquiteturas *Web*

Com o intuito de construir uma aplicação *web* de qualidade, foram estudados conceitos elementares da arquitetura web moderna, nomeadamente o modelo *Client-Server* e serviços *REpresentational State Transfer* (REST). Além disso, foram exploradas duas abordagens contrastantes à construção de aplicações *web* – a arquitetura monolítica e a arquitetura de micro-serviços.

### 2.4.1 Modelo *Client-Server*

Na literatura, o modelo *Client-Server* refere-se a um sistema constituído por dois componentes: o servidor e o cliente. O primeiro possui a informação, ou, pelo menos, o acesso a fontes de informação como BDs ou outros servidores. O segundo, o cliente, recorre ao servidor, para obter esses recursos, sendo ignorante ao resto do sistema [82], [83]. Esta comunicação é realizada através de uma rede de área local ou uma rede de longa distância, como a internet.

Na sua forma mais simples, este modelo é constituído por apenas duas camadas, o(s) cliente(s) e a fonte de dados. Por um lado, esta abordagem mais direta possibilita um acesso mais rápido à informação, sendo o próprio cliente responsável pelo seu processamento. Por outro lado, esta arquitetura é inerentemente frágil. Tanto a existência de mais que uma fonte de dados como alterações na sua estrutura são fatores que facilmente impactariam o sistema, uma vez que a lógica de acesso ou tratamento dos dados seria obsoleta [83]. Por estes motivos, a utilização de apenas duas camadas não é recomendada quando existem prospeções de escalabilidade.

Por outro lado, existe o modelo de três (ou mais) camadas, representado na Figura 10, que integra um intermediário – o servidor – não só abstraindo o processo de obtenção de dados, mas também aumentando o nível de segurança do sistema. Assim, alterações no modelo ou número de fontes de dados são irrelevantes para os clientes, uma vez que o servidor é o único componente diretamente afetado por tais modificações. Neste modelo, o cliente é somente responsável pela apresentação dos dados, sendo a restante lógica executada pelo servidor [83].

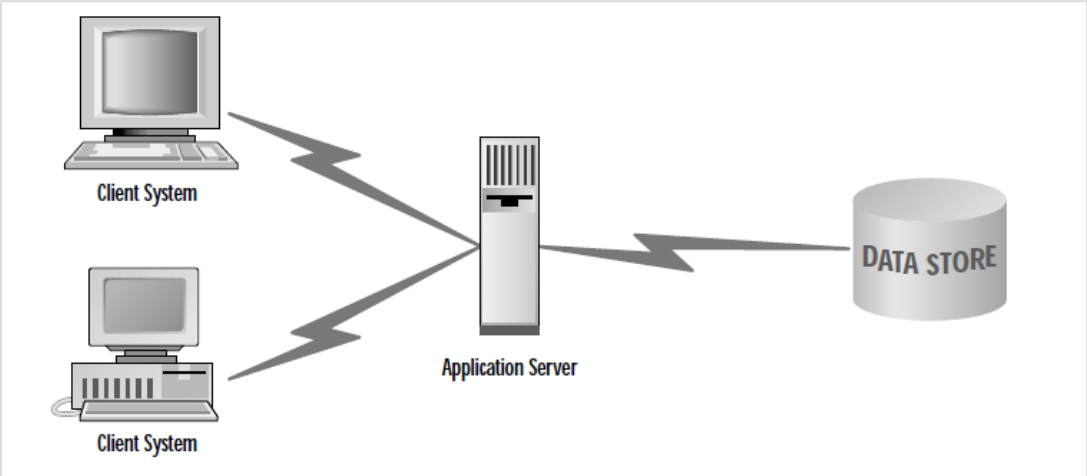



*Figura 10 – Modelo Client-Server de 3 camadas (retirado de* [83]*)*

### 2.4.2 REST API

No ano 2000, Roy Fielding apresentou o conceito de transferência representacional de estado (do inglês, *REpresentational State Transfer*, REST) [84]. Fielding define-o como um estilo arquitetural para sistemas web definido por 6 princípios:

***Client-Server***

Separação das responsabilidades da *User Interface* (UI) e do armazenamento de dados, aumentando a portabilidade da UI e simplificando os componentes do servidor.

***Stateless***

Um pedido deve, obrigatoriamente, conter toda a informação necessária ao seu processamento, não estando dependente de nenhum contexto do servidor.

***Cacheable***

Todas as respostas devem informar se a informação pode ser armazenada na *cache*. Por outras palavras, deve informar se a informação contida na resposta é válida para um outro pedido igual, e por quanto tempo

**Interface uniforme**

A interface deve respeitar 4 restrições

1. Um pedido deve conter a identificação dos recursos a que diz respeito
2. Se um cliente possui a representação de um recurso, pode-o alterar ou apagar
3. As mensagens devem-se auto descrever, por exemplo explicitando o formato do seu conteúdo
4. A partir de um pedido base, as respostas consequentes devem conter informação sobre as possibilidades de pedidos associadas a esse. Ou seja, ao obter um recurso, a resposta deve explicitamente conter os caminhos associados a esse recurso.

**Sistema por camadas**

Como representado na Figura 11, o sistema deve estar dividido por camadas hierárquicas de forma a que cada componente não conheça componentes além dos da camada imediatamente abaixo. Deste modo, o sistema é tornado mais seguro e ainda permite que serviços obsoletos ou pouco utilizados sejam agregados sob uma outra camada com maior nível hierárquico, tornando o sistema menos complexo para o cliente.

**Código sob demanda**

O serviço pode disponibilizar código a ser executado no cliente, permitindo que haja uma extensão das funcionalidades além das previamente instaladas.

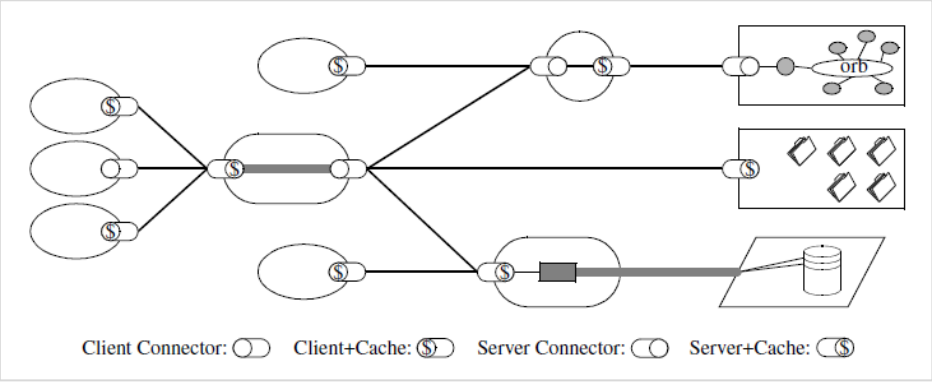



*Figura 11 – Sistema Uniform-Layered-Client-Cache-Stateless-Server (retirado de* [84]*)*

Deste modo, uma *Application Programing Interface* (API) REST é um serviço *web* que respeita as diretrizes descritas.

### 2.4.3 Arquitetura monolítica

Numa arquitetura monolítica, toda a aplicação é construída sobre a mesma base de código como uma única unidade - o monólito [85]. Uma vez que todos os componentes partilham a mesma base de código, esta arquitetura dificulta a integração de bibliotecas ou linguagens de programação adicionais, o que torna a aplicação menos flexível [86].

É de realçar que da sua simplicidade emergem vantagens como a sua facilidade inicial de desenvolvimento, implantação e escalabilidade. Porém, da mesma forma, conforme a complexidade do projeto aumenta, não só estas vantagens se podem transformar em obstáculos, como surgem novos desafios [87]:

**Desenvolvimento e Manutenibilidade**

À medida que a base de código aumenta, as dependências entre diferentes módulos tendem a acumular e o projeto perde a sua modularidade inicial. A equipa deixa de poder trabalhar independentemente em partes diferentes do projeto comprometendo, assim, a sua produtividade.

Além disso, torna-se incrementalmente difícil a integração de novos membros na equipa, que não estão familiarizados com a aplicação.

**Implantação e Escalabilidade**

Devido à natureza da arquitetura, como demonstrado na Figura 12, a implantação inicial é um processo simples. Porém, na eventualidade de um dos serviços da aplicação estar constantemente a sobrecarregar o sistema, a única opção de escalamento horizontal seria adicionar uma nova instância da aplicação completa.

Pelo mesmo motivo, quando existe uma alteração num serviço, toda a aplicação tem de ser recompilada e implantada, o que pode ser um processo moroso.

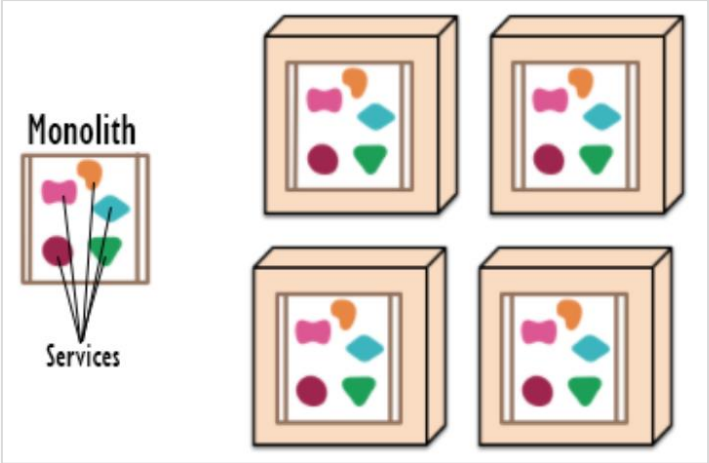



*Figura 12 – Implantação de uma arquitetura monolítica (adaptado de* [85]*)*

### 2.4.4 Arquitetura de micro-serviços

Martin Fowler e James Lewis descrevem a arquitetura de micro-serviços como o desenvolvimento de aplicações constituídas por vários serviços independentes [85]. A popularidade desta arquitetura tem crescido celeremente por solucionar muitos dos problemas recorrentes em arquiteturas monolíticas [88].

Embora o conceito não possua uma definição exata, é caracterizado pela divisão de funcionalidades entre serviços independentes que podem ser geridos por equipas pequenas. Assim, apresentam características como fácil manutenção e testabilidade, baixo nível de acoplamento e independência de implantação [86].

**Desenvolvimento e Manutenibilidade**

Embora exista uma sobrecarga acrescida à implementação inicial do projeto, esta arquitetura permite que, mesmo quando complexidade aumenta, seja conservada a modularidade da aplicação e a equipa possa trabalhar em serviços diferentes em simultâneo.

Além disso, a segregação de funcionalidades por serviços favorece a utilização de tecnologias diferentes para cada, sendo possível a uma seleção mais adequada das tecnologias.

Pelos motivos anteriormente referidos, a integração de novos elementos nas equipas é, também, um processo mais simples: não só a base de código é significativamente menor (por serviço) como existem menos dependências a considerar.

**Implantação e Escalabilidade**

Como demonstrado na Figura 13, a implantação desta arquitetura pode ser feita modularmente. Assim, na eventualidade de um dos serviços da aplicação estar

constantemente a sobrecarregar o sistema, podem ser adicionadas apenas novas instâncias desse serviço. Além disso, o processo de modificação dos serviços também é mais eficiente.

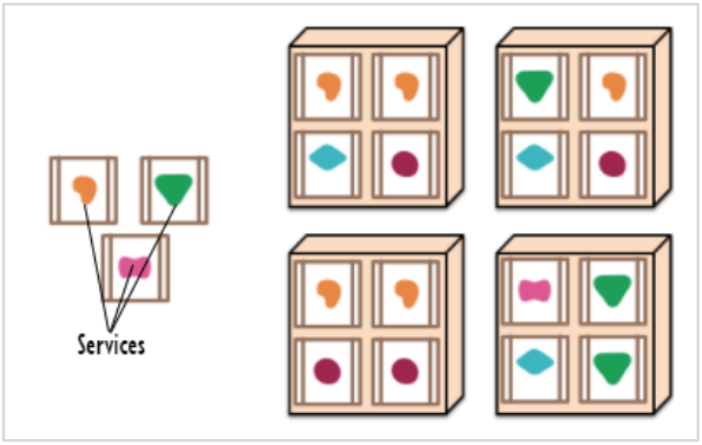



*Figura 13 – Implantação de uma arquitetura de micro-serviços (adaptado de* [85]*)*

## 2.5 Trabalhos relacionados

Com os incentivos europeus para a transição das redes elétricas para SG, têm surgido diversos projetos envolvendo DG e DR que, por norma, requerem a agregação dos participantes. Desta forma, esses projetos possuem uma plataforma para monitorização e gestão da rede, que permite essa agregação.

### *2.5.1 SmartNet*

O *SmartNet* é um projeto de investigação europeu com duração de 3 anos, financiado pelo H2020 e envolvendo 22 parceiros de 9 países europeus. Tem como objetivo solucionar os conflitos gerados pela integração de DER, sobretudo FERs, nas atuais redes de transmissão e distribuição de energia [89], [90].

O projeto está dividido em 4 fases e os 22 parceiros foram distribuídos por 8 grupos de trabalho com objetivos bem definidos para cada fase. Nas fases iniciais do projeto, o foco foi a criação de modelos de coordenação e comunicação TSO-DSO e testes das mesmas em ambientes controlados e simulações.

Por outro lado, na fase atual, a quarta fase, pretende-se validar os resultados recorrendo a demonstrações físicas através 3 projetos piloto que decorrem em Itália, Dinamarca e Espanha. Em cada piloto serão testados diferentes aspetos do sistema. Porém, a agregação de participantes será um ponto crucial em todos, uma vez que é necessária para a participação de produtores e consumidores mais pequenos nos mercados, que é um dos objetivos do projeto [91].

### *2.5.2 GE Energy Aggregator*

A *Energy Aggregator* é uma aplicação de *software* de monitorização e gestão de rede, desenvolvida pela *GE Grid Solutions*, que é um empreendimento da *GE Renewable Energy*. Esta divisão da *General Electric* tem como objetivo solucionar problemas à implementação de energias renováveis e SG [92].

Esta é uma ferramenta de apoio ao agregador e foi desenvolvida como complemento a outra aplicação, a *GE Communicator*, que obtém os dados em tempo real dos aparelhos de medição previamente distribuídos pela rede [93], [94]. Os focos da *Energy Aggregator* são a visualização de dados como padrões de consumo de equipamentos ou grupos de equipamentos e a automação não só de geração de relatórios, mas também da faturação que, devido ao *GE Communicato*r é mais precisa e justa. Segundo o manual, a aplicação permite, também, a agregação de várias entidades [95].

### *2.5.3 NEMOCS*

A *NEMOCS* é uma plataforma de controlo de VPP, desenvolvida pela *NEXT Kraftwerke*. De forma semelhante ao *SmartNet*, tem como objetivo a integração de DERs na rede.

Esta plataforma é uma solução *all in one*, disponibilizando o controlo remoto de pontos individuais da rede; agregação, monitorização e visualização de dados das DERs; otimizações dependendo de picos de demanda e condições climatéricas e, por fim, DR [96].

### *2.5.4 cyberNOC*

A *cyberNOC* é uma plataforma de agregação de flexibilidade desenvolvida pela *cyberGRID* e é um dos contribuintes para o projeto europeu *FutureFlow*, financiado pelo H2020.

Esta é uma plataforma *web* que permite a monitorização das DER da rede, disponibiliza serviços de estabilização da frequência da rede e programas de DR, contribuindo para uma rede mais estável [97].

### 2.5.5 Comparação dos trabalhos relacionados

Devido à especificidade do projeto desenvolvido, não foi encontrado nenhum projeto que oferecesse a totalidade das funcionalidades desenvolvidas. Nomeadamente, nenhum dos projetos analisados refere a possibilidade de importação de dados a partir de ficheiros ou a possibilidade de escolha dos parâmetros para a agregação. No entanto, é espectável que todos eles, em fases de desenvolvimento, tenham tido funcionalidades semelhantes, de modo a testar e avaliar as suas capacidades de agregação. De modo semelhante, nenhum dos

trabalhos menciona funcionalidades de avaliação das agregações na própria plataforma. As diferentes características mencionadas nos projetos estão listadas na Tabela 4.

*Tabela 4 – Comparação dos projetos*

| **Características mencionadas** | ***SmartNet*** | ***GE Aggregator*** | ***NEMOCS*** | ***cyberNOC*** | **Plataforma para o Agregador** |
|---|---|---|---|---|---|
| Agregação | Sim | Sim | Sim | Não | Sim |
| Agregação parametrizada | Não | Não | Não | Não | Sim |
| Avaliação de Agregações | Não | Não | Não | Não | Sim |
| Gestão de cenários | Não | Não | Não | Não | Sim |
| Gestão da Rede | Sim | Sim | Sim | Sim | Não |
| Monitorização da rede | Sim | Sim | Sim | Sim | Sim (através da de *datasets* da API) |
| Plataforma *web* | Não | Não | Não | Sim | Sim |

Através da tabela, é possível concluir que o foco dos trabalhos analisados é sobretudo na monotorização e gestão da rede, onde a agregação é um componente estático previamente estudado. Por outro lado, a plataforma desenvolvida é baseada em *datasets*, permitindo a monotorização da rede através da importação manual de um *dataset* a partir dos dados mais recentes disponibilizados pela API do projeto integrador. Além disso, apenas o projeto *cyberNOC* é uma plataforma web, sendo as restantes aplicações de *software* nativas.

# 3 Análise do Problema

Neste capítulo é apresentado o processo de resolução do problema através da compreensão do domínio do problema e seus requisitos, funcionais e não-funcionais, definidos ao longo do projeto.

## 3.1 Domínio do problema

Nesta secção é decomposto e expandido o problema descrito na Secção 1.3, seccionando o problema por categorias e evidenciando todos os critérios a considerar na construção do modelo de domínio. De seguida é apresentado o modelo de domínio derivado do problema descrito, acompanhado de uma apresentação dos seus conceitos.

### 3.1.1 Problema

A plataforma para o agregador é uma aplicação *web* que permite a importação de cenários para agregação. A importação pode ser realizada tanto através de ficheiros como a partir da API disponibilizada pelo projeto DOMINOES. Um cenário, ou *dataset,* é um conjunto de medições de vários parâmetros como o consumo e a geração de energia dos participantes da rede. Na plataforma, esses dados podem ser visualizados, editados e utilizados para a agregação, que irá separar os participantes em vários grupos.

O Agregador deve ter acesso, em qualquer momento, aos *datasets* e agregações já realizadas. Além disso, é de realçar o facto dos processos de importação e agregação de dados serem morosos, pelo que a solução necessita de mecanismos para superar ou amenizar essas situações.

Deste modo, o problema pode ser decomposto em sete partes: tratamento dos parâmetros; importação de ficheiros; importação dos dados da API; visualização e alteração de *datasets*; gestão de *datasets*; agregação de participantes e visualização dos seus resultados; e sistema de tarefas.

**Tratamento dos parâmetros**

Primeiramente, é de realçar que um participante possui parâmetros de diversos tipos, exemplificando, o consumo é composto por várias medições ao longo de um período, sendo, portanto, um atributo quantitativo contínuo, enquanto que o tipo de consumo é um atributo categórico. A aplicação deverá distinguir esses dados de forma a melhor os representar.

**Importação de ficheiros**

Um *dataset* pode ser criado ao carregar um ficheiro *excel* com uma estrutura previamente definida para a página *web*, arrastando o ficheiro para uma determinada secção da interface gráfica.

**Importação de dados da API DOMINOES**

Numa versão final, a plataforma deverá ainda permitir a obtenção dos dados reais dos participantes da rede, recorrendo à API do projeto integrador. Porém, para períodos longos esta aquisição pode demorar dezenas de minutos. Assim, a ação de busca de dados não pode ser uma operação bloqueante no funcionamento da aplicação e deve ser persistente fora do contexto da sessão, evitando a sua repetição no caso de, por exemplo, a página ser atualizada.

**Visualização e Alteração de *datasets***

Após o carregamento de um *dataset*, a plataforma disponibilizará ferramentas que possibilitam obter rapidamente um parecer da sua constituição. A análise pode ser feita ao conjunto ou a nível individual, isto é, apenas a um participante. A revisão isolada potencia a deteção e correção de erros. Essa correção poderá ser realizada utilizando a própria aplicação, agilizando assim o processo de configuração do *dataset*.

**Gestão de *datasets***

Para que o agregador possa efetivamente comparar diferentes cenários de teste, é crucial que esses cenários sejam persistentes na aplicação. Assim, o Agregador pode optar por guardar os *datasets* importados no servidor para uma consulta posterior. Uma vez que podem existir pedidos de agregação associados a um *dataset*, no caso de alteração, o *dataset* é considerado como não persistido, tornando-se efetivamente outro *dataset* no contexto da plataforma. Para além disso, o Agregador pode, a qualquer momento, baixar um *dataset* para a sua máquina ou apagar o registo de qualquer *dataset* guardado no servidor, após a confirmação de um código. Por fim, quando um *dataset* é removido, todos os pedidos de agregação associados deverão ser apagados.

**Agregação de participantes**

Um dos objetivos da plataforma é a agregação dos participantes, esta é realizada através de algoritmos de agregação em R que deverão ser adaptados ao problema. Com o intuito de

averiguar que dados são relevantes para a agregação, o Agregador poderá escolher não só o intervalo de número de grupos, mas também quais os parâmetros dos participantes a considerar na agregação e qual o seu nível de importância, gerando um pedido de agregação. Por exemplo, um pedido pode ser realizado para 5 a 10 grupos e considerando apenas o tipo de consumo. O Agregador pode ainda optar por fazer todas as combinações entre os parâmetros de um pedido de agregação, executando a agregação para cada uma.

Deste modo, um pedido de agregação resulta em uma ou mais tarefas de agregação, dependendo das combinações. No caso de sucesso de uma tarefa de agregação, é registado o resultado de agregação; alternativamente, no caso de falha, a tarefa gera um erro. Um resultado de agregação contém uma agregação por cada número de grupos a considerar. Ou seja, para o intervalo de 5 a 10 grupos (inclusive), seriam geradas 6 agregações. Deste modo, uma agregação é composta por *clusters* (grupos) de participantes.

Para complementar o processo de agregação, a aplicação disponibiliza a avaliação das agregações feitas. Assim, explicita qual o número de grupos ideal, segundo as metodologias de avaliação *Elbow* e *Silhouette*, também implementados em R.

**Sistema de tarefas**

Para dar resposta aos longos tempos de processamento dos algoritmos de agregação e obtenção de dados a partir da API, foi considerada pertinente a adição de um sistema de tarefas. Este permitirá a criação e monitorização de tarefas a serem executadas em segundo plano. Desta forma, quando o Agregador inicializa uma das ações descritas no início do parágrafo, deverá ser registado o pedido de execução dessa ação e retornado o seu identificador para acompanhamento, não interrompendo o fluxo da aplicação.

### 3.1.2 Modelo de domínio

Considerado o problema detalhado na secção 3.1.1, alcançou-se a representação do domínio descrito na Figura 14, através de um diagrama de classes em *Unified Modeling Language* (UML). Nesta secção, a letra ***k*** refere-se ao número de partes em que se pretende dividir o *dataset*, isto é, a quantidade de grupos a gerar na agregação.

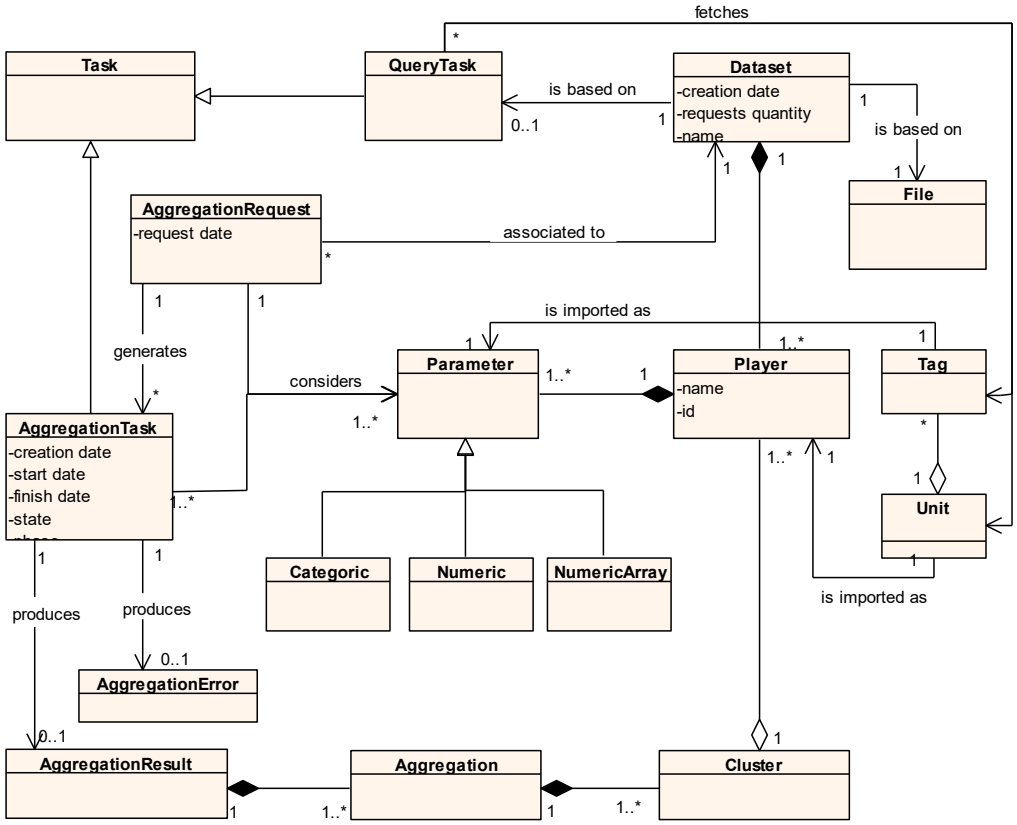



*Figura 14 – Modelo de Domínio*

Um ***dataset*** é um conjunto de dados referentes a um ou mais ***players***. Cada ***player*** possui um identificador, um nome e diversos **parâmetros** de vários tipos. Cada tipo de parâmetro (categórico, numérico e vetor de números) deverá ser interpretado de maneira diferente pela plataforma. Um ***dataset*** pode não só ser importado através de um **ficheiro**, mas também carregado a partir da API do projeto integrador. O processo de recolha de dados da API gera uma ***query task*** que obtém **unidades** e ***tags.*** No carregamento de um ***dataset*** a partir de uma ***query task***, as **unidades** correspondem a ***players*** e as ***tags*** a **parâmetros**.

Uma vez que o Agregador selecione um ***dataset***, poderá simular a agregação dos participantes do mesmo, gerando um **pedido de agregação**. Este contém informação sobre a data da sua criação, qual o ***dataset*** a agregar, o intervalo de ***k*** a testar, os **parâmetros** a considerar e se é pretendido explorar todas as combinações possíveis entre os **parâmetros** selecionados.

Assim, cada combinação resulta numa **tarefa de agregação**, que regista não só o intervalo de ***k*** definido e combinação de **parâmetros**, mas também dados referentes à sua execução.

Quando a execução falha, é registada a causa da mesma – o **erro**. Caso contrário, após o sucesso de **uma tarefa de agregação**, são guardados os seus **resultados**. Um **resultado de agregação** é composto uma ou mais **agregações** (é gerada uma para cada ***k*** no intervalo) e cada **agregação** é composta por um ou mais (***k***) **grupos** de participantes.

## 3.2 Requisitos

Esta secção descreve os requisitos funcionais e não funcionais do sistema, que foram identificados ao início e durante projeto. A classificação dos requisitos foi realizada recorrendo à extensão do modelo *Functionality, Usability, Reliability, Performance, Supportability* (FURPS) descrito em [98] – o FURPS**+**, sendo que o **+** inclui requisitos como limitações de desenho, implementação, interface e físicos.

Nesta fase do projeto, apenas existe um ator no sistema, o Agregador, que pretende utilizar a plataforma como auxiliar à gestão da rede elétrica pela qual é responsável.

### 3.2.1 Requisitos Funcionais

Os requisitos funcionais (***F**unctionality*) encontram-se divididos em cinco partes, nomeadamente, leitura e criação de *datasets*, gestão de *datasets*, visualização de um *dataset*, gestão de pedidos de agregação e monitorização de tarefas de agregação.

#### Criação de *datasets*

Para a criação de *datasets*, o sistema fornece duas opções: ler e importar dados de um ficheiro *excel*, ou importar os dados reais diretamente a partir da API do DOMINOES. As funcionalidades diretamente relacionadas com a criação de *datasets* estão discriminadas na Figura 15.

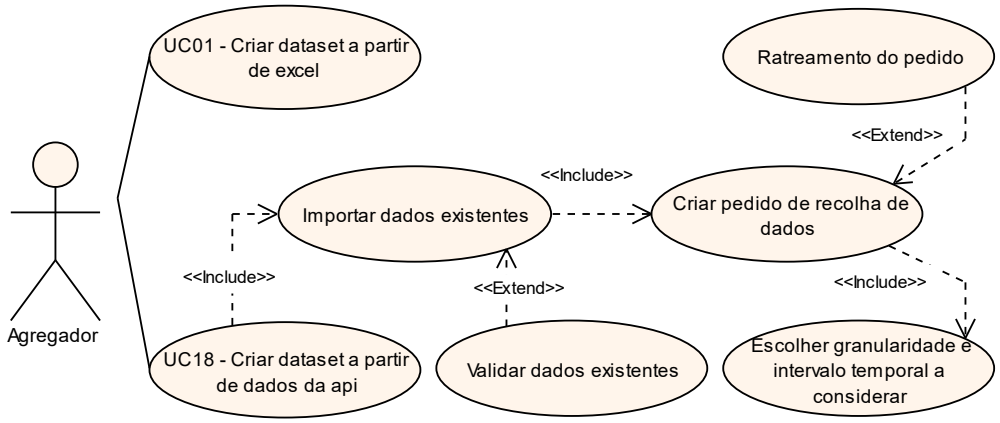



*Figura 15 – Diagrama de casos de uso referentes à criação de datasets*

**UC01 – Importar um *dataset* a partir de um ficheiro excel**

O Agregador arrasta para uma caixa de *drag'n'drop* da aplicação web um ficheiro *excel* representativo de um *dataset*, com uma estrutura previamente definida, com dados os dados dos participantes a avaliar, que é lido e carregado para o navegador.

**UC18 – Importar um *dataset* a partir de uma API externa**

O Agregador seleciona a granularidade – anual, mensal, diária, horária ou instantânea (tetra-horária) – desejada e, de seguida, a data de início e a data de fim a considerar para a recolha dos dados da API. Esta ação gera um pedido que poderá, posteriormente, ser rastreado pelo Agregador. Adicionalmente, o Agregador pode optar por validar os dados obtidos para o intervalo selecionado antes da sua importação, salientando as medições incompletas ou em falta. Por fim, o Agregador importa os dados, carregando-os para o navegador.

### *Gestão de datasets*

Para facilitar a utilização partilhada de *datasets*, a plataforma disponibiliza um sistema de gestão dos mesmos, representado na Figura 16 . Este permite guardar e carregar *datasets*, fazer o seu download para *excel* e, por fim, apagar o seu registo da plataforma.

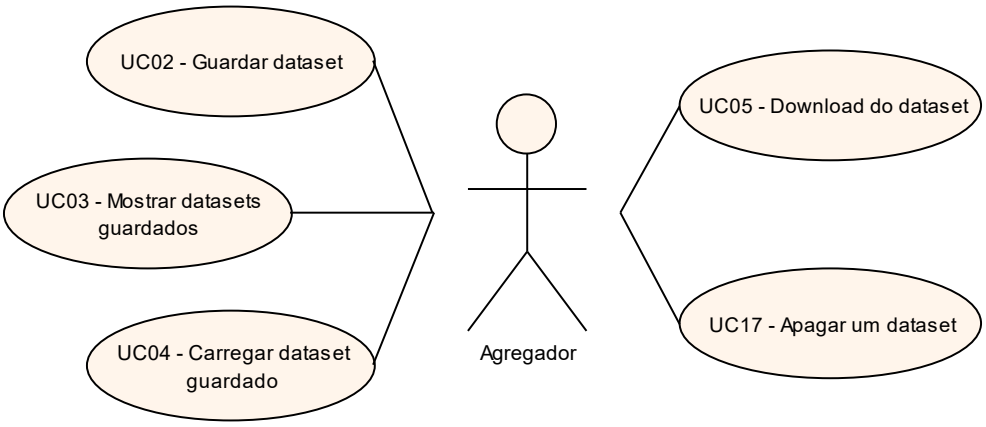



*Figura 16 – Diagrama de casos de uso referentes à gestão de datasets*

**UC02 – Guardar o *dataset***

O Agregador pode optar por guardar o *dataset* no servidor, para que possa ser consultado por outros utilizadores e carregado mais rapidamente. Para a execução de uma agregação é necessário que o *dataset* esteja guardado no servidor.

**UC03 – Listar *datasets* guardados**

Para que o Agregador possa selecionar um dos datasets guardados no servidor com precisão, são listados todos os *datasets* guardados no servidor, ordenados do mais recente para o mais antigo, e os seus detalhes, nomeadamente, data de criação, nome, número de pedidos associados e quantidade de cada tipo de participante (consumidores, produtores e prosumidores).

**UC04 – Carregar *dataset* guardado**

Caso o Agregador saiba que o *dataset* que pretende consultar já se encontra persistido no sistema, este pode optar por carregá-lo diretamente. Este processo é mais rápido que o descrito no UC01 e permite que o Agregador possa associar diversas agregações ao mesmo *dataset* ao longo de várias sessões de utilização.

**UC05 – Download do *dataset***

O Agregador pode, a qualquer momento, exportar o *dataset* que está a utilizar, o ficheiro gerado segue a mesma estrutura que o utilizado no UC01, podendo ser alterado externamente e recarregado para o sistema, se necessário.

**UC17– Eliminar *dataset***

O Agregador, após uma confirmação e validação de acesso, elimina um *dataset* e todos os seus pedidos de agregação associados.

### *Visualização e edição de dataset*

A Figura 17 representa as funcionalidades de visualização e edição de *datasets* disponibilizados pela plataforma. Estas ferramentas visam facilitar o processo de deteção de falhas no *dataset* e correção das mesmas.

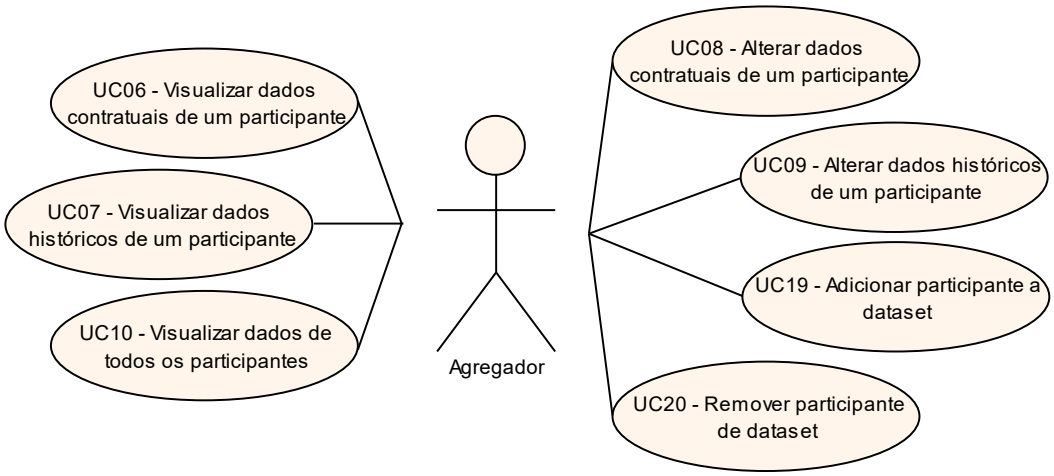



*Figura 17 – Diagrama de casos de uso referentes à visualização e alteração de um dataset*

**UC06 e UC07 – Visualizar dados de um participante**

O Agregador visualiza os dados de um participante. Para enriquecer a análise dos dados, a plataforma distingue dados contratuais (tipo de consumo/geração, tarifas e flexibilidade) de dados históricos (consumo, redução, custo do consumo, geração, lucro de geração). Os dados históricos podem ser representados através de gráficos ou tabelas.

**UC08 e UC09 – Alterar dados de um participante**

O Agregador pode alterar tanto os dados contratuais como históricos de um participante diretamente na plataforma, por exemplo, no caso de encontrar um erro, sem necessidade de carregar um novo *dataset* corrigido. As alterações são imediatamente refletidas nos casos de uso UC06 e UC07.

**UC10 – Visualizar dados de todos os participantes**

O Agregador visualiza os dados totais do *dataset*. Pode, ainda, optar por visualizar esses dados graficamente, divididos por tipo de consumo e tipo de geração.

**UC19 – Adicionar participante a *dataset***

O Agregador pode adiciona um participante ao *dataset* através de um ficheiro excel com a mesma estrutura que o do UC01.

**UC20 – Remover participante de *dataset***

O Agregador remove o registo de um participante de um *dataset*. Esta operação não pode ser revertida, pelo que deve ser confirmada a intenção do ator.

### *Gestão e consulta de pedidos de agregação*

Sendo uma plataforma para o Agregador, esta disponibiliza funcionalidades de agregação dinâmicas – os pedidos de agregação. Tanto esses pedidos como os seus eventuais resultados são registados para consultas posteriores, tal como representado no diagrama da Figura 18.

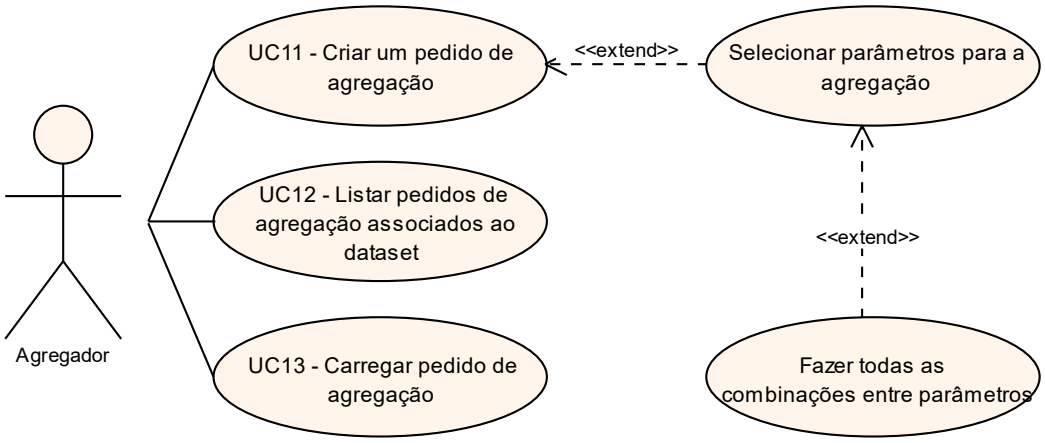



*Figura 18 – Diagrama de casos de uso referentes aos pedidos de agregação*

**UC11 – Criar um pedido de agregação**

O Agregador seleciona os parâmetros que quer considerar para a agregação, pode ainda optar por testar todas as combinações entre os parâmetros que selecionou, cada uma gerando uma tarefa de agregação.

**UC12 – Listar pedidos de agregação associados ao *dataset***

O Agregador lista todos os pedidos de agregação associados ao *dataset* selecionado.

**UC13 – Carregar pedido de agregação**

O Agregador pode carregar qualquer pedido de agregação que tenha sido feito para o *dataset* selecionado, podendo consultar os resultados obtidos.

### *Monitorização e resultados de tarefas de agregação*

Uma vez que um pedido pode gerar várias tarefas de agregação e que o processo de agregação tende a ser demorado é essencial que o Agregador consiga saber sempre que agregações estão a ser executadas e qual o seu estado no processo. Além disso, permite a visualização e comparação de resultados das agregações já concluídas, antes do término do pedido de agregação. Assim, surgem os casos de uso representados na Figura 19.

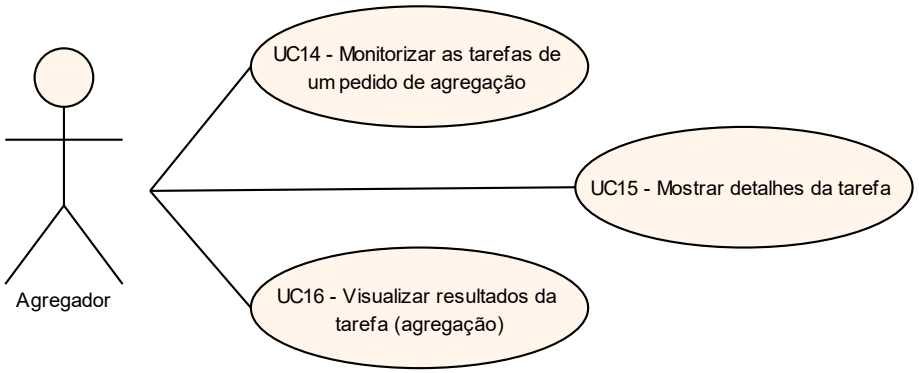



*Figura 19 – Diagrama de casos de uso referentes às tarefas*

**UC14 – Monitorizar as tarefas de um pedido de agregação**

O utilizado, após selecionar o pedido de agregação a consultar, pode monitorizar o estado e a fase das agregações (tarefas) associadas ao pedido. As tarefas distinguem-se pelos parâmetros considerados para essa agregação.

**UC15 – Mostrar detalhes da tarefa**

O Agregador consulta os detalhes da tarefa, nomeadamente, os instantes de criação da tarefa, começo e finalização da sua execução (se aplicáveis) e informações sobre cada centro da agregação, caso esta tenha terminado.

**UC16 – Visualizar resultados da tarefa**

O Agregador poder optar por uma análise mais completa da agregação, recorrendo a gráficos. Permite a avaliação dos grupos - qual o melhor número de grupos para o *dataset*, visualizar informação sobre os centros dos grupos formados e, ainda, a constituição dos grupos.

### 3.2.2 Requisitos Não Funcionais

Os requisitos não funcionais, correspondem a todos os requisitos do modelo FURPS+ à exceção da funcionalidade, ou seja, URPS+. Referem-se às imposições e limitações para a

concretização do projeto, incluindo, mas não se limitando a restrições estéticas, tecnológicas e físicas. Estes requisitos servem também como avaliação da solução, pelo que devem ser considerados em todas as fases do desenvolvimento do sistema [99].

**Usabilidade**

A usabilidade (***U**sability*) refere-se à interface gráfica e à experiência do utilizador durante essa interação. Define, por exemplo, normas para a aparência da interface gráfica, prevenção de erros e tempos de resposta [100] .

Uma vez que a plataforma está dependente de constantes interações entre a página web e o servidor, é importante garantir que o utilizador sabe sempre o estado atual da página, sendo claro se ocorreu um erro, ou a aplicação está apenas a carregar dados. Assim, foram definidos os seguintes requisitos:

- A interface gráfica deve ser simples, intuitiva para qualquer utilizador conhecedor do domínio*;*
- O tema (cores, tipos e tamanho de letra, estilo) da interface deve ser consistente em toda a interface*;*
- O tempo de resposta deve ser curto, sendo qualquer operação com tempo de resposta superior a 1 segundo acompanhada de um indicador de carregamento*;*
- Erros na plataforma (e.g. pedidos para o *backend* sem resposta), não devem interromper o funcionamento da aplicação.

**Confiabilidade**

A confiabilidade (***R**eliability*) avalia a integridade, conformidade e interoperabilidade do sistema. Alguns dos aspetos considerados são a disponibilidade, tolerância a falhas, expectativa de recuperação após uma falha e exatidão das computações realizadas [99].

Deste modo, foi realçado o requisito:

- No caso de erros ou falhas, a plataforma deve continuar funcional.

**Desempenho**

O desempenho (***P**erformance*) envolve conceitos relacionados com a eficiência e rapidez do sistema como tempo de resposta, utilização de memória e/ou processador e tempo de inicialização do sistema [100].

Assim, é destacado o requisito:

- Se possível, guardar os dados localmente para evitar pedidos morosos a APIs externas

**Suportabilidade**

A suportabilidade (***S**upportability*) refere-se a características como a escalabilidade, testabilidade, manutenibilidade, compatibilidade e configurabilidade do sistema, isto é, requisitos facilitadores da manutenção e extensão do projeto desenvolvido [99].

Deste modo, foram definidos os seguintes requisitos:

- A aplicação deverá possibilitar a integração de novos algoritmos de agregação*;*
- O código deverá obedecer às normas e refletir boas práticas da programação, como a utilização de padrões de software e obedecer aos princípios de *Single-responsibility, Open-closed, Liskov substitution, Interface segregation* e *Dependency inversion* (SOLID) e *General Responsibility Assignment Software Principles* (GRASP), de modo a simplificar a sua manutenibilidade*;*
- A aplicação (*backend*) deverá suportar tanto plataformas baseadas em *Unix* como *Windows*.

**Limitações de desenho**

Uma limitação ou requisito de desenho (**+**, *design constraints*) é um requisitos que impacta diretamente o desenho do sistema [99] .

Assim, foram distinguidos os requisitos abaixo:

- Os algoritmos de agregação deverão ser implementados em R*;*
- A aplicação deverá seguir o modelo *Client-Server.*

**Limitações de implementação**

Limitações ou requisitos de implementação (**+**, *implementation constraints*) referem-se a imposições à codificação e construção do sistema, como por exemplo ambientes operacionais, linguagens e normas de programação [99].

Desta forma, foram levantados os seguintes requisitos:

- O servidor deverá ser desenvolvido em *Python;*
- Deverá ser uma aplicação web;
- A aplicação deverá seguir o modelo *Client-Server.*

**Requisitos de interface**

Limitações ou requisitos de interface (**+**, *interface constraints*) são referentes a interações com sistemas externos [99], [100].

São realçados os seguintes requisitos:

- A aplicação deverá disponibilizar os seus serviços REST para a agregação;
- A aplicação devera interagir com os serviços a API REST do DOMINOES para obter; dados de participantes reais.

**Limitações físicas**

Limitações ou requisitos físicos (**+**, *physical constraints*) definem características do hardware a ser utilizado, como peso e forma e a estrutura de rede necessária ao funcionamento do sistema [101].

Assim, é considerado o requisito seguinte:

- A implantação deverá ser feita em *dockers* conectados à rede privada do GECAD.

# 4 Desenho da Solução

O modelo C4 tem como principal objetivo uniformizar os níveis de abstração utilizados ao representar a arquitetura de um software, dividindo-os em 4 níveis e sem se restringir a uma linguagem de representação de diagramas [102]:

- **Contexto do sistema** – neste nível são representados os diferentes sistemas, permitindo visualizar a interação do sistema desenvolvido com sistemas externos;
- **Contentores** – dentro de um sistema, existem vários contentores, sendo um contentor definido como uma unidade de *software* separada, como uma BD ou aplicação *web;*
- **Componentes** – um contentor é definido por vários componentes, cada um representando partes importantes da aplicação como controladores e serviços*;*
- **Código** – este nível detalha a implementação ao nível de código discriminando as classe e métodos utilizados.

Deste modo, este capítulo será dividido por níveis de abstração, começando pelo mais generalizado. O nível de código apenas será abordado no capítulo da Implementação da Solução.

O modelo C4 descrito será utilizado em combinação com o modelo de vistas 4+1, que propõe a utilização de 5 vistas – lógica, processo, desenvolvimento, física e cenários – para descrever o sistema, como apresentado na Figura 20, das quais se destacam [103]:

- **Vista Lógica** – descreve os elementos de software definindo as suas dependências e fronteiras. A representação desta vista é comumente feita através de diagramas de classes ou componentes;
- **Vista de processo** – descreve o fluxo de uma funcionalidade, captando a sua concorrência e sincronização. Pode ser representada através de diagramas de sequência;
- **Vista física** – descreve o sistema a um nível físico refletindo a sua implantação e protocolos de comunicação. Por norma esta vista corresponde a diagramas de implantação.

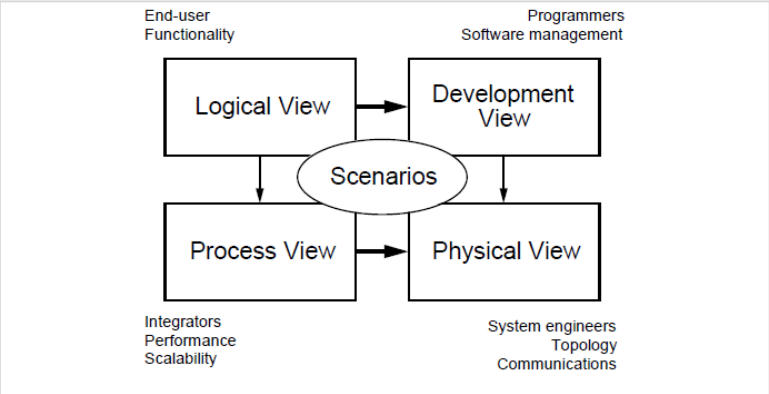



*Figura 20 – Modelo de vistas 4+1 (retirado de* [103]*)*

## 4.1 Nível de Sistema

Como referido anteriormente, este nível pretende enquadrar o sistema desenvolvido, explicitando as pessoas (atores) e sistemas externos com o qual interage, como representado no diagrama da Figura 21.

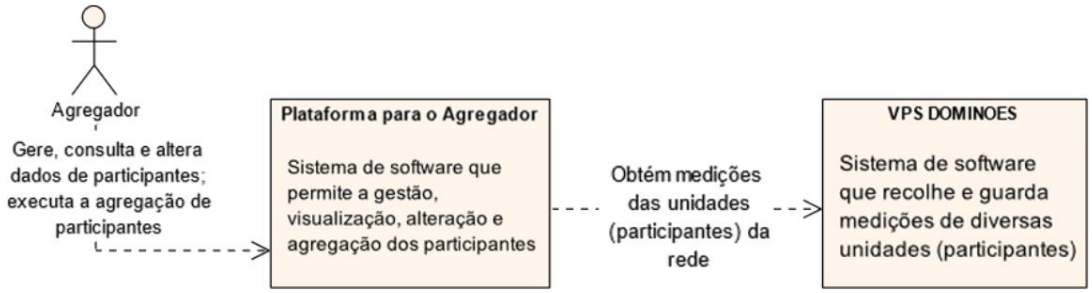



*Figura 21 – Diagrama de sistema – Vista lógica*

A **VPS DOMINOES** é a API REST desenvolvida pela *Virtual Power Solutions* (VPS) para o projeto integrador DOMINOES. Esta API é referida no documento como API DOMINOES e é responsável pela disponibilização das medições como o consumo e a geração das diferentes unidades da rede. A **Plataforma para o Agregador** interage com a **VPS DOMINOES** para a obtenção das medições das unidades. O único ator do sistema, o **Agregador**, interage com a **Plataforma para o Agregador** para usufruir das suas funcionalidades de gestão e monotorização da rede, alteração de dados relativos a participantes e a agregação de participantes.

## 4.2 Nível de Contentores

Ao longo deste projeto, uma das prioridades foi o desenvolvimento célere das funcionalidades propostas, para que pudessem ser avaliadas e, posteriormente, melhoradas conforme as recomendações. Por esse motivo, é de realçar que, o tempo de familiarização com as tecnologias e algoritmos impostos também foi um fator fulcral na escolha da arquitetura.

Não está prevista a definição de novos requisitos além dos já levantados podendo, no entanto, existir a extensão de requisitos já existentes. De forma semelhante, dado que a aplicação é uma plataforma para o agregador de energia, é improvável o aumento exponencial de utilizadores no mesmo servidor, não sendo priorizada a escalabilidade da aplicação.

Desta forma, no desenho da arquitetura foram considerados, principalmente, os fatores complexidade inicial de implementação e facilidade de desenvolvimento rápido de versões estáveis.

### 4.2.1 Alternativa: Arquitetura Monolítica

Pelos motivos referidos, uma das alternativas mais apelativas é a arquitetura monolítica. Como descrito no subcapítulo 2.4.3, esta arquitetura tem um tempo inicial de configuração relativamente baixo e a reutilização da mesma base de código para diferentes componentes permite um desenvolvimento rápido de novas funcionalidades. Deste modo, uma possível solução seria a representada na Figura 22.

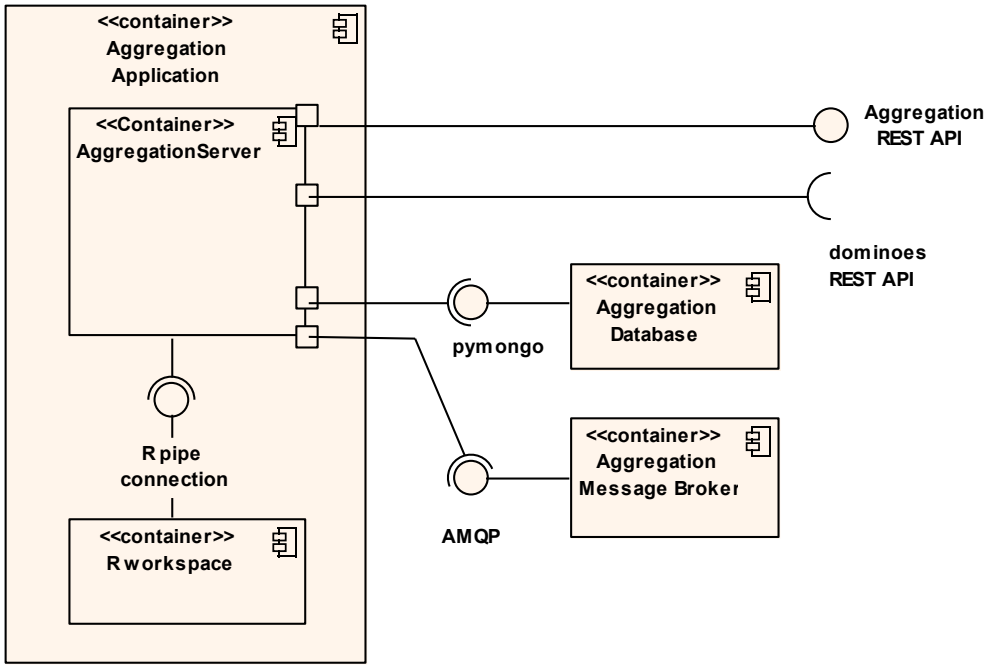



*Figura 22 – Diagrama de contentores da alternativa monolítica – Vista Lógica*

No entanto, uma vez que uma das restrições de implementação é o servidor ser desenvolvido em *Python*, numa arquitetura puramente monolítica, isso obrigaria a utilização de alternativas de *fullstack* em *Python,* como por exemplo, *Django.* Embora esta *framework* permita a utilização das bibliotecas e *frameworks* de *frontend* mais populares como o *Angular*, *React* e *Vue*, esse é um processo complexo quando comparado ao da *framework NodeJS*, principalmente na instalação de bibliotecas que extendem tecnologias de *frontend* referidas. De forma semelhante, devido à diferença de popularidade entre o *Django* e *NodeJS* [104], existe uma maior escassez de questões respondidas, tutoriais e *templates* para a integração de tecnologias de *frontend* na primeira opção. Deste modo, a liberdade de escolha de tecnologias não é um fator negligenciável aquando a escolha da arquitetura.

Concluiu-se que a deveria existir, pelo menos, uma separação entre o servidor que fornece a página *web* e o servidor que fornece os restantes recursos. Assim, não só existe uma melhor separação de responsabilidades e menos acoplamento, como é, também, possibilitada a utilização de tecnologias de *frontend* que potenciem a rapidez de desenvolvimento.

### 4.2.2 Alternativa: Arquitetura de Micro-serviços

A outra alternativa analisada foi a arquitetura de micro-serviços. Em contraste com a arquitetura monolítica, neste exemplo, as diversas funcionalidades da plataforma são divididas em vários serviços *web* como representado na Figura 23.

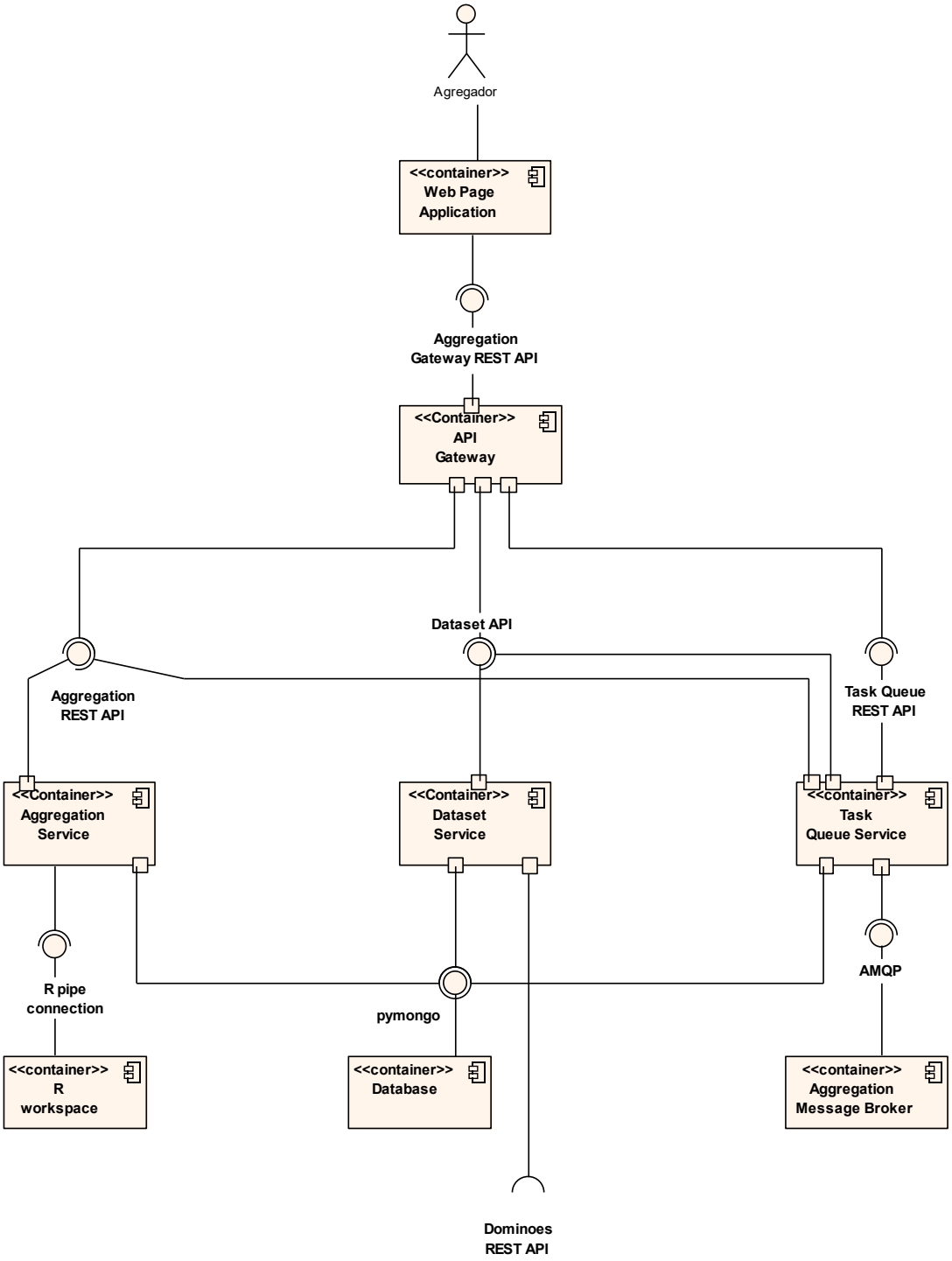



*Figura 23 – Diagrama de contentores da alternativa de micro-serviços – Vista Lógica*

Assim, destacam-se os contentores:

- ***Web Page Application***: Aplicação *frontend* em *NodeJS* e *React*- é o único contentor com o qual o utilizador interage diretamente e, por sua vez, interage com a ***API Gateway***, para aceder aos diversos serviços.

- ***API Gateway***: Implementação do padrão homónimo [105], permite que os diversos serviços sejam acedidos através de um intermediário, uniformizando o seu acesso.
- ***Dataset Service***: Serviço de gestão de *datasets*, permite a persistência de datasets carregados no *frontend*, ou a obtenção de *datasets* a partir da API REST do DOMINOES.
- ***Task Queue Service***: O serviço de TQ permite a adição de tarefas previamente definidas a uma lista para serem executadas, ordenadamente, em segundo plano. É responsável pela criação, execução e rastreamento das tarefas e devolução dos resultados.
- ***Aggregation Service:*** Permite a execução total ou parcial de algoritmos de agregação, recorrendo a ambientes de trabalho em R para a execução dos mesmos. Oferece um sistema de sessões, para que os dados não sejam perdidos entre pedidos parciais.

De um ponto de vista de acoplamento, manutenibilidade e escalabilidade, esta alternativa seria a ideal. No entanto, como descrito na secção 2.4.4, o desenvolvimento desta arquitetura vem com um custo acrescido na fase inicial de configuração, que já será demorada devido a necessidade de habituação às tecnologias utilizadas. Ademais, considerando que a escalabilidade não é um fator decisivo na escolha da arquitetura, a utilização de uma arquitetura mais simples é vantajosa.

### 4.2.3 Solução proposta

Considerando as alternativas analisadas, a solução escolhida foi uma abordagem intermédia entre esses dois extremos. Existe uma separação entre o *frontend* e o *backend* da aplicação, no entanto, os serviços de agregação e TQ estão agrupados no contentor ***Task Queue Service*** e apenas podem ser acedidos através do servidor principal, como representado na Figura *24*.

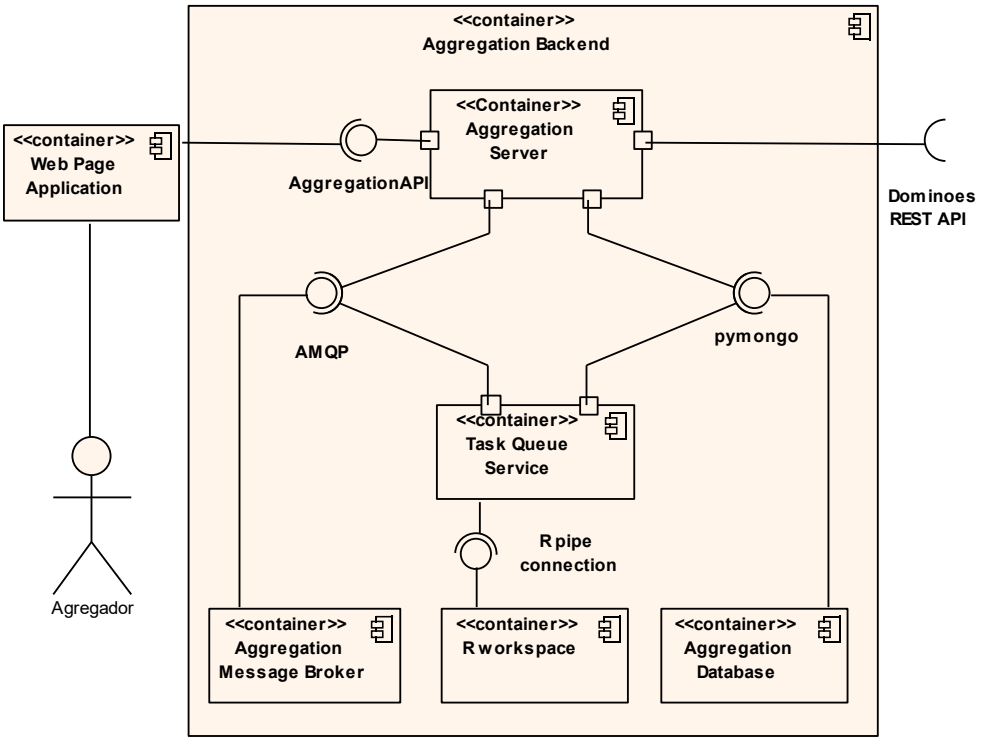



*Figura* 24 – *Contentores da plataforma – Vista Lógica*

Assim, a solução final é composta por 6 contentores, nomeadamente:

- ***Web Page Application***: Aplicação *frontend* - é o único contentor com o qual o utilizador interage diretamente e, por sua vez, interage com o contentor do ***Aggregation Server***.
- ***Aggregation Server***: Servidor REST que disponibiliza serviços de gestão de *datasets*, pedidos e tarefas de agregação. Utiliza os serviços facultados pelos contentores ***Aggregation Message Broker*** e ***Aggregation Database***.
- ***Task Queue Service***: o serviço de fila de tarefas (TQ) é uma implementação da biblioteca *Celery* que, em combinação com o ***Aggregation Message Broker***, permite que tarefas demoradas possam ser executadas em segundo plano sem interromper o bom funcionamento do serviço *REST*. Este contentor interage, com os contentores ***Aggregation Database,*** para obter/atualizar informação relativa à tarefa, e ***R workspace***, para executar algoritmos escritos em R. Este contentor não estava inicialmente planeado, mas foi integrado como resposta a longos tempos de execução de alguns algoritmos de agregação.
- ***Aggregation Message Broker***: MB *RabbitMQ* que permite que as tarefas sejam executadas em segundo plano ou armazenadas ordenadamente para posterior execução, caso não exista nenhum serviço a consumir essas tarefas.

- ***Aggregation Database:*** Conjunto de BD em *Mongo* utilizadas para armazenar dados relativos a *datasets,* pedidos e tarefas de agregação.
- ***R workspace:*** Ambiente de trabalho em R, este é necessário para a execução dos algoritmos de agregação e avaliação.

Uma vez que o serviço de TQ é maioritariamente utilizado em combinação com o serviço de algoritmos de agregação, o facto de estes estarem juntos não deverá representar um problema na eventual necessidade de escalar a aplicação. Ademais, a monitorização destas tarefas foi movida para o servidor principal, uma vez que o contentor *Task Queue Service* não disponibiliza uma API REST.

Esta abordagem mista permite que a aplicação tenha um período inicial de desenvolvimento consideravelmente mais baixo que a arquitetura pura de micro-serviços sem abdicar totalmente da sua modularidade.

Para melhor demonstrar a interação entre os contentores, é explorado na Figura 25 o UC11 - Criação de um pedido de agregação - através da sua vista de processo.

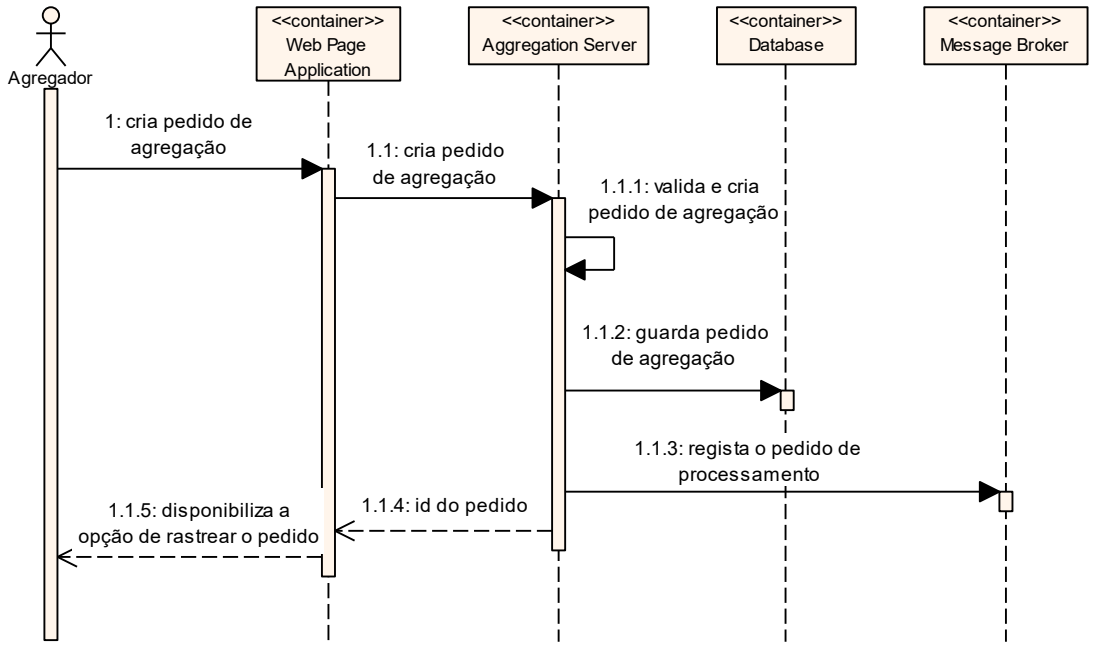



*Figura 25 – Criação de um pedido de agregação – Vista de processo*

É possível observar que o processo de criar o pedido e o adicionar à fila de tarefas é um processo simples e linear. Não obstante, este caso de uso desencadeia vários outros processos assíncronos para o tratamento do pedido de agregação e das suas tarefas, representados na Figura 26 e que serão abordados na implementação da Agregação de participantes.

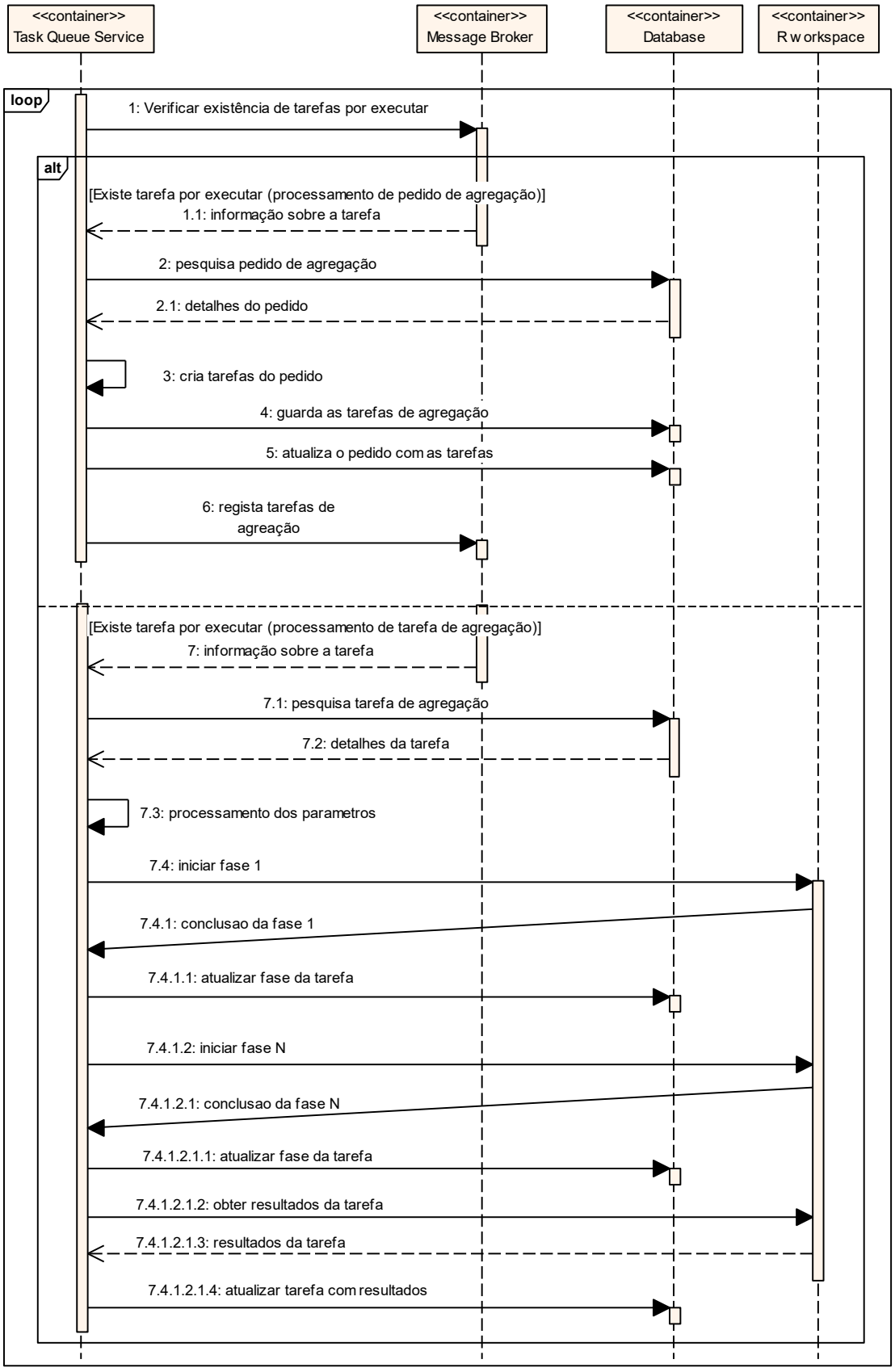



*Figura 26 – Processamento de um pedido e tarefas de agregação – Vista de processo*

## 4.3 Nível de componentes

Nesta secção são expandidos e analisados os dois principais contentores do sistema: o ***Aggregation Server*** e o ***WebPage Application***. São representadas as suas vistas lógicas no nível 3 do modelo C4: componentes.

### *4.3.1 Aggregation Server*

A Figura 27 representa a vista lógica do diagrama de componentes do contentor ***Aggregation Server***, sendo esta complementada por uma breve descrição dos mesmos.

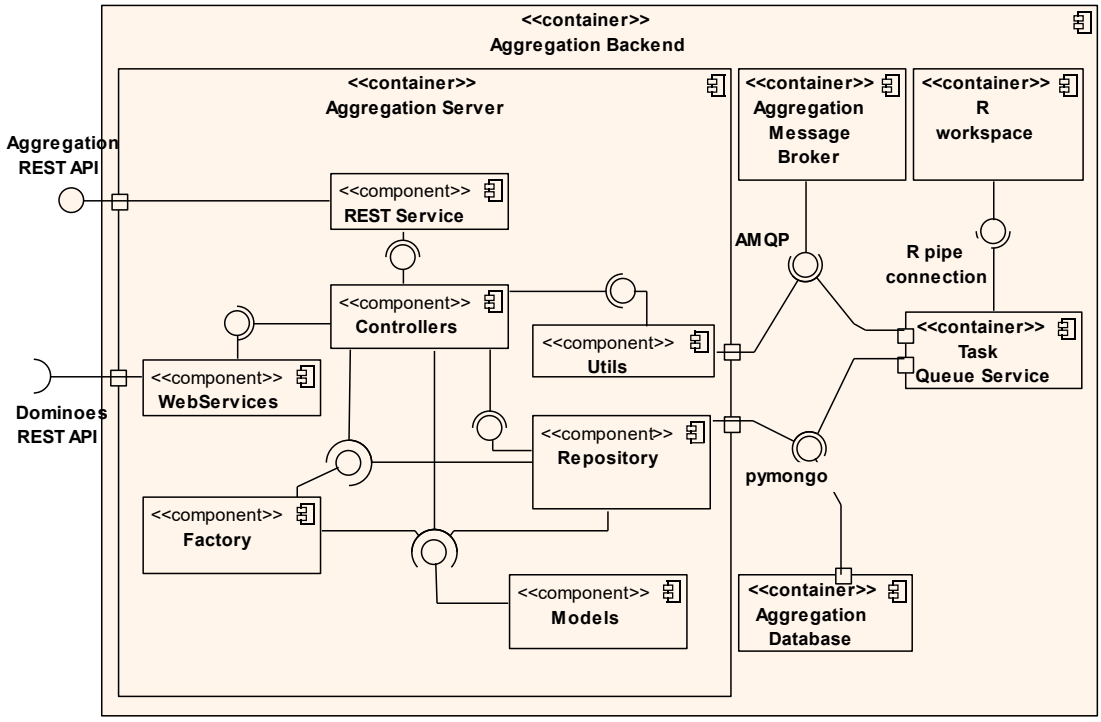



*Figura 27 – Diagrama de componentes do Aggregation Server – Vista Lógica*

- ***REST Service*** – Este componente representa um serviço REST e é o único que pode ser acedido por outros containers. Tem como objetivo redirecionar os pedidos HTTP externos para o controlador e método corretos.
- ***Models*** – Este componente corresponde ao modelo apresentado no subcapítulo 3.1 da Análise do Problema. É responsável pelo cumprimento das regras de negócio.
- ***Controllers*** – Controladores das funcionalidades disponibilizadas pelo servidor. Estes incluem os controladores de *Datasets*, Pedidos e Tarefas de agregação e definem o fluxo das ações a serem executadas de modo a concluir a execução do pedido.
- ***Utils*** – Este componente agrega todas as classes utilitárias que são comuns a várias partes da aplicação.

- ***WebServices*** – Componente responsável pela comunicação com *APIs* externas, abstraindo, assim, todos os outros componentes das suas formalidades.
- ***Repository*** – Componente responsável pelo acesso à base de dados. É a implementação do padrão *Repository* que abstrai os restantes componentes da lógica de interação com a BD.

### 4.3.2 Web Page Application

A Figura *28* representa a vista lógica do diagrama de componentes do contentor ***Web Page Application*** e é acompanhada por uma breve explicação dos mesmos.

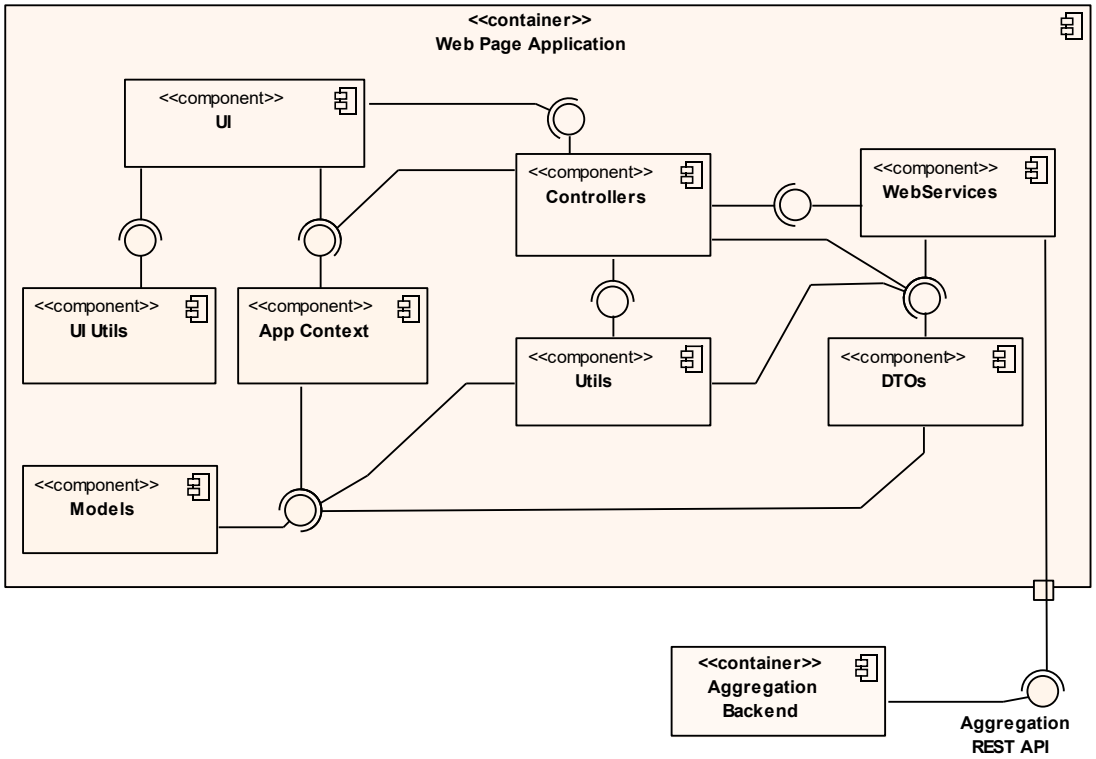



*Figura* 28 *– Diagrama de componentes da Web Page Application – Vista Lógica*

- **UI** – Este componente é responsável pela interação com o utilizador. Isto é, apresenta a informação que recebe do *App Context* ao utilizador e recebe os seus inputs, passando-os para os *Controllers*, quando necessário.
- **UI *Utils*** – Este componente agrega todas as funcionalidades utilitárias diretamente relacionadas à visualização dos dados, necessárias em várias interfaces. Um exemplo é a criação dinâmica de estruturas de dados para geração de gráficos.

- ***App Context*** – Este componente é responsável por guardar os estados da aplicação, como o modelo atual, para que possam ser acedidos por diversos níveis hierárquicos na UI.
- ***Models*** – Este componente corresponde ao domínio do problema apresentado no subcapítulo 3.1 da Análise do Problema.
- ***Controllers*** – Este componente é responsável pelo controlo das funcionalidades requeridas pela UI, serve como intermediário entre a UI e os componentes de serviços.
- ***Utils*** – Este componente agrega funcionalidades utilitárias como leitura e escrita de ficheiros, processamento de datas e arredondamentos.
- **DTOs** – Este componente corresponde aos *Data Transfer Objects* (DTO) criados pela aplicação para facilitar a comunicação com o *Aggregation Backend*.
- ***Web services*** – Este componente é responsável pela comunicação com o contentor do servidor, abstraindo, assim, todos os outros componentes das suas formalidades.

# 5 Implementação da Solução

Este capítulo descreve os detalhes e resultados da implementação da solução analisada e desenhada anteriormente. Primeiramente, é descrita a implementação dos vários sistemas do projeto. De seguida são discriminados os testes realizados e, por fim é feita a avaliação da solução desenvolvida.

## 5.1 Detalhes de implementação

Esta secção está organizada por fases, sendo que cada uma necessita da implementação de fases anteriores. Deste modo, começa por descrever a configuração inicial do sistema, seguida do desenvolvimento dos sistemas base da aplicação e, por fim, a implementação de funcionalidades assentes nesses sistemas.

### 5.1.1 Configuração inicial

Nesta secção é descrito o processo de configuração inicial da aplicação e uma breve familiarização com as tecnologias utilizadas em cada parte.

**Ambiente virtual *Python***

Em *Python*, por defeito, todos os projetos guardam as bibliotecas utilizadas no mesmo diretório da máquina. Embora este comportamento não seja problemático para um projeto isolado, quando existem vários projetos nessa máquina, podem-se gerar conflitos de dependências. Por exemplo, se cada projeto requerer diferentes versões da mesma biblioteca. Deste modo, é recomendada a utilização de ambientes virtuais, um por projeto, isolando, assim, as suas dependências [106]. A criação e ativação destes ambientes pode ser realizada em apenas dois comandos, como explícito Excerto de Código 1. Após o ambiente estar ativo, é, então, possível instalar as dependências do projeto sem receio de conflitos.

```
#sendo .venv o nome do ambiente virtual a criar
#macOS / linux
Python3 -m venv .venv          #criação
source .venv/bin/activate      #ativação

#windows
Python -m venv .venv           #criação
.venv/Scripts/activate         #ativação
```

*Excerto de Código 1 – Criação e ativação de um ambiente virtual em Python, em linha de comandos*

**Servidor REST**

No panorama do desenvolvimento de aplicações e serviços *web* em *Python* existem duas *frameworks* dominantes - *Django* e *Flask.* O consenso é que a primeira é mais robusta disponibilizando mais funcionalidades. Contudo, isso é refletido na sua curva de aprendizagem elevada, principalmente para desenvolvedores sem experiência prévia com a linguagem de programação. Por outro lado, o *Flask* possibilita a criação simples e rápida de protótipos, mas a longo prazo, tende a tornar-se mais difícil de trabalhar que a primeira [107], [108]. Deste modo, considerando a falta de familiaridade com a linguagem *Python* e a escala da aplicação optou-se pela simplicidade da *microframework Flask,* amenizando o processo familiarização com a linguagem em si.

**Filas de Tarefas**

Consideradas as alternativas analisadas na secção *Brokers* e Filas de Tarefas, foi optado pela combinação da biblioteca da TQ *Celery* com o MB *RabbitMQ*, sobretudo por motivos de compatibilidade multiplataforma. Este sistema foi configurado utilizando diferentes filas. Deste modo, é possível distribuir os recursos de processamento de tarefas pelas filas, consoante o necessário. O Excerto de Código 2, descreve o processo de configuração e utilização do *Celery*.

```
1.  #Utils.celeryConfig---------------------------------------------------------
2.  broker_url = 'pyamqp://guest@localhost//'
3.  task_ignore_result = True
4.  track_started = False
5.  include=['App.Utils.queueTasks','queueTasks','dominoesServiceTasks']
6.
7.  #Utils.celery---------------------------------------------------------------
8.  import Utils
9.  app = Celery('Utils')
10. app.config_from_object('celeryConfig')
11.
12. #Utils.queueTasks-----------------------------------------------------------
13. from Utils.celery import app
14. @app.task (queue=”aggregationQueue”)
15. def metodo_agregacao(parametros):
16.    #lógica do método
17.
18. #Qualquer parte da aplicação-----------------------------------------------
19. from queueTasks import metodo_agregacao
20. metodo_agregacao.delay(parametros)
```

*Excerto de Código 2 – Configuração e utilização do Celery, em Python*

Primeiramente, nas linhas 2 a 4 está representado o ficheiro de configuração com os detalhes do MB a ser utilizado e scripts que contêm tarefas. De seguida, nas linhas 8 a 10, inicializa-se um objeto *Celery* e é-lhe aplicado as configurações previamente definidas. A criação de uma tarefa a ser executada em segundo plano é realizada nas linhas 13 a 16, explicitamente, a

anotação da linha 14 transforma o método abaixo numa tarefa pertencente à fila da agregação, "*aggregationQueue*". Por último, na linha 19 é importado e, na linha 20, adicionado à fila de tarefas o método anteriormente definido através do comando *".delay( )".*

No entanto, o processo acima descrito apenas corresponde à criação e adição das tarefas à fila. Para o processamento das mesmas é necessário executar o serviço de execução tarefas - um *worker*. No Excerto de Código 3 é exposto o comando para a criação de um *worker* para a fila "*aggregationQueue*" definida previamente.

```
celery -A Utils.celery workers -Q aggregationQueue  - -name=aggregationWorker
```

*Excerto de Código 3 – Instanciação de um Celery workers, em linha de comandos*

**Comunicação com o *R***

Para a comunicação entre o *Python* e o *R*, foi escolhida a biblioteca *PypeR*, que é multiplataforma e possibilita a criação de vários ambientes de trabalho em *R* independentes. O Excerto de Código 4 exemplifica a sua utilização.

```
1.  import pyper as pyper
2.  r = pyper.R()                               #criação do ambiente R
3.  r.r_data = dados_criados_em_python          #atribuição de um valor a uma variável no R
4.  r.run('source("caminho_do_script.R")')      #carregamento de um script
5.  r('result<- metodoR1(r_data)')              #execução de um comando em R
6.
7.  task.updatePhase("Fase 2")                  #atualização do rastreamento
8.  taskRep.persist(task)
9.  #... outras etapas
10. resultadoFinal=r["result2"]["dado1"]        #obtenção do valor de uma variável no R
```

*Excerto de Código 4 – Utilização da biblioteca PypeR, em Python*

O primeiro passo é a criação de um ambiente R como descrito na linha 2 do excerto. Posteriormente, é possível o carregamento de um ficheiro *R* para o ambiente (linha 4), executar funções (linha 5) e recolher os seus resultados (linha 10) através do *Python*, num processo síncrono, facilitando a sua implementação com o sistema de rastreamento.

### 5.1.2 Persistência

Para a persistência dos dados foi analisada a sua natureza e tipo de operações realizadas, tendo sido tiradas as seguintes conclusões:

- Os *datasets* apenas são guardados uma vez, sendo gerado um novo em caso de modificações, portanto nunca existirá concorrência de acessos de escrita e leitura;
- Os dados de um objeto do domínio são sempre necessários na sua totalidade;

- O processo de rastreamento de tarefas executa quantidades altamente variáveis de acessos à BD;
- A maioria dos acessos à BD são operações de leitura.

Deste modo, concluiu-se que a complexidade acrescida de uma BD relacional não era justificada, tendo-se optado pela utilização de uma BD não relacional – o *MongoDB*.

**Padrão *Repository***

Com o intuito de uniformizar e encapsular o acesso à fonte de dados, foi implementado o padrão *Repository*, sendo gerado um repositório por objeto de domínio. Assim, o processo de escrita e leitura na BD podem ser genericamente representados pelos diagramas de sequência da Figura 29 e Figura 30, respetivamente.

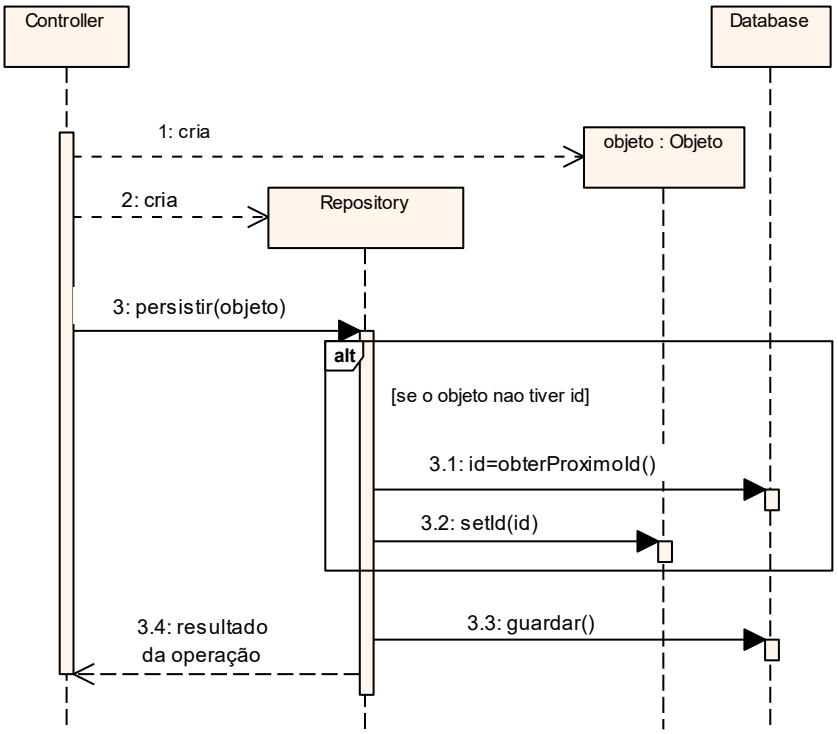



*Figura 29 – Guardar objeto de domínio (nível de componentes)*

O processo de escrita começa pela instanciação de um objeto e do respetivo repositório. Posteriormente, o objeto é persistido através do repositório que cuida de toda a lógica associada à sua persistência na BD.

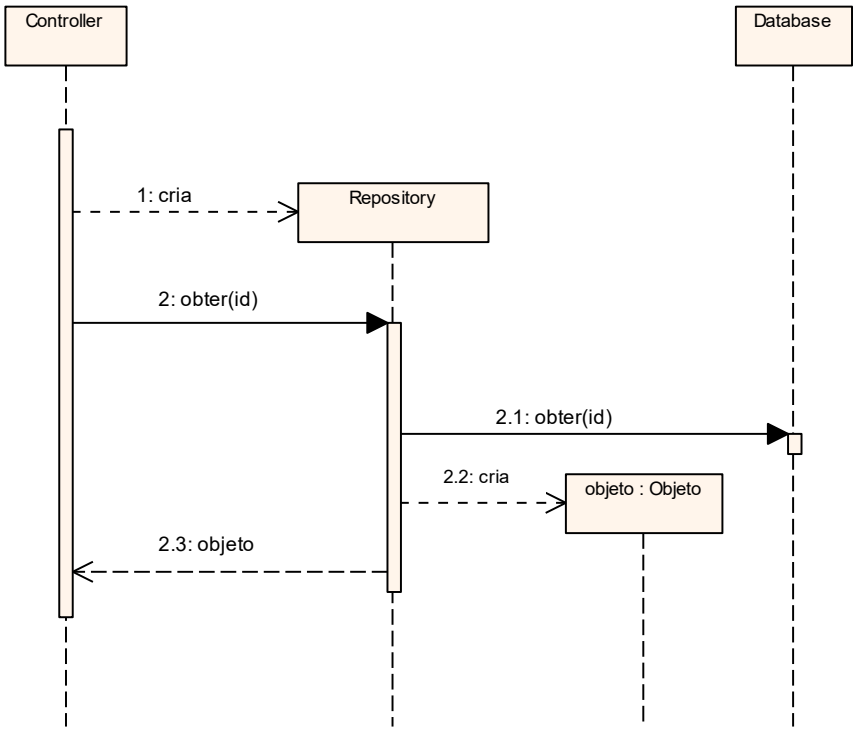



*Figura 30 – Obter objeto de domínio (nível de componentes)*

A leitura de um objeto é realizada através do seu id. Primeiramente, o repositório associado a esse tipo de objetos é instanciado. De seguida, é-lhe requerido o objeto através do seu id. Por fim, o repositório obtém o objeto da BD, instancia-o e devolve-o, isolando uma vez mais a lógica de persistência.

### 5.1.3 Rastreamento de tarefas

Para complementar os serviços de TQ oferecidos pelo *Celery*, para cada tarefa é instanciado um objeto *Task*. Estes objetos servem sobretudo para rastreamento, guardando o estado e a fase em que a tarefa se encontra, além do instante em que cada fase começou. Deste modo existem 3 estados: pendente, a executar e terminada.

À medida que uma tarefa, isto é, o método executado em segundo plano, é processada, são atingidos vários patamares, atualizando o objeto *Task* referente a essa tarefa. A forma como cada *Task* é atualizada depende, sobretudo, do seu subtipo e implementação. Por exemplo, uma tarefa de pesquisa possui um atributo extra para rastrear o processo a um nível percentual e um método para o atualizar.

#### Generalização das tarefas e Padrão *Factory*

Considerando a existência de diferentes tipos de tarefa com características diversificadas, foram generalizados todos os atributos e métodos comuns a todas as tarefas para uma classe

abstrata, permitindo a sua extensão quando necessário, tal como representado no diagrama da Figura 31.

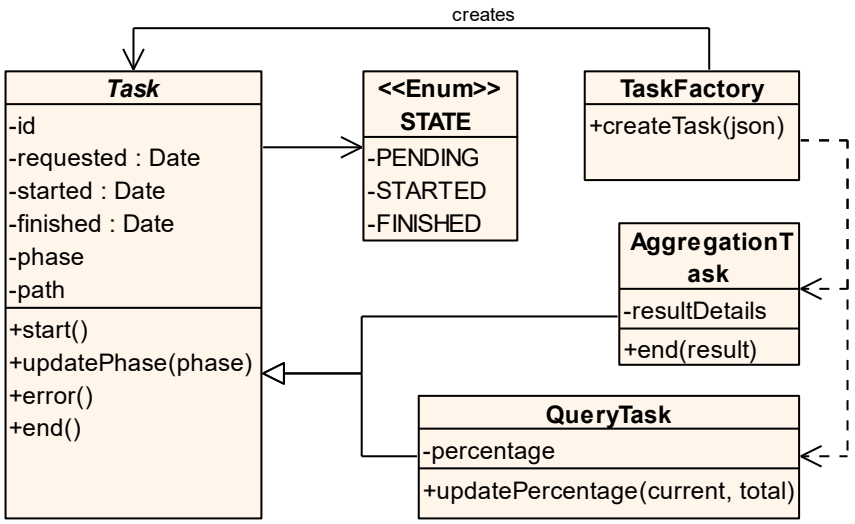



*Figura 31 – Diagrama de classes referentes às tarefas (nível de código)*

Assim, o comportamento da classe *Task* é generalizado, permitindo a sua substituição pelos seus subtipos que são responsáveis pela implementação de alterações ao comportamento original da *Task*. Deste modo são respeitados os princípios GRASP *Low Coupling*, *High Cohesion*, *Polymorphism* e *Protected Variations* e SOLID *Single-Responsibility, Open-closed* e *Liskov Substitution*. Além disso, considerando o princípio GRASP *Pure Fabrication* foi utilizado o padrão *Factory*, atribuindo a generalização da instanciação de uma tarefa a uma nova classe não pertencente ao domínio, de forma a reduzir o acoplamento.

**Criação de uma tarefa**

Independentemente do tipo de tarefa, devido ao padrão *Factory*, o processo de criação segue sempre a estrutura descrita do diagrama da Figura 32.

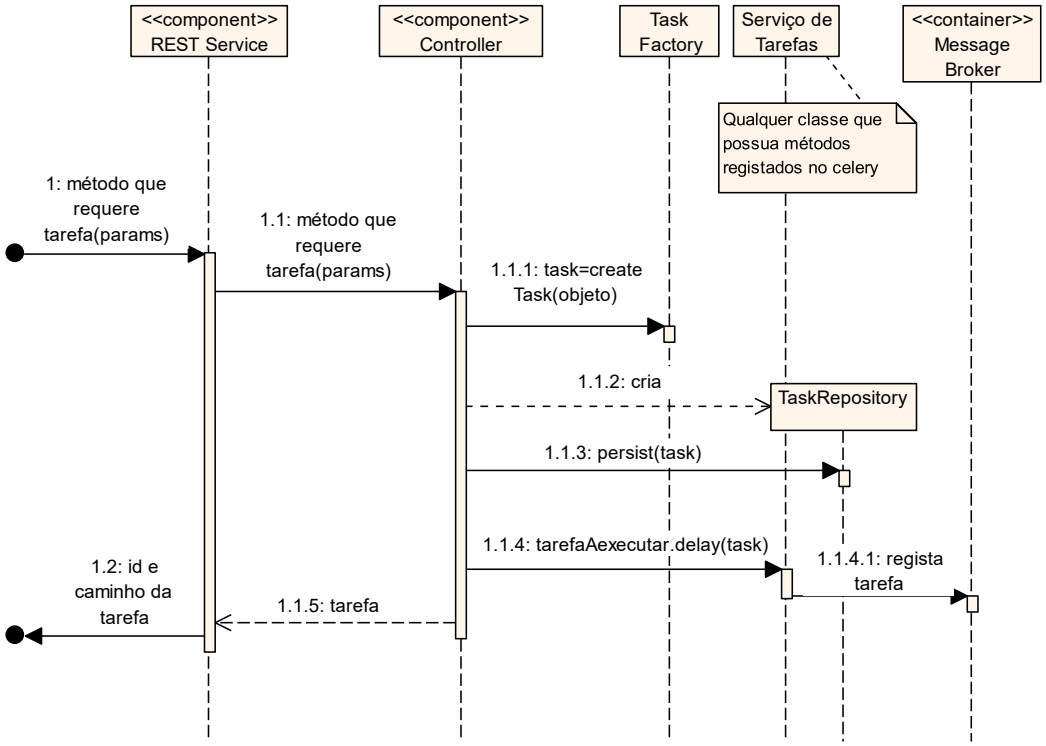



*Figura 32 – Diagrama de sequência generalizado da criação de uma tarefa (nível de código)*

Primeiramente, o controlador da operação recolhe os dados recebidos e organiza-os num objeto. É de notar que cada tipo de tarefa é gerado por um controlador diferente, pelo que estes são agrupados e representados ao nível de componentes. Posteriormente, esse objeto é enviado para a *TaskFactory,* que retorna uma *Task*, com a subclasse explícita no parâmetro "*type*" do objeto. Por fim essa tarefa é adicionada à BD e a tarefa a executar é adicionada à fila.

**Atualização e rastreamento da tarefa**

O rastreamento de uma tarefa é dividido em duas partes: a alteração dos atributos da tarefa e a sua obtenção.

A primeira é realizada nos respetivos métodos, no *backend*, que alteram e persistem o estado e fase da tarefa consoante o seu desenvolvimento. Como já foi mencionado, a atualização de uma tarefa depende do seu tipo, no entanto, a metodologia utilizada para as atualizar é consistente como representado no Excerto de Código 5.

```
1.  def backgroudAggregationTask(taskId):
2.      task=taskRep.getTask(taskId) #já convertida para AggregationTask
3.
4.      task.start()
5.      task.updatePhase("Method X")
6.      taskRep.persist(task)
7.
8.      methodX(...)
9.
10.     task.updatePhase("Method Y")
11.     taskRep.persist(task)
12.
13.     result=methodY(...)
14.
15.     task.end(result.numberOfGroups)
16.     taskRep.persist(task)
```

*Excerto de Código 5 – Atualização dos atributos de rastreamento de uma tarefa, em Python*

Em primeiro lugar, na linha 2, a tarefa é obtida através do seu repositório. De seguida, são alterados os atributos correspondentes à etapa, através de métodos como os das linhas 4, 5 e 10. Por fim, a tarefa é persistida (linhas 6, 11 e 16), recorrendo ao mesmo repositório.

A segunda parte, corresponde a pedidos consecutivos por parte do *frontend*, de modo a obter frequentemente as últimas atualizações de uma tarefa, transmitindo a sua evolução ao utilizador, como representado no diagrama da Figura 33.

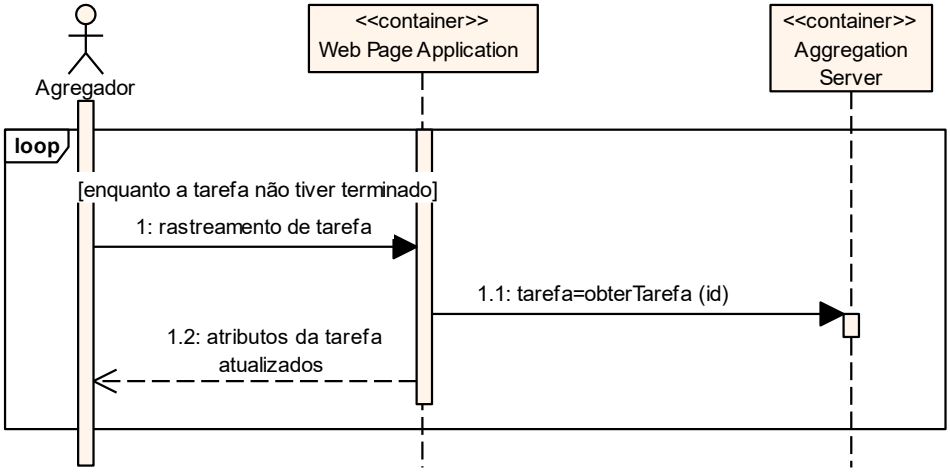



*Figura 33 – Diagrama de sequência do processo de rastreamento de uma tarefa (nível de contentores)*

Deste modo a interface gráfica recorre a pedidos *GET*, repetindo o pedido após algum tempo, enquanto a tarefa não tiver sido registada como terminada. A interpretação desses pedidos pela interface será abordada em mais detalhe nas secções de Carregamento de *datasets* a partir da API DOMINOES e Agregação de participantes.

### 5.1.4 Carregamento de *datasets* a partir da API DOMINOES

A API DOMINOES disponibiliza medições de um determinado tipo, ou *tags* referentes a participantes, ou *units*. As medições podem ser facultadas em cinco granularidades: instantânea (períodos de 15 minutos), horária, diária, mensal e anual.

#### Comunicação com a API DOMINOES

O processo de recolha de dados da API DOMINOES, requer a autenticação no sistema, de modo a obter um *token*. Este é, depois, guardado na BD, juntamente da sua data de expiração, para que possa ser reutilizado em vários pedidos.

Normalmente, o processamento de um pedido para a API seguiria o processo descrito na alternativa da Figura 34.

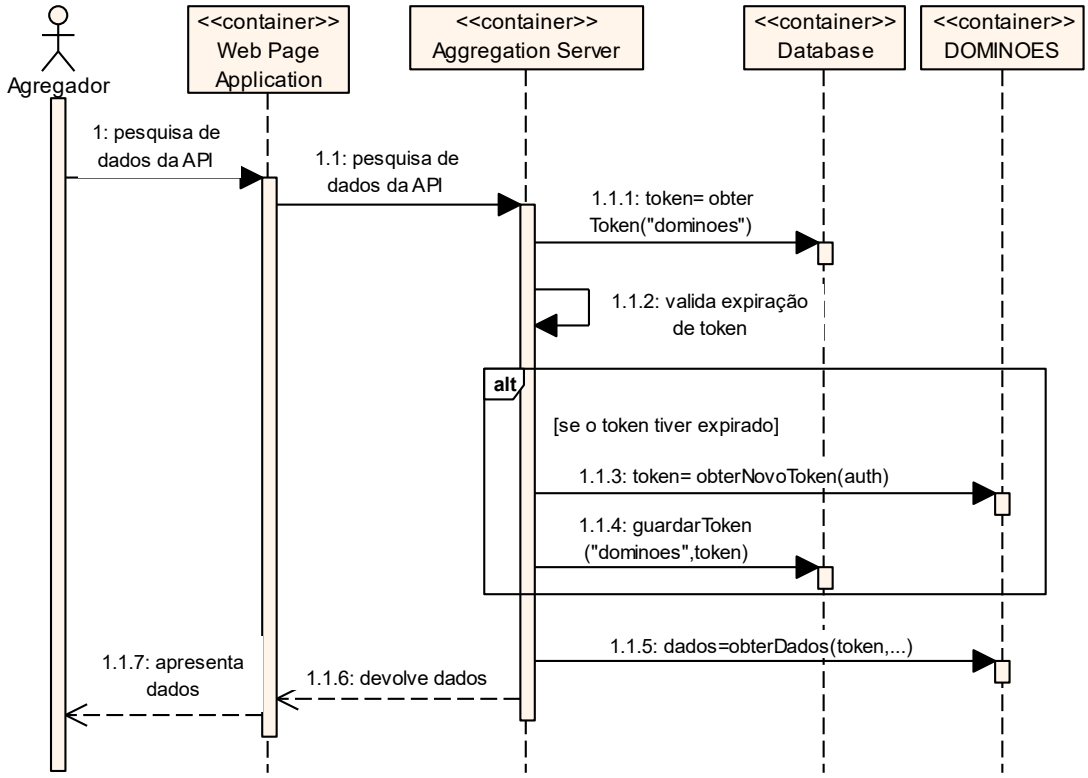



*Figura 34 – Diagrama de sequência da retribuição linear de dados da API DOMINOES (nível de contentores)*

O servidor recebe o pedido e obtém o último *token* utilizado, caso este tenha expirado, é requerido um novo *token*. De seguida, o servidor reencaminha o pedido para a API DOMINOES, apenas dando uma resposta ao cliente quando a própria API retornasse o seu resultado. Porém, esta abordagem é insuficiente quando os pedidos são demorados, como é o caso da recolha de medições com granularidade instantânea. Para solucionar esse problema

foram implementadas duas medidas: a persistência local das medições obtidas e a utilização do sistema de tarefas para a recolha de dados.

**Persistência local**

Como referido, a persistência local tem como principal objetivo guardar os resultados dos pedidos de pesquisa realizados para que não seja necessário recorrer excessivamente à API, funcionando como *cache*. Além disso, concluiu-se que seria mais vantajoso que esses dados fossem persistidos no servidor permanentemente. Por motivos de consistência, foi decidido que seriam localmente guardadas as medições na granularidade mais baixa. Deste modo, mesmo que um pedido requeira uma granularidade anual, o servidor irá realizar um pedido com granularidade instantânea à API, transformando-a posteriormente na granularidade desejada.

Considerando as limitações de tamanho máximo por documento na BD, foram analisados vários modelos de dados, dos quais se destacam os dois representados na Figura 35.

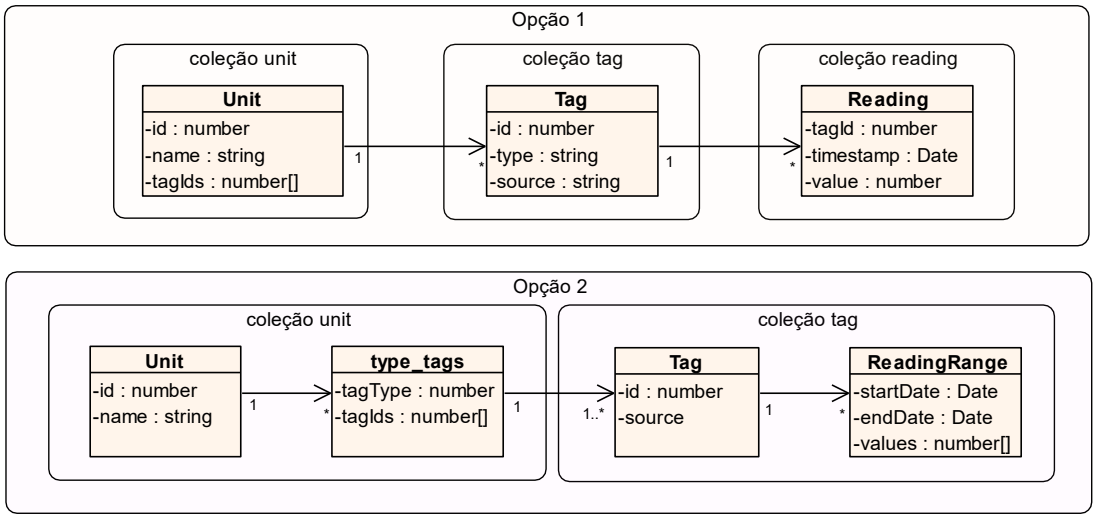



*Figura 35 – Modelos de dados não relacionais para a persistência de medições*

No primeiro modelo, existem três coleções: *units*, *tags* e *readings.* Esta divisão permite que todas as entradas das coleções tenham um tamanho previsível, garantindo, assim, o cumprimento da restrição de tamanho. Porém, o principal foco deste modelo está na última coleção - *reading*. Esta guarda o id da *tag*, o instante e o valor da medição, o que torna a sua compreensão mais fácil, mas obriga à iteração por milhares de entradas para descobrir, por exemplo, intervalos em falta.

Por outro lado, no segundo modelo, existem apenas duas coleções. A primeira, *unit*, representa a unidade e tem um mapa que divide as *tags* por tipos. Na segunda coleção, *tag*, é de notar que a objeto *Tag* passa a ter associada intervalos de medições em vez de medições. Este formato permite localizar mais facilmente quais os intervalos em falta, o que é mais vantajoso para a pesquisa de dados. Uma desvantagem deste modelo é que não há garantias que o limite de tamanho não seja, eventualmente, ultrapassado. Porém, foi computado e concluído que para tal acontecer, teriam de ser persistidos o equivalente a 30 anos de medições constantes. Deste modo, foi optado pelo segundo modelo apresentado. Foram, então, criadas as classes apresentadas na Figura 36.

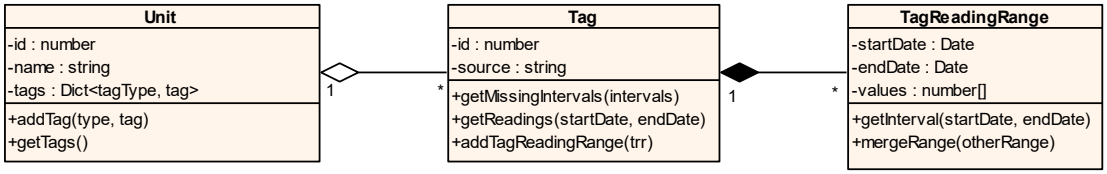



*Figura 36 – Diagrama de classes referente às medições (nível de código)*

De modo semelhante ao modelo de dados, existe uma classe *Unit* que possui um dicionário que faz o mapeamento entre os tipos de *Tags* e os seus ids para essa unidade. Cada *Tag* possui vários intervalos de leituras, *TagReadingRange*, que regista a data de início e fim do intervalo, e os valores medidos. Esta implementação resulta numa lógica bastante refinada para a junção de vários intervalos sobrepostos, sendo esta assegurada através de testes unitários.

**Implementação do sistema de tarefas para pedidos de pesquisa**

Como descrito na secção 5.1.3, a atualização do estado do objeto tarefa é realizada diretamente pelo processo que a executa. Ademais, é de realçar que a API possibilita a recolha coletiva dos dados. Esta configuração implica longos períodos sem *feedback*, mas é a alternativa de pesquisa mais eficiente.

Deste modo, a plataforma disponibiliza dois modos de pesquisa – a pesquisa coletiva, que resulta num menor número de atualizações na tarefa, e a pesquisa faseada, que permite uma maior quantidade de atualizações ao estado da tarefa, resultando em *feedback* mais preciso e frequente.

Considerando que a segunda opção inclui todos os passos da primeira, é apenas apresentada essa vista de processo, no diagrama da Figura 37. É de realçar que este diagrama é referente à execução da tarefa de pesquisa. A criação da tarefa já foi descrita no capítulo anterior. Além

disso, destaca-se a omissão da instanciação dos repositórios, a obtenção/validação do *token* e parte da persistência das unidades, por motivos de legibilidade.

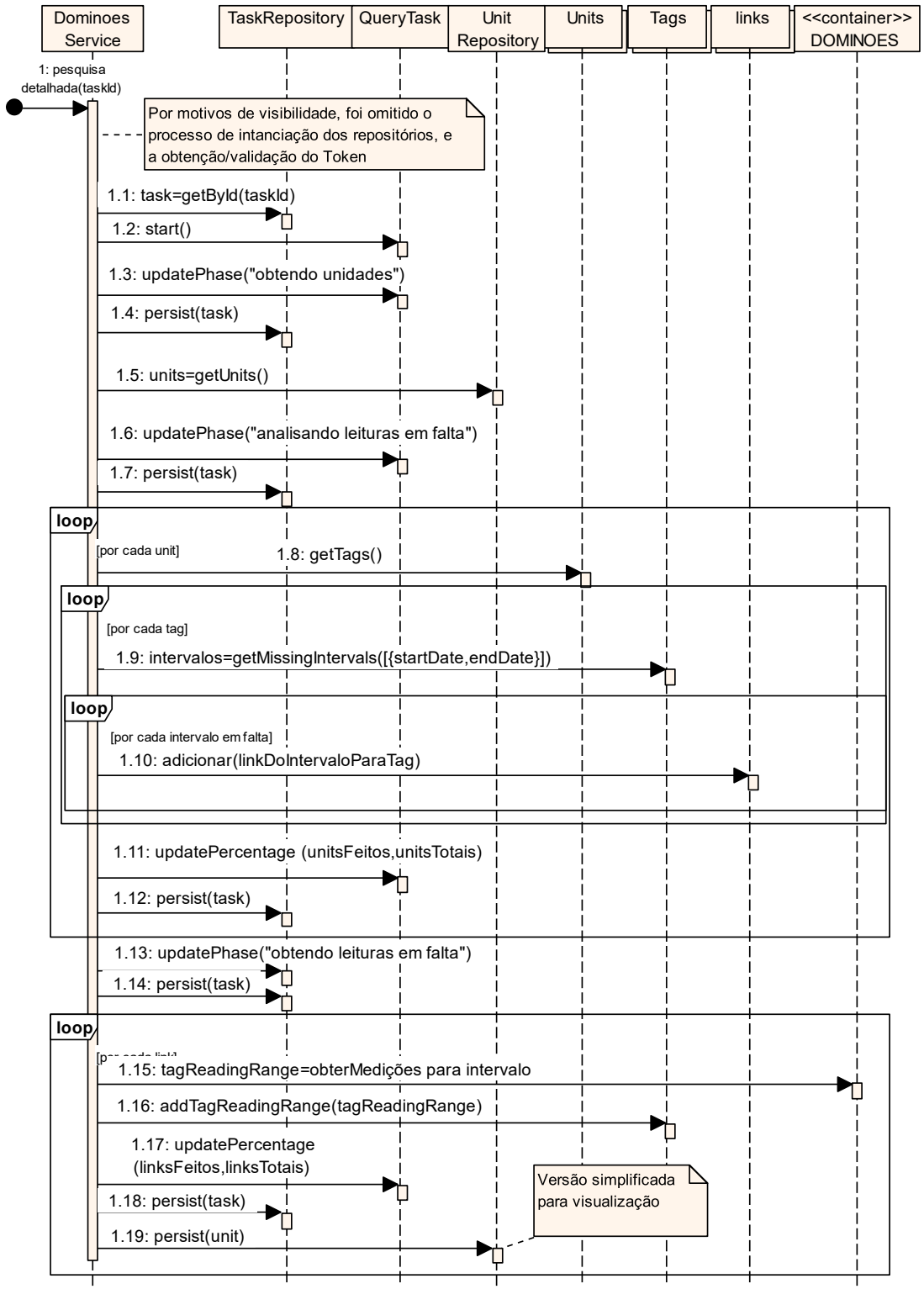



*Figura 37 – Diagrama de sequência da obtenção faseada de dados da API*

A obtenção de dados a partir da API do projeto integrador começa pela consulta dos detalhes da pesquisa através da tarefa associada (passo 1.1). Posteriormente (passo 1.8), são

encontradas as *Tags* de cada unidade e, por cada *Tag* é averiguada a diferença entre o período a pesquisar e os intervalos já registados na BD como demonstrado na Figura 38. De seguida (passo 1.10), é computado o *Uniform Resource Link* (URL) correspondente a cada intervalo em falta e é adicionado a uma lista. Por fim, por cada URL, é realizado um pedido síncrono para a API, retornando o intervalo em falta, que é adicionado e fundido com os já existentes nessa *Tag* (passo 1.16). Durante todo o processo ocorrem várias atualizações à fase da tarefa, nomeadamente nos passos 1.4, 1.7, 1.12 e 1.19.


*Figura 38 – Visualização de intervalos em falta num período*

Na figura, o intervalo de pesquisa é de 2 a 11 (inclusive) dos quais apenas o momento 2, 4, 5 e 10 estão em falta. Deste modo, são detetados 3 intervalos em falta: {2}, {4,5} e {10}.

**Integração na interface gráfica**

Na página de configuração existe uma aba dedicada à importação de dados a partir da API DOMINOES, apresentada na Figura 39.

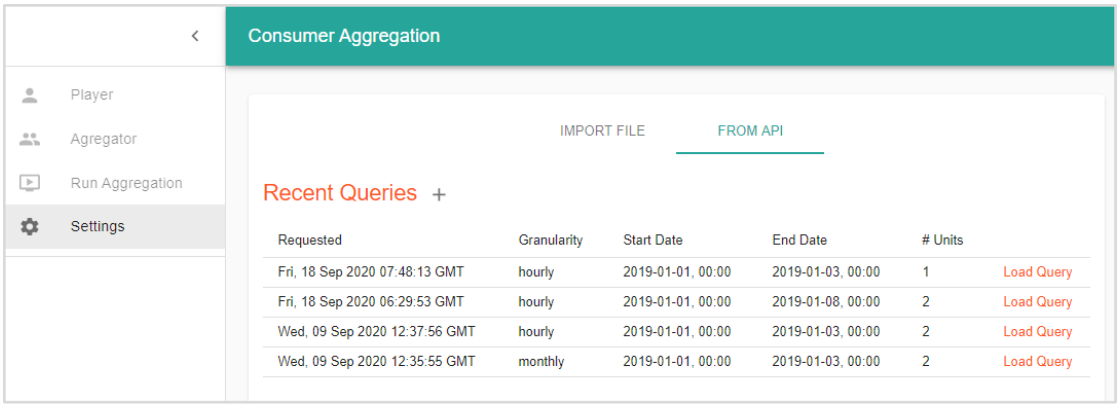



*Figura 39 – Interface gráfica da plataforma, divisão de configuração, aba da API, pedidos recentes*

Ao entrar nessa secção, é apresentada uma lista de pedidos recentes, detalhando as suas características como granularidade, período e número de unidades selecionadas. A partir desta interface, carregando no botão "+", é exibida outra interface, que gera os *popups* explícitos na Figura 40 e que permitirá a criação de um novo pedido.

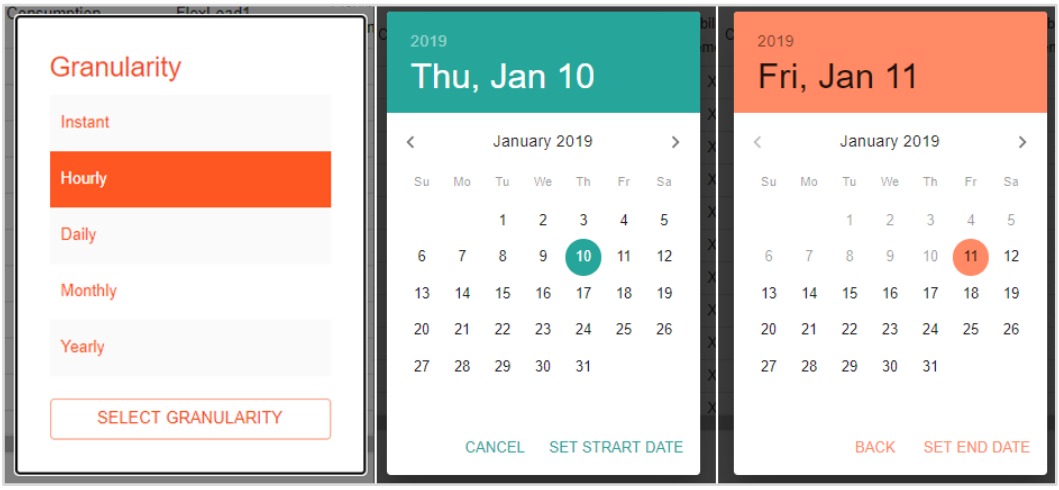



*Figura 40 – Importação de dados da API, Popups de seleção de granularidade (esquerda), data de início (centro) e data de fim (direita)*

Para gerar um novo pedido, é obrigatória a seleção da granularidade e do intervalo temporal a considerar, pelo que esses parâmetros são imediatamente requeridos pela interface, individualmente, como demonstrado na Figura 40. Caso o utilizador cancele o seu preenchimento, volta à interface dos pedidos recentes. Em caso de sucesso, é mostrada a interface da Figura 41.

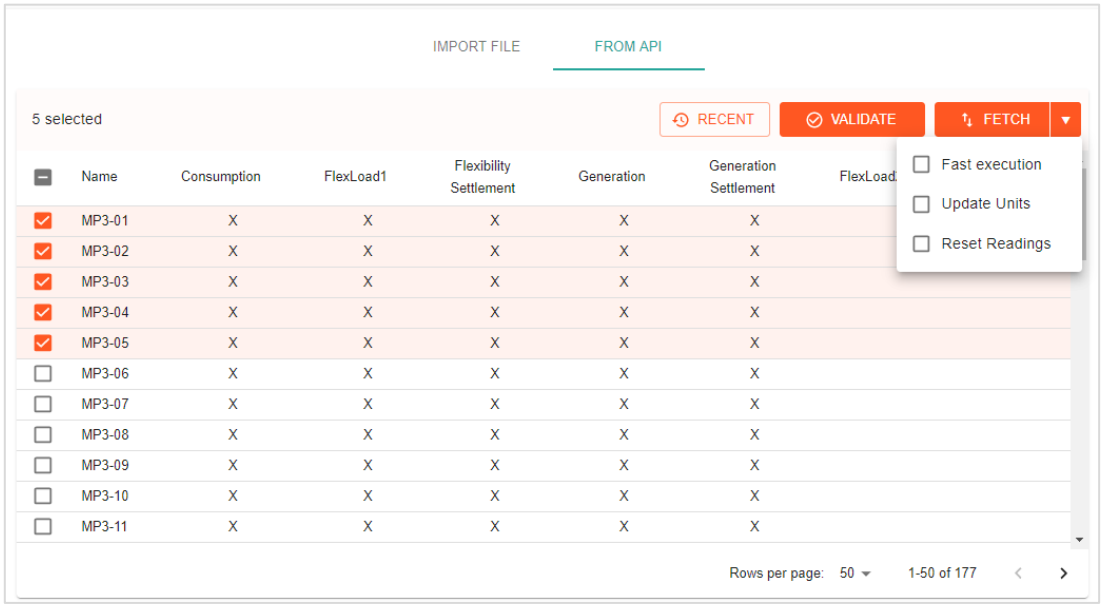



*Figura 41 – Interface gráfica da plataforma, divisão de configuração, aba API*

Após a seleção dos parâmetros de pesquisa, é apresentada a interface na Figura 41 onde é possível verificar que parâmetros é que cada unidade contém e selecionar as unidades a serem importadas para a plataforma. Ao selecionar uma unidade, são desbloqueados os

botões de validação e busca. A opção de busca tem um menu que permite, entre outros, ativar o modo de pesquisa rápido e com menos *feedback* discutido anteriormente.

Ambas as ações (validação e busca) desencadeiam a mesma tarefa de pesquisa, porém os seus resultados serão obtidos e apresentados de forma diferente. Por um lado, os resultados obtidos na validação focam-se em realçar genericamente as partes do intervalo que apresentam falhas, como exemplificado na Figura 42.

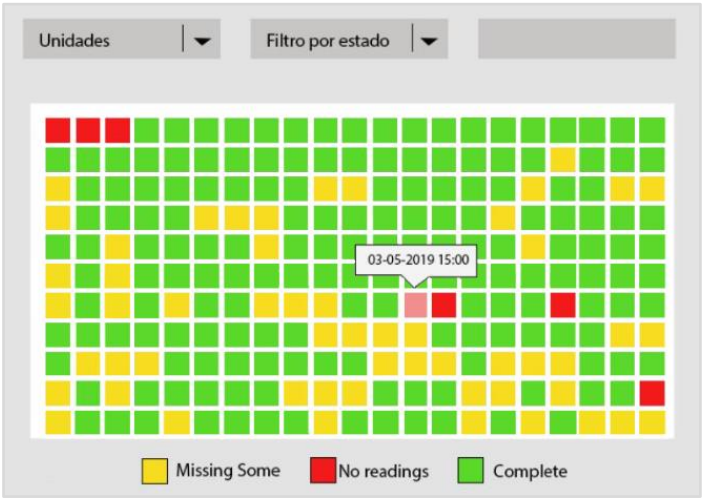



*Figura 42 –Interface gráfica de validação*

Esta disposição permite ao Agregador facilmente identificar períodos em que existiram falhas e obter ainda mais detalhes ao carregar num dos quadrados.

Por outro lado, a tarefa de pesquisa tem como finalidade a importação do *dataset* para o sistema, pelo que os seus resultados serão estruturados de forma a facilitar a essa conversão.

Todo o processo descrito acima, desde o momento em que o botão de pesquisa é pressionado até à criação da tarefa de pesquisa está descrito abaixo no diagrama da Figura 43. Uma vez mais, é de realçar que este é apenas o processo de criação de uma tarefa, sendo esta posteriormente executada, como exemplificado na Figura 37.

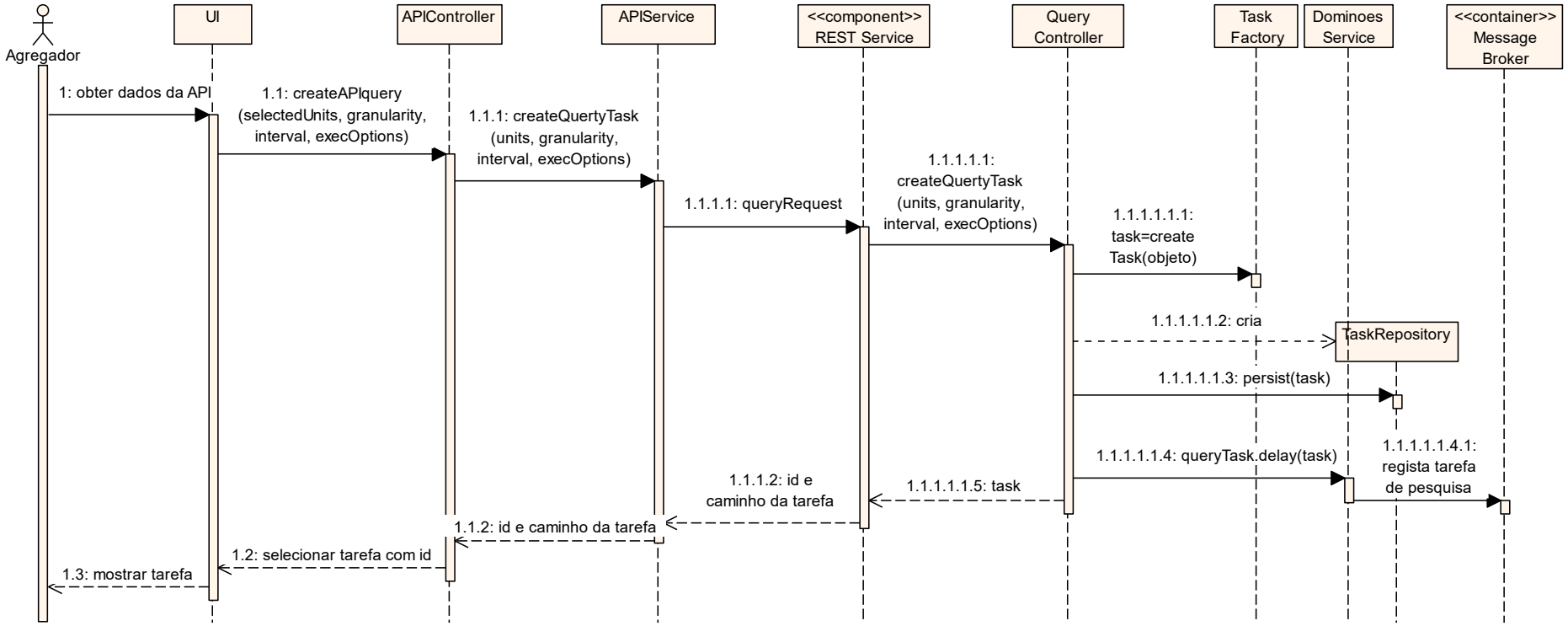



*Figura 43 – Diagrama de sequência do processo de criação de uma tarefa de pesquisa (nível de código*

O processo de criação de uma tarefa é bastante linear, sendo enviada um pedido POST do *frontend* para o servidor, que o dirige para o controlador correto. Após a criação da tarefa, é aberta a interface experimental de rastreamento representada na Figura 44.


*Figura 44 – Interface de rastreamento e importação de uma tarefa de pesquisa*

Esta interface expõe a fase e percentagem em que a tarefa se encontra e pode ser acedida para qualquer tarefa de pesquisa, independentemente do seu estado ou data de criação. Além disso, caso a tarefa tenha finalizado, é possível a importação dos dados pesquisados como *dataset,* completando assim o UC18. Esse processo é descrito na Figura 45, por motivos de visibilidade, as interações com o servidor foram discriminadas a um nível de contentores.

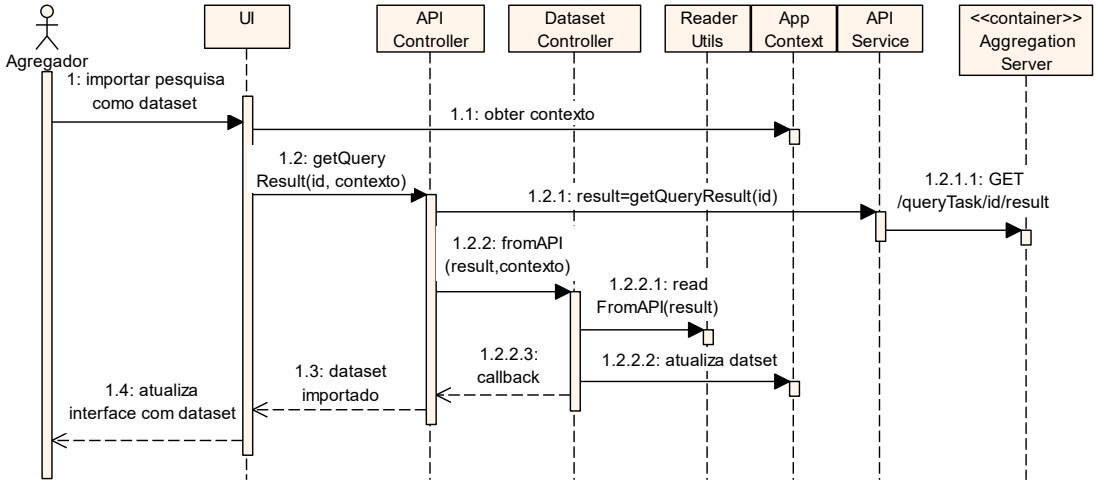



*Figura 45 – Diagrama de sequência da importação de um dataset a partir de uma pesquisa (nível de código)*

Inicialmente o *APIController* requer ao serviço correspondente os resultados de uma pesquisa. Deste modo, o APIService executa um pedido *GET* para o servidor, que retorna as unidades e as suas medições ao longo do período, para cada parâmetro, num formato fácil de interpretar pelo *frontend*. De seguida o *APIController* recorre a outro controlador, o *DatasetController*, para a leitura a importação dos dados como *dataset*, mantendo assim o *Single Responsibility Principle* dos princípios SOLID. Por fim, o *DatasetController* utiliza a classe *ReaderUtils* para mapear os dados recebidos para um *dataset* e atualiza-o no contexto.

### 5.1.5 Carregamento de *datasets* a partir de ficheiro

Comparativamente ao Carregamento de *datasets* a partir da API DOMINOES, este processo é simples e linear. A importação pode ser realizada para ficheiros que sigam a estrutura descrita na Figura 46.

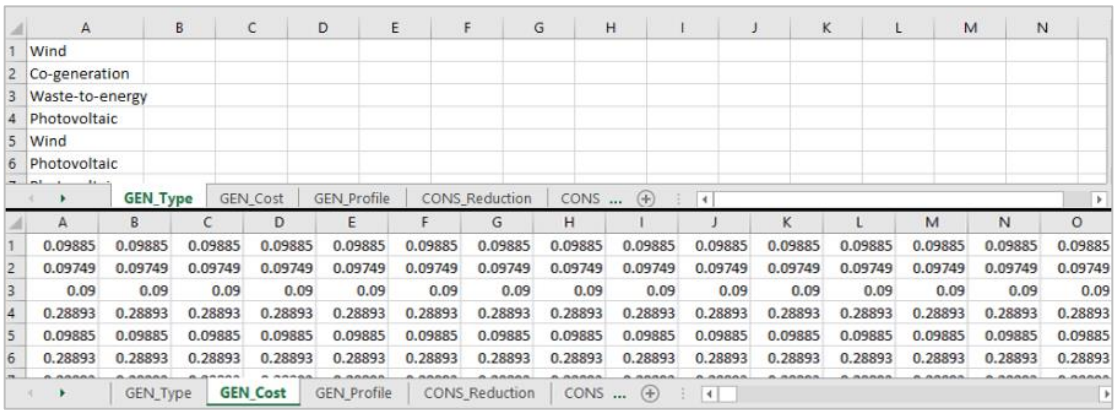



*Figura 46 – Estrutura do ficheiro de importação Excel*

No ficheiro, cada folha corresponde a um parâmetro e cada linha corresponde a um participante, sendo o número da linha o id do participante. No caso de o parâmetro ser categórico ou numérico, apenas a primeira coluna da folha se encontra preenchida, como no exemplo da folha *GEN_Type*. Por outro lado, como exemplificado pela folha *GEN_Cost*, os parâmetros de vetores numéricos contêm os valores das várias medições ao longo do período, sendo cada medição representada numa coluna diferente. A importação deste ficheiro pode ser realizada através da interface exibida na Figura 47.

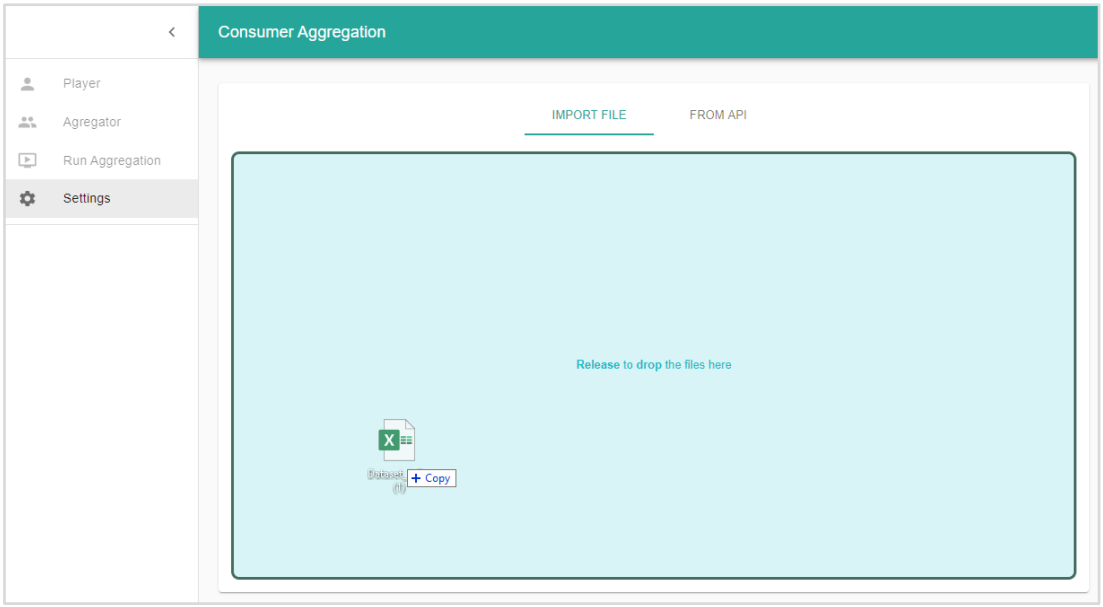



*Figura 47 – Interface gráfica da plataforma, divisão de configuração (reduzida)*

Na página inicial da divisão de configuração, na aba "*Import File*", existe uma área para onde é possível arrastar um ficheiro. É ainda, possível a seleção manual do ficheiro, carregando nessa mesma secção. Após o ficheiro ser selecionado, é iniciado o processo descrito no diagrama da Figura 48. Considerando que a leitura de um ficheiro é um processo demorado e bloqueante, a interface apresenta um *spinner*, informando o utilizador que o processo irá demorar.

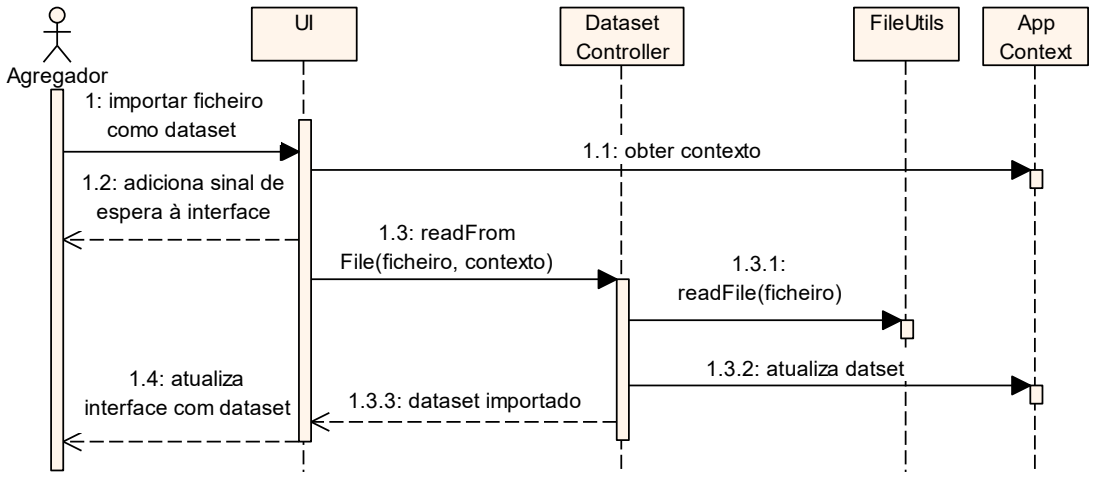



*Figura 48 – Diagrama de sequência da importação de um dataset a partir de um ficheiro*

Observando o diagrama, após ser escolhido um ficheiro a interface é atualizada mostrando um sinal de espera – o *spinner*. De seguida, o *DatasetController* recorre à classe utilitária *FileUtils* para a leitura do ficheiro. Por fim, o mesmo controlador atualiza o *dataset* no contexto.

### 5.1.6 Gestão de *datasets*

A gestão de *datasets* é feita na divisão de configuração, abaixo das opções de carregamento descritas em 5.1.4 e 5.1.5. Como é possível observar na Figura 49, na secção "*Recent Datasets*" estão discriminados os *datasets* guardados recentemente e o atualmente selecionado. A Figura 50 enumera os componentes mais importantes desta interface.

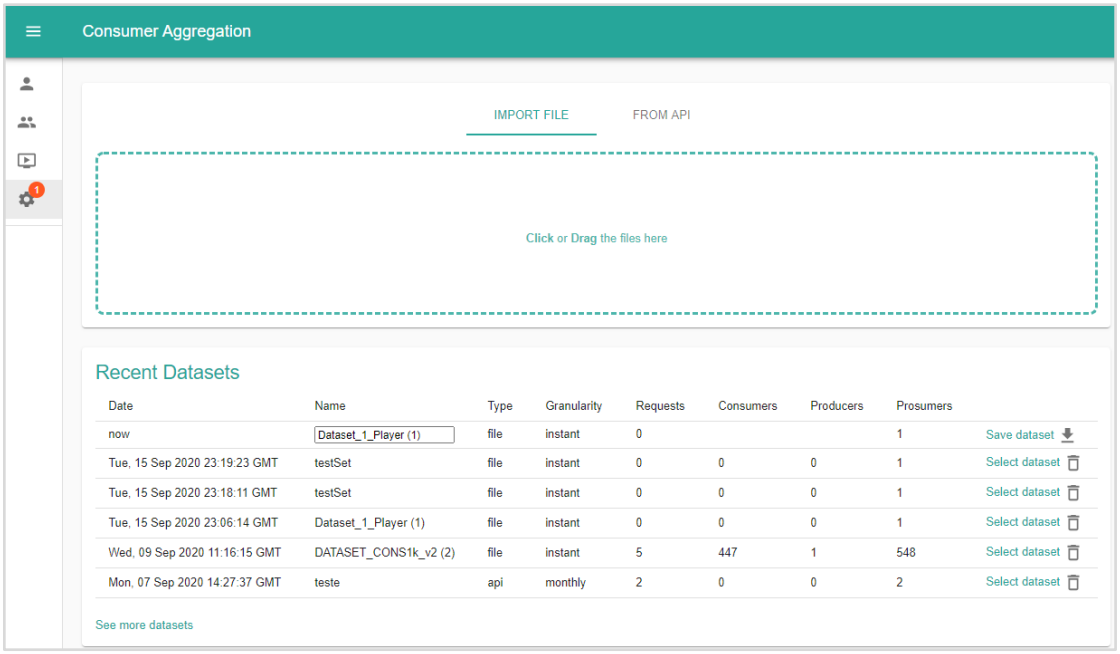



*Figura 49 – Interface gráfica da plataforma, divisão de configuração*

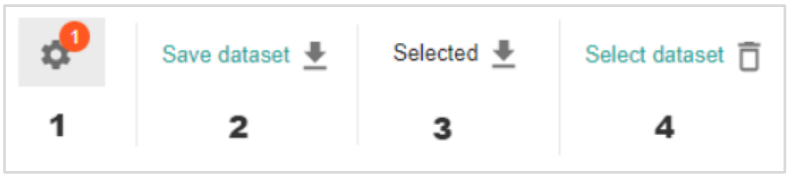



*Figura 50 – Componentes gráficos da gestão de dataset*

Na gestão de *datasets*, existem alguns componentes da interface gráfica que devem ser considerados, discriminados na Figura 50. O primeiro, permite ao Agregador saber se a sua versão atual do *dataset* está persistida no servidor. Caso apareça a notificação, como é o caso do exemplo, significa que o *dataset* atual é novo ou foi alterado. Quando o primeiro componente tem a configuração, a primeira linha dos "*Recent Datasets*" irá sempre corresponder ao *dataset* atual, e aparecerá a opção de o guardar – o componente 2. Por outro lado, caso o *dataset* corresponda à versão guardada no servidor, aparecerá o componente 3, indicando que o mesmo está selecionado. Ambos os componentes 2 e 3 são acompanhados de um botão que permite o *download* do *dataset*. Por fim, o último componente é revelado

para todos os outros *datasets* da lista não selecionados, permitindo a sua seleção ou eliminação.

Devido às limitações de tamanho do *MongoDB*, um *dataset* não é diretamente persistido na BD. Porém, como observado na Persistência, um *dataset* nunca é alterado, existindo apenas acessos concorrentes de leitura, que são operações não conflituantes. Assim, o *dataset* é guardado diretamente no servidor em formato *Javascript Object Notation* (JSON) (de modo a facilitar a leitura), sendo persistido na BD o seu id, nome, alguns dos seus detalhes e o caminho para o ficheiro.

Este processo é desencadeado ao carregar no componente 2 da Figura 50 e o seu fluxo pode ser verificado no diagrama de sequência do UC02, representado na Figura 51. Apenas a parte do servidor é apresentada a nível de código, uma vez que é a parte mais relevante à persistência do *dataset*.

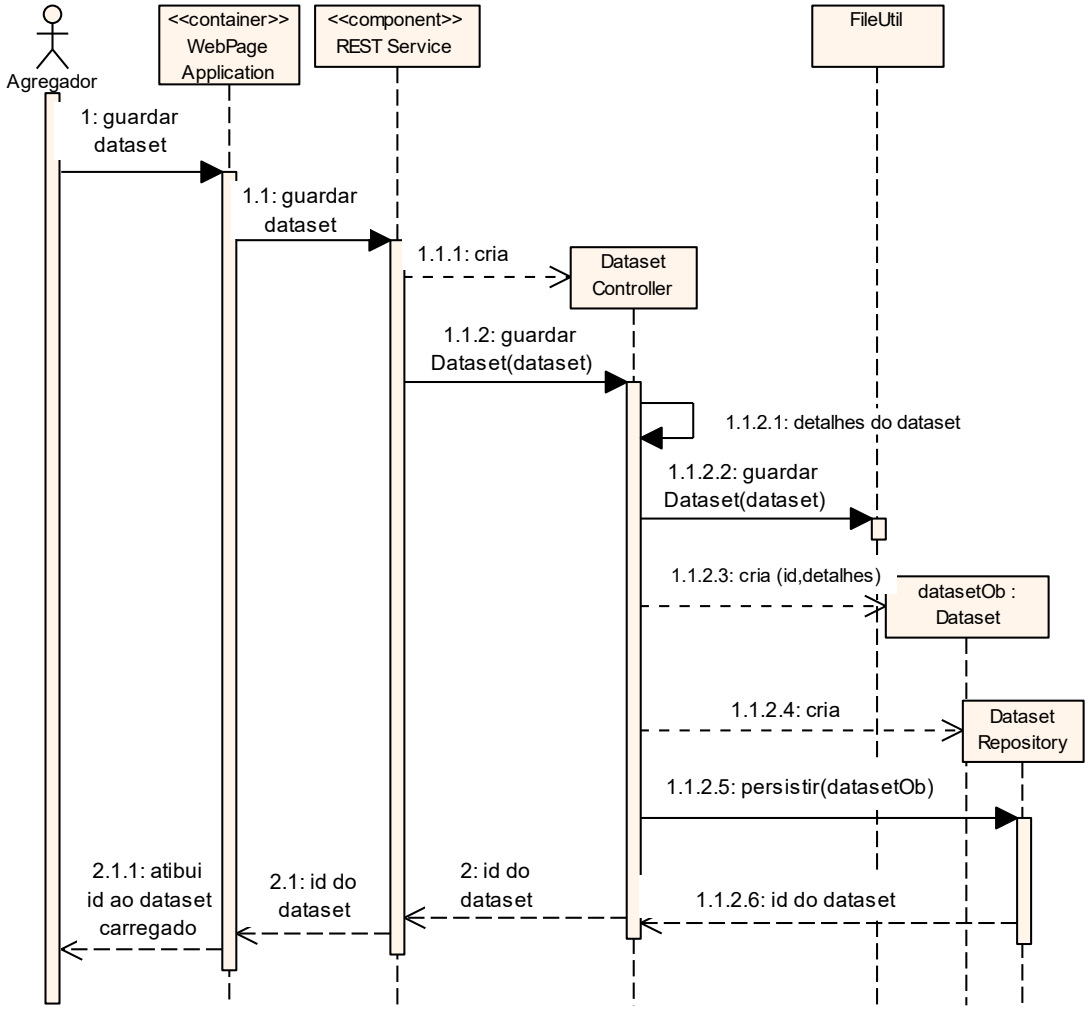



*Figura 51 – Diagrama de sequência do UC 02 – Guardar um dataset (nível de código)*

Por motivos de visualização a Figura 51 apresenta o *frontend* a nível de contentores. O *frontend* realiza um pedido *POST* para o servidor, como descrito anteriormente, o *dataset* é guardado localmente, através da classe *FileUtils* no passo 1.1.2.2. De seguida, é criado um objeto da classe *Dataset* que contém a informação sobre o *dataset* como o seu id e caminho. Por fim, esse objeto é persistido e o seu id é retornado.

De modo semelhante, a obtenção de um *dataset* é desencadeada pelo componente 4 – "*Select dataset*" – da Figura 50. Este corresponde ao UC04 e o seu processo é descrito no diagrama da Figura 52. Naturalmente este utiliza as mesmas classes que o UC02, mas recorre a métodos referentes ao carregamento do ficheiro em vez da sua persistência.

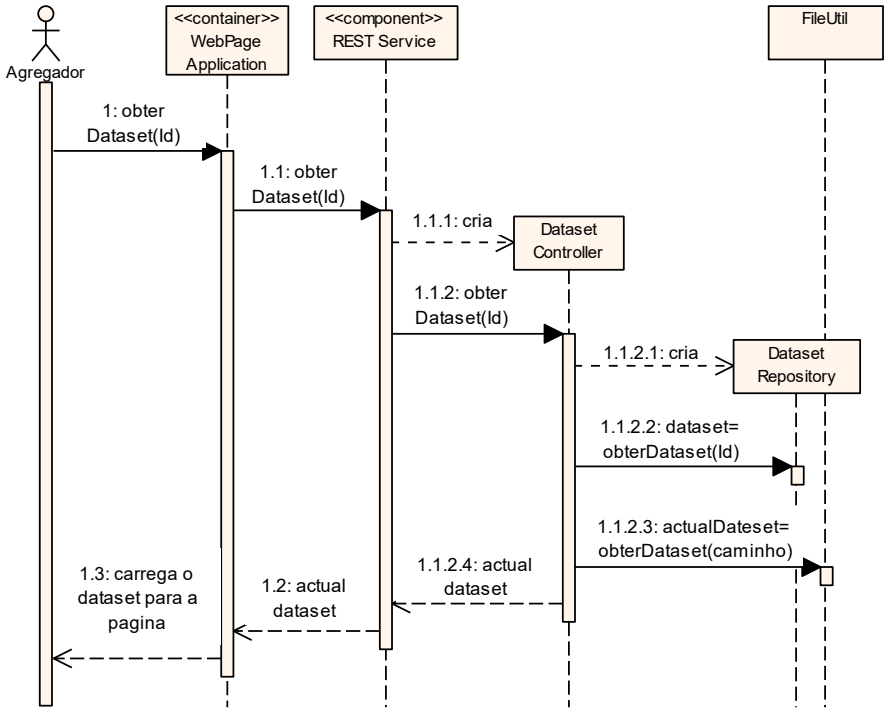



*Figura 52 – Diagrama de sequência do UC04 – Carregar um dataset (nível de código)*

O *frontend* envia um pedido *GET* para o servidor com o id do *dataset* a importar. Seguidamente, o servidor pesquisa na BD a sua existência e, em caso de sucesso, recorre ao *FileUtils* para a sua leitura. Por fim, o documento lido é retornado.

O UC17, apagar um *dataset* guardado, é iniciado pelo botão presente no componente 4 da Figura 50. O seu processo é em tudo semelhante ao anterior, diferenciando-se pela remoção do ficheiro após este ser encontrado, ao invés do seu retorno, como representado na Figura 53.

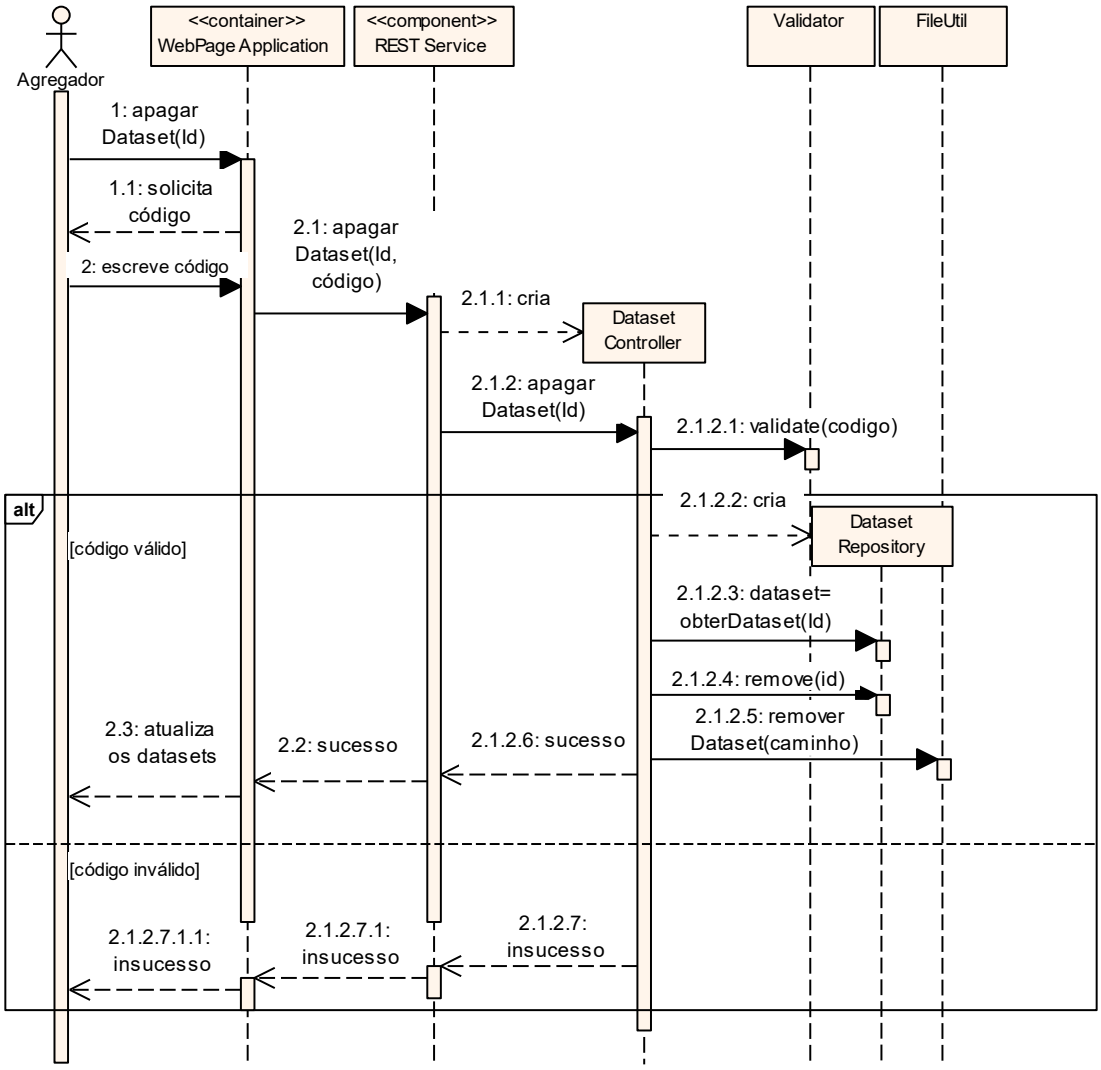



*Figura 53 – Diagrama de sequência do UC17 – Remover um dataset (nível de código)*

O servidor recebe um pedido *DELETE* do *frontend* com o id do *dataset* a remover e o código de validação. Esse pedido é encaminhado para o *DatasetController*, que através do *Validator* verifica a legitimidade do código. Em caso de sucesso, recorre ao *DatasetRepository* para verificar a existência do *dataset* no sistema no passo 2.1.2.3. Caso o ficheiro exista, é então apagado da BD e, por fim, removido do servidor através da classe utilitária *FileUtils*, retornando o sucesso da operação.

### 5.1.7 Criação dinâmica de Parâmetros

Para a implementação de parâmetros dinâmicos, são necessários três requisitos – configuração dos parâmetros recebidos, a criação dinâmica de classes que os reflitam e a interpretação dos dados pela plataforma. O último requisito apenas será abordado em 5.1.8

**Configuração dos Parâmetros**

O primeiro requisito visa resolver o mapeamento dinâmico dos parâmetros para os tipos corretos. Deste modo, foram criados três ficheiros de configuração:

- ***Units.json***

Este primeiro ficheiro faz o mapeamento de um tipo de medição para a sua unidade ou representação como exemplificado no Excerto de Código 6.

```
1.  {
2.       “energy”:          "Watts",
3.       “time”:            "Time (dd, hh:ss)",
4.       “price”:           "Monetary Unts",
5.       “percentage”:      "Representation (%)",
6.       “quantity”:        "Quantity",
7.  }
```

*Excerto de Código 6 – Ficheiro Units.json*

- ***ParameterSettings.json***

O segundo ficheiro, representado no Excerto de Código 7, define os detalhes sobre cada parâmetro e como é que a plataforma o deve interpretar. Desta forma, é possível a definição de novos parâmetros, sem necessidade de alterar o código. Uma vez que este é o ficheiro mais importante para a Interpretação dos Parâmetros, uma explicação mais detalhada do mesmo será realizada nessa secção.

```
1.  {
2.      “consumption”: {“name”: ”Consumption”,      “type”: ”numeriArray”,  ...},
3.      “generation”:  {“name”: ”Generation” ,      “type”: ”numeriArray”,  ...},
4.      “genType”:     {“name”: ”Generation Type”, “type”: ”categoric”,  ...},
5.      ,
6.     ...
7.  }
```

*Excerto de Código 7 – Ficheiro ParameterSettings.json (Adaptado)*

- ***Alias.json***

Este último ficheiro, representado no Excerto de Código 8, complementa o anteriormente descrito e surge da premissa que o mesmo parâmetro poder ter designações diferentes dependendo da fonte de dados. Além disso, podem existir parâmetros que o Agregador considere irrelevantes e, portanto, devam ser descartados.

```
{
    "Generation":  "generation",
    "gen":         "generation",
    ...
    "FlexLoad1":   "@ignore"
}
```

*Excerto de Código 8 – Ficheiro Alias.json (Adaptado)*

Assim, de modo a reduzir o número de entradas no ficheiro "*ParameterSettings.json"*, aquando a importação dos dados, a aplicação recorre a este novo ficheiro para fazer a correspondência entre diferentes designações do mesmo parâmetro, como exemplificado no excerto anterior.

Este caso está exemplificado pelas entradas "*Generation*" e "gen", sendo que ambas representam o parâmetro "*generation*" no primeiro ficheiro. Por fim, é também possível ignorar o parâmetro na sua leitura através da correspondência *"@ignore"*.

**Implementação dos Parâmetros**

Uma vez definidos os ficheiros de configuração, é possível a implementação da solução descrita anteriormente. Para tal, foi utilizado o superconjunto de *JavaScript*, o *TypeScript* que, inclusive, mune o programador do acesso a interfaces e classes abstratas, permitindo uma implementação de heranças mais linear.

- **Classes de Parâmetros**

Deste modo, foi criada a classe abstrata *Parameter*, como apresentada no Excerto de Código 9, que define o comportamento de um parâmetro. Posteriormente, esta classe foi estendida pelos diferentes tipos de parâmetros, como exemplificado na Figura 54.

```
1.  export default abstract class Parameter {
2.      name:string
3.      units:string
4.      isPeriodDependent:boolean
5.      useScientific:boolean
6.      chartTypes:string[]
7.
8.      constructor(name:string, unitName:string,isPeriodDependent:Boolean
9.                                  ,chartTypes:string[],useScientific:boolean){
10.          //Inicialização
11.      }
12.
13.      abstract isArray():boolean
14.      abstract type():string
15.      abstract get():any
16.      abstract set(value:any):boolean
17.
18.      getName(){  return this.name }
19.      getUnits(){  return this.units }
20.   }
```

*Excerto de Código 9 – Implementação da classe abstrata Parameter, em TypeScript*

A superclasse abstrata *Parameter* contém os atributos comuns a todos os parâmetros, nomeadamente, o nome, unidades a utilizar, dependência de período, se deve ser apresentado em notação científica e os tipos de gráfico a utilizar. Além disso, requer a implementação dos métodos abstratos que distinguem se o parâmetro é vetorial, a obtenção do seu tipo e métodos de definição e obtenção do valor do parâmetro.

A utilização de uma classe abstrata para caracterizar um parâmetro está em concordância com princípios GRASP (*Low Coupling*, *High Cohesion*, *Polymorphism* e *Protected Variations)* e SOLID (*Single-Responsibility* e *Open-closed*), uma vez que as subclasses são responsáveis pela implementação de variação ao normal comportamento da superclasse.

- **Padrão Factory**

Com o intuito de generalizar a criação dinâmica dos parâmetros, foi, ainda, criada a classe *ParameterFactory* que, como o próprio nome indica, é uma implementação do padrão *Factory,* representada na Figura 54.

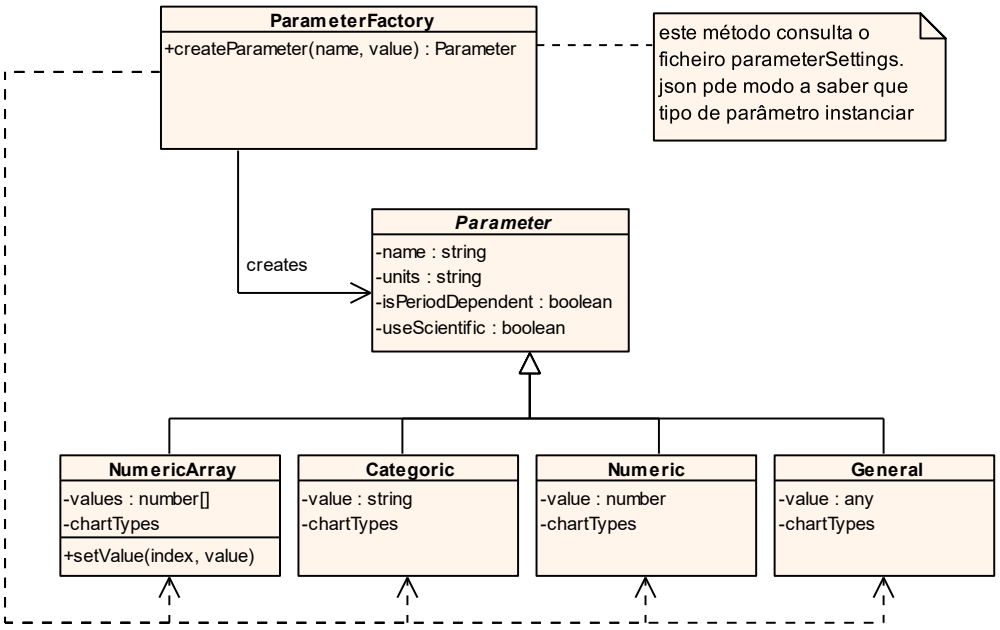



*Figura 54 – Diagrama de classes referente à implementação do padrão Factory (nível de código)*

A implementação do padrão *Factory* surge através do princípio GRASP *Pure Fabrication*, de modo a manter a responsabilidade da criação e diferentes objetos isolada para garantir a alta coesão e baixo acoplamento. Em combinação com a classe abstrata *Parameter*, toda a interação com os parâmetros é totalmente encapsulada, reduzindo as dependências na restante aplicação.

### 5.1.8 Interpretação dos Parâmetros

A criação e classificação dinâmica de parâmetros abordada na secção anterior tem como principal objetivo habilitar a sua interpretação pela interface. Deste modo, nesta secção será documentado o processamento desses dados pela interface.

#### Descrição das configurações

O ficheiro de configuração dos parâmetros apresentado no Excerto de Código 10 permite definir como é que a plataforma deve interpretar cada parâmetro, nomeadamente:

- "***name***": texto de apresentação do parâmetro na interface
- "***units***": tipo de unidades que o parâmetro representa
- "***type***": tipo de parâmetro (numérico, vetor numérico ou categórico)
- "***scientific***": define se o(s) valor(es) deve(m) ser apresentado(s) em notação científica
- "***isPeriodDependent***": define se o parâmetro é variável consoante o período.

```
{
    “generation”:
     {
        “name”:              "Generation",
        “units”:             "energy",
        “type”:              "numericArray",
        “isPeriodDependent”: true,
        “scientific”:        true
     },
    “genType”:
     {
        “name”:              "Generation Type",
        “units”:             "percentage",
        “type”:              "categoric",
        “isPeriodDependent”: false,
        “scientific”:        false
     },
   ...
}
```

*Excerto de Código 10 – Ficheiro ParameterSettings.json detalhado (Adaptado)*

No exemplo anterior é apenas discriminada a definição de dois parâmetros: a geração e o tipo de geração, sendo a primeira do tipo vetor numérico e variável com o período e a segunda um parâmetro categórico.

**Transformação em objetos**

Através da implementação do padrão *Factory* descrito em 5.1.2, a instanciação de um parâmetro pode ser feita apenas através do seu nome e valor inicial. Esta abstração garante que o parâmetro é sempre mapeado para a subclasse de tipo de parâmetro correta.

**Flexibilidade de configuração**

As variadas possibilidades de configurações tornam a plataforma flexível e preparada para futuras alterações. Com o intuito de demonstrar essa versatilidade, são de seguida apresentados os dois exemplos (Figura 55 e Figura 56) configurados no ficheiro do Excerto de Código 10.

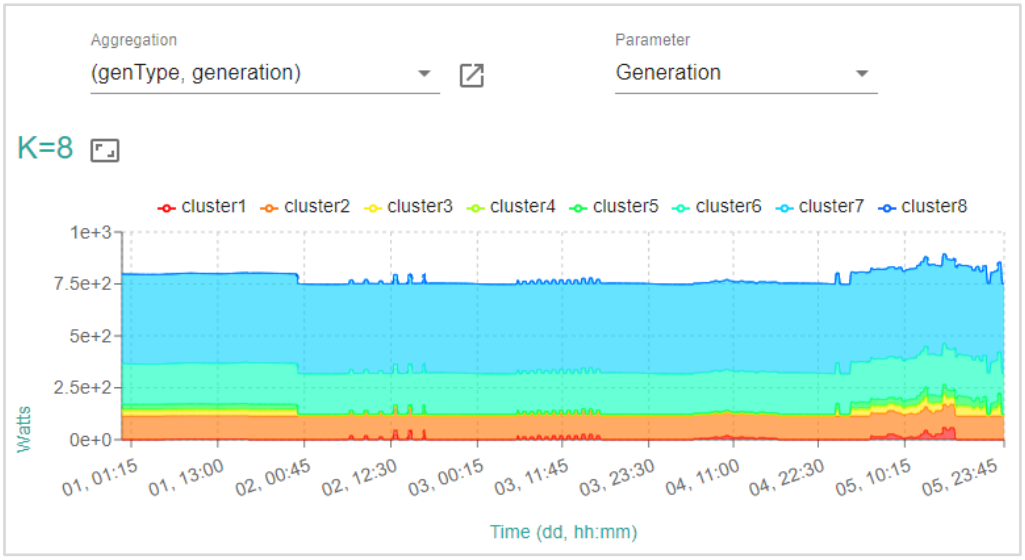



*Figura 55 – Gráfico interativo de avaliação da agregação para 8 grupos, parâmetro geração*

A Figura 55 demonstra o gráfico gerado para o parâmetro geração no contexto de uma agregação. Isto é, representa a geração média de cada grupo (*cluster*). É possível verificar que tanto as características definidas para o parâmetro geração como as inerentes ao seu tipo de parâmetro foram consideradas na criação do gráfico:

- é um vetor numérico, pelo que a sua representação corresponderá a um gráfico de áreas acumulado;
- é dependente do período, sendo representado o período no eixo horizontal;
- o tipo da unidade, energia, corresponde à unidade *Watts*, legendando o eixo vertical;
- os valores do eixo vertical são apresentados em notação científica.

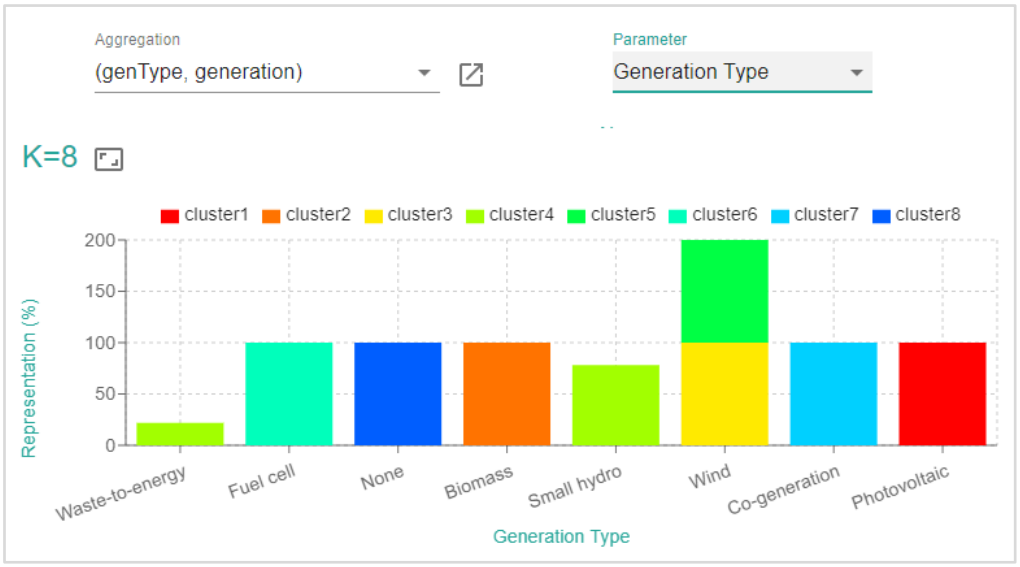



*Figura 56 – Gráfico interativo de avaliação da agregação para 8 grupos, parâmetro geração*

Seguindo o mesmo processo de configuração, para parâmetros categóricos, o resultado é bastante distinto. O gráfico apresentado na Figura 56 é proveniente do mesmo contexto que anterior. Porém, em vez de geração, discrimina a representação de cada tipo de geração de energia por grupo (*cluster*). Deste modo, é também possível verificar que as configurações definidas são refletidas no gráfico:

- o parâmetro é categórico, sendo representado através de um gráfico de barras;
- sendo categórico e independente do período, a legenda horizontal corresponde aos tipos de geração;
- o tipo de unidade corresponde à representação, legendando o eixo vertical;
- os valores do eixo vertical não são apresentados em notação científica.

### 5.1.9 Visualização do *dataset*

Na interface gráfica da plataforma existem duas divisões que permitem a visualização inalterada do *dataset* – *Player* e *Aggregator*, permitindo a visualização de dados relativos a apenas um participante ou ao conjunto, respetivamente.

#### Abas dinâmicas

Para abas que apenas representem gráficos de parâmetros dependentes do período do tipo vetor numérico, é possível customizar o conteúdo apresentado através de ficheiro de configuração como o apresentado no Excerto de Código 11.

```
1.  {"tabs": [
2.          {
3.              "name": "Overview",
4.              "charts": [
5.                  {
6.                      "name": "Energy",
7.                      "type": "normal",
8.                      "params": {
9.                          "consumption": {},
10.                         "reduction"  : {},
11.                         "generation" : {}
12.                     }
13.                 },
14.                 {
15.                     "name": "Cost / Profit",
16.                     "type": "normal",
17.                     "params" : {
18.                         "consEnergyPrice": {
19.                             "multiplyBy" : "consumption"
20.                         },
21.                         "genEnergyPrice" : {
22.                             "multiplyBy" : "generation"
23.                         }
24.                     }
25.                 }]},
26.         {
27.             "name" : "Generation",
28.             "charts" : [
29.                 {
30.                     "name" : "Energy",
31.                     "type" : "filtered",
32.                     "param" : "generation",
33.                     "filterBy" : "genType"
34.                 },
35.                 {
36.                     "name" : "Profit",
37.                     "type" : "filtered",
38.                     "param" : "genEnergyPrice",
39.                     "filterBy" : "genType",
40.                     "multiplyBy" : "generation"
41.                 }]},
42.         ...
43.  ] }
```

*Excerto de Código 11 – Ficheiro AggregatorTabs.json (Adaptado)*

O ficheiro aceita duas configurações para a formação de gráficos:

- "***normal***": cada linha do gráfico corresponde à soma vetorial de um parâmetro. Podem ser explícitos vários parâmetros no atributo "*params*" (linhas 8 e 17). Estes parâmetros têm de ser do tipo vetor numérico;
- "***filtered***": o gráfico corresponde a apenas um parâmetro do tipo vetor numérico - o atributo "*param*" (linhas 32 e 38). Este é filtrado por um parâmetro categórico (linhas 33 e 39) e cada linha do gráfico irá corresponder à soma vetorial do parâmetro do "*param*" para uma categoria.

A soma vetorial corresponde aos totais, para cada índice, de um parâmetro do tipo vetor numérico para um conjunto de participantes. Além disso, a configuração permite ainda multiplicar o parâmetro por outro parâmetro vetorial (linhas 19, 22 e 40). O exemplo da aba gerada pelo ficheiro é apresentado abaixo, na divisão do Agregador.

**Interpretação do Ficheiro**

Considerando que a responsabilidade da interface gráfica é apresentar dados e receber os *inputs* do Agregador, a lógica de interpretação do ficheiro e cálculo dos valores a mostrar nos gráficos é encapsulada pelo componente *UI Utils*, como representado no diagrama de sequência da Figura 57.

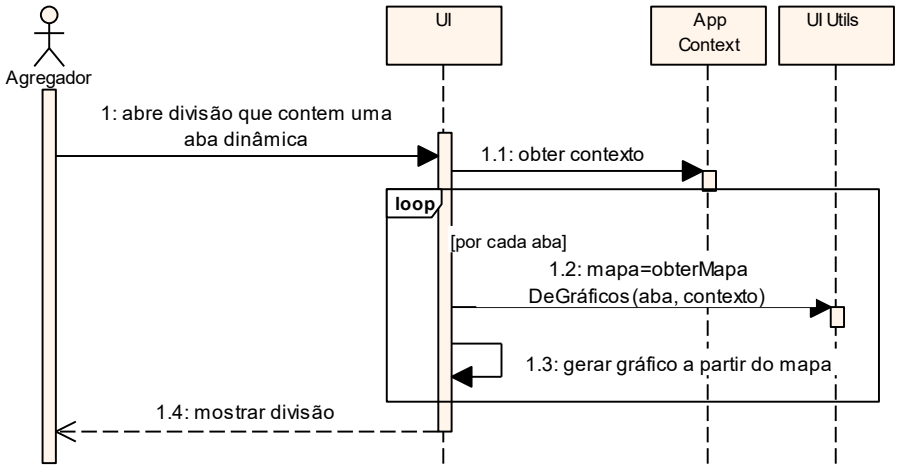



*Figura 57 – Diagrama de sequência do processamento das abas dinâmicas (nível de componentes)*

Na abertura de uma divisão que contém uma aba dinâmica, a UI interpreta o ficheiro correspondente, e, por cada aba, recorre às suas classes utilitárias para a geração de mapas que são interpretados pelos componentes da UI para a formação de gráficos.

**Divisão Agregador**

Nesta divisão são apresentados os dados cumulativos de todos os participantes do *dataset*. A divisão é totalmente gerada pelo ficheiro representado no Excerto de Código 11. Especificamente, a Figura 58 e a Figura 59 são a concretização das duas abas visíveis no excerto.

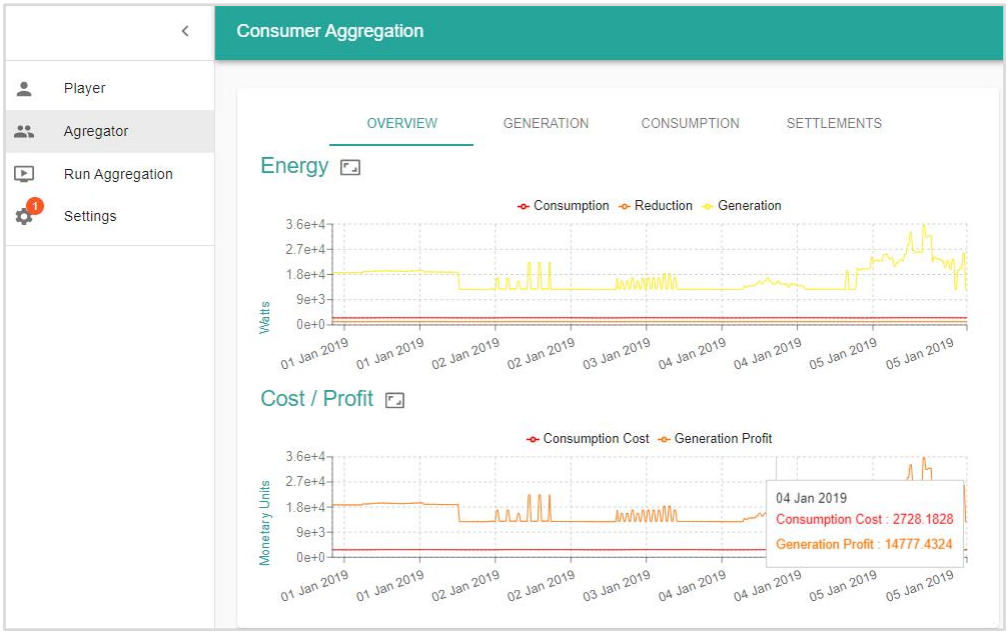



*Figura 58 – Interface gráfica da plataforma, divisão Agregador, aba Overview*

A primeira aba, "*Overview*", é constituída por dois gráficos normais. O primeiro apresenta o consumo, redução do consumo e geração ao longo do período. Por sua vez, o segundo apenas contém o custo do consumo e o lucro da geração, sendo multiplicado o preço da energia consumida e gerada pelo consumo e geração, respetivamente.

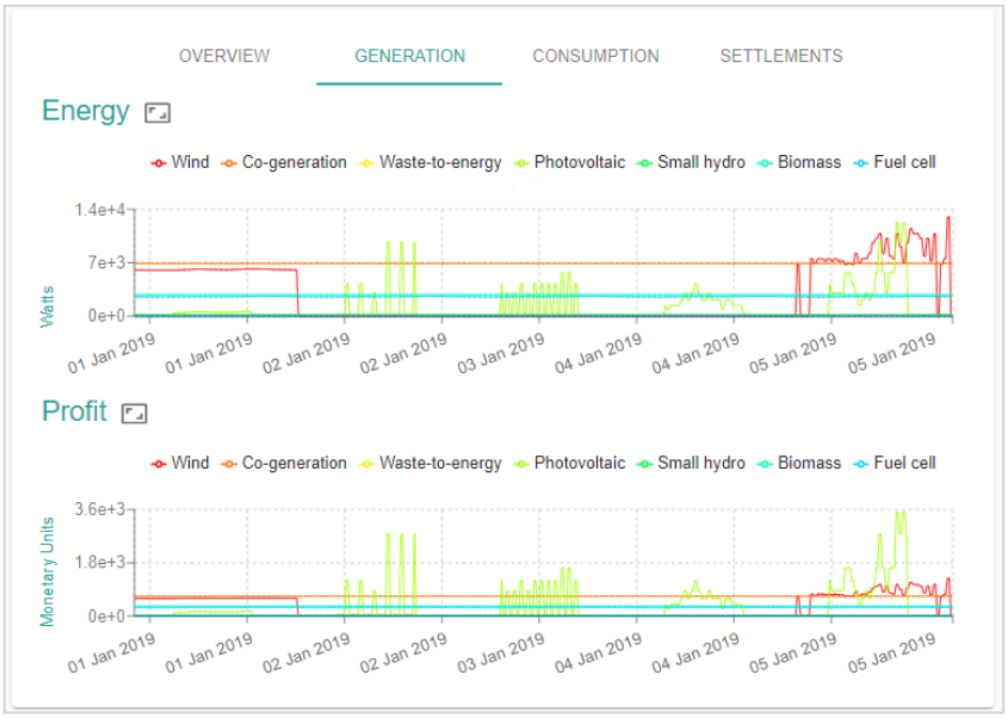



*Figura 59 – Interface gráfica da plataforma, divisão Agregador, aba Generation*

Por outro lado, a segunda aba, "*Generation*", tira partido da opção de filtragem, apresentando os resultados separados por tipo de geração. Deste modo, o primeiro gráfico evidencia a geração acumulada de cada tipo de geração, durante o período. De forma semelhante, o segundo gráfico apresenta o lucro gerado por cada tipo de geração, durante o período. Além disso, para a obtenção do lucro, recorreu-se uma vez mais à opção de multiplicar o preço da energia pela geração.

### Divisão *Player*

Nesta divisão são representados todos os dados relativos a um participante, distinguindo-os entre dados contratuais e dados históricos representados na Figura 60 e Figura 61, respetivamernte.

- **Dados Contratuais**

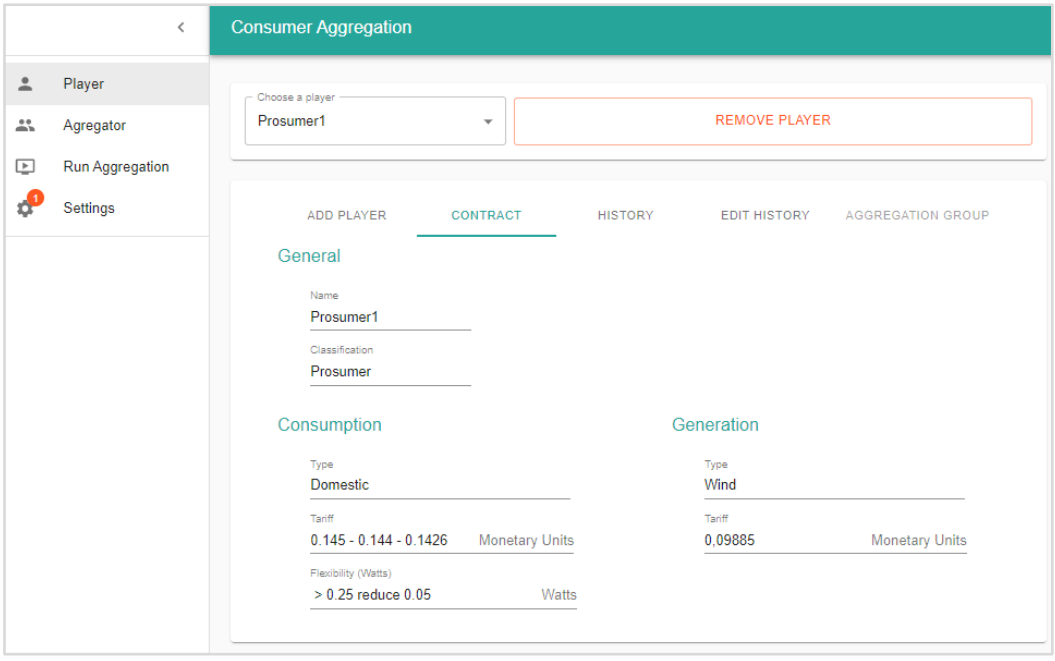



*Figura 60 – Interface gráfica da plataforma, divisão Player, aba contratual*

A aba dos dados contratuais, apresentada na Figura 60, é composta por, sobretudo, parâmetros categóricos e numéricos. Porém, esta secção é excecionalmente definida estaticamente. Isto deve-se à inconsistência da representação dos seus parâmetros, nomeadamente as tarifas e flexibilidade de consumo.

- **Dados Históricos**

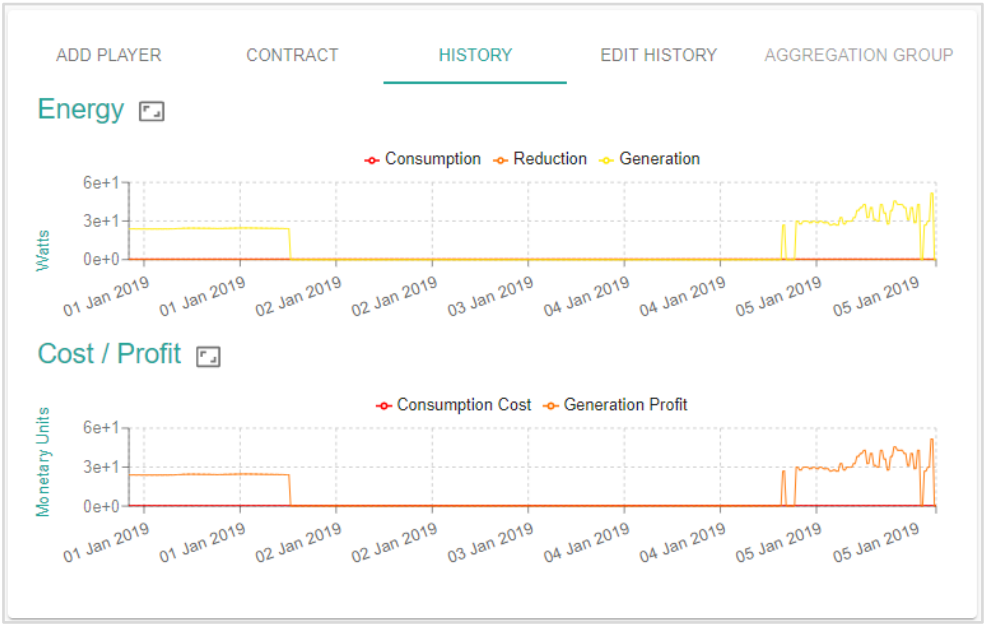



*Figura 61 – Interface gráfica da plataforma, divisão Player, aba do histórico*

A aba dos dados históricos representa as medições feitas para aquele participante durante um período através de gráficos interativos. Além disso, é uma aba dinâmica, sendo totalmente configurável. O exemplo da Figura 61 está apenas configurado para desenhar dois gráficos.

### *5.1.10* Edição do *dataset*

A edição de datasets engloba três operações distintas: a alteração de dados de um participante, a adição de novos participantes ao *dataset* e, por fim, a remoção de participantes. Assim, nesta secção serão analisadas e exploradas essas componentes, discriminando, primeiramente o fluxo das operações através de diagramas de sequência e, por fim, a apresentação da funcionalidade na interface.

#### Alteração de dados de um participante

A edição de um *dataset* apenas pode ser realizada ao nível individual, na aba contratual e na aba de edição do histórico. Não obstante, este processo segue sempre a mesma estrutura, definida no diagrama da Figura 62.

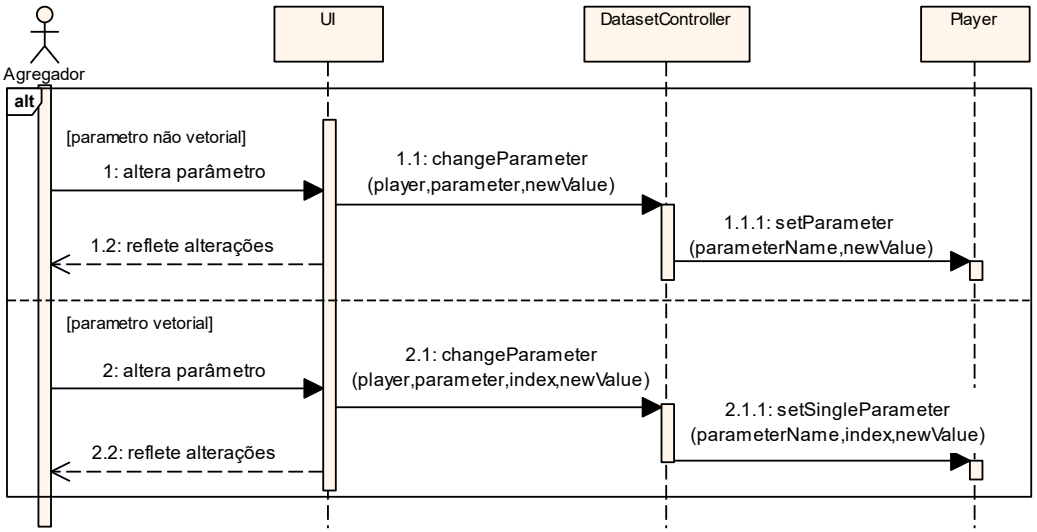



*Figura 62 – Diagrama de sequência da alteração de um parâmetro (nível de código)*

O processo da alteração de um parâmetro é bastante linear: a UI recebe a alteração e encaminha-a para o controlador de *datasets*. Este processa essa alteração, alterando o domínio, refletindo essa alteração na UI. Embora o processo de alteração seja semelhante, as interfaces para a alteração de parâmetros contratuais e históricos são bastante diferentes, estando contrastadas nas figuras Figura 63 e Figura 64.

- **Dados Contratuais**

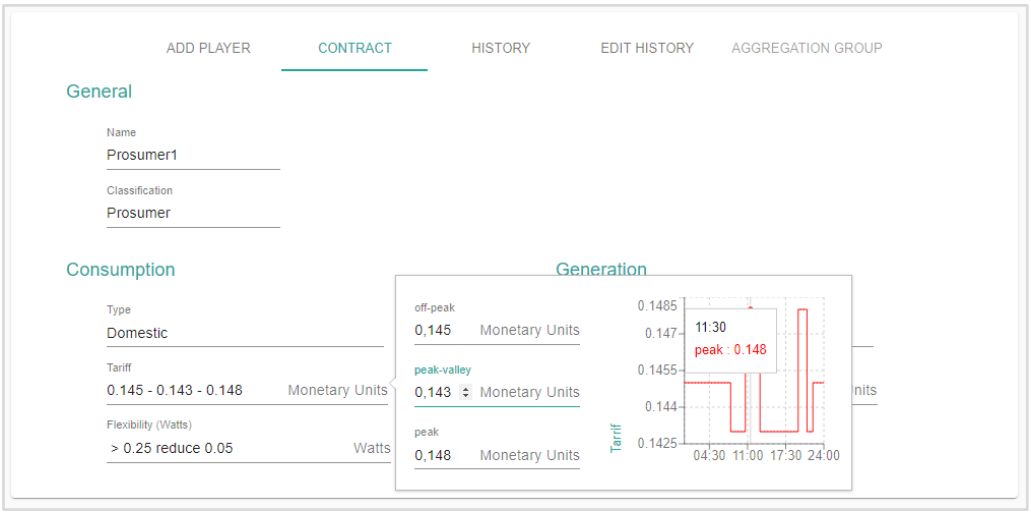



*Figura 63 – Alteração da tarifa de consumo*

A alteração dos dados contratuais é realizada através da alteração dos conteúdos dos respetivos campos. Porém, a classificação do participante não pode ser diretamente alterada, uma vez que é-lhe atribuída consoante o seu consumo e geração. Além disso, é de realçar que a alteração das tarifas de consumo é acompanhada de um gráfico interativo que permite visualizar o seu impacto ao longo do dia.

- **Dados Históricos**

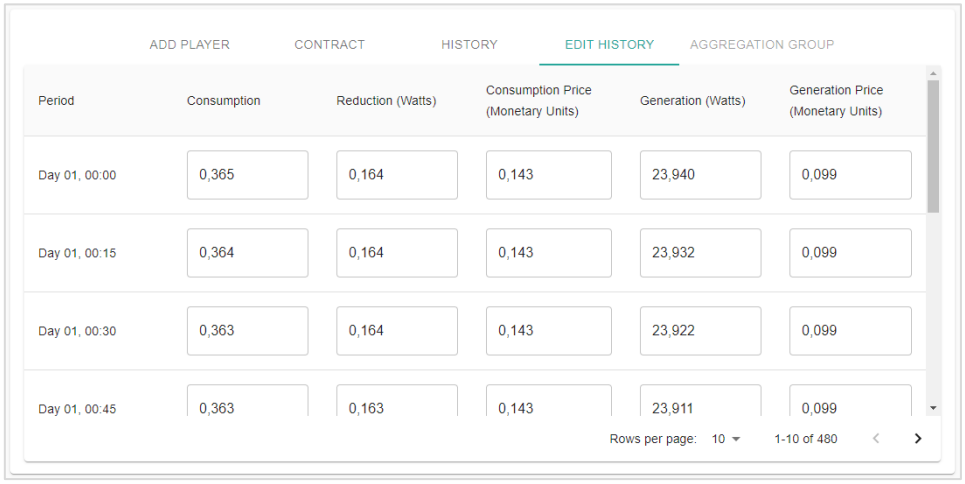



*Figura 64 – Alteração de dados históricos*

De modo semelhante, a alteração de dados históricos é realizada através da alteração do campo respetivo na interface gráfica. Porém, neste caso, esses campos representados numa tabela dinamicamente criada com todos os parâmetros vetoriais variáveis consoante o período.

### Adição de participante ao *dataset*

A adição de um participante ao *dataset* é realizada na aba "*Add Player*". Através da interface do lado esquerdo da Figura 65 é possível a importação de um ficheiro *excel* seguindo o mesmo processo do carregamento de um *dataset* por ficheiro. Porém, neste caso apenas será carregado o primeiro participante encontrado. Após a sua importação a interface é substituída pela do lado direito, que visa a confirmação da adição ou o descartar do novo participante. Mesmo antes da confirmação já é possível consultar e alterar os dados do participante.

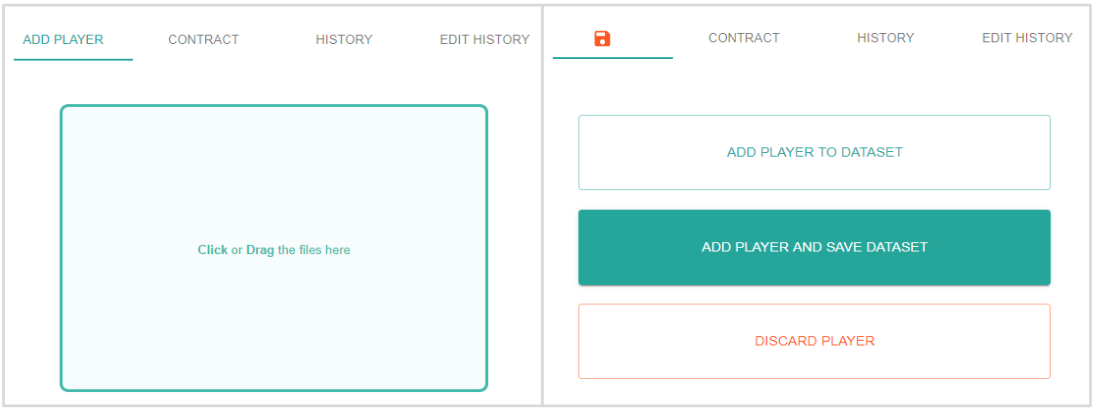



*Figura 65 – Aba de adição de participante antes (esquerda) e depois (direita) do carregamento de um ficheiro*

Como referido a confirmação da adição é realizada na interface do lado direito da Figura 65. Esta permite adicionar o participante ao *dataset*, adicioná-lo e guardar o *dataset* imediatamente ou descartar o participante, voltando à interface inicial. Sendo a segunda opção a mais complexa, esta está discriminada na Figura 66.

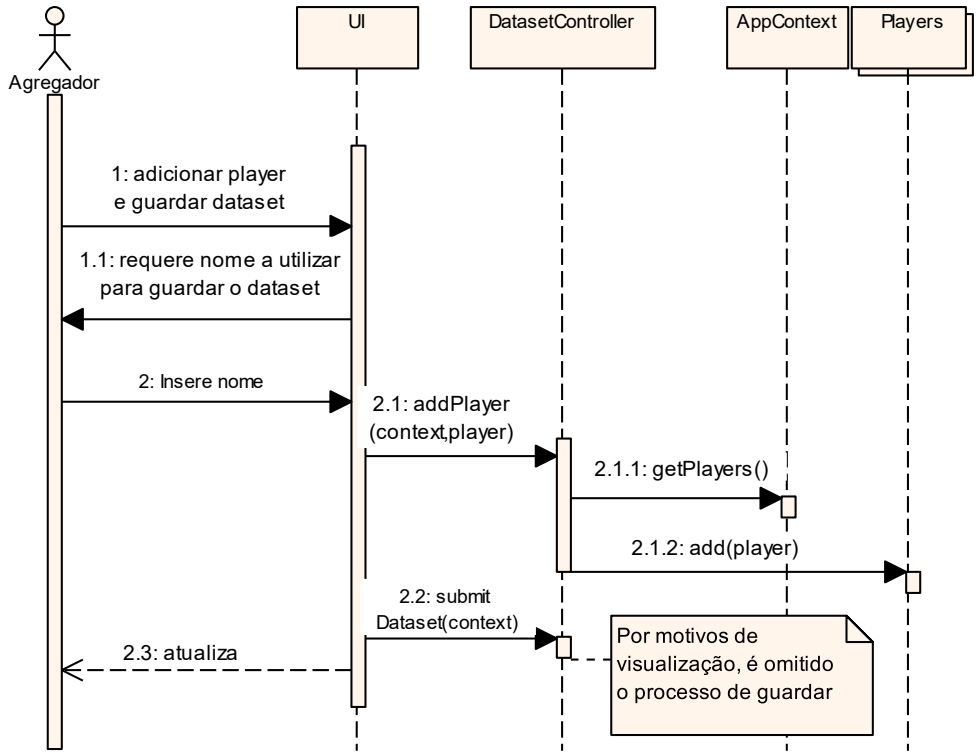



*Figura 66 – Diagrama de sequência do processo de adicionar um participante ao dataset e guardá-lo (nível de código)*

O diagrama descreve parcialmente o processo de adição de um participante ao *dataset*, seguido da sua persistência. Este processo é referido na secção da Gestão de *datasets*, pelo que foi omitido no diagrama. Quando o Agregador seleciona a opção de adicionar o participante e persistir o *dataset*, é-lhe requerido o nome a utilizar através de um *popup*. Uma vez que o nome tenha sido definido, é então adicionado o participante ao *dataset*. Por fim, é desencadeado o processo já definido na Figura 51, de guardar o *dataset*.

**Remoção de um participante**

A remoção de um participante é uma ação não reversível e que, por esse motivo, necessita de uma confirmação extra que avisa o Agregador desse facto, evitando, também, remoções acidentais. A Figura 67 representa parte da interface gráfica da divisão "*Player*" que é responsável por essa operação.

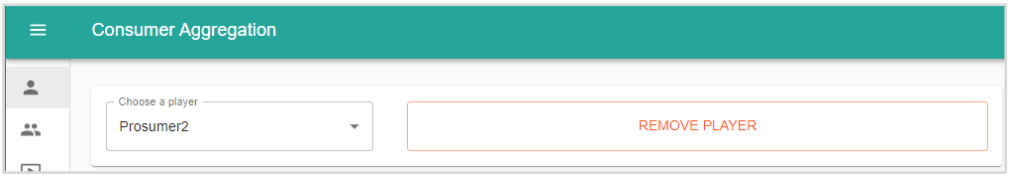



*Figura 67 – Botão de remoção de participante*

Esta operação é iniciada ao carregar no botão "*Remove Player*" apresentada acima, seguindo-se da confirmação da operação através de uma mensagem de confirmação. Este processo é descrito na Figura 68.

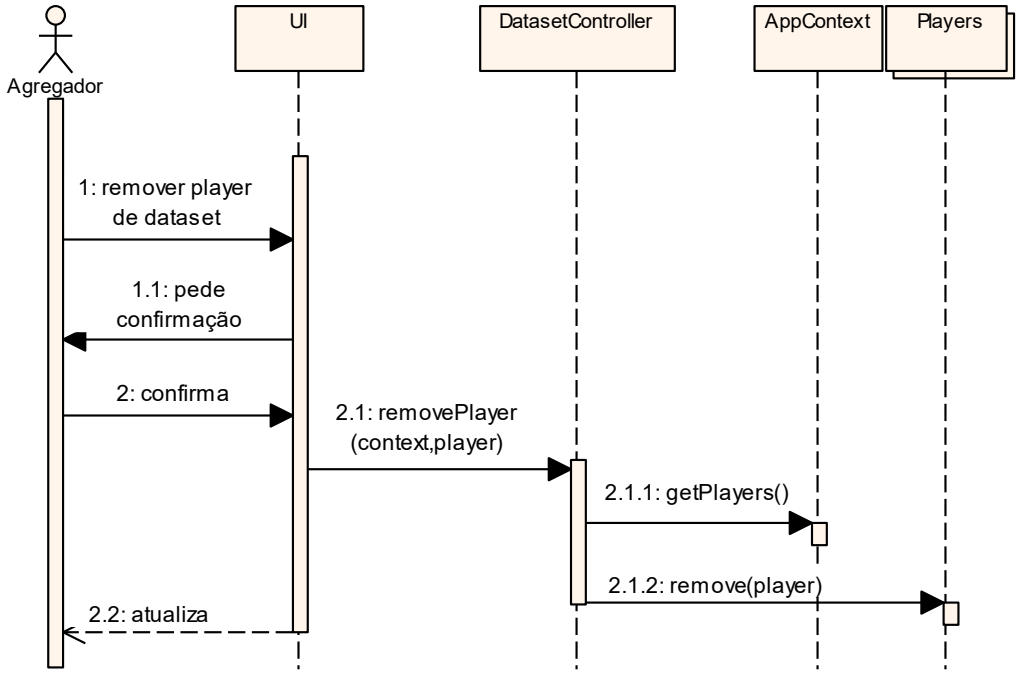



*Figura 68 – Diagrama de sequência do processo de remoção de um participante (nível de código)*

Após o botão de remoção ser carregado, como referido, é necessária a confirmação da ação, que corresponde ao passo 1.1 do diagrama. Uma vez confirmada a ação, o participante é removido do *dataset* e a página reflete a alteração.

### 5.1.11 Agregação de participantes

A agregação de participantes é um dos pontos principais da aplicação. Como referido na secção 3.1.1, esta é realizada através de pedidos que especificam os parâmetros e intervalo de número de grupos a serem testados. Deste modo, esta secção aborda primeiramente os algoritmos utilizados para a agregação e avaliação da mesma. Em segundo lugar, é detalhada a criação dos pedidos de agregação, seguindo-se do seu processamento, isto é, da criação das tarefas necessárias para completar esse pedido. Depois é abordado o funcionamento

assíncrono das tarefas geradas, e dissecado o rastreamento de tarefas. Por fim, são detalhados os processos de download e visualização dos resultados da agregação.

**Algoritmos de agregação e avaliação**

Como descrito na secção 2.2.4, foi utilizado o *Centroid-Based Clustering*, ou *K-Means*, para a agregação de participantes. Além disso, foram utilizados os métodos de avaliação *Elbow* e *Silhoette*. Inicialmente, era utilizado o método "*fviz_nbclust*" da biblioteca "*factoextra*", que fazia a agregação e a avaliação automaticamente, mas devido a problemas de desempenho, não só a nível de tempo de execução, como a nível de memória, foram procuradas alternativas mais eficientes. Deste modo, foi utilizado o método "*kmeans*" da biblioteca "*cluster*" para a criação dos grupos como apresentado no Excerto de Código 12.

```
1.  aggregation <- function( dataset, #row (player) x col (periods)
2.                           kmin, #numero mínimo de grupos
3.                           kmax  #numero máximo de grupos
4.  )
5.  {
6.    res<-tryCatch(
7.      {
8.        library(cluster)
9.        listk<-list()
10.       for (k in kmin:kmax) {
11.         namek<-paste0("k=",k)
12.         listk[[namek]]<-kmeans(dataset, k,nstart=25,algorithm=c("Lloyd"))
13.       }
14.       res<-list()
15.       res$dis<-dist(dataset)^2
16.       res$ks<-listk
17.       return (list(result=res,error=NULL))
18.       # for the condition handlers for warnings and error below)
19.     },
20.     error=function(cond) {
21.       error=cond
22.       return(list(result=NULL,error=cond$message))
23.     }
24.     )
25.     return(res)
26. }
```

*Excerto de Código 12 – Implementação do algoritmo k-means*

O algoritmo de agregação tem como parâmetros o *dataset*, o número mínimo de grupos a gerar, *kmin* e o número máximo de grupos a gerar, *kmax*. Deste modo, por cada número de grupos a gerar, *k*, é executado o método "kmeans" e são guardados os seus resultados numa lista (linha 12). É de realçar que este é um algoritmo semi-aleatório, pelo que é executado diversas vezes e escolhida a melhor execução. O método *kmeans* permite definir quantas vezes é executado a partir do parâmetro "*nstart*". Além disso, é também guardado no

resultado o quadrado das distâncias, para utilizar na avaliação *Silhouette* que se seguirá. Por fim, é retornado o resultado ou, em caso de erro, a mensagem gerada.

A avaliação dos grupos através dos métodos descritos no Excerto de Código 13 e Excerto de Código 14 é constituída por três passos principais: a obtenção da medida de avaliação para cada *k*, a sua adição ordenada a um vetor (para possibilitar o desenho de gráficos) e a computação do seu *k* ideal.

```
1.  library (vegan)
2.
3.  silhouetteWidth<- function(node, dis){
4.    sil<-silhouette(node,dis)
5.
6.    if(is.na(sil)){
7.      return (0)
8.    }
9.    return (summary(sil)$avg.width)
10. }
11.
12.
13. silhouetteEvaluation<-function(res){
14.   result<-list()
15.   result$k<-0
16.   result$array<-list()
17.   highestValue <- -1
18.   for (k in res$ks){
19.     if(nrow(k$centers)>1){
20.       tempW<- silhouetteWidth(k$cluster,res$dis)
21.       if(tempW>highestValue){
22.         highestValue=tempW
23.         result$k=nrow(k$centers)
24.       }
25.       result$array[[nrow(k$centers)]]<-tempW
26.     }else{
27.       result$array[[1]]<-0
28.     }
29.   }
30.   return(result)
31. }
32.
```

*Excerto de Código 13 – Avaliação Silhouette*

A avaliação *Silhouette* é descrita no Excerto de Código 13 e consiste na obtenção do *Silhouette Score* (SS), através do método "*silhouette*" da biblioteca "*cluster*" para cada *k*. Uma vez que a obtenção do SS não é um processo direto, está encapsulada na função "*silhouetteWidth*", que recebe os grupos a avaliar e a matriz de disparidade por parâmetro e devolve a computação do SS. De seguida, esses valores são colocados num vetor. É por fim avaliado qual foi o *k* com SS mais elevado, sendo esse registado como o *k* ótimo segundo a avaliação *Silhouette*.

```
elbowEvaluation<- function(res){
    result<-list()
    result$k<-0
    result$array<-list()  #lista de tot withniss(TW) desse k
    delta1<-list()        #lista da diferença(Delta1): TW(k)-TW(k-1)
    delta2<-list()        #lista da diferença(Delta2): D(k) - D(k-1)

    i<-0
    for(k in res$ks){
      i<-i+1
      result$array[[i]]<-k$tot.withinss
      if(i>1){
        delta1[[i]]<-result$array[[i]]-result$array[[i-1]]
        if(i>2){
          delta2[[i]]<-delta1[[i]]-delta1[[i-1]]
        }else{
          delta2[[i]]<-0
        }
      }else{
        delta1[[i]]<-0
        delta2[[i]]<-0
      }
    }
    i<-0
    maxDP<-0
    maxDP_k<-0
    for(k in res$ks){
      i<-i+1
      #como delta1 é negativo, soma-se em vez de subtrair
      delta2_1<-delta2[[i]]+delta1[[i]]
      if(delta2_1>0){
        delta2_1Pesado<-delta2_1/nrow(k$centers)
        if(maxDP<delta2_1Pesado){
          maxDP<-delta2_1Pesado
          result$k<-nrow(k$centers)-1
        }
      }
    }
    return (result)
}
```

*Excerto de Código 14 – Avaliação Elbow*

Por outro lado, a avaliação *Elbow* descrita no Excerto de Código 14, obtém a *Within Sum of Square* (WSS) de cada grupo diretamente dos resultados do método "*kmeans*" e ordena-as num vetor. Porém, a obtenção do *k* ideal através do método *Elbow* tende a ser um processo visual, onde é encontrado o "cotovelo" do gráfico. Deste modo, foi necessária a pesquisa da sua automatização, tendo sido implementado o algoritmo descrito em [109]. O algoritmo calcula a diferença entre o WSS de um *k* e o WSS do seu antecessor, guardando-a na lista "*delta1*". De seguida, compara a diferença entre "*delta1*" de um *k* e o "*delta1*" do seu antecessor, registando-a na lista "delta2". Depois são somados os vetores "*delta1*" e "*delta2*" e a cada elemento é aplicada a divisão pelo seu *k*. O *k* que obtiver o resultado mais elevado será considerado o ideal segundo o método de avaliação *Elbow*.

**Criação ou carregamento de um pedido de agregação**

O pedido de agregação pode ser realizado ou carregado na divisão de Agregação, na secção de seleção dos dados. Através da interface gráfica apresentada na Figura 69, é possível selecionar as configurações de agregação.

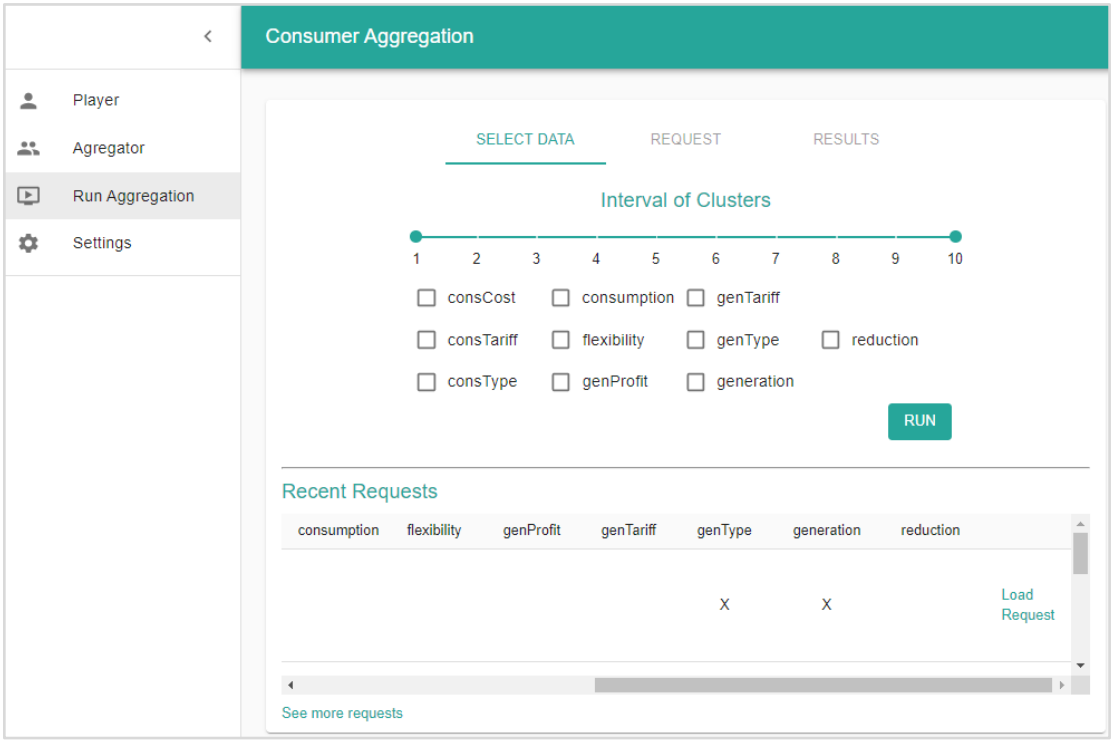



*Figura 69 – Interface gráfica da plataforma, divisão de agregação, aba "Select Data"*

No pedido de agregação, pode ser selecionado o intervalo de número de grupos a avaliar, através do *slider* abaixo de "*Interval of Clusters*", neste exemplo apenas está configurado para permitir de 1 a 10 grupos. Além disso, podem ser selecionados os parâmetros a considerar para a agregação, a partir dos apresentados. A plataforma apenas mostra os parâmetros relevantes ao *dataset* atual.

**Processamento de um pedido de agregação**

Ao carregar no botão "*Run*" da Figura 69 é desencadeado o processo representado na Figura 25, que regista a tarefa a executar na TQ. Além disso, como referido anteriormente na secção 3.1.1, um pedido de agregação está, por norma, associado a um *dataset* pode gerar várias tarefas de agregação. Porém, o servidor também disponibiliza uma interface pública, que permite a criação de pedidos de agregação sem a persistência do *dataset*. Por este motivo, no

processamento de um pedido de agregação, aquando a criação de uma tarefa de agregação é guardado no seu diretório a matriz com os dados de execução da agregação. Deste modo, o processamento do pedido está representado pela Figura 70.

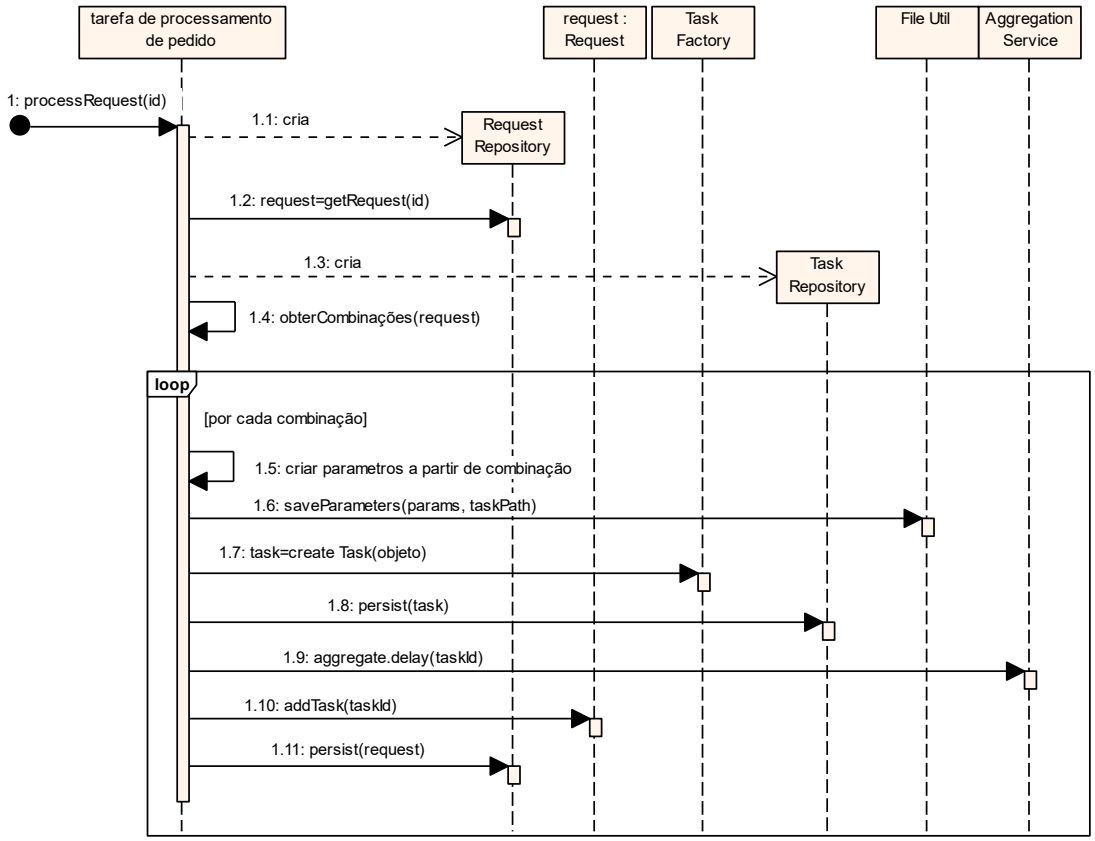



*Figura 70 – Diagrama de sequência do processamento assíncrono de um pedido de agregação (nível de código)*

O processamento de um pedido de agregação começa pela obtenção das configurações do pedido, através da sua obtenção pelo *RequestRepository*. De seguida, no passo 1.4, são obtidas todas as combinações possíveis entre os parâmetros selecionados, sendo que se o pedido não exigir combinações, esse passo apenas devolve a combinação original – os parâmetros selecionados. Por cada combinação obtida, é então calculada a matriz de execução da agregação (passo 1.5), que é persistida recorrendo à classe *FileUtils*. De seguida, a tarefa de agregação é criada e persistida com os respetivos detalhes. Por fim é adicionada à TQ no passo 1.9 e associada ao pedido de agregação nos passos 1.10 e 1.11.

Com a adição das tarefas de agregação à TQ, serão desencadeados vários outros processos assíncronos de execução de agregações.

**Tarefas de agregação**

Devido à natureza da biblioteca utilizada para aceder ao *R*, *PypeR*, é possível a geração de sessões em *R.* Deste modo, o processo de agregação pode recorrer a um ambiente de trabalho de forma síncrona e sem necessidade de adaptadores, tal como representado na Figura 71.

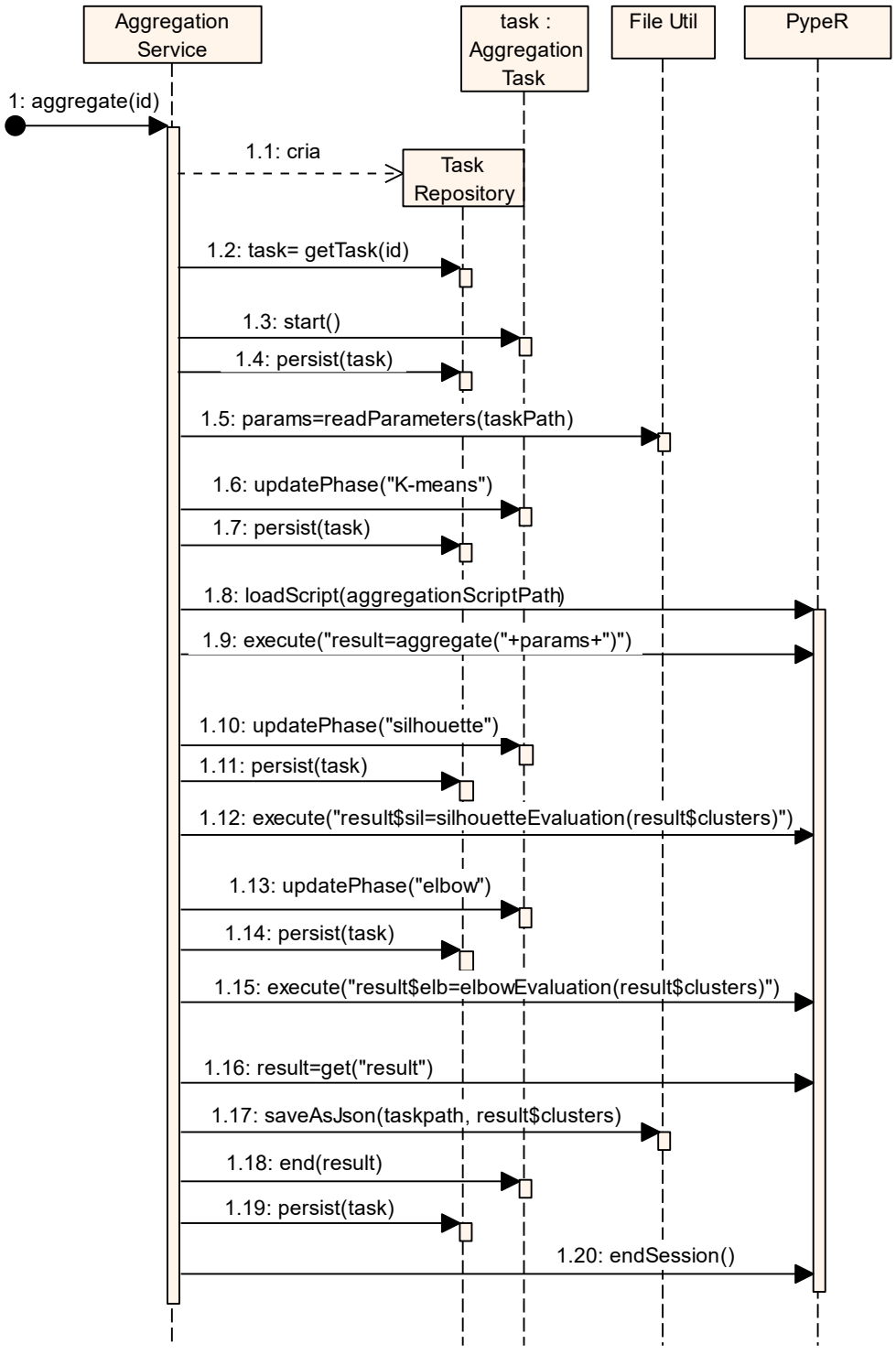



*Figura 71 – Diagrama de sequência da execução de uma tarefa de agregação (nível de código)*

O processo de agregação é dividido em várias fases, atualizando a tarefa ao iniciar cada fase. Primeiramente, é obtida a tarefa e o seu caminho, para importar o ficheiro com a matriz de agregação a utilizar (passos 1.1 a 1.5). De seguida, começa a fase de agregação. Nesta é criado o ambiente *R*, carregado o ficheiro que contém os algoritmos e executado o *K-Means*. De

seguida, é executada a avaliação de *Silhouette*, nos passos 1.10 a 1.13. Semelhantemente, segue-se a avaliação *Elbow*, nos três passos seguintes. Por fim, são gravados os resultados dos grupos, no passo 1.17, recorrendo à classe *FileUtils*, e guardados os restastes detalhes dos resultados na tarefa de agregação.

**Rastreamento do pedido de agregação**

Nesta secção já foram descritos os processos de criação de tarefas de agregação e atualização das suas fases. O procedimento para o rastreamento também já foi apresentado na Figura 33, pelo que nesta secção apenas será demonstrada a implementação desse rastreamento na interface gráfica. Na aba do pedido, é possível acompanhar os progressos de cada uma das suas tarefas, como demonstrado na Figura 72.

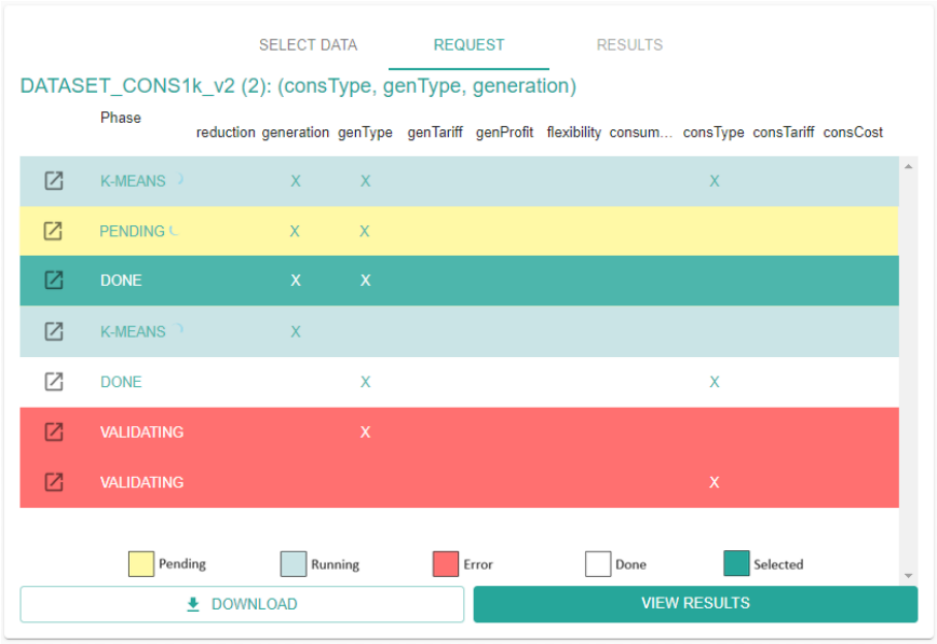



*Figura 72 – Interface gráfica da plataforma, divisão de agregação, aba do pedido*

Neste caso, existem duas tarefas em execução (a primeira e a quarta), uma tarefa pendente (a segunda), duas tarefas em que ocorreu um erro (as duas últimas) e duas tarefas que terminaram com sucesso (a terceira e a quinta), estando a primeira selecionada. Além disso, carregando no botão de *popup* de uma tarefa são revelados os seus detalhes de execução ou erro, através das interfaces da Figura 73. Após o término de uma tarefa é possível selecioná-la. Podem ser selecionadas várias tarefas ao mesmo tempo. Quando pelo menos uma tarefa está selecionada, os botões "*Download*" e "*View Results*" ativam-se, desbloqueando o acesso a essas funcionalidades.

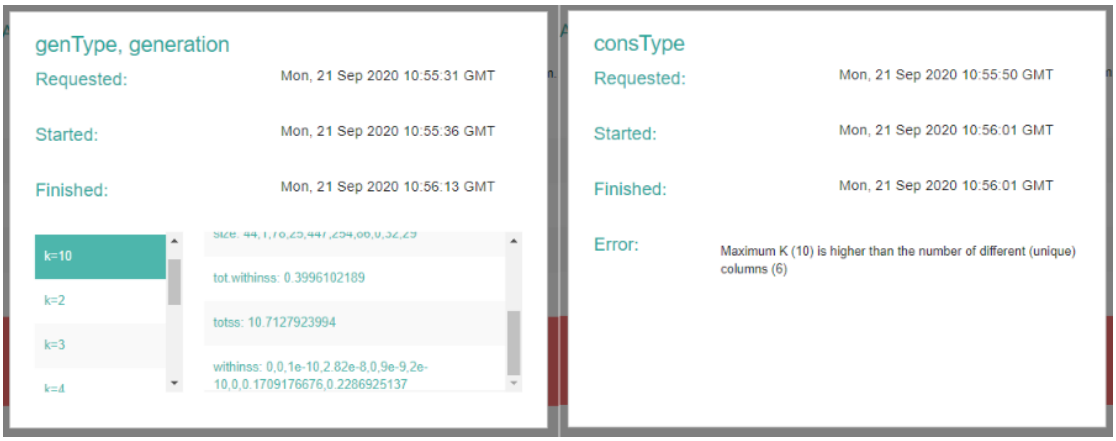



*Figura 73 – Popup de detalhes da tarefa em caso de sucesso (esquerda) e erro (direita)*

A interface de detalhes da tarefa de agregação explicita as suas datas de criação, começo e finalização. Além disso, no caso de a tarefa de agregação ter sucedido (lado esquerdo) são apresentados pormenores sobre os grupos gerados, como o WSS de cada grupo. Em caso de erro, é apresentado o erro gerado.

### ***Download* resultados**

A opção de download gera um ficheiro *excel* por cada tarefa selecionada. O ficheiro *excel* gerado pode ser consultado no Anexo A. É composto por uma folha geral, que contém dados referentes à execução, informação relativa ao *dataset*, quais os parâmetros considerados e os resultados das avaliações *Elbow* e *Silhouette*. Considerando uma tarefa de gera várias agregações, além da folha inicial, o ficheiro contém 3 folhas por cada agregação, sendo a primeira informação genérica sobre os grupos, a segunda a média dos grupos para cada coluna e a terceira o grupo atribuído a cada participante nessa agregação. O *download* de uma agregação pode ser descrito pelo diagrama da Figura 74.

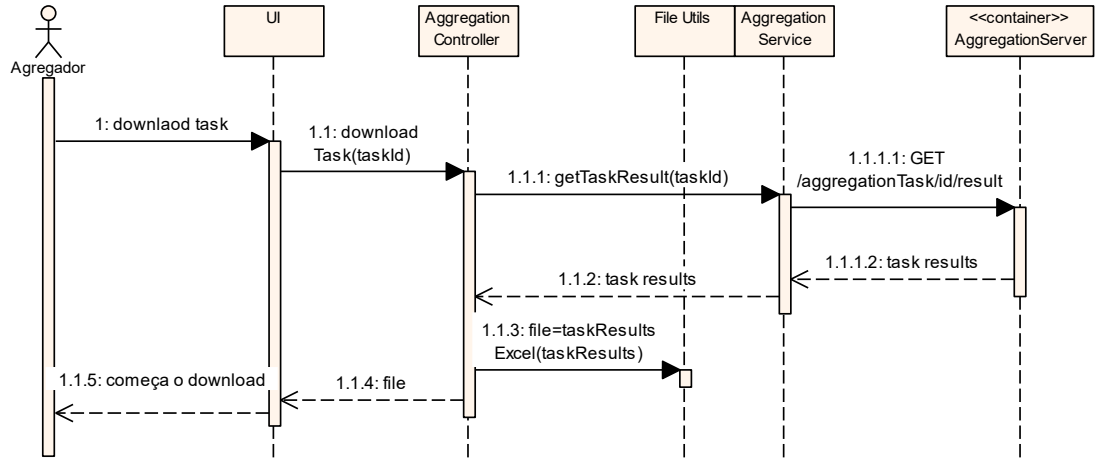



*Figura 74 – Diagrama de sequência do download de resultados de uma agregação (nível de código)*

Neste diagrama o servidor é representado ao nível de componentes, uma vez que o processo de obtenção de resultados de uma tarefa de agregação será detalhado abaixo. Deste modo, o processo de *download* começa com a obtenção dos resultados da tarefa de agregação. De seguida, o *AggregationController* serve-se da classe utilitária *FileUtils* para a criação do ficheiro acima descrito. Por fim, o ficheiro é retornado para a interface e o *download* é iniciado.

**Visualização de resultados**

Para a visualização dos resultados, é primeiramente enviado um pedido de recolha dos mesmos para o servidor, descrito na Figura 75.

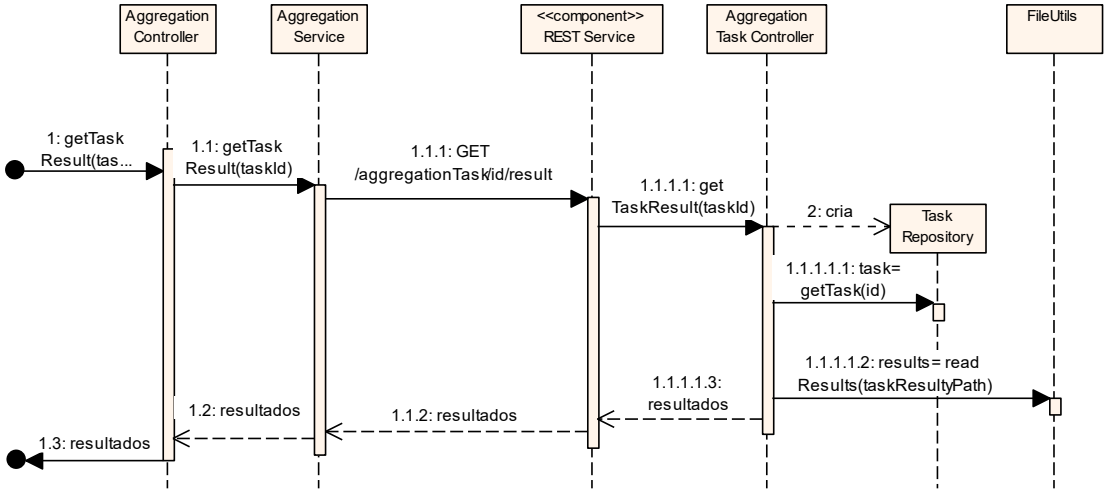



*Figura 75 – Diagrama de sequência do processo de obtenção do resultado de uma tarefa de agregação (nível de código)*

Para a obtenção do resultado de uma tarefa de agregação, em primeiro lugar, é enviado um pedido *GET* contendo o id da tarefa para o servidor, que é processado pelo *AggregationTaskController*. De seguida, é criado o repositório de tarefas e obtida a respetiva através do seu id. Por fim, recorre-se à classe utilitária *FileUtils* para a leitura do resultado guardado no diretório da tarefa. Este foi escrito no processo representado na Figura 71.

A visualização de resultados está dividida em quatro abas, representadas na Figura 76, Figura 77, Figura 78 e Figura 79, cada uma permitindo analisar aspetos diferentes da agregação.

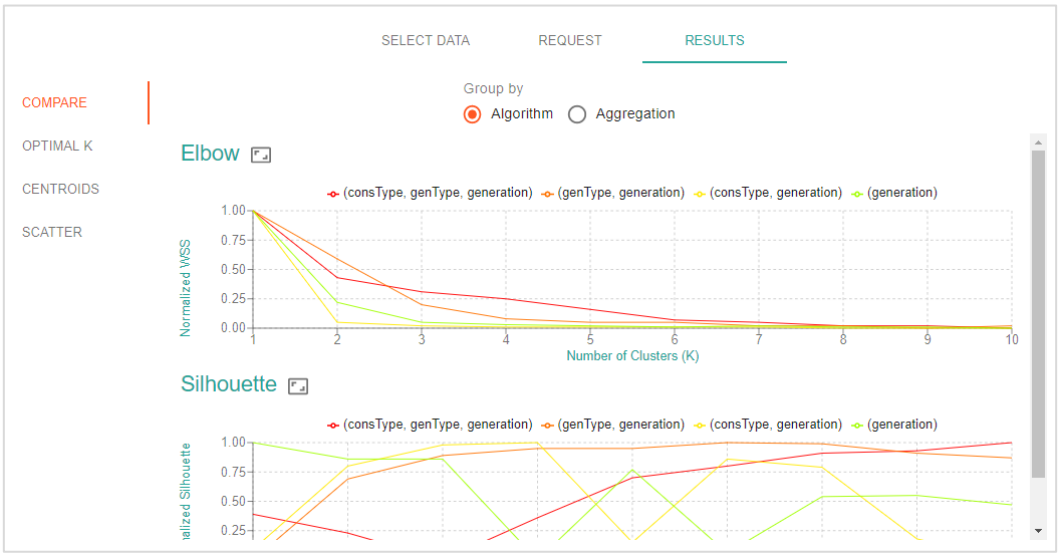
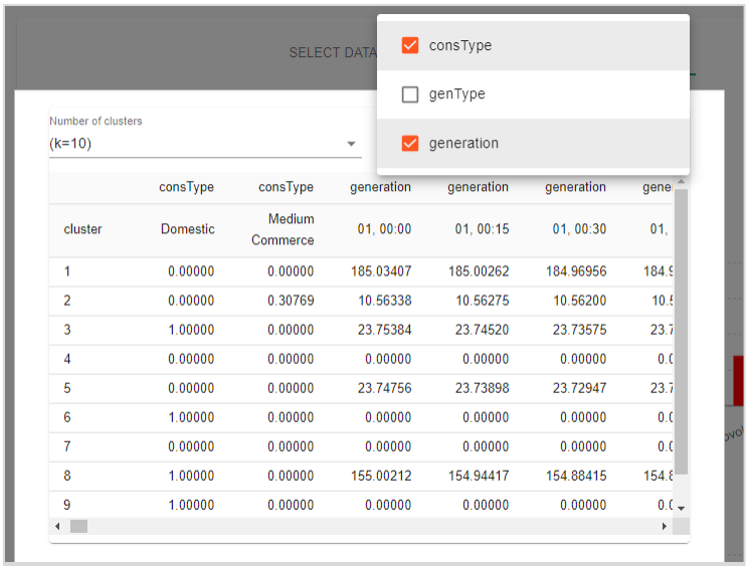



*Figura 76 – Comparação da avaliação dos grupos das tarefas por algoritmo*

A primeira aba permite comparar as avaliações de todas as tarefas selecionadas. Esta comparação pode ser realizada por algoritmo, onde é gerado um gráfico para cada método de avaliação em que cada linha representa uma tarefa. Alternativamente, é possível comparar os resultados por tarefa, sendo gerado um gráfico por cada tarefa onde cada linha representa a avaliação de um algoritmo.


*Figura 77 – Avaliação dos grupos e indicação do melhor número*

A segunda aba, "*Optimal K*" permite escolher uma das tarefas selecionadas e visualizar qual o melhor número de grupos gerado para essa agregação, segundo cada algoritmo de avaliação. É gerado um gráfico para cada método de avaliação, e explicitado qual o melhor número de grupos a utilizar.




| cluster | consType Domestic | consType Medium Commerce | generation 01, 00:00 | generation 01, 00:15 | generation 01, 00:30 | gene 01, |
|---|---|---|---|---|---|---|
| 1 | 0.00000 | 0.00000 | 185.03407 | 185.00262 | 184.96956 | 184.9 |
| 2 | 0.00000 | 0.30769 | 10.56338 | 10.56275 | 10.56200 | 10.5 |
| 3 | 1.00000 | 0.00000 | 23.75384 | 23.74520 | 23.73575 | 23.7 |
| 4 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.0 |
| 5 | 0.00000 | 0.00000 | 23.74756 | 23.73898 | 23.72947 | 23.7 |
| 6 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.0 |
| 7 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.0 |
| 8 | 1.00000 | 0.00000 | 155.00212 | 154.94417 | 154.88415 | 154.8 |
| 9 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.0 |

*Figura 78 – Tabela de representação dos centroides*

A terceira aba diz respeito aos centroides dos grupos, e pode ser dividido em duas partes. A primeira é a geração de gráficos dinâmicos já apresentados na Figura 55 e na Figura 56. A segunda parte é a consulta dos dados dos centroides através de uma tabela *popup* interativa, que permite selecionar que parâmetros são apresentados.


*Figura 79 – Gráfico de dispersão entre a geração e o custo de consumo para 10 grupos*

Por fim, a quarta aba permite a geração dinâmica de gráficos de dispersão, variáveis com o período. Estes gráficos podem ser gerados para qualquer par de parâmetros vetoriais e dependentes do período. Na Figura 79, está representado o gráfico gerado para 10 grupos, considerando os parâmetros geração e custo do consumo.

### 5.1.12 Implantação da solução

A implantação da solução foi realizada utilizando o *software* de virtualização *Docker*. Desta forma, a plataforma foi dividida e implantada em vários *Docker Containers*, como representado na Figura 80.

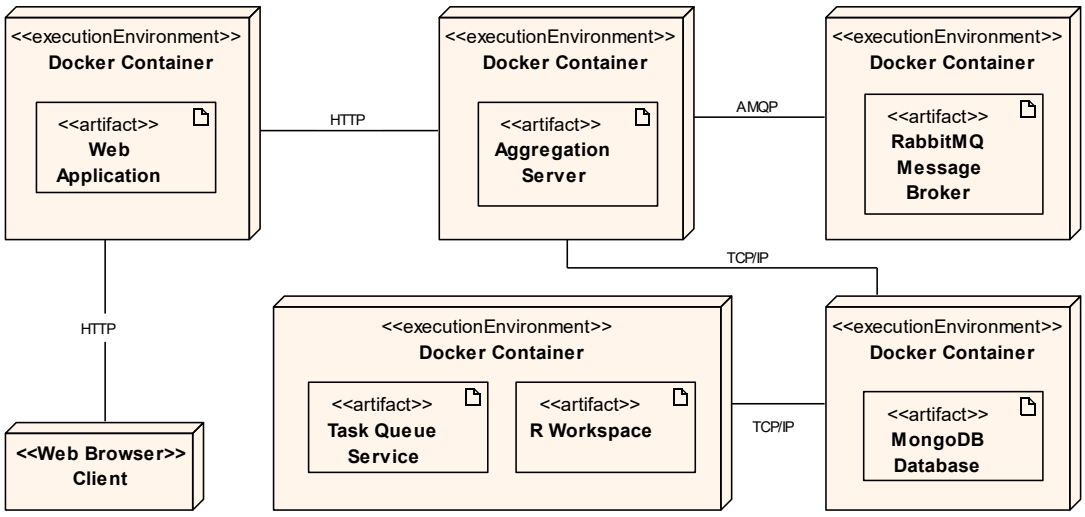



*Figura 80 – Diagrama de implantação simplificado da plataforma*

Deste modo, os diversos contentores referidos na secção 4.2.3 foram implantados separadamente, à exceção do ambiente de trabalho R, que foi implantado juntamente com o *Task Queue Service*. Assim, todos os contentores podem escalar independentemente consoante as necessidades do sistema, sem estarem limitados à partilha de recursos de um dispositivo.

## 5.2 Testes

No presente subcapítulo é demonstrado o processo de desenvolvimento de testes que garantem a qualidade do software desenvolvido. Deste modo, esta secção encontra-se subdividida entre testes unitários e testes de integração.

### 5.2.1 Testes à qualidade dos grupos formados na agregação

A qualidade dos grupos formados foi avaliada através das funcionalidades disponibilizadas pela plataforma. Foram avaliados os grupos formados para a agregação de participantes considerando como parâmetros a geração e o tipo de geração, ambos com o peso de 1. A Figura 81 discrimina a quantidade de participantes por tipo de geração no *dataset* avaliado.

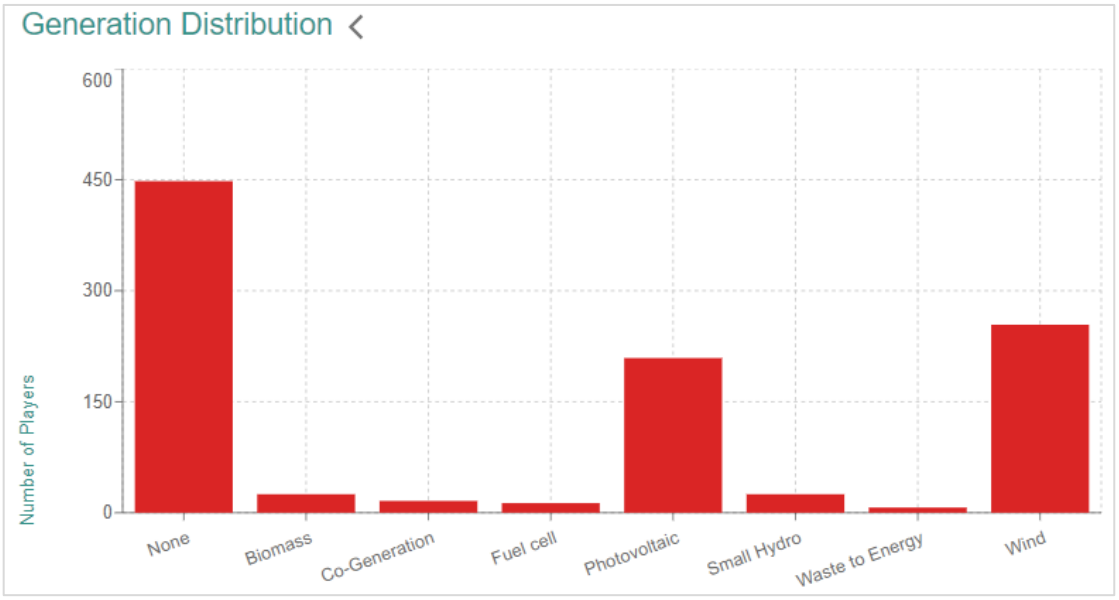



*Figura 81 – Distribuição dos tipos de geração no dataset*

Primeiramente, será analisada a avaliação *Elbow*, representada na Figura 82. Esta avaliação apenas considera a homogeneidade dos grupos, onde uma WSS mais baixa significa um grupo mais coerente. Naturalmente, quantos mais grupos forem gerados mais específicos (e homogéneos) serão. Deste modo, o objetivo desta avaliação é descobrir o número de grupos a partir do qual essa variação estabiliza.

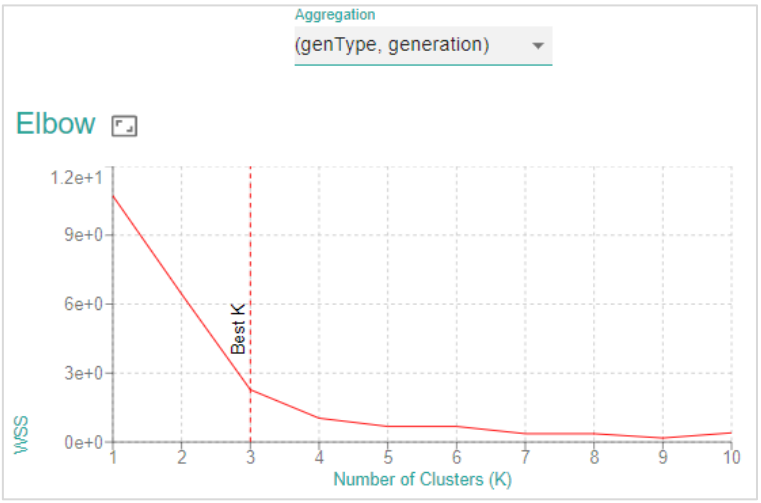



*Figura 82 – Avaliação Elbow para a agregação com os parâmetros geração e tipo de geração*

O método de avaliação *Elbow* sugere a utilização de 3 grupos, uma vez que a partir desse ponto, a redução da WSS é menos acentuada. Porém, analisando o gráfico, é possível verificar que a utilização de 4 grupos poderia também ser uma opção viável. Deste modo, serão analisados os resultados da divisão entre 3 e 4 grupos, com o auxílio da Figura 84 e da Figura 85.

O outro método de avaliação – *Silhouette* – distingue-se do *Elbow* por considerar as diferenças entre os grupos além da homogeneidade dos mesmos. Deste modo, foram obtidas as classificações (SS) representadas no gráfico da Figura 83.

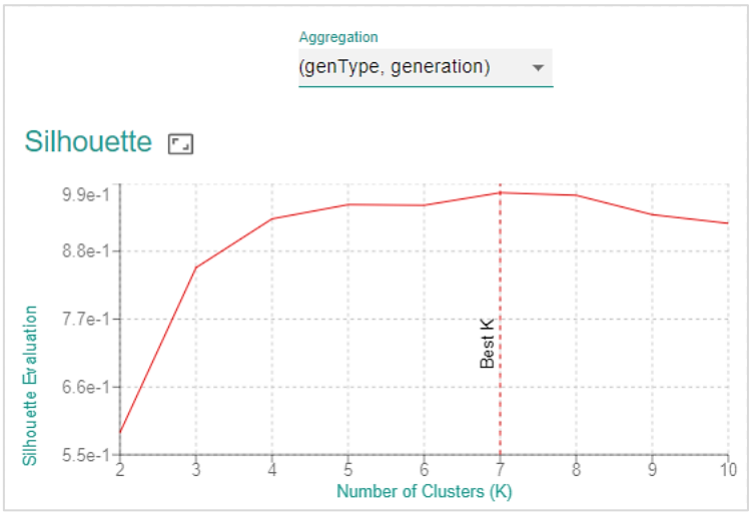



*Figura 83 – Avaliação Silhouette para a agregação com os parâmetros geração e tipo de geração*

A avaliação *Silhouette* sugere a criação de 7 grupos, pois é o número de grupos que apresenta um melhor SS. Ou seja, quando existem 7 grupos, as diferenças entre elementos do mesmo

grupo são minimizadas enquanto as diferenças entre grupos são maximizadas. A Figura 86 representa a constituição dos 7 grupos gerado.

Deste forma, procedeu-se à análise das agregações que geraram 3, 4 e 7 grupos, como sugeridas pelos métodos de avaliação.

**3 Grupos**

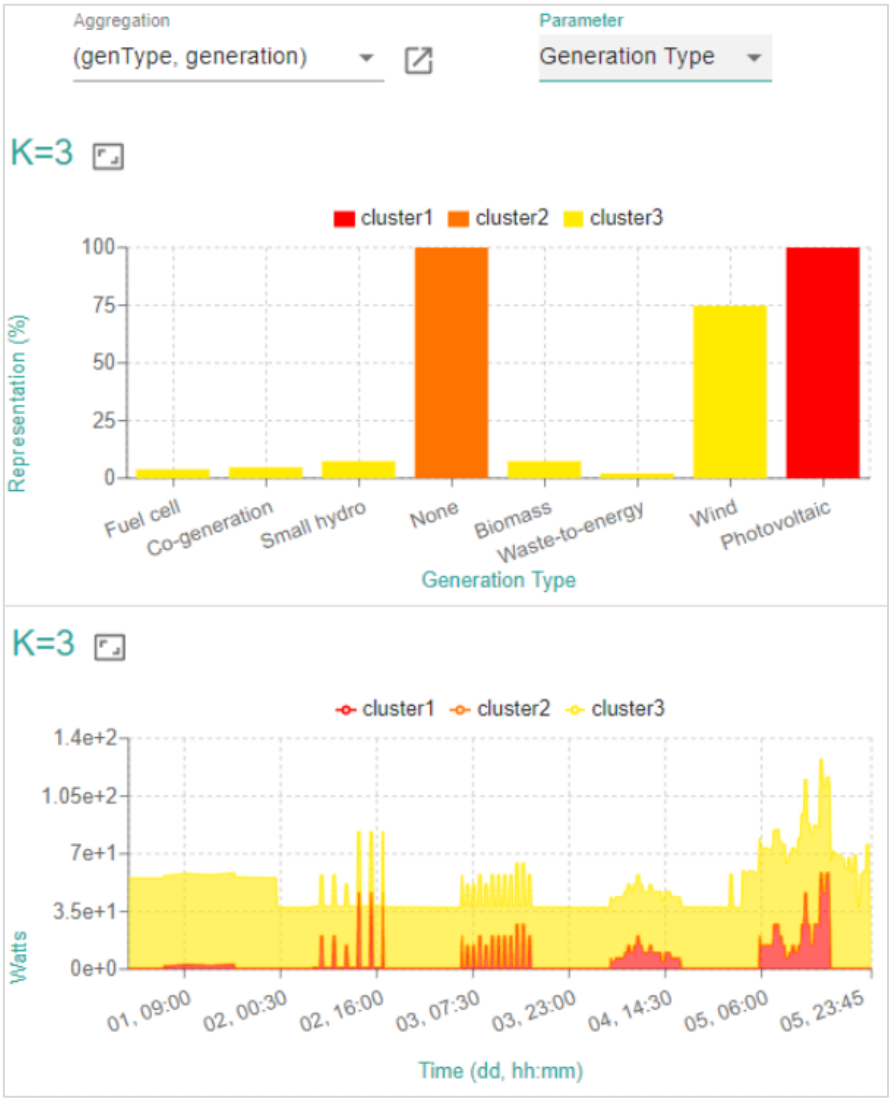



*Figura 84 – Distribuição dos tipos de geração energia e geração média por grupo, para 3 grupos*

O grupo 2 representa todos os participantes com que não produzem energia. O grupo 2 é constituído apenas por fontes de energia fotovoltaicas e por isso apresenta grandes variações na produção de energia, existindo períodos com produção média quase nula e períodos com uma produção média de 60W. O grupo 3 contém todas os restantes tipos de geração e é o grupo com a geração média mais consistente variando entre os 40 *Watts (W)* e os 70W.

Considerando que 448 participantes não produzem energia seria esperado que existisse um grupo que os isolasse, tal como se verifica. A energia eólica é o segundo tipo de geração com maior representação no *dataset* (254 participantes), pelo que também seria esperado que tivesse o seu próprio grupo. No entanto, esse grupo foi atribuído à energia fotovoltaica, com 209 participantes, devido ao facto das suas medições médias serem menos semelhantes às dos restantes tipos de energia. Não obstante, é de realçar que o tipo de geração eólica representa 75% dos participantes do seu grupo, pelo que, se poderá justificar a sua separação num novo grupo.

**4 Grupos**


*Figura 85 – Distribuição dos tipos de geração energia e geração média por grupo, para 4 grupos*

Comparando com a alternativa de 3 grupos, a geração de 4 grupos distingue-se por separar a energia eólica, que tende a ser intermitente, num novo grupo (3). Neste grupo, a geração média é variável entre os 0W e os 50W. Por outro lado, o grupo 1 é composto pelos participantes sem geração. Além disso, o grupo 4 corresponde ao grupo 1 no exemplo anterior, sendo composto pelas fontes de energia do tipo fotovoltaica. Por fim, o grupo 2 representa todos os outros tipos de geração, mais estáveis com uma geração média de 145W.

Esta opção seria mais adequada que a anterior, uma vez que a separa ambas as fontes de energia inconsistentes, isto é, solar e eólica, em grupos diferentes. Deste modo, o grupo 2 contém apenas fontes que produzem energia de modo constante e previsível.

**7 grupos**

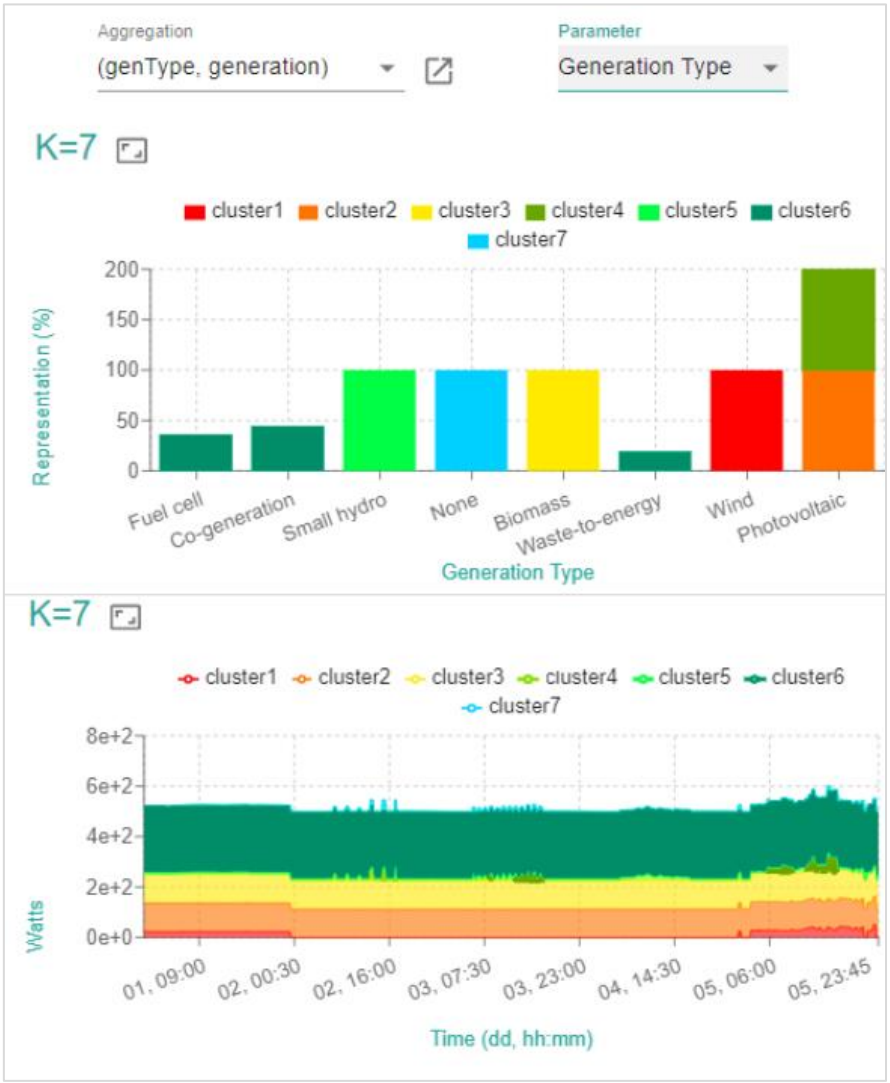



*Figura 86 – Distribuição dos tipos de geração energia e geração média por grupo, para 7 grupos (Cores adaptadas para realçar diferenças)*

Da mesma forma que no exemplo de 4 grupos, na agregação de 7 grupos dedica um grupo à energia eólica. Porém, separa a energia fotovoltaica em dois grupos: O grupo 2, que apresenta uma geração média de 113W ao longo de todo o período; e o grupo 4, que varia a sua geração média entre os 0W e os 60W. Além disso, em vez de agregar todos os restantes tipos de energia, separa os tipos de geração "*Small Hydro*" e "*Biomass*" em dois grupos diferentes: grupo 5 e grupo 3, respetivamente. O primeiro apresenta uma produção média entre os 8W e os 9W, enquanto que o segundo apresenta consistentemente 113W. Por fim, os restantes tipos de geração são agregados no grupo 4, e têm uma geração média de 265W ao longo de todo o período.

É ainda de realçar que neste exemplo o grupo 2 é, provavelmente, derivado de tipos de energia erradamente classificado. A existência de uma fonte de energia fotovoltaica com uma geração tão estável e consistente ao longo de todo o período é irrealista. Não obstante, esse erro realça que os grupos estão a ser formados corretamente, visto que a anomalia foi detetada e devidamente isolada num novo grupo. Ou seja, a agregação com 7 grupos não só atribui um grupo diferente a cada fonte intermitente (eólica e fotovoltaica), mas também separa a o(s) participante(s) com uma geração fotovoltaica inesperadamente consistente num novo grupo.

Além disso, as fontes de energia "*Small Hydro*" e "*Biomass*" foram separadas das restantes fontes pertencentes ao grupo 4 do exemplo da Figura 85, uma vez que as suas gerações médias (9W e 113W) não se enquadram com a geração média (265W) do restante grupo.

Em suma, a opção de divisão por 7 grupos é a alternativa melhor dividida, mas a opção 4 seria igualmente válida num contexto onde fosse mais valorizada distinção entre fontes intermites face à quantidade de energia produzida.

### 5.2.2 Testes Unitários

Os testes unitários focam-se em avaliar o comportamento isolado de porções do software. Deste modo, a procedimento seguido foi o de primeiramente garantir o cumprimento da lógica de negócio dos componentes independentes. Assim, a sua lógica faltosa não poderá comprometer os testes de componentes com maior nível hierárquico [110].

Ademais, com o intuito de tornar os testes unitários consistentes e fáceis de interpretar, foi seguida a abordagem trifásica *Arrange, Act and Assert* (AAA). A primeira fase, *Arrange*, diz respeito à construção do alvo do teste, este pode ser tanto um objeto ou um conjunto de objetos. Seguidamente, na fase *Act*, é executada a ação (ou conjunto de ações) cujo efeito se

pretende avaliar. Por fim, na fase *Assert*, é verificado se os resultados obtidos são coerentes com os esperados. É de realçar que, um resultado obtido pode ser um efeito colateral das ações tomadas e não o resultado direto das mesmas [111].

Para o desenvolvimento e execução destes testes unitários, no *backend*, foi utilizada a *framework* de testes unitários nativa do *Python* – a *unittest*. Por outro lado, no *frontend*, foi utilizada a *framework Mocha*. O Excerto de Código 15 é um exemplo um teste unitário seguindo a abordagem AAA e utilizando a *framework unittest*.

```
1.  def test_start_task(self):
2.      #Arrange
3.      task=Task()
4.
5.      #Act
6.      task.start()
7.
8.      #Assert
9.      expected=True
10.     self.assertEqual(expected, task.hasStarted())
```

*Excerto de Código 15 – Teste unitário de uma tarefa utilizando a abordagem AAA, em Python*

Com o intuito de exemplificar o emprego completo dos testes unitários, no Anexo C, estão discriminados todos os testes unitários efetuados à classe *TagReadingRange*. Foi escolhida esta classe por ser independente e com apenas dois métodos, mas com uma lógica de negócio deveras complexa. Assim, os testes relativos a cada método foram colocados em classes de teste diferentes, existindo, então, uma melhor separação dos testes e permitindo a sua execução separada.

Por fim, a Tabela 5 descreve brevemente as vertentes testadas em cada categoria de alvos. O exemplo anterior era um teste de lógica de negócio a um objeto do domínio, correspondendo, portanto, à primeira linha da tabela.

*Tabela 5 – Descrição dos testes unitários por categoria de alvos*

| ***Alvo*** | **Teste** | **Descrição** |
|---|---|---|
| *Objetos do Domínio* | Lógica de Negócio | Testar o comportamento isolado de um objeto e garantir o cumprimento das regras do negócio. |
| *Utilitários de Ficheiros* | Leitura de ficheiros | Testar a construção de objetos do modelo através de um ficheiro. |
| *Utilitários de Ficheiros* | Escrita de ficheiros | Testar a construção de um ficheiro através de objetos do modelo. |
| *Utilitários Genéricos* | Comparação de resultados | Testar se os resultados computados pelo utilitário são os esperados sobre diferentes cenários. |

| | | |
|---|---|---|
| *DTOs* | Criação a partir de Domínio | Criação de DTOs a partir de objetos do domínio, verificando a correção da conversão |
| *DTOs* | Conversão para Domínio | Conversão de DTOs para objetos do domínio, verificando a correção da mesma. |

### 5.2.3 Testes de Integração

Para a validação dos serviços disponibilizados pela API, foram realizados testes de integração. Para tal, recorreu-se à funcionalidade de testes automáticos do *Postman* [112]. Esta permite a definição de vários conjuntos de testes que podem, posteriormente, ser executados em cadeia. Além disso, através do uso de variáveis globais, o *Postman* permite utilizar os resultados de um teste da API noutro teste. Deste modo, foram realizados testes de integração para todas as funcionalidades disponibilizadas pela API, como refletido na

Figura 87.

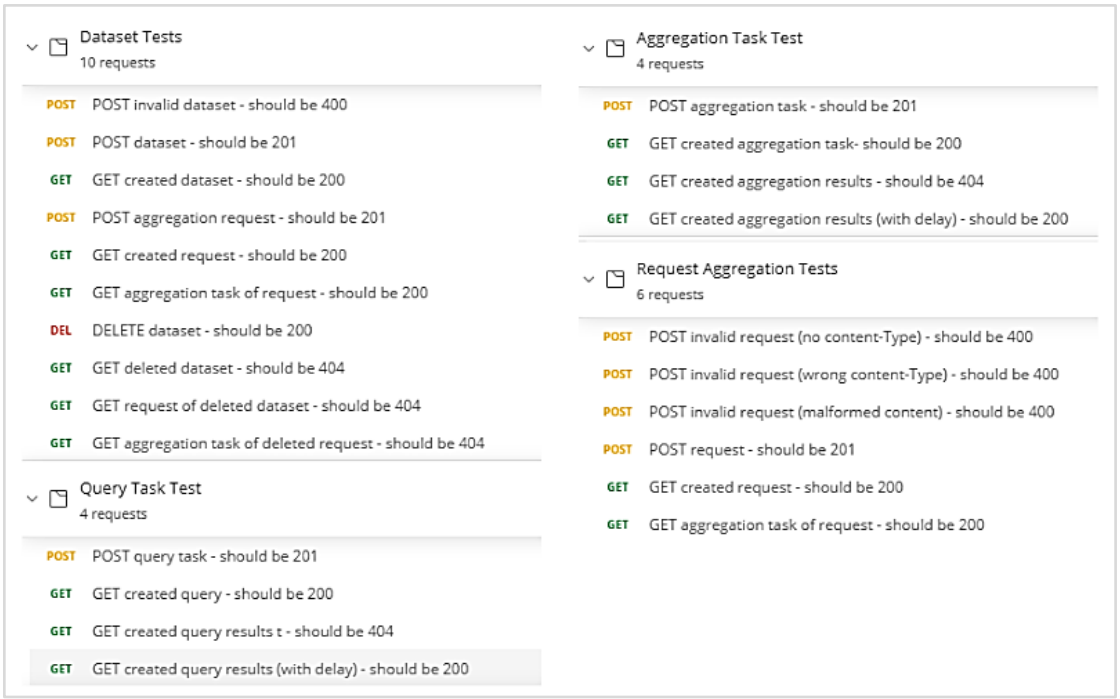



*Figura 87 – Testes de Integração realizados através do Postman*

Para melhor explorar a natureza dos testes acima descritos, o Anexo B detalha o primeiro bloco de testes, isto é, os testes relativos à gestão de *datasets*.

Finalmente, na Figura 88 é exposta a implementação dos testes de criação de um *dataset* e de feita a verificação deste, respetivamente. É de realçar, uma vez mais, a utilização de uma variável global, "*DATASET_ID*" para a execução sequencial de testes.


*Figura 88 – Implementação dos testes da criação do dataset no Postman*

## 5.3 Avaliação da solução

Considerando que a abordagem utilizada para a realização do projeto foi baseada em *scrum*, a avaliação da solução foi realizada bimensalmente, verificando o cumprimento dos requisitos.

Em cada reunião foram avaliadas partes diferentes da plataforma, e, consoante o *feedback*, foram realizadas alterações que seriam reavaliadas na reunião seguinte, juntamente com os requisitos dessa nova iteração. Desta forma, foi garantido que, a cada iteração, a solução se aproximava cada vez mais da desejada.

Além disso, tendo em conta um dos objetivos da abordagem *scrum* - a entrega iterativa de soluções funcionais - a versão atual da solução é totalmente utilizável. Deste modo, é ainda

de realçar que os requisitos não implementados não são impeditivos para o funcionamento previsto da plataforma.

Por fim, considerando que a aplicação se encontra numa versão estável e cumpre os requisitos funcionais e não funcionais impostos (ver Anexo D), a aplicação pode ser apreciada como totalmente funcional.

# 6 Conclusões

Este capítulo visa resumir o cumprimento dos objetivos definidos no início do projeto, refletir sobre as falhas e limitações do mesmo, apresentar possíveis soluções ou novos requisitos que enaltecerão o projeto e, por fim, apresentar o ponto de vista do autor sobre a experiência de realização do projeto.

## 6.1 Objetivos concretizados

Nesta secção é feita uma reflexão sobre o cumprimento dos objetivos definidos na secção 1.3.1 do capítulo introdutório através da Tabela 6 e posterior análise.

*Tabela 6 – Concretização dos objetivos*

| ***Objetivo*** | **Estado da implementação** | **Secções** |
|---|---|---|
| Estudo da área do projeto | Concluído | 2.1 |
| Estudo de algoritmos de *clustering* | Concluído | 2.2 |
| Estudo de tecnologias semelhantes | Concluído | 2.5 |
| Levantamento e análise de requisitos | Concluído | 3 |
| Desenho da solução | Concluído | 4 |
| Desenvolvimento *clustering* | Concluído | 5.1 |
| Desenvolvimento do *frontend* | Concluído | 5.1 |
| Desenvolvimento do *backend* | Concluído | 5.1 |
| Testes e avaliação | Concluído | 5.2 e 5.3 |

Atualmente, o projeto encontra-se numa versão estável que corresponde aos requisitos necessários à sua utilização, tendo sido cumpridos todos os objetivos definidos para os sprints enumerados na secção 1.3.4.

## 6.2 Limitações e trabalho futuro

Considerando o estado atual do projeto, apesar de estar numa versão estável, a plataforma apresenta algumas limitações, nomeadamente:

- **Medições em falta** – A execução da agregação com dados em falta resulta num erro, devido à natureza dos algoritmos utilizados. A validação é uma solução que permite ao Agregador contornar este problema, mas não o corrige.
- **Parâmetros em falta** – Apesar da plataforma fornecer meios para a interpretação de parâmetros novos, ao importar dados a partir da API do DOMINOES existem parâmetros contratuais que não são, ainda, disponibilizados, pelo que essa aba perde a sua relevância.

## 6.3 Trabalho Futuro

O trabalho a curto prazo forcar-se-á, sobretudo, na implementação das funcionalidades do sprint atual, como a remoção de períodos faltosos na agregação. Além disso, outros aspetos poderão ser melhorados, definindo-se assim o seguinte trabalho futuro:

- **Preenchimento ou remoção de falhas nas medições** – Considerando que a agregação não permite falhas nas medições, a solução a esse problema passa por excluir os períodos em que são detetadas falhas ou pelo seu preenchimento, com o valor médio dos intervalos vizinhos, por exemplo;
- **Agregação parcial –** Permitir que a agregação possa ser realizada para apenas parte do *dataset*, como por exemplo, apenas com participantes que sejam consumidores domésticos;
- **Preenchimento das abas contratuais** – Tornar a aba dinâmica, percebendo quais os dados relevantes;
- **Customização da plataforma** – Possibilitar a edição da estrutura da plataforma (alteração dos ficheiros de configuração) a partir da página *web*;
- **Uniformizar a interface** – Estudar os temas e padrões utilizados no *website* do projeto DOMINOES e aplicá-los a interface. Embora a interface gráfica cumpra os requisitos impostos, a interface ser semelhante à do projeto integrador pode ser relevante.

Estas melhorias tornariam a plataforma aplicável a um maior número de cenários, nomeadamente, a agregação poderia ser feita mesmo quando existem falhas nas medições e ser direcionada a apenas um conjunto de participantes com determinadas características; o Agregador poderia decidir os parâmetros que quer ver e definir a sua disposição; a uniformização com outras interfaces do projeto integrador tornaria a sua utilização mais intuitiva para o Agregador.

## 6.4 Apreciação final

Devo começar por refletir sobre a área em que senti maior evolução durante o projeto: *soft skills*. Estou imensamente grato por todas as repreensões e ensinamentos ao longo dos últimos meses, mesmo que por vezes tenha necessitado que esses fossem soletrados. Sinto que cresci como pessoa e fico feliz de ter compreendido que a diferença entre o que é dito e o que se quer dizer é assertividade e experiência. Não considero que estou ao nível que desejaria, mas sei que, pelo menos, estou mais ciente das minhas falhas, que é o primeiro passo para as minimizar.

Em termos de trabalho desenvolvido, todas as funcionalidades inicialmente propostas foram implementadas. Embora nem todas as funcionalidades adicionais descritas neste relatório estejam atualmente implementadas, o trabalho encontra-se numa versão estável e estou grato de poder continuar a trabalhar nessas e em futuras funcionalidades ao longo dos próximos meses.

Não considero que o projeto tenha sido tão desafiador como o previsto, apesar do tempo de adaptação a temáticas, linguagens e *frameworks* novas. Um dos maiores fatores foi o facto de na maioria das vezes em que existia uma dificuldade na implementação, existir também um supervisor com uma solução. Além disso, o cariz do projeto era o desenvolvimento de funcionalidades e não no desenvolvimento de algoritmos, que tendem a ser mais complexos.

Considerando que o projeto a desenvolver era uma aplicação *fullstack*, uma das minhas metas pessoais era a criação de uma interface gráfica apelativa e moderna, por sentir que foi algo que, durante o curso, não tive oportunidade de dedicar o tempo que gostaria. Porém, como o foco do trabalho foi sempre o desenvolvimento e entrega de funcionalidades, não sobrou o tempo desejado para a exploração da interface gráfica, que é refletido em alguns componentes. Contudo, considero que as horas extracurriculares dedicadas a esse tema foram recompensadas, não só por ter resultado num projeto mais gratificante, mas também pelos conhecimentos obtidos que irão acelerar esse processo em projetos futuros.

Em suma, estou grato. Grato pela oportunidade de participar num projeto europeu que contribui para um futuro mais verde. Grato pelo apoio incansável por parte dos supervisores e orientador. Grato a todos os meus amigos e família que me ajudaram em momentos mais difíceis. Grato a mim por ter desenvolvido algo em que me orgulho.

Obrigado.

# Referências

[1] DOMINOES, “dominoesproject – Smart Distribution Grid: a Market Driven Approach for the Next Generation of Advanced Operation Models and Services.” http://dominoesproject.eu/ (accessed Aug. 13, 2020).

[2] “Energy | Horizon 2020.” https://ec.europa.eu/programmes/horizon2020/en/area/energy (accessed Sep. 23, 2020).

[3] “H2020 Projects - Summary .” https://webgate.ec.europa.eu/dashboard/sense/app/93297a69-09fd-4ef5-889f-b83c4e21d33e/sheet/a879124b-bfc3-493f-93a9-34f0e7fba124/state/analysis (accessed Aug. 13, 2020).

[4] Parlamento Europeu, “DIRETIVA (UE) 2019/944,” *J. Of. da União Eur.*, vol. 2018, no. 4, pp. 210–230, 2018.

[5] S. M. Kaplan, “Electric power transmission: Background and policy issues,” *Smart Grid Electr. Power Transm.*, pp. 47–85, 2011.

[6] A. Ricci *et al.*, “Smart grids / Energy grids. The techno-scientific developments of smart grids and the related political, societal and economic implications,” *Sci. Technol. Options Assess.*, p. 138, 2012, [Online]. Available: http://www.europarl.europa.eu/RegData/etdes/etudes/join/2012/488797/IPOL-JOIN_ET(2012)488797_EN.pdf.

[7] T. R. Oliveira, C. A. G. Marques, W. A. Finamore, S. L. Netto, and M. V. Ribeiro, “A Methodology for Estimating Frequency Responses of Electric Power Grids,” *J. Control. Autom. Electr. Syst.*, vol. 25, no. 6, pp. 720–731, 2014, doi: 10.1007/s40313-014-0151-5.

[8] H. Farhangi, “The path of the smart grid,” *IEEE Power Energy Mag.*, vol. 8, no. 1, pp. 18–28, Jan. 2010, doi: 10.1109/MPE.2009.934876.

[9] L. Chhaya, P. Sharma, G. Bhagwatikar, and A. Kumar, “Wireless sensor network based smart grid communications: Cyber attacks, intrusion detection system and topology control,” *Electron.*, vol. 6, no. 1, 2017, doi: 10.3390/electronics6010005.

[10] P. Siano, “Demand response and smart grids - A survey,” *Renew. Sustain. Energy Rev.*,

vol. 30, pp. 461–478, 2014, doi: 10.1016/j.rser.2013.10.022.

[11] X. Yan, Y. Ozturk, Z. Hu, and Y. Song, “A review on price-driven residential demand response,” *Renewable and Sustainable Energy Reviews*, vol. 96. Elsevier Ltd, pp. 411–419, Nov. 01, 2018, doi: 10.1016/j.rser.2018.08.003.

[12] C. Silva, P. Faria, and Z. Vale, “Demand response and distributed generation remuneration approach considering planning and operation stages,” *Energies*, vol. 12, no. 14, 2019, doi: 10.3390/en12142722.

[13] L. Gkatzikis, I. Koutsopoulos, and T. Salonidis, “The role of aggregators in smart grid demand response markets,” *IEEE J. Sel. Areas Commun.*, vol. 31, no. 7, pp. 1247–1257, 2013, doi: 10.1109/JSAC.2013.130708.

[14] M. Parvania and M. Fotuhi-Firuzabad, “Demand response scheduling by stochastic SCUC,” *IEEE Trans. Smart Grid*, vol. 1, no. 1, pp. 89–98, 2010, doi: 10.1109/TSG.2010.2046430.

[15] S. Burger and J. P. Chaves-ávila, “The Value of Aggregators in Electricity Systems Carlos Batlle , and Ignacio J . Pérez-Arriaga,” *MIT Cent. Energy Environ. Policy Res.*, vol. CEEPR WP 2, no. January, p. 29, 2016.

[16] BEUC, “Electricity Aggregators: Starting off on the right foot with consumers,” vol. 32, no. 9505781573, pp. 0–11, 2018, [Online]. Available: http://www.beuc.eu/publications/beuc-x-2018-010_electricity_aggregators_starting_off_on_the_right_foot_with_consumers.pdf.

[17] J. Iria, F. Soares, and M. Matos, “Optimal bidding strategy for an aggregator of prosumers in energy and secondary reserve markets,” *Appl. Energy*, vol. 238, no. January, pp. 1361–1372, 2019, doi: 10.1016/j.apenergy.2019.01.191.

[18] AENA, “What are distributed energy resources and how do they work? - Australian Renewable Energy Agency,” 2018. https://arena.gov.au/blog/what-are-distributed-energy-resources/ (accessed Sep. 14, 2020).

[19] M. Warcholinski, “Lean, Agile And Scrum: A Simple Guide [2020] | Brainhub,” 2020. https://brainhub.eu/blog/differences-lean-agile-scrum/ (accessed Sep. 15, 2020).

[20] DOMINOES, “About – dominoesproject.” http://dominoesproject.eu/about/ (accessed Sep. 15, 2020).

[21] “History of Electrification Sites.” http://edisontechcenter.org/HistElectPowTrans.html

(accessed Aug. 17, 2020).

[22] Litos Strategic Communication, “The smart grid: An introduction,” *Smart Grid Electr. Power Transm.*, pp. 1–45, 2011, doi: 10.1002/9781119521129.ch15.

[23] A. Moshari, G. R. Yousefi, A. Ebrahimi, and S. Haghbin, “Demand-side behavior in the smart grid environment,” *IEEE PES Innov. Smart Grid Technol. Conf. Eur. ISGT Eur.*, pp. 1–7, 2010, doi: 10.1109/ISGTEUROPE.2010.5638956.

[24] Parlamento Europeu, “DIRETIVA (UE) 2018/2001,” *J. Of. da União Eur.*, vol. 2001, pp. 82–209, 2018.

[25] R. Fares, “Renewable Energy Intermittency Explained: Challenges, Solutions, and Opportunities - Scientific American Blog Network,” 2015. https://blogs.scientificamerican.com/plugged-in/renewable-energy-intermittency-explained-challenges-solutions-and-opportunities/ (accessed Jun. 21, 2020).

[26] National Energy Technology Laboratory, “A System view of the modern grid:Provides power quality for 21st century needs,” *Electr. Deliv. Energy Reliab. U.S. Dep. Energy,Tech. rep.*, no. January, pp. 0–21, 2007.

[27] S. E. Fleten, K. M. Maribu, and I. Wangensteen, “Optimal investment strategies in decentralized renewable power generation under uncertainty,” *Energy*, vol. 32, no. 5, pp. 803–815, 2007, doi: 10.1016/j.energy.2006.04.015.

[28] H. Chandler, *Harnessing Variable Renewables*. 2011.

[29] M. H. Albadi and E. F. El-Saadany, “Demand response in electricity markets: An overview,” *2007 IEEE Power Eng. Soc. Gen. Meet. PES*, pp. 1–5, 2007, doi: 10.1109/PES.2007.385728.

[30] A. Asadinejad and K. Tomsovic, “Optimal use of incentive and price based demand response to reduce costs and price volatility,” *Electr. Power Syst. Res.*, vol. 144, pp. 215–223, 2017, doi: 10.1016/j.epsr.2016.12.012.

[31] A. S. Gazafroudi, “Universidad de Salamanca Tesis Doctoral Multi-agent architecture for local electricity trading in power distribution systems,” 2019.

[32] H. Saboori, M. Mohammadi, and R. Taghe, “Virtual power plant (VPP), definition, concept, components and types,” in *Asia-Pacific Power and Energy Engineering Conference, APPEEC*, 2011, doi: 10.1109/APPEEC.2011.5749026.

[33] G. Di Bella, L. Giarrè, M. Ippolito, A. Jean-Marie, G. Neglia, and I. Tinnirello, “Modeling

energy demand aggregators for residential consumers," *Proc. IEEE Conf. Decis. Control*, pp. 6280–6285, 2013, doi: 10.1109/CDC.2013.6760882.

[34] J. Aengenvoort, "What's the business model of a Virtual Power Plant (VPP)?," 2020. https://www.next-kraftwerke.com/energy-blog/business-model-virtual-power-plant-vpp (accessed Sep. 22, 2020).

[35] H. Heinrich, "Aggregators," *E-Journal Invasion*, pp. 101–125, 2007, doi: 10.1016/b978-1-84334-144-4.50003-3.

[36] S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah, "Time-series clustering - A decade review," *Inf. Syst.*, vol. 53, pp. 16–38, 2015, doi: 10.1016/j.is.2015.04.007.

[37] H. Steinhaus, "Sur la division del corps matériels en parties," *Bull. l'Académie Pol. del Sci. - Cl. III*, vol. IV, no. 12, pp. 801–804, 1957.

[38] J. Macqueen, "SOME METHODS FOR CLASSIFICATION AND ANALYSIS OF MULTIVARIATE OBSERVATIONS," The Regents of the University of California, 1967. Accessed: Sep. 26, 2020. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992.

[39] J. Starmer, "StatQuest: K-means clustering - YouTube," May 23, 2018. https://www.youtube.com/watch?v=4b5d3muPQmA&t=359s (accessed Aug. 03, 2020).

[40] A. Bouguettaya, Q. Yu, X. Liu, X. Zhou, and A. Song, "Efficient agglomerative hierarchical clustering," *Expert Syst. Appl.*, vol. 42, no. 5, pp. 2785–2797, 2015, doi: 10.1016/j.eswa.2014.09.054.

[41] J. Starmer, "StatQuest: Hierarchical Clustering - YouTube," Jun. 20, 2017. https://youtu.be/7xHsRkOdVwo?t=286 (accessed Jul. 28, 2020).

[42] H. Kriegel and P. Kr, "Density-based clustering," vol. 1, no. June, pp. 231–240, 2011, doi: 10.1002/widm.30.

[43] I. Journal, A. C. Science, I. Science, and T. Bangalore, "Appraising Research Direction & Effectiveness of Existing Clustering Algorithm for Medical Data," vol. 8, no. 3, pp. 343–351, 2017.

[44] D. Aggarwal and D. Sharma, "Application of clustering for student result analysis," *Int. J. Recent Technol. Eng.*, vol. 7, no. 6, pp. 50–53, 2019.

[45] D. M. SAPUTRA, D. SAPUTRA, and L. D. OSWARI, "Effect of Distance Metrics in

Determining K-Value in K-Means Clustering Using Elbow and Silhouette Method," vol. 172, no. Siconian 2019, pp. 341–346, 2020, doi: 10.2991/aisr.k.200424.051.

[46] A. Lengyel and Z. Botta-Dukát, "Silhouette width using generalized mean—A flexible method for assessing clustering efficiency," *Ecol. Evol.*, vol. 9, no. 23, pp. 13231–13243, 2019, doi: 10.1002/ece3.5774.

[47] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of data clusters via the gap statistic," *Journal of the Royal Statistical Society: Series B*, vol. 63, no. Part 2. pp. 411–423, 2001.

[48] F. F. H. Nah, "A study on tolerable waiting time: How long are Web users willing to wait?," *Behav. Inf. Technol.*, vol. 23, no. 3, pp. 153–163, 2004, doi: 10.1080/01449290410001669914.

[49] "Asynchronous Task Execution In Python." https://bhavaniravi.com/blog/asynchronous-task-execution-in-python (accessed Jul. 30, 2020).

[50] "Thanks Pivotal, Hello Redis Labs - <antirez>," 2015. http://antirez.com/news/91 (accessed Jul. 28, 2020).

[51] "Introduction to Redis – Redis." https://redis.io/topics/introduction (accessed Jul. 26, 2020).

[52] "Redis Persistence – Redis." https://redis.io/topics/persistence (accessed Jul. 28, 2020).

[53] "Redis." https://redis.io/ (accessed Jul. 26, 2020).

[54] "DB-Engines Ranking - popularity ranking of key-value stores," 2020. https://db-engines.com/en/ranking/key-value+store (accessed Jul. 26, 2020).

[55] "Pub/Sub – Redis." https://redis.io/topics/pubsub (accessed Jul. 28, 2020).

[56] "Redis Clients." https://redis.io/clients (accessed Jul. 26, 2020).

[57] "RabbitMQ - News," 2013. https://web.archive.org/web/20130602054940/http://www.rabbitmq.com/news.html (accessed Jul. 28, 2020).

[58] G. Olah, "An introduction to RabbitMQ - What is RabbitMQ? | Erlang Solution blog," Apr. 03, 2020. https://www.erlang-solutions.com/blog/an-introduction-to-rabbitmq-

what-is-rabbitmq.html (accessed Jul. 26, 2020).

[59] “Erlang Programming Language.” https://www.erlang.org/about (accessed Jul. 28, 2020).

[60] N. Chechina, M. M. Hernandez, and P. Trinder, “A Scalable Reliable Instant Messenger using the SD Erlang Libraries,” in *Erlang 2016 - Proceedings of the 15th International Workshop on Erlang, co-located with ICFP 2016*, Sep. 2016, pp. 33–41, doi: 10.1145/2975969.2975973.

[61] N. Q. Uy and V. H. Nam, “A comparison of AMQP and MQTT protocols for Internet of Things,” in *Proceedings - 2019 6th NAFOSTED Conference on Information and Computer Science, NICS 2019*, Dec. 2019, pp. 292–297, doi: 10.1109/NICS48868.2019.9023812.

[62] “Persistence Configuration — RabbitMQ.” https://www.rabbitmq.com/persistence-conf.html (accessed Jul. 28, 2020).

[63] T. Treat, “Benchmarking Message Queue Latency – Brave New Geek,” 2016. https://bravenewgeek.com/benchmarking-message-queue-latency/ (accessed Jul. 27, 2020).

[64] “Clients Libraries and Developer Tools — RabbitMQ.” https://www.rabbitmq.com/devtools.html (accessed Jul. 27, 2020).

[65] “Supported Platforms — RabbitMQ.” https://www.rabbitmq.com/platforms.html (accessed Jul. 26, 2020).

[66] “Change history for Celery 1.0.” http://www.pythondoc.com/celery-3.1.11/history/changelog-1.0.html#version-0-1-0 (accessed Jul. 30, 2020).

[67] “Configuration and defaults — Celery 4.4.6 documentation.” https://docs.celeryproject.org/en/stable/userguide/configuration.html (accessed Jul. 30, 2020).

[68] “Brokers — Celery 4.4.6 documentation.” https://docs.celeryproject.org/en/latest/getting-started/brokers/ (accessed Jul. 30, 2020).

[69] “Frequently Asked Questions — Celery 4.4.6 documentation - Task Priorities.” https://docs.celeryproject.org/en/latest/faq.html#does-celery-support-task-priorities (accessed Jul. 30, 2020).

[70] “Periodic Tasks — Celery 4.4.6 documentation.”

https://docs.celeryproject.org/en/stable/userguide/periodic-tasks.html (accessed Jul. 30, 2020).

[71] “Flower - Celery monitoring tool — Flower 1.0.0 documentation.” https://flower.readthedocs.io/en/latest/ (accessed Jul. 30, 2020).

[72] “What’s new in Celery 4.0 (latentcall) — Celery 4.4.6 documentation.” https://docs.celeryproject.org/en/latest/history/whatsnew-4.0.html#removed-features (accessed Jul. 27, 2020).

[73] “Releases · rq/rq.” https://github.com/rq/rq/releases?after=0.3.2 (accessed Jul. 30, 2020).

[74] “rq/rq: Simple job queues for Python - Project History.” https://github.com/rq/rq (accessed Jul. 30, 2020).

[75] “RQ: Workers.” https://python-rq.org/docs/workers/ (accessed Jul. 27, 2020).

[76] “rq/rq-scheduler: A lightweight library that adds job scheduling capabilities to RQ (Redis Queue).” https://github.com/rq/rq-scheduler (accessed Jul. 30, 2020).

[77] “Parallels/rq-dashboard: Flask-based web front-end for monitoring RQ queues.” https://github.com/Parallels/rq-dashboard (accessed Jul. 30, 2020).

[78] “RQ: Documentation Overview.” https://python-rq.org/docs/ (accessed Jul. 30, 2020).

[79] “huey · PyPI.” https://pypi.org/project/huey/#history (accessed Aug. 02, 2020).

[80] “huey — huey 2.2.0 documentation.” https://huey.readthedocs.io/en/latest/ (accessed Aug. 02, 2020).

[81] “Error running Getting Started on Windows · Issue #208 · coleifer/huey.” https://github.com/coleifer/huey/issues/208 (accessed Aug. 02, 2020).

[82] H. S. Oluwatosin, “Client-Server Model,” *IOSR J. Comput. Eng.*, vol. 16, no. 1, pp. 57–71, 2014, doi: 10.9790/0661-16195771.

[83] G. Reese, “The Distributed Application Architecture,” *Enterp. Integr. Model.*, pp. 126–145, 2000, doi: 10.7551/mitpress/2768.003.0058.

[84] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” vol. 6, no. 2, p. 103, 2000.

[85] J. Lewis and M. Fowler, “Microservices - a definition of this new architectural term.” https://martinfowler.com/articles/microservices.html (accessed Sep. 13, 2020).

[86] C. Richardson, "Monolithic Architecture pattern." https://microservices.io/patterns/monolithic.html (accessed Sep. 13, 2020).

[87] D. Namiot and M. Sneps-sneppe, "On Micro-services Architecture," no. September, 2014.

[88] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, 2015, doi: 10.1109/MS.2015.11.

[89] "SmartNet - Integrating renewable energy in transmission networks." http://smartnet-project.eu/ (accessed Aug. 08, 2020).

[90] "Partners - SmartNet." http://smartnet-project.eu/partners/ (accessed Aug. 08, 2020).

[91] "The project - SmartNet." http://smartnet-project.eu/the-project/ (accessed Aug. 08, 2020).

[92] "About Us : GE Grid Solutions." https://www.gegridsolutions.com/ourcompany.htm (accessed Aug. 08, 2020).

[93] "GE Communicator - Instruction Manual," pp. 1–183, 2003.

[94] "GE Energy Aggregator v4.0 Release Notes," vol. 0, pp. 1–5, 2017.

[95] "GE Energy Aggregator - Brochure," [Online]. Available: https://www.gegridsolutions.com/multilin/catalog/energy-aggregator.htm.

[96] "NEMOCS | Virtual Power Plant (VPP) as a Service Solution." https://www.next-kraftwerke.com/products/vpp-solution (accessed Sep. 22, 2020).

[97] "cyberNOC - Flexibility Aggregation Platform." https://www.cyber-grid.com/cybernoc/ (accessed Sep. 22, 2020).

[98] R. B. Grady, *Practical software metrics for project management and process improvement | Guide books*. Prentice-Hall, Inc., 1992.

[99] "Capturing Architectural Requirements." https://www.ibm.com/developerworks/rational/library/4706.html#N100A7 (accessed Aug. 25, 2020).

[100] "FURPS+ | QualidadeBR." https://qualidadebr.wordpress.com/2008/07/10/furps/ (accessed Aug. 25, 2020).

[101] "What is FURPS+? – Business Analyst Training in Hyderabad – COEPD." https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/ (accessed Aug. 25, 2020).

[102] “The C4 model for visualising software architecture.” https://c4model.com/ (accessed Sep. 22, 2020).

[103] P. B. Kruchten, “The 4+1 View Model of Architecture,” *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, 1995, doi: 10.1109/52.469759.

[104] “Node.js vs Django: Key differences, popularity, use cases and more.” https://www.simform.com/nodejs-vs-django/ (accessed Sep. 23, 2020).

[105] “The API gateway pattern versus the direct client-to-microservice communication | Microsoft Docs.” https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern (accessed Sep. 14, 2020).

[106] RealPython, “Python Virtual Environments: A Primer – Real Python.” https://realpython.com/python-virtual-environments-a-primer/ (accessed Sep. 18, 2020).

[107] M. Herman, “Django vs. Flask in 2019: Which Framework to Choose | TestDriven.io.” https://testdriven.io/blog/django-vs-flask/ (accessed Sep. 18, 2020).

[108] V. Singh, “Flask vs Django in 2020: Which Framework to Choose?” https://hackr.io/blog/flask-vs-django (accessed Sep. 18, 2020).

[109] V. Granville, “How to Automatically Determine the Number of Clusters in your Data - and more - Data Science Central,” 2019. https://www.datasciencecentral.com/profiles/blogs/how-to-automatically-determine-the-number-of-clusters-in-your-dat (accessed Sep. 24, 2020).

[110] M. Gälli, M. Lanza, O. Nierstrasz, and R. Wuyts, “Ordering broken unit tests for focused debugging,” *IEEE Int. Conf. Softw. Maintenance, ICSM*, no. June 2014, pp. 114–123, 2004, doi: 10.1109/icsm.2004.1357796.

[111] Kolodiy Sergey, “Unit Testing and Coding Best Practices for Unit Tests: A Test-Driven Perspective | Toptal.” https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters (accessed Sep. 16, 2020).

[112] Postman, “Automated Testing with Postman.” https://www.postman.com/automated-testing/ (accessed Sep. 16, 2020).

# Anexo A Ficheiro de Resultados da Agregação

1

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Proccess: | | | | | | | | | | | | |
| 2 | requested | Mon, 21 Sep 2020 10:55:31 GMT | | | | | | | | | | | |
| 3 | started | Mon, 21 Sep 2020 10:55:36 GMT | | | | | | | | | | | |
| 4 | finished | Mon, 21 Sep 2020 10:56:13 GMT | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | |
| 6 | Dataset: | | | | | | | | | | | | |
| 7 | name | DATASET_CONS1k_v2 (2) | | | | | | | | | | | |
| 8 | datasetId | 47 | | | | | | | | | | | |
| 9 | consumers | 447 | | | | | | | | | | | |
| 10 | producers | 1 | | | | | | | | | | | |
| 11 | prosumers | 548 | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | |
| 13 | Parameters: | | | | | | | | | | | | |
| 14 | consType | genType | generation | | | | | | | | | | |
| 15 | | X | X | | | | | | | | | | |
| 16 | | | | | | | | | | | | | |
| 17 | Optimal k | | | | | | | | | | | | |
| 18 | Elbow result | 3 | | | | | | | | | | | |
| 19 | Silhouette result | 7 | | | | | | | | | | | |
| 20 | Elbow array | 10.71279 | 6.439234 | 2.276979 | 1.037391 | 0.678431 | 0.678414 | 0.362174 | 0.362191 | 0.174914 | 0.39961 | | |
| 21 | Silhouette array | 0 | 0.585969 | 0.853543 | 0.932798 | 0.955836 | 0.954833 | 0.975179 | 0.970976 | 0.939565 | 0.925475 | | |

Details | (k=10)general | (k=10)general centers | (k=10) player groups ...

2

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | betweenss | 10.712 | | | | | | | | | |
| 2 | tot.withinss | 0.0006 | | | | | | | | | |
| 3 | totss | 10.712 | | | | | | | | | |
| 4 | withinss | 2.99E-09 | 0.00E+00 | 3.41E-08 | 0.00E+00 | 3.20E-11 | 1.92E-05 | 3.52E-04 | 9.05E-09 | 2.82E-08 | 2.31E-04 |
| 5 | Groups Size | 208 | 1 | 13 | 447 | 7 | 25 | 11 | 254 | 25 | 5 |
| 6 | | | | | | | | | | | |

Details | (k=10)general | (k=10)general centers | (k=10) player groups ...

3

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | genType | genType | genType | genType | genType | genType | genType | genType | generation | generation | generation |
| 2 | Group | Fuel cell | Co-generation | Small hydro | None | Biomass | Waste-to-energy | Wind | Photovoltaic | generation0 | generation1 | g |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 113.33065 | 113.3306496 | |
| 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 6 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 113.063098 | 113.0630976 | 113.063098 |
| 7 | 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 8 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 23.7517632 | 23.7431808 | 23.7336768 |
| 9 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 10 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 11 | 8 | 0 | 0 | 0.78125 | 0 | 0 | 0.21875 | 0 | 0 | 8.701056 | 8.6819328 | 8.6632704 |
| 12 | 9 | 0.448276 | 0.551724138 | 0 | 0 | 0 | 0 | 0 | 0 | 326.143123 | 326.0444544 | 325.940544 |
| 13 | | | | | | | | | | | | |

Details | (k=10)general | (k=10)general centers | (k=10) player groups ...

4

| | A |
|---|---|
| 1 | 6 |
| 2 | 10 |
| 3 | 9 |
| 4 | 7 |
| 5 | 6 |
| 6 | 3 |
| 7 | 3 |
| 8 | 6 |
| 9 | 7 |
| 10 | 1 |
| 11 | 9 |
| 12 | 7 |
| 13 | 7 |
| 14 | 7 |

... | (k=10)general | (k=10)general centers | (k=10) player groups | (k=1) p ...

*(1) Dados gerais sobre as agregações; (2) Dados gerais sobre a agregação de 10 grupos; (3) Centroides / médias dos 10 grupos; (4) Distribuição dos participantes (linha) pelos 10 grupos*

# Anexo B Detalhe dos Testes de Integração

| ***Método e caminho*** | **Descrição do teste** | **Código** |
|---|---|---|
| **POST** */dataset* | Tentar guardar o *dataset* inválido recebido no corpo do pedido no *backend*. | **400** |
| POST */dataset* | Guardar o *dataset* recebido no corpo do pedido no *backend*. | **201** |
| **GET** */dataset/id* | Validar a criação e persistência do *dataset* no *backend*. | **200** |
| **POST** */requestAggregation* | Criar um pedido de agregação para o *dataset* criado. | **201** |
| **GET** *requestAggregation/reqId* | Obter o pedido de agregação criado, incluindo as suas tarefas associadas. | **200** |
| **GET** */aggregationTask/taskId* | Verificar a existência de uma das tarefas de agregação associadas ao pedido de agregação. | **200** |
| **DELETE** */dataset/id* | Remover o registo do *dataset* do *backend* através do identificador. | **200** |
| **GET** */dataset/id* | Verificar que o *dataset* não existe após a sua remoção. | **404** |
| **GET** */requestAggregation/reqId* | Verificar que o pedido não existe após a remoção do *dataset* associado. | **404** |
| **GET** */aggregationTask/taskId* | Verificar que a tarefa de agregação não existe após o pedido ao qual está associada ser apagado | **404** |
| **POST** */dataset* | Guardar o *dataset* recebido no corpo do pedido no *backend*. | **201** |
| **GET** */dataset/id* | Validar a criação e persistência do *dataset* completamente. | **200** |
| **POST** */requestAggregation* | Criar um pedido de agregação para o *dataset* criado. | **201** |
| **GET** */requestAggregation/reqId* | Obter o pedido de agregação criado, incluindo as suas tarefas associadas. | **200** |
| **GET** */aggregationTask/taskId* | Verificar a existência de uma das tarefas de agregação associadas ao pedido de agregação. | **200** |
| DELETE */dataset/id* | Remover o registo do *dataset* do *backend* através do identificador. | **200** |
| **GET** */dataset/id* | Verificar que o *dataset* não existe após a sua remoção. | **404** |
| **GET** */requestAggregation/reqId* | Verificar que o pedido não existe após a remoção do *dataset* associado. | **404** |
| **GET** */aggregationTask/taskId* | Verificar que a tarefa de agregação não existe após o pedido ao qual está associada ser apagado | **404** |

# Anexo C Testes unitários da classe TRR

```
class testTRRMerges(unittest.TestCase):

    def test_add_trr_empty(self):

  #RIGHT MERGE--------------------------------
    def test_add_trr_right_no_overlap(self):
    def test_add_trr_right_consecutive(self):
    def test_add_trr_right_edge(self):
    def test_add_trr_right_overlap(self):

  #LEFT MERGE---------------------------------
    def test_add_trr_left_no_overlap(self):
    def test_add_trr_left_consecutive(self):
    def test_add_trr_left_edge(self):
    def test_add_trr_left_overlap(self):

  #OTHER--------------------------------------
    def test_add_trr_containedFirst(self):
    def test_add_trr_containedSecond(self):
    def test_add_trr_connect_one(self):
    def test_add_trr_connect_all(self):
    def test_add_trr_connect_all2(self):


class testMissingTRR(unittest.TestCase):

    def test_empty(self):

  #RIGHT MISSING------------------------------
    def test_missing_right_edge(self):
    def test_missing_right_consecutive(self):
    def test_missing_right_overlap(self):
    def test_missing_right_no_overlap(self):

  #LEFT MISSing-------------------------------
    def test_missing_left_edge(self):
    def test_missing_left_consecutive(self):
    def test_missing_left_overlap(self):
    def test_missing_left_no_overlap(self):

  #OTHER --------------------------------------
    def test_missing_middle(self):
    def test_missing_side(self):
    def test_missing_middle_and_side(self):
```

# Anexo D Avaliação dos Requisitos

| *Requisito* | **Método de Avaliação** | **Resultado** |
|---|---|---|
| **(Funcionalidade)**<br><br>Funcionalidades desenvolvidas | Ao longo das reuniões foram sendo demostradas e avaliadas todas as funcionalidades desenvolvidas. | 100% |
| **(Usabilidade)**<br><br>A interface gráfica deve ser simples, intuitiva para qualquer utilizador conhecedor do domínio. | Foi realizada uma sessão de avaliação com membros conhecedores do domínio onde foi pedido para executar casos de uso da plataforma e avaliada a sua acessibilidade. | 80% |
| **(Usabilidade)**<br><br>O tema (cores, tipos e tamanho de letra, estilo) da interface deve ser consistente em toda a interface. | Foram analisadas todas as interfaces da aplicação e verificado se os tipos de letra, estrutura da página e a paleta de cores eram consistentes. | 90% |
| **(Usabilidade)**<br><br>O tempo de resposta deve ser curto, sendo qualquer operação com tempo de resposta superior a 1 segundo acompanhada de um indicador de carregamento | Foram executadas todas as ações que têm um tempo de resposta previsto superior a um segundo (ler ficheiros e gerar ficheiros, carregar *datasets*, acompanhamento de tarefas, gráficos de dispersão dos resultados de agregação) e garantido que aparecia sempre um indicador de execução. | 100% |
| **(Usabilidade)**<br><br>Erros na plataforma (e.g. pedidos para o *backend* sem resposta), não devem interromper o funcionamento da aplicação. | Foi desligado o *backend* da plataforma a meio de pedidos do *frontend* e verificado que as funcionalidades não dependentes do *backend* continuavam disponíveis. | 100% |

| | | |
|---|---|---|
| **(Confiabilidade)**<br><br>No caso de erros ou falhas, a plataforma deve continuar funcional. | No *frontend* foi testado o carregamento de ficheiros inválidos e garantido (após uma mensagem de erro) o normal funcionamento da aplicação. Foram retiradas algumas verificações de modo a simular erros fatais no *backend*. | 100% |
| **(Desempenho)**<br><br>Se possível, guardar os dados localmente para evitar pedidos morosos a APIs externas | O cumprimento deste requisito é detalhado na secção 5.1.4. | 100% |
| **(Suportabilidade)**<br><br>A aplicação deverá possibilitar a integração de novos algoritmos de agregação. | O sistema de execução de tarefas está desenhado de forma a permitir a leitura e execução de diferentes scripts R, permitindo, assim, a adição de novos algoritmos quando necessário. | 100% |
| **(Suportabilidade)**<br><br>O código deverá obedecer às normas e refletir boas práticas da programação, como a utilização de padrões de software e obedecer aos princípios SOLID e GRASP de modo a simplificar a sua manutenibilidade. | Durante o desenho e implementação, tal como representado ao longo do relatório, foi priorizado o cumprimento dos padrões SOLID e GRASP, sempre que aplicáveis. | 100% |
| **(Suportabilidade)**<br><br>A aplicação (*backend*) deverá suportar tanto plataformas baseadas em Unix como Windows. | A plataforma foi desenvolvida numa máquina *Windows* e implantada em *Docker Containers* baseados em *Ubuntu* 18.04. | 100% |

| | | |
|---|---|---|
| **(Limitação de Desenho)**<br><br>Os algoritmos de agregação deverão ser implementados em *R* | O *backend* da plataforma foi desenhado de forma a incluir um ambiente de trabalho *R*, para executar os algoritmos desenvolvidos. | 100% |
| **(Limitação de Desenho e Implementação)**<br><br>A aplicação deverá seguir o modelo *Client-Server*. | Tal como documentado ao longo do relatório, a plataforma é uma aplicação web e tem um *backend* dedicado, seguindo o modelo *Client-Server.* | 100% |
| **(Limitação de Implementação)**<br><br>O servidor deverá ser desenvolvido em *Python* | O servidor foi desenvolvido em *Python.* | 100% |
| **(Limitação de Implementação)**<br><br>A plataforma deverá ser uma aplicação *web* | Tal como documentado ao longo do relatório, a plataforma é uma aplicação *web.* | 100% |
| **(Requisitos de Interface)**<br><br>A aplicação deverá disponibilizar os seus serviços REST para a agregação | O servidor disponibiliza os serviços de agregação através de uma interface pública. | 100% |
| **(Requisitos de Interface)**<br><br>A aplicação devera interagir com os serviços a API REST do DOMINOES para obter dados de participantes reais. | A plataforma possibilita a criação de *datasets* a partir de dados provenientesda API REST do DOMINOES. | 100% |

| | | |
|---|---|---|
| **(Limitações Físicas)**<br>A implantação deverá ser feita em *Dockers* conectados à rede privada do GECAD. | Os *Docker Containers* gerados foram implantados em *Dockers* hospedados por máquinas ligadas à rede privada do GECAD. | 100% |