



# UMinho

## Mestrado Engenharia Informática

### Requisitos e Arquiteturas de Software

#### (2023/24)

## PROBUM v1.0

*Grupo 1.B, PL1/Entrega 2*

PG52683	Gonçalo Nuno Pereira Senra
PG52684	Henrique Miguel da Silva Costa
PG53944	João Pedro Moreira Brito
PG53926	João Miguel Pinheiro Machado
PG54042	Maria Eugénia Bessa Cunha
PG54056	Maria Rita Dias Braga Lino
PG54121	Nuno Miguel Leite da Costa
PG54188	Ricardo Lopes Santos Silva
PG54198	Rodrigo Manuel Matos Pereira
PG54249	Tiago André Mendes Oliveira
PG54256	Tiago Luís Pereira Ferreira



Braga, 1 de dezembro de 2023

# Prefácio

Com o decorrer do trabalho, fomos encontrando dificuldades em vários aspetos devido, maioritariamente, pela nossa má organização enquanto equipa. Estas dificuldades surgiram em grande parte por defeito de trabalhos distribuídos pelos elementos do grupo estarem muito interligados entre si, o que, desde logo, exigia bastante comunicação entre todos.

Por fim, após uma reflexão crítica sobre o contributo de cada membro da equipa e com base na possibilidade de valorizar o contributo de cada um através da afetação da nota individual num valor entre -2.0 e 2.0, estipulamos as seguintes avaliações entre-pares.

- PG52683, Gonçalo Nuno Pereira Senra : 0
- PG52684, Henrique Miguel da Silva Costa : 0
- PG53926, João Miguel Pinheiro Machado : 0
- PG53944, João Pedro Moreira Brito : 0
- PG54042, Maria Eugénia Bessa Cunha : 0
- PG54056, Maria Rita Dias Braga Lino : 0
- PG54121, Nuno Miguel Leite da Costa : 0
- PG54188, Ricardo Lopes Santos Silva : 0
- PG54198, Rodrigo Manuel Matos Pereira : 0
- PG54249, Tiago André Mendes Oliveira : 0
- PG54256, Tiago Luís Pereira Ferreira : 0

# Conteúdo

<b>1</b>	<b>Introdução e Objetivos</b>	<b>6</b>
1.1	Introdução . . . . .	6
1.2	Objetivos do Projeto . . . . .	7
<b>2</b>	<b>Restrições do Projeto</b>	<b>9</b>
2.1	Restrição da Infraestrutura . . . . .	9
2.2	Restrição da Isolamento da Aplicação durante Provas . . . . .	9
2.3	Restrição Temporais . . . . .	9
2.4	Restrição Orçamentais . . . . .	9
2.5	Restrição Segurança de Dados e Privacidade . . . . .	10
<b>3</b>	<b>Requisitos de Qualidade</b>	<b>11</b>
<b>4</b>	<b>Domínio do Projeto e Contextualização</b>	<b>13</b>
4.1	Domínio . . . . .	13
4.2	Contexto . . . . .	13
<b>5</b>	<b>Estratégia de Solução</b>	<b>15</b>
5.1	Decomposição Funcional . . . . .	15
5.2	Arquitetura . . . . .	16
5.3	Tecnologia . . . . .	18
<b>6</b>	<b>Vista <i>Building Block</i></b>	<b>19</b>
6.1	<i>Building Block View</i> . . . . .	19
6.2	Diagramas de Classe . . . . .	19
6.2.1	Gestão de Contas . . . . .	19
6.2.2	Disseminação de Eventos . . . . .	20
6.2.3	Calendarização . . . . .	20
6.2.4	Gestão de Provas . . . . .	21
6.2.5	<i>Frontend</i> . . . . .	22
6.3	Diagrama de Entidades Relacionais . . . . .	22
<b>7</b>	<b>Vista <i>Runtime</i></b>	<b>24</b>
7.1	Registrar Docentes . . . . .	24
7.2	Registrar Alunos . . . . .	26
7.3	Autenticar . . . . .	27
7.4	Editar Perfil . . . . .	27
7.5	Criar Prova . . . . .	28
7.6	Criar Questões . . . . .	29
7.7	Editar Prova . . . . .	31
7.8	Editar Questões . . . . .	32
7.9	Consultar detalhes da Prova . . . . .	32

7.10 Partilhar Prova . . . . .	33
7.11 Responder a Prova . . . . .	34
7.12 Classificar respostas . . . . .	35
7.13 Publicar classificações da prova . . . . .	37
7.14 Consultar prova corrigida . . . . .	37
7.15 Gerir Salas . . . . .	38
7.16 Aceder às notificações . . . . .	39
<b>8 Vista Deployment</b>	<b>41</b>
8.1 Docker Containers . . . . .	41
8.2 Amazon AWS EC2 . . . . .	41
8.3 AMAZON AWS RDS for MySQL . . . . .	42

# Listas de Figuras

3.1	Diagrama "Quality Tree". . . . .	12
4.1	Diagrama de Contexto. . . . .	14
5.1	Diagrama de blocos . . . . .	15
5.2	Arquitetura inicial . . . . .	17
5.3	Arquitetura final . . . . .	18
5.4	Tecnologias usadas . . . . .	18
6.1	Building block view . . . . .	19
6.2	Diagrama de classe do micro-serviço "Gestão de Contas" . . . . .	20
6.3	Diagrama de classe do micro-serviço "Disseminação de Eventos" . . . . .	20
6.4	Diagrama de classe do micro-serviço "Calendarização" . . . . .	21
6.5	Diagrama de classe do micro-serviço "Gestão de Provas" . . . . .	21
6.6	Diagrama de classe do FrontEnd . . . . .	22
6.7	Diagrama de Entidades Relacionais . . . . .	23
7.1	Diagrama de sequência do use case “Registrar Docentes” . . . . .	25
7.2	Diagrama de sequência do use case “Registrar Alunos”. . . . .	26
7.3	Diagrama de sequência do use case “Autenticar”. . . . .	27
7.4	Diagrama de sequência do use case “Editar Perfil”. . . . .	28
7.5	Diagrama de sequência do use case “Criar Prova”. . . . .	29
7.6	Diagrama de sequência do use case “Criar Questões”. . . . .	30
7.7	Diagrama de sequência do use case “Editar Prova”. . . . .	31
7.8	Diagrama de sequência do use case “Editar Questões”. . . . .	32
7.9	Diagrama de sequência do use case “Consultar detalhes da Prova”. . . . .	33
7.10	Diagrama de sequência do use case “Partilhar Prova”. . . . .	34
7.11	Diagrama de sequência do use case “Responder a Prova”. . . . .	35
7.12	Diagrama de sequência do use case “Classificar respostas”. . . . .	36
7.13	Diagrama de sequência do use case “Publicar classificações da prova”. . . . .	37
7.14	Diagrama de sequência do use case “Consultar prova corrigida”. . . . .	38
7.15	Diagrama de sequência do use case “Gerir Salas”. . . . .	39
7.16	Diagrama de sequência do use case “Aceder às notificações”. . . . .	40
8.1	Diagrama de <i>deployment</i> . . . . .	41

## **Lista de Tabelas**

5.1	Tabela de Ligações . . . . .	16
5.2	Tabelas de Requisitos Funcionais . . . . .	16

# 1. Introdução e Objetivos

## 1.1 Introdução

Este relatório serve de suporte à entrega da segunda fase do trabalho prático. Nesta etapa crucial do projeto **Probum**, embarcamos na definição e elaboração da arquitetura que fundamentará o desenvolvimento e implementação do nosso sistema. A arquitetura, sendo a espinha dorsal do **Probum**, será delineada com o intuito de orientar as decisões de *design* e proporcionar uma visão clara da estrutura global do sistema.

Este relatório abordará oito pontos fundamentais:

1. **Introdução e Objetivos:** O presente capítulo, com uma breve abordagem ao conteúdo do relatório.
2. **Restrições do Projeto:** Identificaremos as restrições e limitações que podem influenciar a arquitetura, considerando fatores como recursos disponíveis, tecnologias específicas e prazos definidos.
3. **Requisitos de Qualidade:** Abordaremos os requisitos de qualidade que orientarão o *design* da arquitetura, destacando critérios como desempenho, segurança, escalabilidade e eficiência.
4. **Domínio do Projeto e Contextualização:** Delimitaremos o contexto e o alcance da arquitetura, especificando as interações do **Probum** com o ambiente circundante e definindo claramente os limites do sistema.
5. **Estratégia de Solução:** Apresentaremos a estratégia de solução, indicando as abordagens e técnicas que serão adotadas para atender aos requisitos e objetivos previamente estabelecidos.
6. **Vista *Building Block*:** Descreveremos a visão dos blocos de construção, identificando os principais componentes e módulos que compõem a arquitetura do **Probum**.
7. **Vista *Runtime*:** Apresentaremos a visualização dinâmica da arquitetura, destacando como os diferentes componentes interagem durante a execução do sistema.
8. **Vista *Deployment*:** Exploraremos a visão de implementação, descrevendo como os elementos da arquitetura são distribuídos e implantados no ambiente de execução.

Ao explorar estes tópicos, delinear-se-á uma visão clara e abrangente da arquitetura do **Probum**, estabelecendo as bases para um sistema sólido, inovador e alinhado com as necessidades específicas já delineadas.

## 1.2 Objetivos do Projeto

Para responder a estes desafios, propomos o desenvolvimento de um produto, o **Probum**, que tenha algumas características diferenciadoras e até inovadoras, que aborda estes problemas, relacionados com a realização de provas de avaliação, de forma integrada. O **Probum** permitirá:

- **Criação de provas de avaliação digitais:** Criar provas de avaliação personalizadas será simples e fácil e permitirá incorporar questões de diferentes tipos, temas e níveis de dificuldade. Deve ser possível criar questões alternativas, com respostas também diferenciadas, se for conveniente cada Aluno ter uma prova diferente das dos restantes colegas.
- **Calendarização inteligente das provas:** Utilizando ferramentas matemáticas (e.g., algoritmos de otimização), o **Probum** poderá gerar automaticamente um calendário para uma dada prova de avaliação, tendo em consideração os espaços disponíveis, o número de Alunos inscritos, e as necessidades específicas de cada unidade curricular ou curso. Se uma dada prova tiver, por exemplo, 96 Alunos inscritos e a sala para a realização dessa prova só tiver capacidade para 20 Alunos, então têm que ser organizadas pelo menos 5 rondas de realização do teste. Este facto pode obrigar a que as provas não sejam iguais para todos os Alunos, como atrás se referiu.
- **Realização das provas:** Um Aluno poderá realizar as provas de avaliação de forma digital, via uma plataforma dedicada, proporcionando assim uma experiência simples, robusta, e flexível. Esta plataforma será disponibilizada em espaços e equipamentos da própria IES, garantindo todos os requisitos exigidos (e.g., autenticidade, confidencialidade, equidade, duração, plágio). Deve ser analisada e explorada a possibilidade de compaginar a realização de provas em computadores da IES com provas realizadas em computadores dos Alunos, desde que se mantenham todas as garantias de confidencialidade e autenticidade.
- **Correção:** Na fase de correção, todas as questões de resposta fechada serão facilmente avaliadas e pontuadas de forma automática. O **Probum** deverá poder ser estendido com componentes baseados em técnicas de processamento de linguagem natural, para auxiliar a correção de questões de resposta livre. Se tal funcionalidade não estiver disponível, cabe ao Docente pontuar essas questões abertas.
- **Sustentabilidade:** A substituição do papel pelo produto contribuirá para a sustentabilidade ambiental. Além disso, evitam-se sobras de papel, i.e., as cópias da prova de avaliação que se tiraram a mais devido ao facto de o número de Alunos que apareceram para a realizar ser menor que o esperado.

Se assumirmos que:

- um curso (de 3 anos) tem 6 unidades curriculares em cada semestre (ou seja, 36 unidades curriculares);
- 90% das provas de avaliação desse curso podem ser realizadas de forma digital;
- em cada unidade curricular se realizam 3 provas de avaliação, com uma média de 100 Alunos por prova;
- cada Aluno consome 4 folhas de papel A4 em cada prova de avaliação em que participa;

Então, em cada ano letivo, são poupadadas 38.800 folhas de papel por curso, se for usado o produto **Probum**. Se cada impressão custar 0,05 EUR, então há uma poupança de quase 4.000 EUR (assume-se que cada folha é impressa em ambos os lados). Se uma universidade tiver o equivalente a 100 cursos destes, então a poupança cifra-se em 400.000 EUR anuais. Trata-se de um número bem expressivo que mostra a relevância deste produto.

A isto há ainda que acrescentar os ganhos de tempo de correção, aspeto que não pode ser negligenciado e que consome muito do tempo de um professor nos períodos, por vezes longos, em que tem de corrigir provas.

- **Consulta das provas:** Assim que todas as classificações de uma dada prova forem divulgadas, o Docente poderá permitir que cada Aluno consulte a sua prova. Cada prova (que possa ser consultada) ficará disponível, no perfil do respetivo Aluno, para consulta, durante dois anos, altura em que poderá fazer sentido eliminá-la.

## 2. Restrições do Projeto

O desenvolvimento da aplicação Probum para otimização do processo de avaliação em instituições de ensino superior (IES) exige uma análise meticulosa das restrições fundamentais que guiam sua conceção e implementação. Este capítulo explora de maneira abrangente as diretrizes que moldam o caminho do projeto, abordando especificamente as restrições relacionadas à infraestrutura, isolamento durante as provas, limitações temporais, orçamentais e requisitos rigorosos de segurança de dados e privacidade.

### 2.1 Restrição da Infraestrutura

A aplicação Probum deve ser desenvolvida de forma a executar na infraestrutura atual da respetiva IES. Isso implica que todos os componentes de software necessários devem estar instalados em máquinas da instituição, permitindo que todas as funcionalidades da plataforma para os alunos operem plenamente nas máquinas designadas para as provas de avaliação. Esta restrição visa evitar investimentos adicionais em novos equipamentos, garantindo a compatibilidade com a infraestrutura existente.

### 2.2 Restrição da Isolamento da Aplicação durante Provas

Durante a realização de provas de avaliação, o computador disponibilizado a cada aluno deve permitir apenas o acesso à aplicação Probum. Isso visa evitar que os alunos recorram a outras aplicações, como *email*, *navegadores web*, *WhatsApp*, *Skype*, entre outras, durante o exame. A aplicação deve ser projetada levando em consideração essa restrição, assegurando que, enquanto a prova estiver em andamento, não seja possível aceder a nenhuma outra aplicação que não seja o Probum.

### 2.3 Restrição Temporais

O desenvolvimento do projeto segue um cronograma rigoroso, com a entrega inicial do documento até *22 de dezembro de 2023*. Essa restrição temporal é necessária para avaliar o progresso do projeto, abrangendo a contextualização e a definição dos requisitos na primeira fase.

### 2.4 Restrição Orçamentais

O orçamento total para o desenvolvimento do projeto é fixado em *20 000€*, distribuídos ao longo de um período de 4 meses. Esse montante inclui os salários dos onze engenheiros de software envolvidos no projeto, além das despesas relacionadas à aquisição de um domínio e de um computador para hospedar os dados da aplicação.

Ao considerar essas restrições, busca-se garantir a viabilidade técnica, operacional e financeira do projeto Probum, assegurando que as soluções propostas estejam alinhadas com as necessidades e recursos disponíveis.

## **2.5 Restrição Segurança de Dados e Privacidade**

A aplicação Probum deve aderir a padrões rigorosos de segurança de dados e privacidade. Todos os dados dos alunos, incluindo informações pessoais e resultados de avaliações, devem ser armazenados de forma segura e em conformidade com as leis de proteção de dados vigentes. Mecanismos de autenticação robustos devem ser implementados para garantir o acesso autorizado apenas a pessoal autorizado, protegendo assim a confidencialidade das informações.

### 3. Requisitos de Qualidade

Os requisitos de qualidade — requisitos não funcionais — desempenham um papel crucial na tomada de decisões de arquitetura, moldando diretamente o desenho e a implementação do sistema. Neste capítulo, exploraremos os objetivos de qualidade que orientarão a arquitetura do **Probum**. Estes requisitos representam critérios essenciais para avaliar o desempenho e a eficácia do sistema.

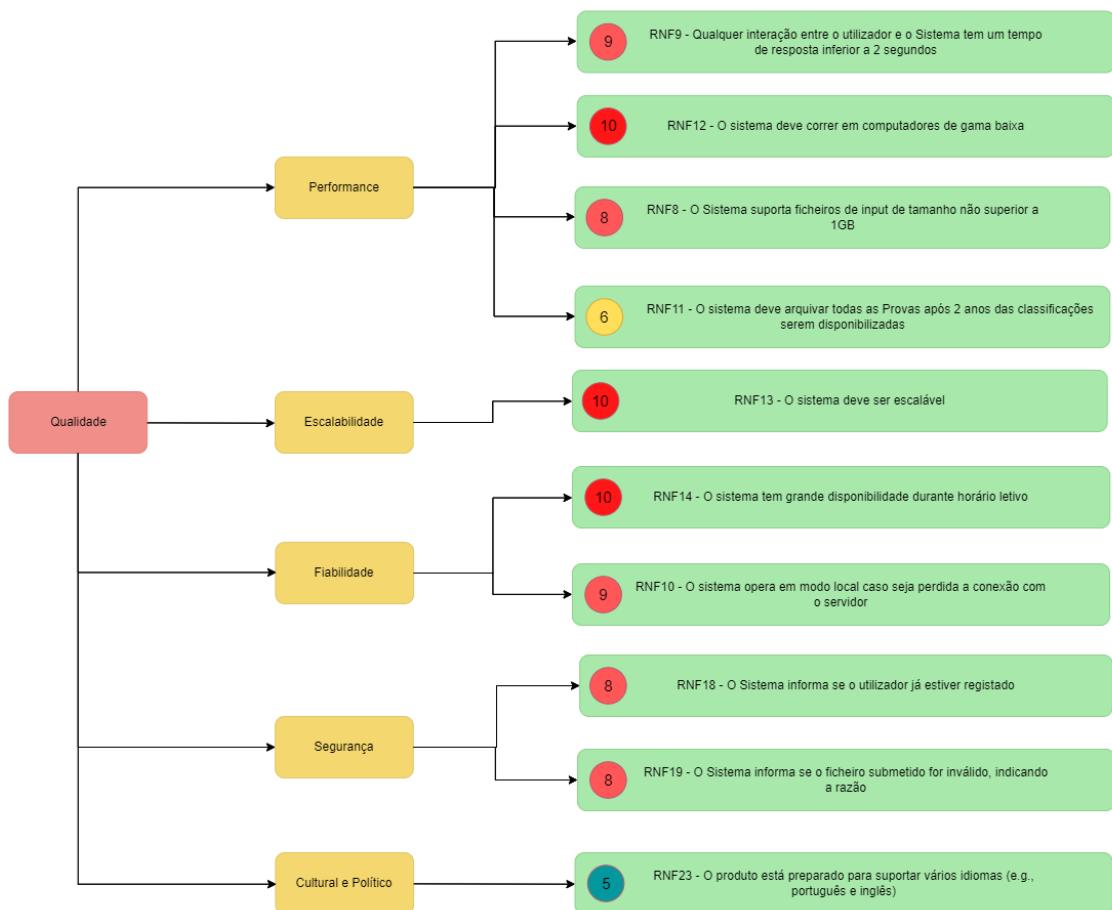
Para melhor compreensão e organização, os requisitos não funcionais podem ser apresentados de forma hierárquica mediante uma *quality tree*, destacando as metas prioritárias. Ao delinearmos claramente estes requisitos, proporcionamos um alicerce sólido para as decisões a tomar, assegurando que o **Probum** não apenas atende às expectativas funcionais, mas também cumpre padrões elevados de desempenho, segurança, escalabilidade e eficiência. Aqui, decidimos destacar os requisitos de qualidade que assumem a mais alta prioridade na conceção e desenvolvimento da arquitetura do **Probum**, e organizá-los hierarquicamente.

Começamos por selecionar 10 dos requisitos que achamos mais relevantes no contexto do projeto para tomar decisões de arquitetura, e organizamos os mesmos numa *quality tree*, indicando a sua prioridade numa escala de 0 a 10, associando uma características de arquitetura que os representem.

Desta forma, as qualidades que a equipa considerou mais relevantes para o sistema foram a fiabilidade, a performance, a segurança e a escalabilidade. De acordo com as prioridades definidas na figura 3.1, os requisitos não funcionais que consideramos prioritários foram os 12,13 e 14 que representam a **performance**, **escalabilidade** e **fiabilidade** respetivamente. Se confrontarmos os conceitos de um programa **monolítico** com uma arquitetura de **micro-serviços**, as duas últimas qualidades referidas beneficiam mais com o uso de **micro-serviços**. Outra das grandes razões pela qual foi decidido o uso deste tipo de *design* arquitetural está relacionada com a modularidade do mesmo, sendo que facilita bastante o trabalho numa equipa de maior dimensão e a divisão do mesmo.

## Need → Drivers → Crítico para a qualidade

Difícil de medir ← → Fácil de medir



Geral ← → Específico

**Figura 3.1:** Diagrama "Quality Tree".

## 4. Domínio do Projeto e Contextualização

### 4.1 Domínio

Este documento surge na sequência do contacto do Reitor da Universidade de Vigo (Espanha) visando criar um produto de *software* que permita a realização de provas de avaliação académicas.

Este produto, que se designa **Probum**, permite que alunos de uma dada unidade curricular de um curso universitário ou politécnico (i.e. do ensino superior) realizem as suas provas académicas, utilizando as infraestruturas informáticas da sua própria instituição de ensino superior (IES), mesmo que estas sejam muito limitadas quanto à sua dimensão, disponibilidade e capacidade. Assim, o **Probum** deve incluir requisitos funcionais que permitam a sua utilização em diversas IES e permitir alguma parametrização e configuração. No essencial, o **Probum** permite que os professores criem provas de avaliação e que as calendarizem, que os alunos as realizem (devidamente calendarizada) e que essas provas sejam corrigidas, tendencialmente de forma automática.

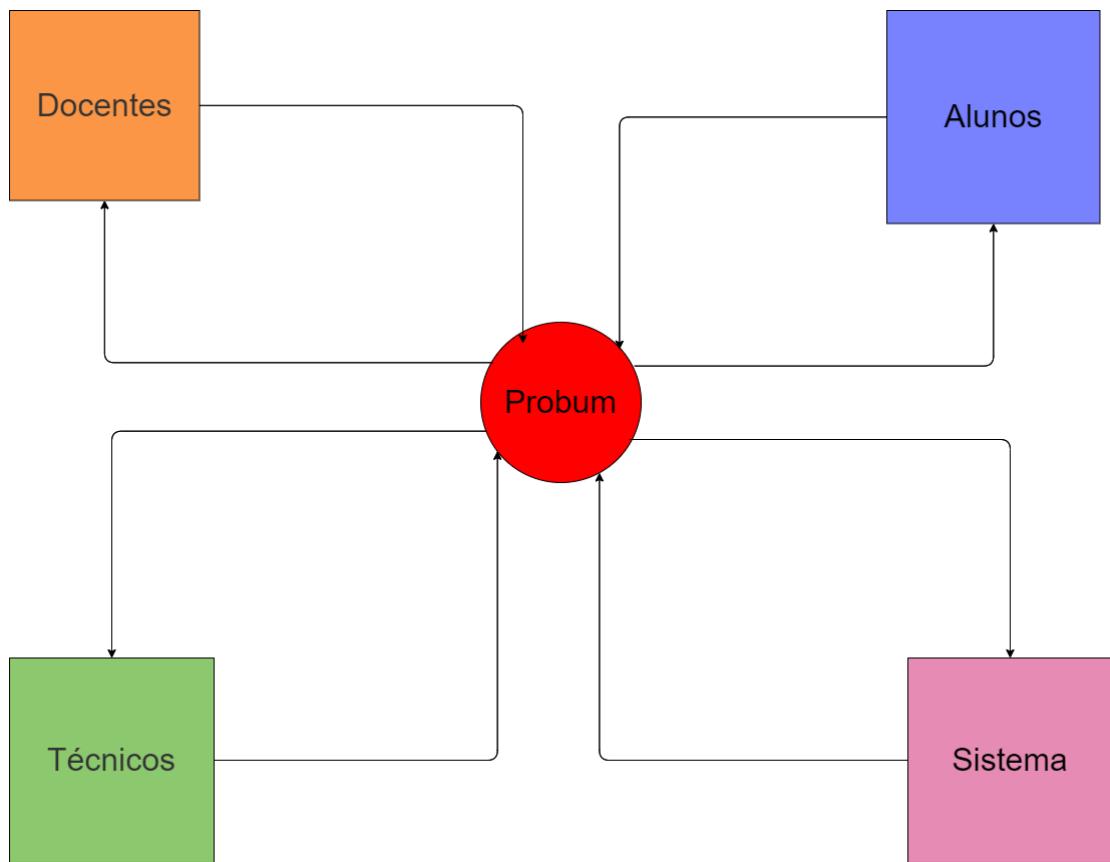
### 4.2 Contexto

Nos últimos anos, tem-se assistido a uma crescente integração da tecnologia no ambiente educativo e esse fenómeno não passa despercebido no ensino superior. Para otimizar processos e melhorar a experiência de Alunos e Docentes, muitas organizações exploram a digitalização de várias atividades, incluindo a realização de provas de avaliação.

Um exemplo interessante desta tendência é aquele que se observa em algumas provas de aferição do ensino básico português, que tentam explorar o potencial de transformação que as soluções digitais podem proporcionar.

No entanto, aplicar no imediato estes métodos numa instituição de ensino superior levanta novos desafios, essencialmente relacionados com as limitações das estruturas informáticas. O crescente número de Alunos inscritos em muitas unidades curriculares é também um obstáculo à aplicação desses métodos. Dentro desta realidade, destacam-se os seguintes desafios:

- **Gestão de espaços e recursos:** A realização de provas de avaliação para inúmeros Alunos coloca uma pressão significativa sobre os espaços disponíveis. As salas revelam-se muitas vezes insuficientes, resultando em desafios logísticos na calendarização e alocação destes espaços.
- **Eficiência na correção:** A correção manual de inúmeras provas é um processo demorado e sujeito a erros humanos. Fazê-la, especialmente em cursos com muitos Alunos, torna-se ineficiente e onerosa em termos de tempo e custo.
- **Sustentabilidade ambiental:** O uso de papel na impressão e resposta às provas de avaliação resulta num desperdício muito significativo, com um impacto negativo no ambiente. A necessidade de encontrar alternativas mais sustentáveis para a realização das provas é um aspecto crítico.

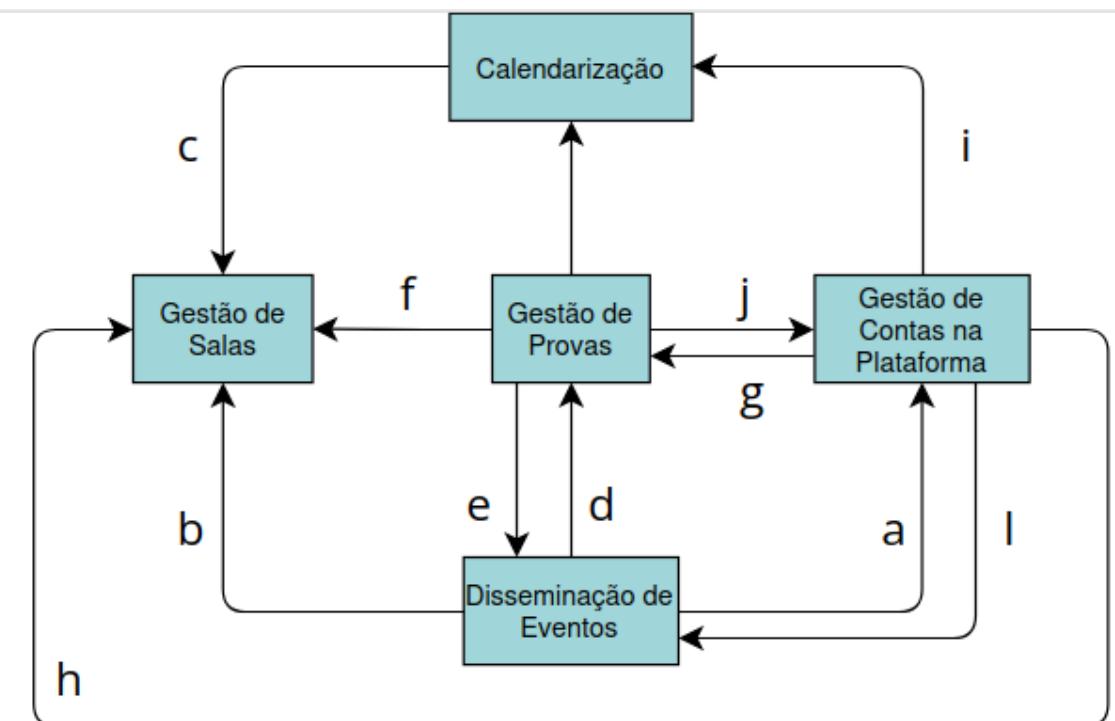


**Figura 4.1:** Diagrama de Contexto.

## 5. Estratégia de Solução

Nesta secção, expomos uma visão resumida das decisões e estratégias fundamentais que moldam a arquitetura do sistema **Probum**. Este capítulo destaca mais a fundo as escolhas tecnológicas e a decomposição de alto nível. Começamos por demonstrar a decomposição funcional do sistema, nomeadamente o **diagrama de blocos** e a alocação dos requisitos funcionais aos respetivos blocos, bem como as ligações existentes entre estes.

### 5.1 Decomposição Funcional



**Figura 5.1:** Diagrama de blocos

A Tabela 5.2 ilustra a proposta de decomposição funcional, destacando a alocação de requisitos funcionais aos respectivos blocos, enquanto a Tabela 5.1 delinea os requisitos que estabelecem as conexões entre os blocos apresentados no diagrama funcional.

Ligações	BlocoA	BlocoB	Reqs
a	Disseminacao de Eventos	Gestao de contas	req47, req61
b	Disseminacao de Eventos	Gestao de salas	req16
c	Disseminacao de Eventos	Gestao de prova	req20, req55, req56, req60, req61
d	Calendarizacao	Gestao de salas	req17
e	Gestao de provas	Disseminacao de Eventos	req54
f	Gestao de provas	Gestao das salas	req67, req21, req30, req31
g	Gestao de provas	Gestao de contas	req 19, req30, req32, req44
i	Gestao de provas	Calendarizacao	req14, req21, req30, req31
h	Gestao de contas	Gestao de provas	req62, req63, req64
j	Gestao de contas	Disseminacao de Eventos	req68

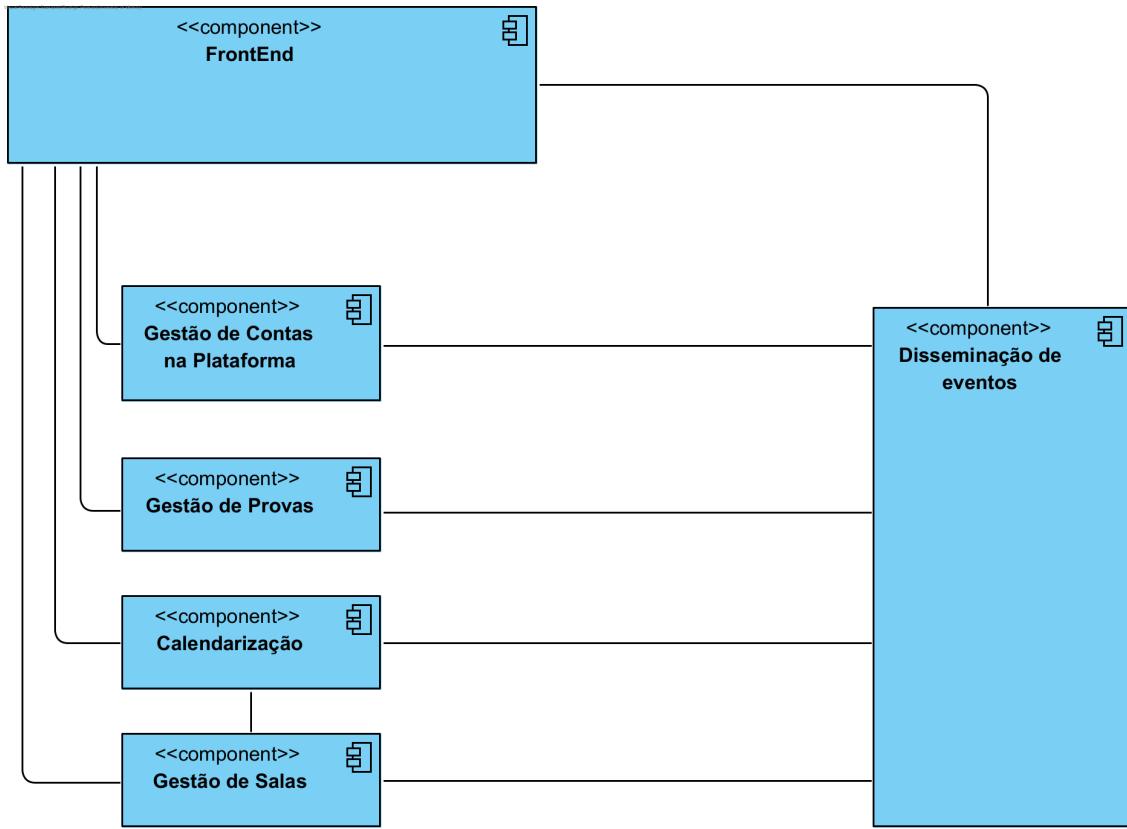
**Tabela 5.1:** Tabela de Ligações

Gestão de Contas na Plataforma
RF01, RF02, RF03, RF04, RF05, RF07, RF10, RF11, RF12, RF13, RF45, RF46, RF62, RF63, RF64
Disseminação de Eventos
RF06, RF08, RF09, RF16, RF20, RF44, RF47, RF55, RF61, RF67, RF68
Gestão de Provas
RF14, RF19, RF21, RF22, RF23, RF24, RF25, RF26, RF27, RF28, RF29, RF30, RF31, RF32, RF33, RF34, RF35, RF36, RF37, RF38, RF39, RF40, RF41, RF42, RF43, RF48, RF50, RF51, RF52, RF53, RF54, RF56, RF57, RF58, RF59, RF60
Calendarização
RF15, RF17, RF18
Gestão de Salas
RF65, RF66

**Tabela 5.2:** Tabelas de Requisitos Funcionais

## 5.2 Arquitetura

Após esta decomposição funcional, tendo em conta tudo analisado previamente, escolhemos um estilo arquitetural de micro-serviços. Foi definida a seguinte arquitetura inicial em cada componente à parte do *Frontend* corresponde a um micro-serviço:

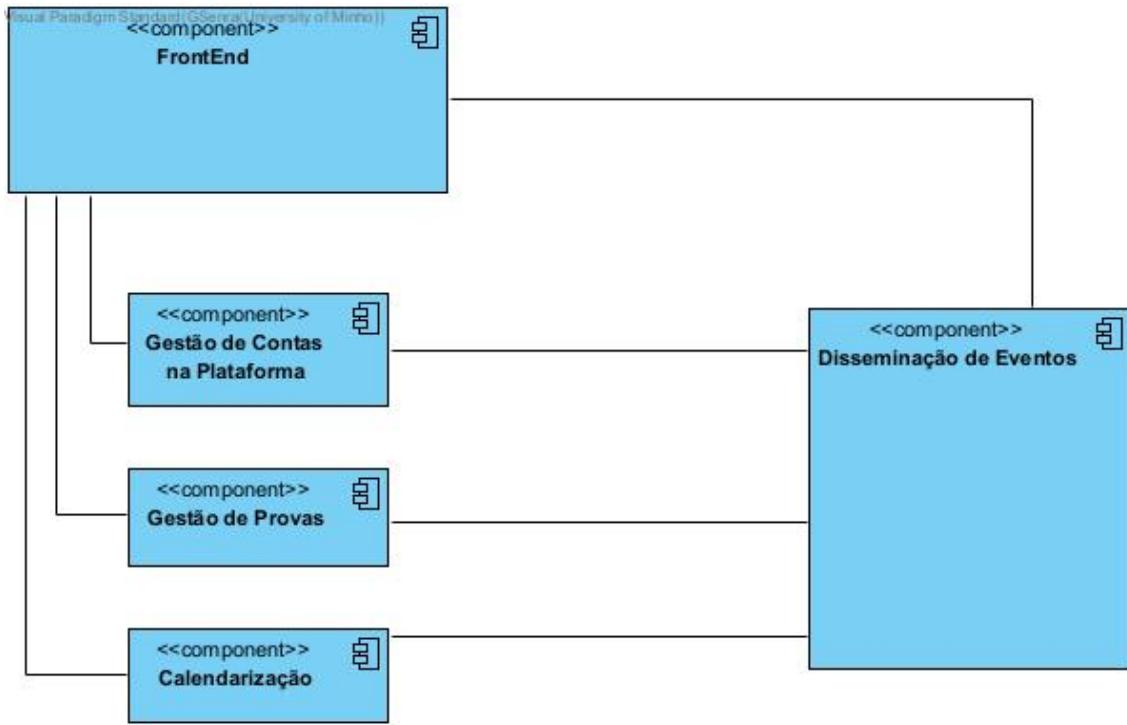


**Figura 5.2:** Arquitetura inicial

Depois, para validação da arquitetura, foi feito uma análise por use case de quantas mensagens seriam passadas entre micro-serviços, para conseguir decidir se existem microserviços que poderiam ser unidos num só. Os resultados encontram-se na seguinte tabela:

Use Cases	Iteração 1	Iteração 2
Registrar Docentes	1	1
Registrar Alunos	1	1
Autenticar	0	0
Editar Perfil	0	0
Criar Prova	8	6
Criar Questões	0	0
Editar Prova	6	4
Editar Questões	0	0
Consultar Detalhes de Prova	0	0
Partilhar Prova	1	1
Responder a Prova	0	0
Classificar Respostas	0	0
Publicar Classificações da Prova	4	4
Consultar Prova Corrigida	0	0
Gerir Salas	1	1
Aceder às Notificações	0	0

Atravez desta tabela chegamos à conclusão que o microserviço Gestao de Salas pode ser unido com o microserviço Calendarização. Este foi unido e manteve o nome Calendarização, chegando à nova arquitetura:



**Figura 5.3:** Arquitetura final

### 5.3 Tecnologia

Tomadas as decisões arquiteturais, o grupo definiu implementar todos os micro-serviços com a framework de *python Django* e o *Frontend* com a framework SvelteKit.

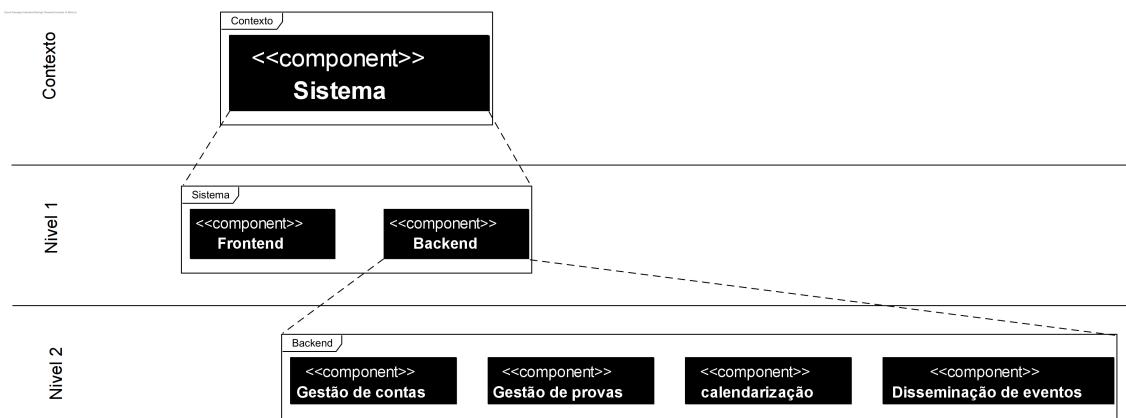


**Figura 5.4:** Tecnologias usadas

## 6. Vista *Building Block*

### 6.1 *Building Block View*

Para melhor segmentar o sistema nas suas partes atomicas foi criado o seguinte *Building Block View*:



**Figura 6.1:** Building block view

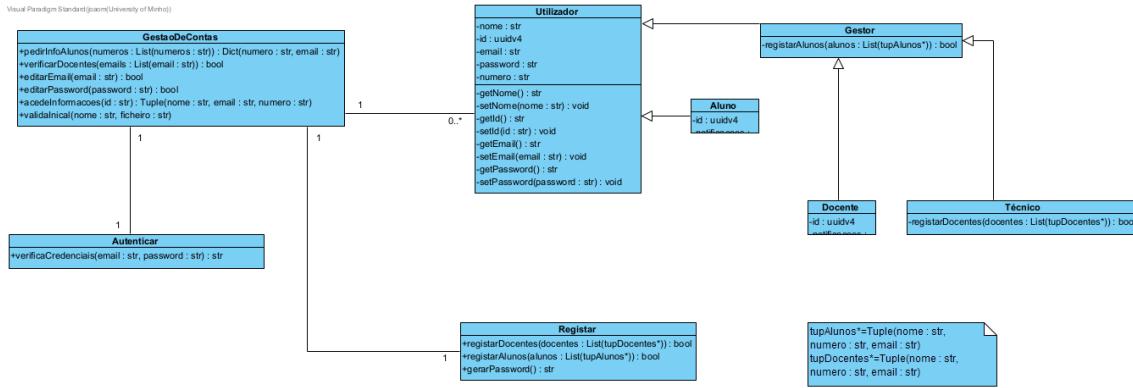
Este diagrama é composto por camadas de abstração e blocos. Esses blocos podem ser *White boxes* onde o interior do bloco corresponde ao que o compõe, e *Black boxes* que têm o seu interior escondido. As *White boxes* de cada camada correspondem às *Black boxes* que podem ser decompostas em mais blocos.

### 6.2 Diagramas de Classe

Incluímos a seguir os **diagramas de classe** dos micro-serviços e um diagrama de componentes correspondente ao *Frontend* das *Black Boxes* que não se conseguiram decompor mais no building block diagram.

#### 6.2.1 Gestão de Contas

Para possibilitar a existência de contas na plataforma, existe o micro-serviço de **Gestão de contas**. O diagrama de classes associado a este serviço é composto pela classe *GestaoDeContas* que está associada às classes *Autenticar* e *Registar*. Estas têm como funcionalidade, como o nome indica, autenticar e registrar *Utilizadores* (apenas *Alunos* e *Docentes*). *Gestor* e *Aluno* são os tipos de *Utilizadores* sendo que o *Técnico* e o *Docente* herdam os métodos do *Gestor* que, neste caso, é apenas a funcionalidade de registar alunos.



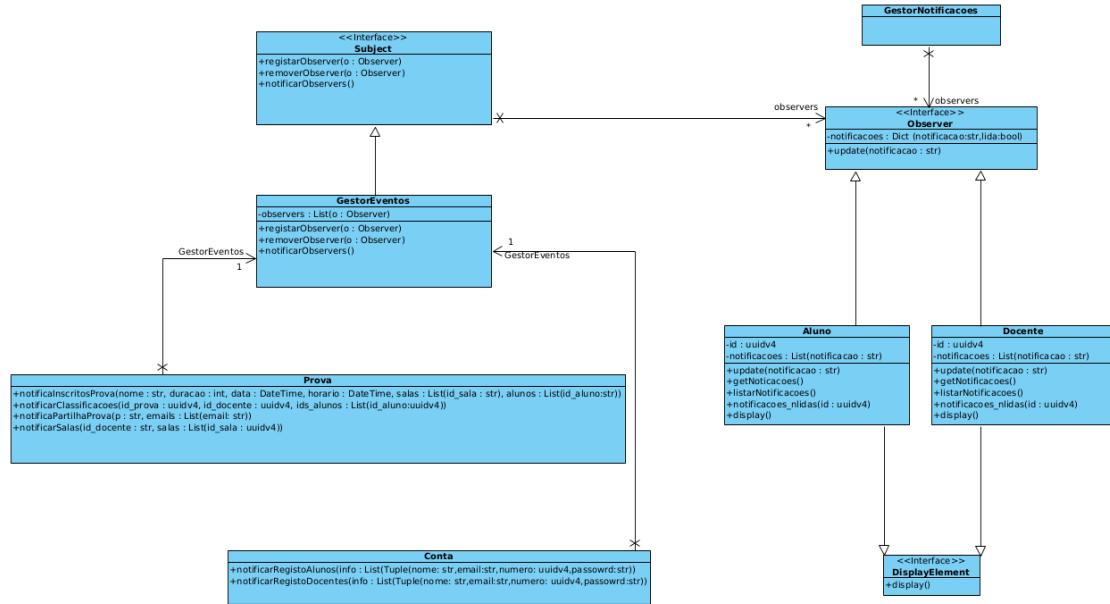
**Figura 6.2:** Diagrama de classe do micro-serviço "Gestão de Contas"

## 6.2.2 Disseminação de Eventos

O diagrama de classes do micro-serviço de **Disseminação de Eventos** é composto por um **Facade GestorNotificacoes** que contém uma lista de *Observers*.

No projeto, o padrão *Observer* é usado de forma a que este micro-serviço "oiça" os restantes e espere por notificações para dar **update** à sua lista de *Observers* que são os *Alunos* e os *Docentes*. A solução para representar o *Subject*, passou por representá-lo neste diagrama, mesmo que a sua implementação aconteça noutras micro-serviços.

Deste modo, a *Prova* irá ser um *Subject* e implementar os métodos que lhe pertencem, como por exemplo, a função de notificar os alunos inscritos para uma prova. Da mesma forma irão funcionar as *Contas*.



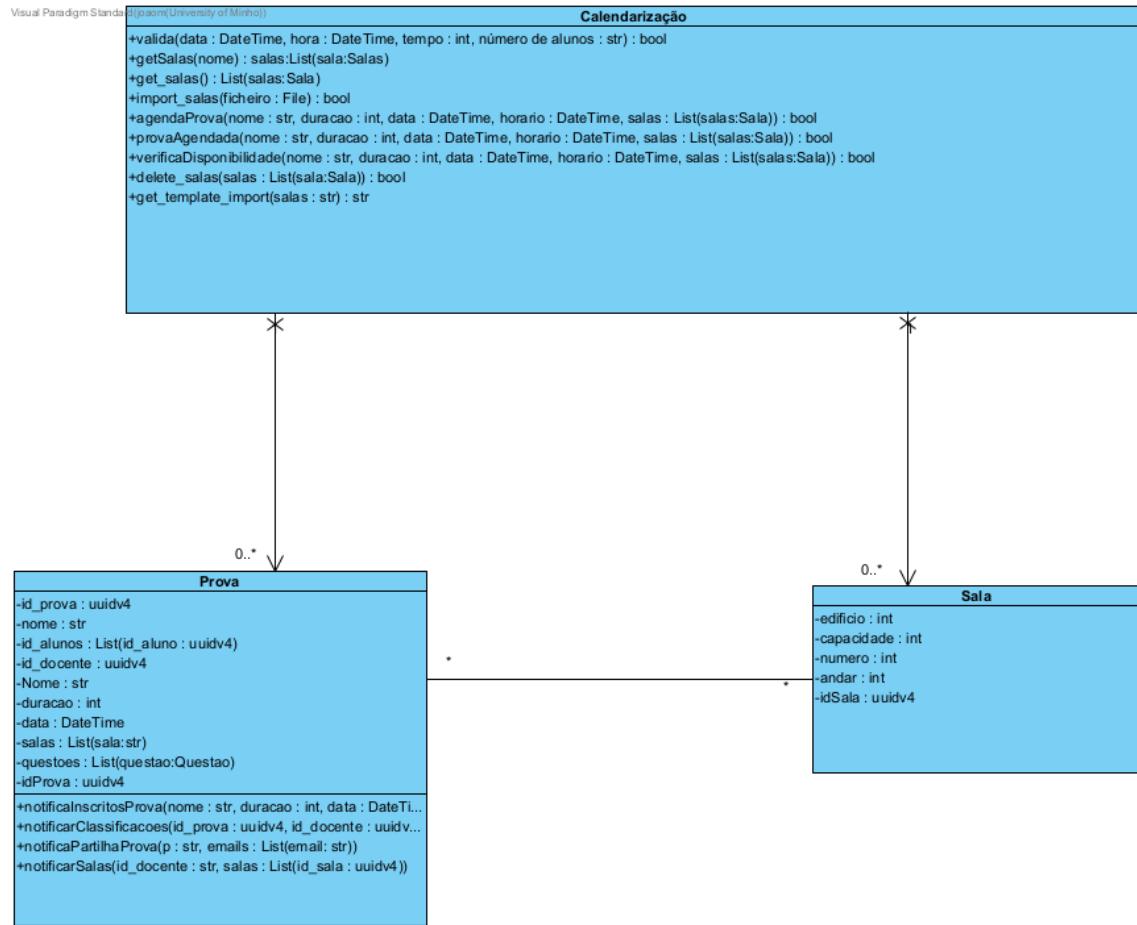
**Figura 6.3:** Diagrama de classe do micro-serviço "Disseminação de Eventos"

## 6.2.3 Calendarização

O micro-serviço da Calendarização é composto pela classe *Calendarizacao* que é composto por todos os métodos associados à gestão de salas e ao agendamento das mesmas para a realização provas que acontecem em certas datas e horários.

Esta classe principal tem como atributos uma lista de *Salas* e uma lista de *Provas*. Estas classes por sua vez possuem os seus próprios atributos como a duração de uma prova ou a capacidade

de uma sala.

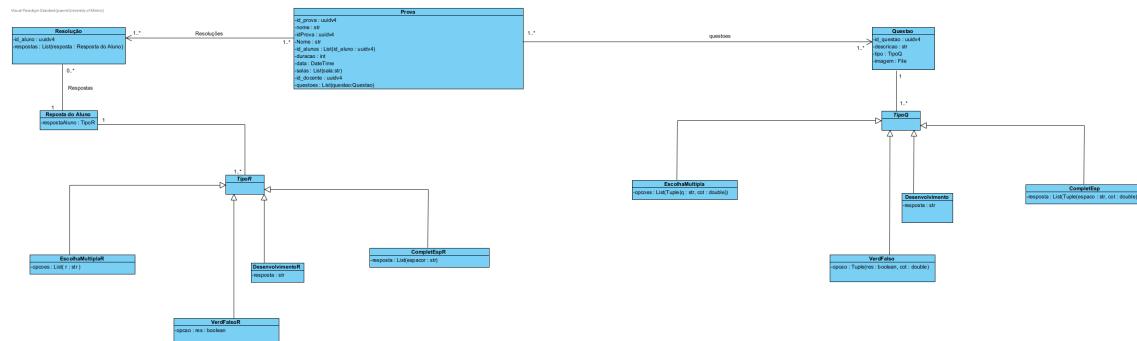


**Figura 6.4:** Diagrama de classe do micro-serviço "Calendarização"

#### 6.2.4 Gestão de Provas

A Gestão de Provas representa o último componente desta arquitetura e incorpora a classe *Prova*.

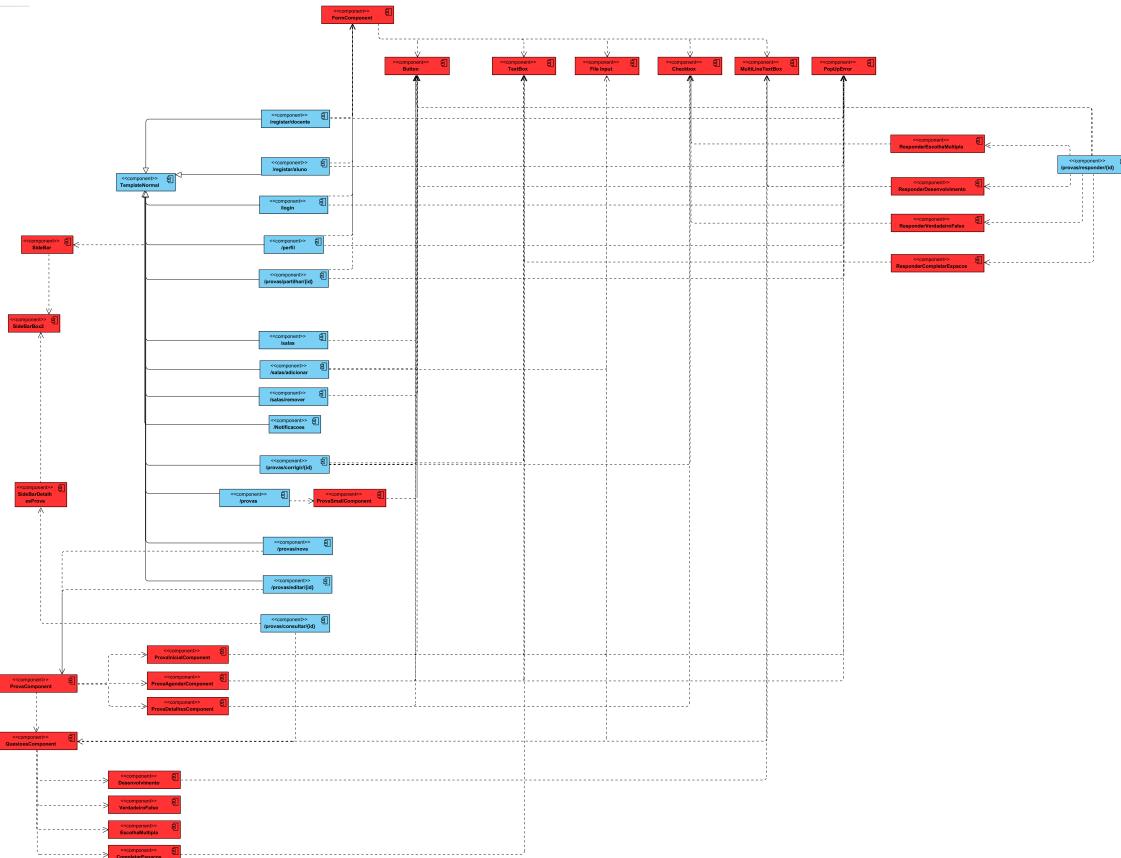
Deste modo, a *Prova* alberga todos os métodos relacionados com as possíveis ações que se podem realizar sobre uma prova, bem como todos os atributos necessários para a execução dessas ações. Cada instância da classe *Prova* representa uma coleção de *Questões*, sendo que cada questão é caracterizada pelo seu próprio tipo (*TipoQ*). Esses tipos podem variar, incluindo opções como verdadeiro e falso, ou ainda escolha múltipla.



**Figura 6.5:** Diagrama de classe do micro-serviço "Gestão de Provas"

### 6.2.5 *Frontend*

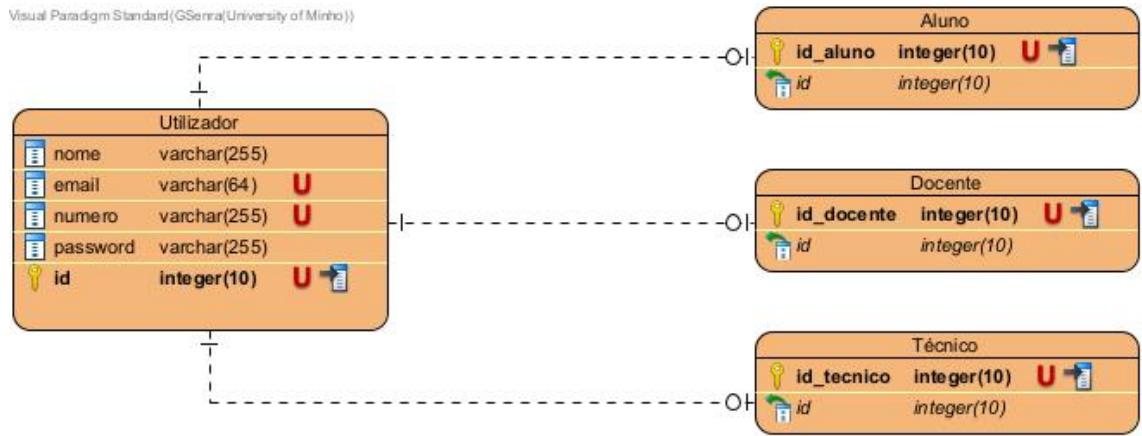
Neste diagrama podemos encontrar as diferentes partes que compõem o *frontend*. Estes dividem-se em dois tipos, as páginas (identificadas a azul) e os componentes (identificados a vermelho). A grande diferença entre estes é que as páginas são acedidas pelos links e os componentes são usados dentro das páginas. Este diagrama foi obtido através de uma análise extensiva aos use cases que se encontram no documento de requisitos.



**Figura 6.6:** Diagrama de classe do FrontEnd

## 6.3 Diagrama de Entidades Relacionais

Nesta secção apresentamos a primeira versão de um modelo relacional de base de dados, do micro-serviço *Gestão de Contas*. Nele podemos encontrar os atributos da entidade "Utilizador" e as suas relações relações com os diferentes tipos de "Utilizador" ("Aluno", "Docente" e "Técnico").



**Figura 6.7:** Diagrama de Entidades Relacionais

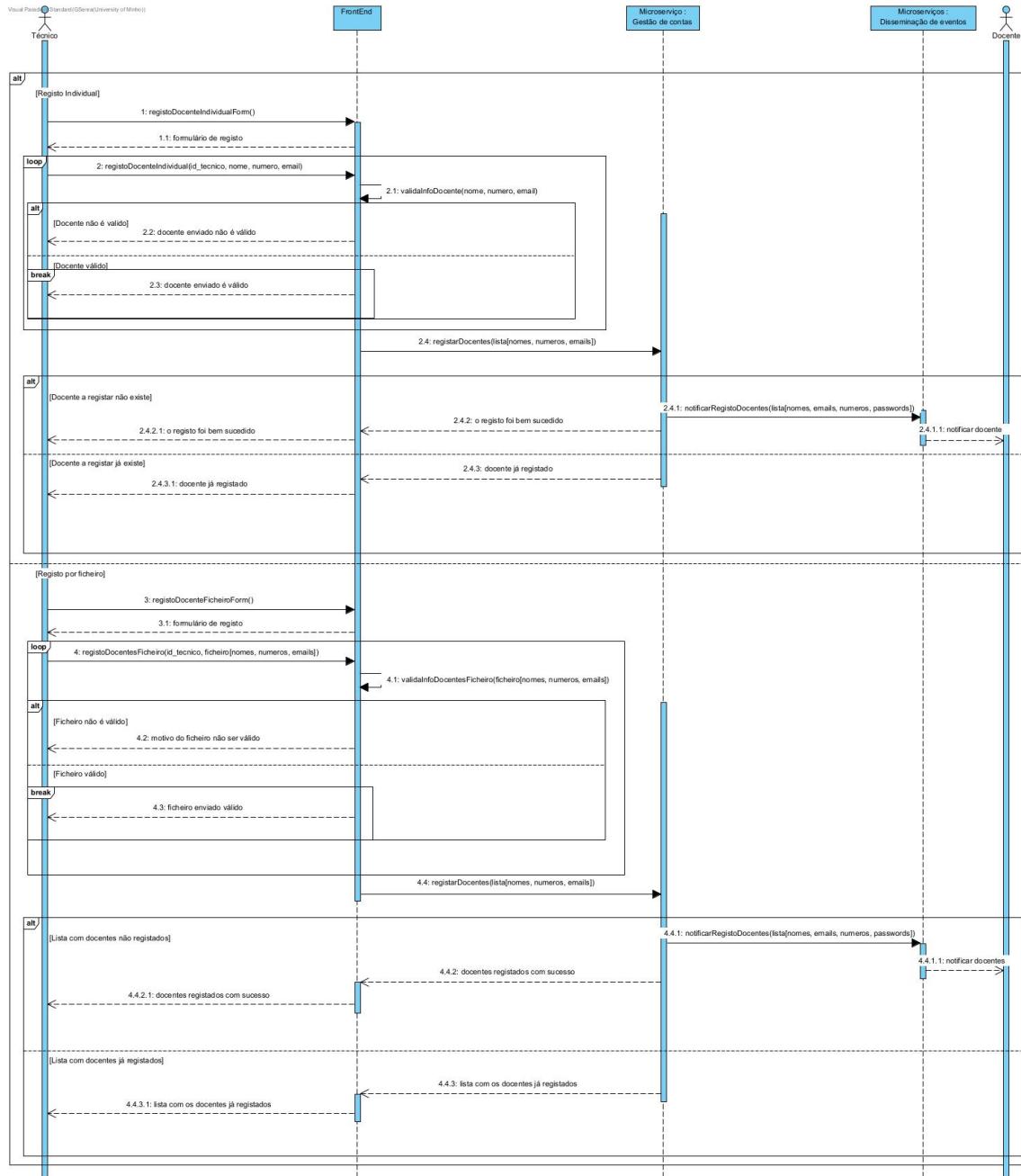
## 7. Vista *Runtime*

Neste capítulo, serão apresentados **diagramas de sequência** para explorar o comportamento dinâmico do sistema. Estes diagramas, permitem visualizar interações entre micro-serviços destacando a **ordem de execução, fluxos de mensagens e colaborações entre objetos**.

### 7.1 Registar Docentes

Neste diagrama, apresentamos o fluxo de interações para o use case “Registar Docentes” para permitir que um ator do tipo “Técnico”, consiga registar atores do tipo “Docente” individualmente ou por ficheiro no sistema.

O *FrontEnd* gera toda a parte de formulários necessários ao registo de Docentes, invocando o método *registarDocentes* do micro-serviço *Gestão de Contas* permitindo que sejam geradas todas as mensagens necessárias para este processo. Também, desencadeia a chamada do método *notificarRegistoDocentes* do micro-serviço *Disseminação de eventos* por parte do micro-serviço anterior, com vista a que cada Docente registado com sucesso, seja notificado com a sua password de acesso à plataforma, via endereço eletrónico.

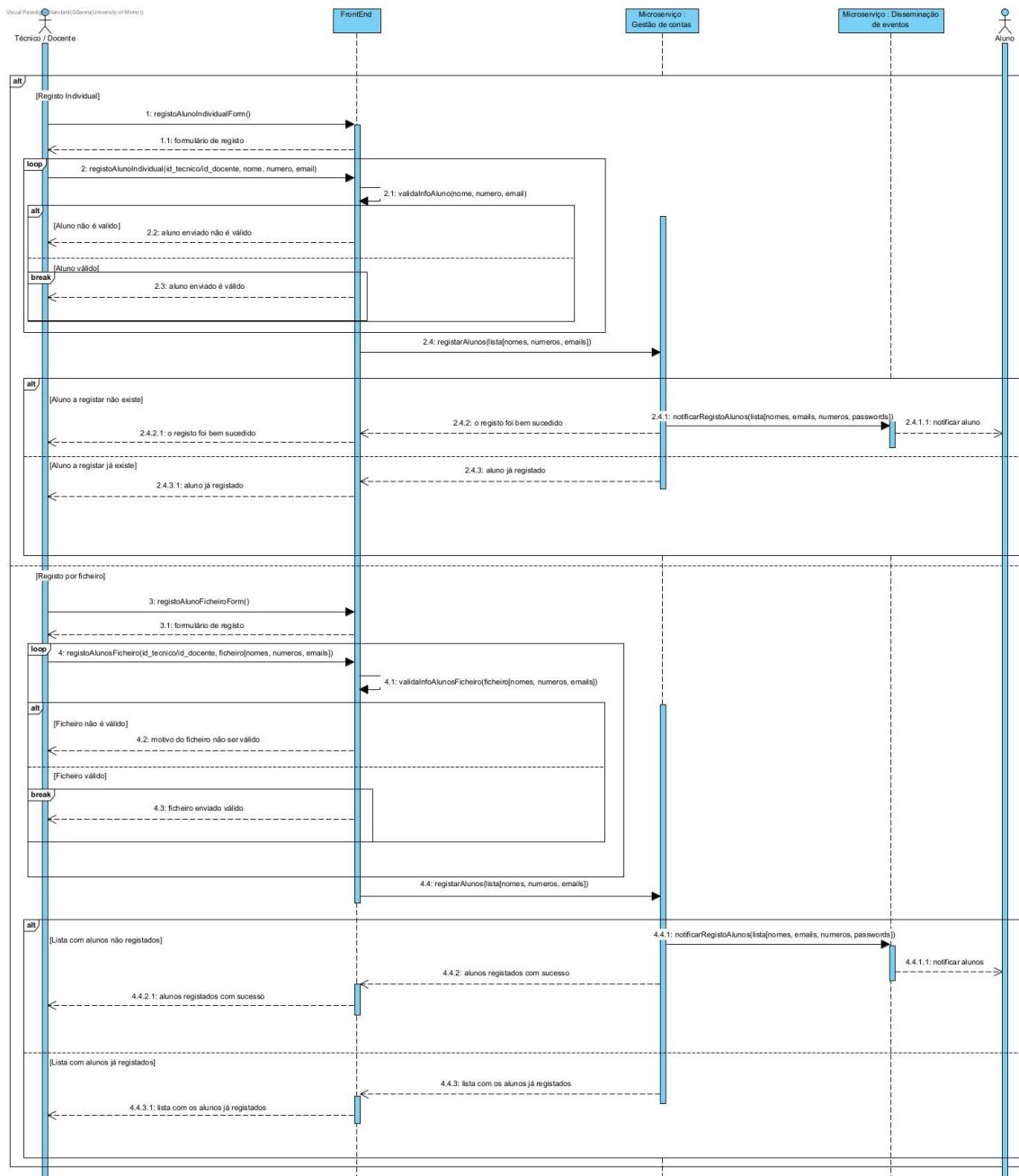


**Figura 7.1:** Diagrama de sequência do use case “Registrar Docentes”.

## 7.2 Registar Alunos

Neste diagrama, apresentamos o fluxo de interações para o use case “Registar Alunos” com vista a permitir que um ator do tipo “Técnico” ou “Docente”, consiga registar atores do tipo “Alunos” de forma individual ou por ficheiro no sistema.

O *FrontEnd* gera toda a parte de formulários necessários ao registo de alunos, invocando o método *registrarAlunos* do micro-serviço *Gestão de Contas* permitindo que sejam geradas todas as mensagens necessárias para este processo. Também desencadeia a chamada do método *notificarRegistoAlunos* do micro-serviço *Disseminação de eventos* por parte do micro-serviço anterior, para que cada Aluno registado com sucesso, seja notificado com a sua *password* de acesso à plataforma, via endereço eletrónico.

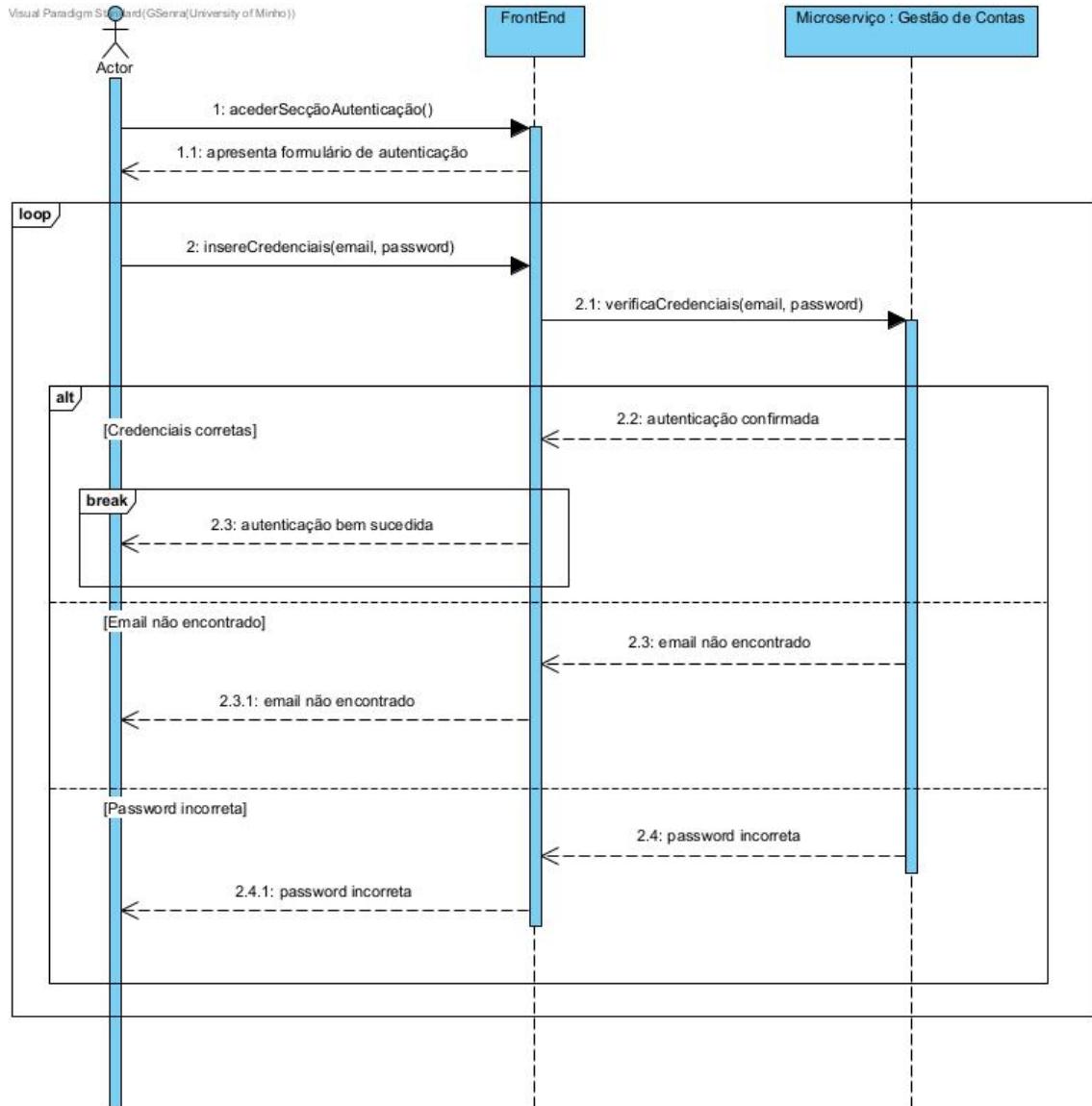


**Figura 7.2:** Diagrama de sequência do use case “Registar Alunos”.

### 7.3 Autenticar

Neste diagrama, apresentamos o fluxo de interações para o use case “Autenticar” para permitir que um ator “utilizador”, consiga aceder às funcionalidades do sistema.

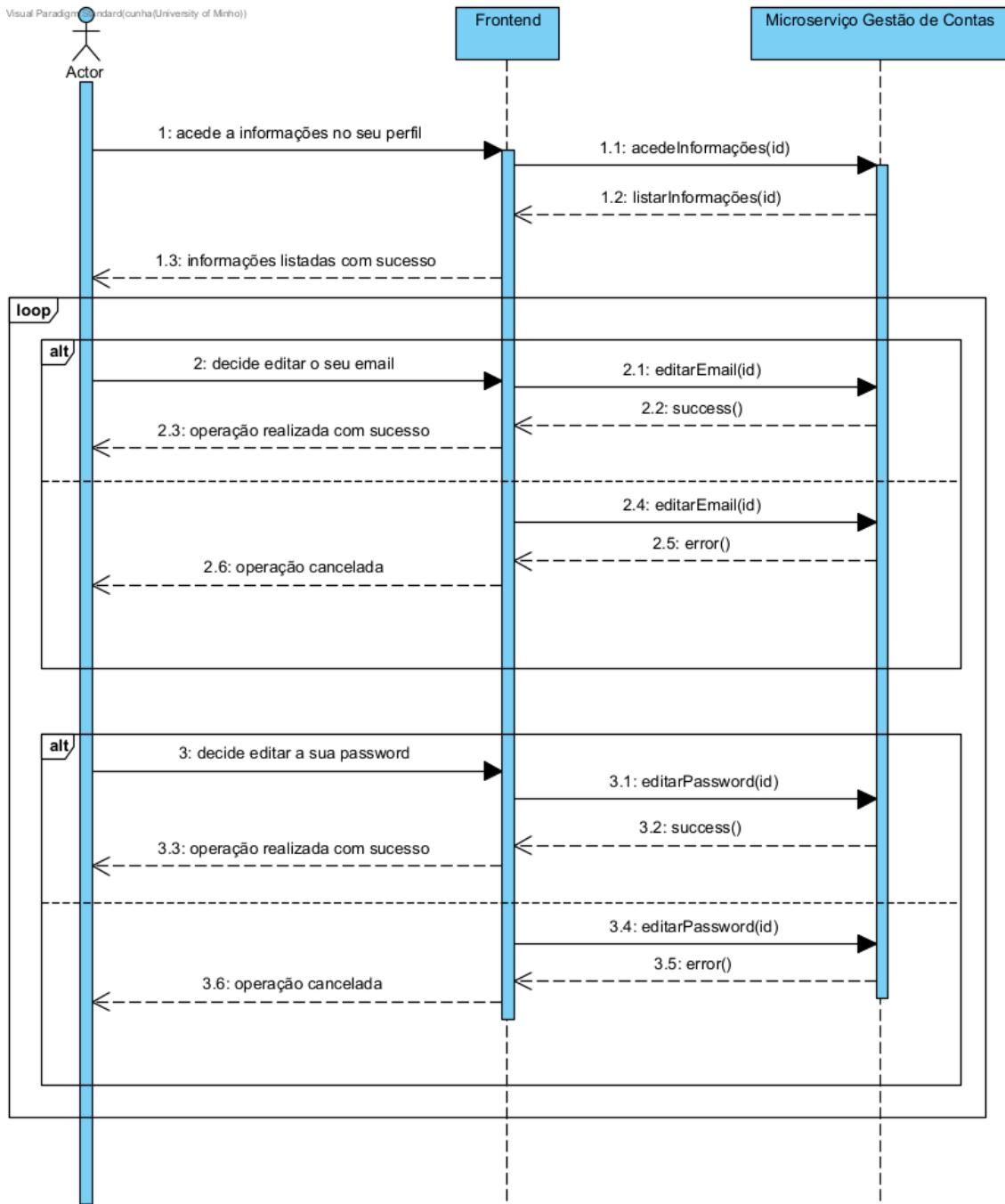
O *FrontEnd* gera toda a parte de formulários necessários à autenticação (incluindo verificação de erros) evitando que exista uma sobrecarga do lado do micro-serviço *Gestão de Contas*. No caso, de não existir erros associados é invocado o método *verificaCredenciais* para possibilitar a autenticação.



**Figura 7.3:** Diagrama de sequência do use case “Autenticar”.

### 7.4 Editar Perfil

Neste diagrama, apresentamos o fluxo de interações para o use case “Editar Perfil” de modo que um ator consiga atualizar as suas informações presentes no sistema através de interações (pedido, resposta) entre *FrontEnd* e o micro-serviço *Gestão de Contas*.



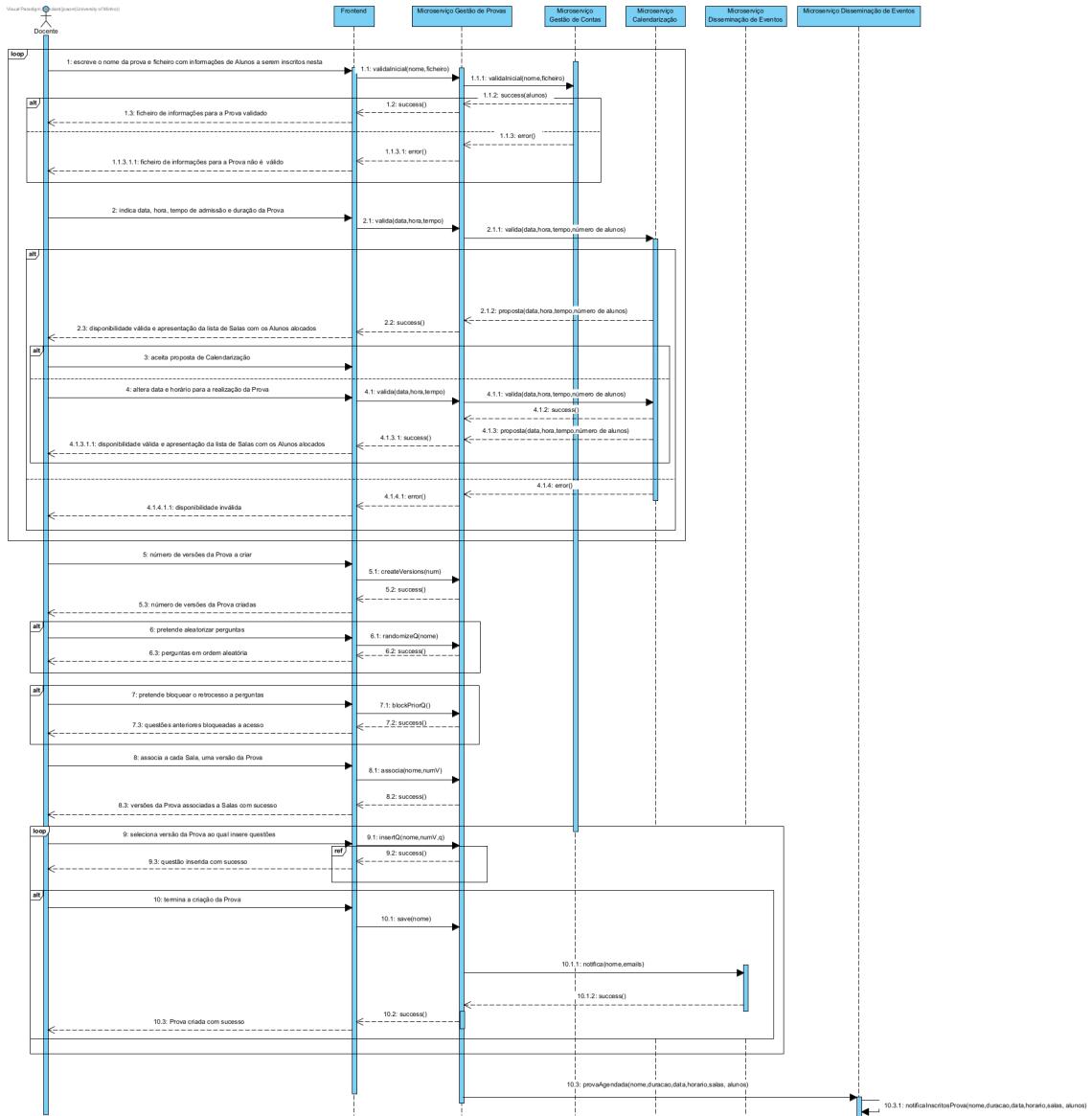
**Figura 7.4:** Diagrama de sequência do use case “Editar Perfil”.

## 7.5 Criar Prova

Neste diagrama, apresentamos o fluxo de interações para o use case “Criar Prova” que permite aos atores do tipo “Docente” introduzir detalhes básicos acerca da prova, bem como permite o registo de alunos e a criação de questões.

Acerca dos detalhes da prova, o micro-serviço *Gestão de Provas* fornece métodos que possibilitam introduzir o ficheiro dos alunos a serem inscritos e até detalhes como data, hora, duração e tempo de admissão. No seguimento da introdução destes detalhes, o micro-serviço *Calendarização* responde com uma proposta de calendarização, à qual o Docente terá de aceitar ou recusar.

Por fim, o Docente seleciona o número de versões que pretende criar para a prova, incluindo se pretende aleatorizar as perguntas e outros tipos de detalhes mais específicos à criação de prova.

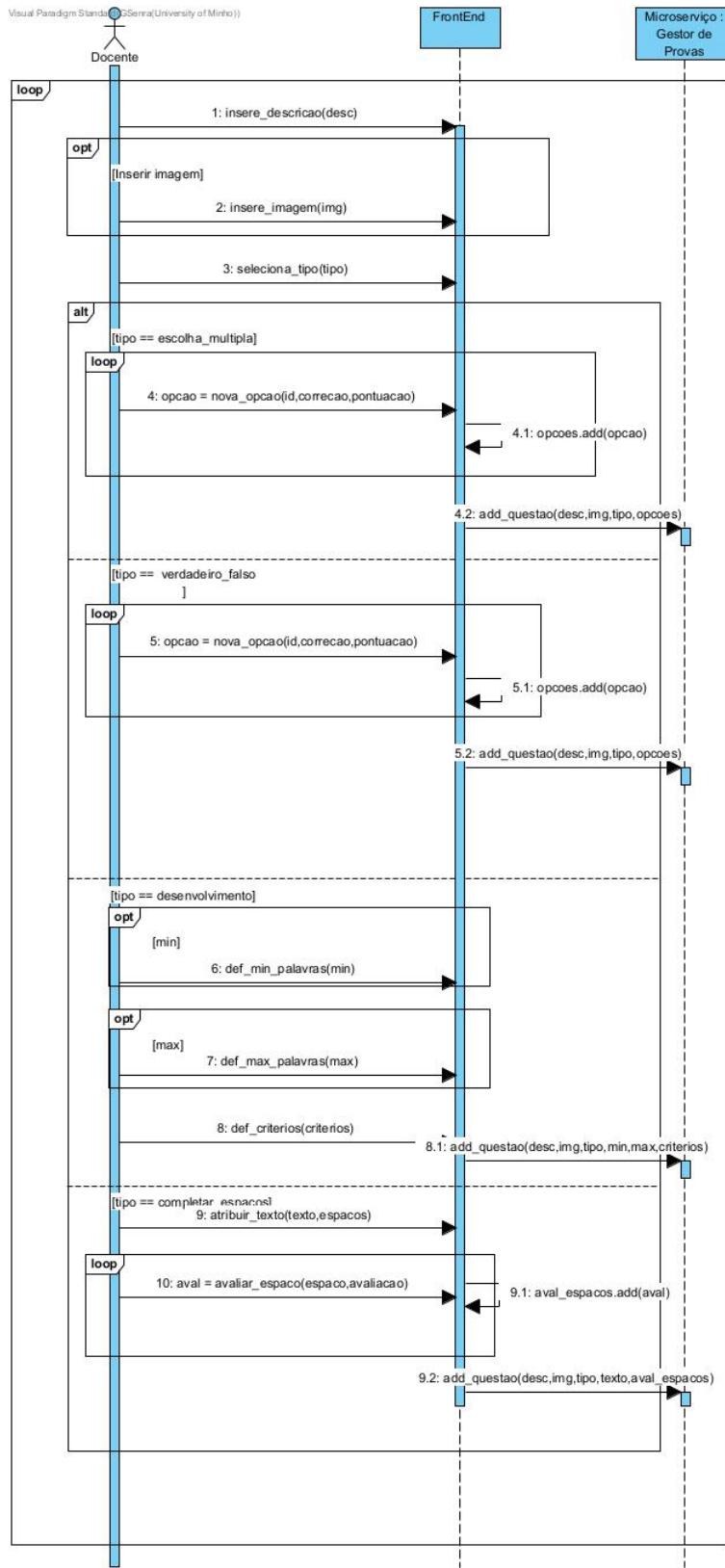


**Figura 7.5:** Diagrama de sequência do use case “Criar Prova”.

## 7.6 Criar Questões

Neste diagrama, apresentamos o fluxo de interações para o use case “Criar Questões” que se baseia na interação dos atores do tipo "Docente" com o *FrontEnd* com vista a criar uma prova com base num formulário disponibilizado pelo mesmo.

Após a conclusão de cada questão de qualquer tipo (V/F, escolha múltipla, desenvolvimento e completar espaços) é chamado o método *add\_questao* do micro-serviço *Gestão de Provas* para que a mesma seja registada, repetindo-se este processo até que o "Docente" deseje terminar a criação de perguntas para a prova.



**Figura 7.6:** Diagrama de sequência do use case “Criar Questões”.

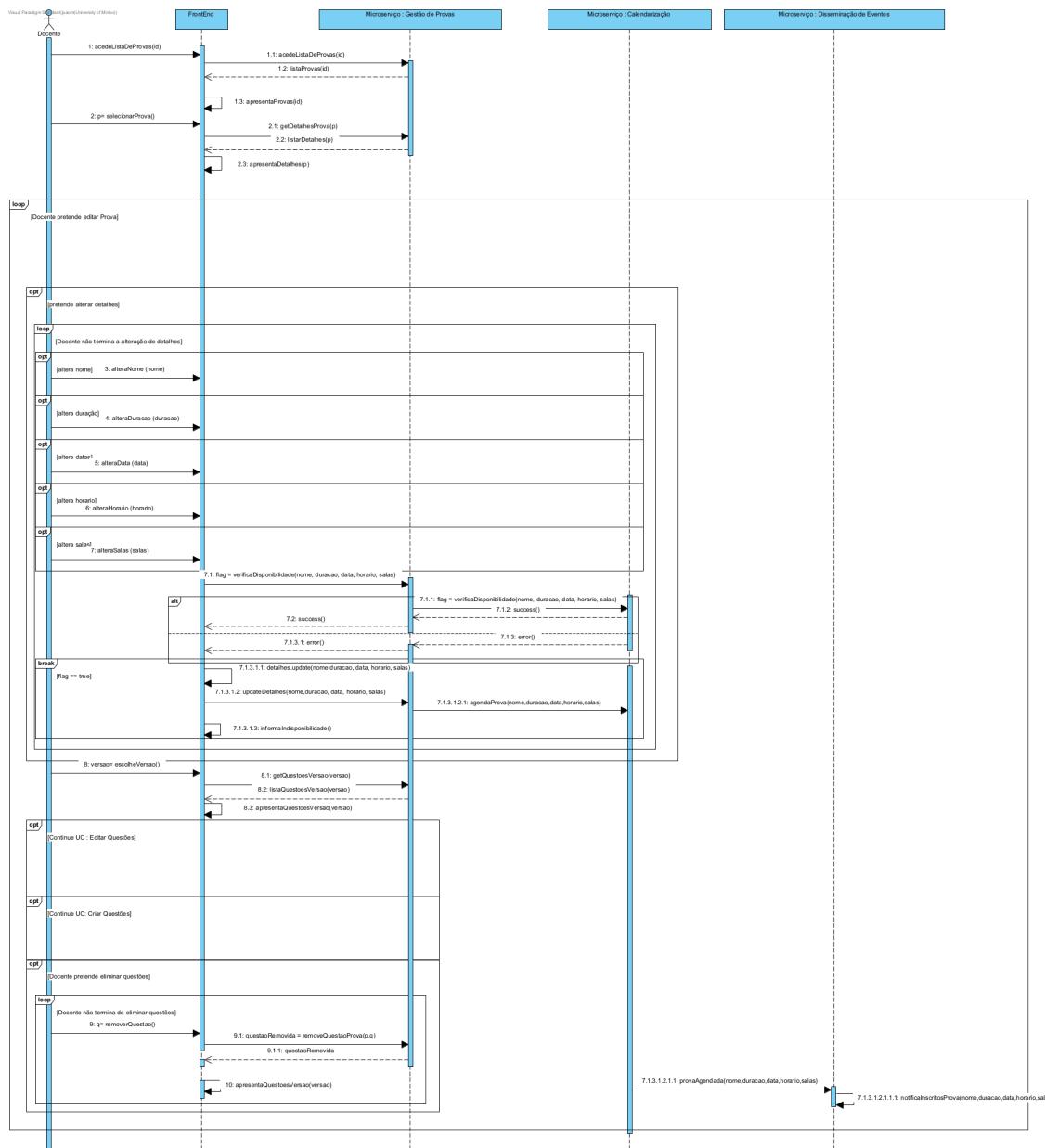
## 7.7 Editar Prova

Neste diagrama, apresentamos o fluxo de interações para o use case “Editar Prova” que se baseia na interação dos atores do tipo "Docente" com o *FrontEnd* com vista a editar uma ou mais provas já criadas.

Depois da seleção da prova a editar e da conclusão da edição da mesma, é chamado o método *updateDetalhes* do micro-serviço *Gestão de Provas* de forma a que as alterações sejam registadas, passando este por algumas verificações por exemplo o método *verificaDisponibilidade* do micro-serviço *Calenderização* que trata de validar os novos detalhes.

Após a alteração dos detalhes, o docente também tem a possibilidade de "Editar Questões" e "Criar Questões", não especificados no diagrama visto que são outros use cases e a sua especificação já é abordada noutros diagramas.

Por fim, o Docente decide se pretende eliminar Questões e é chamado o método *provaAgendada* do micro-serviço *Disseminação de Eventos* de forma a que os novos detalhes sejam atualizados e notificando os alunos inscritos na prova da alteração de salas, datas ou até horas.

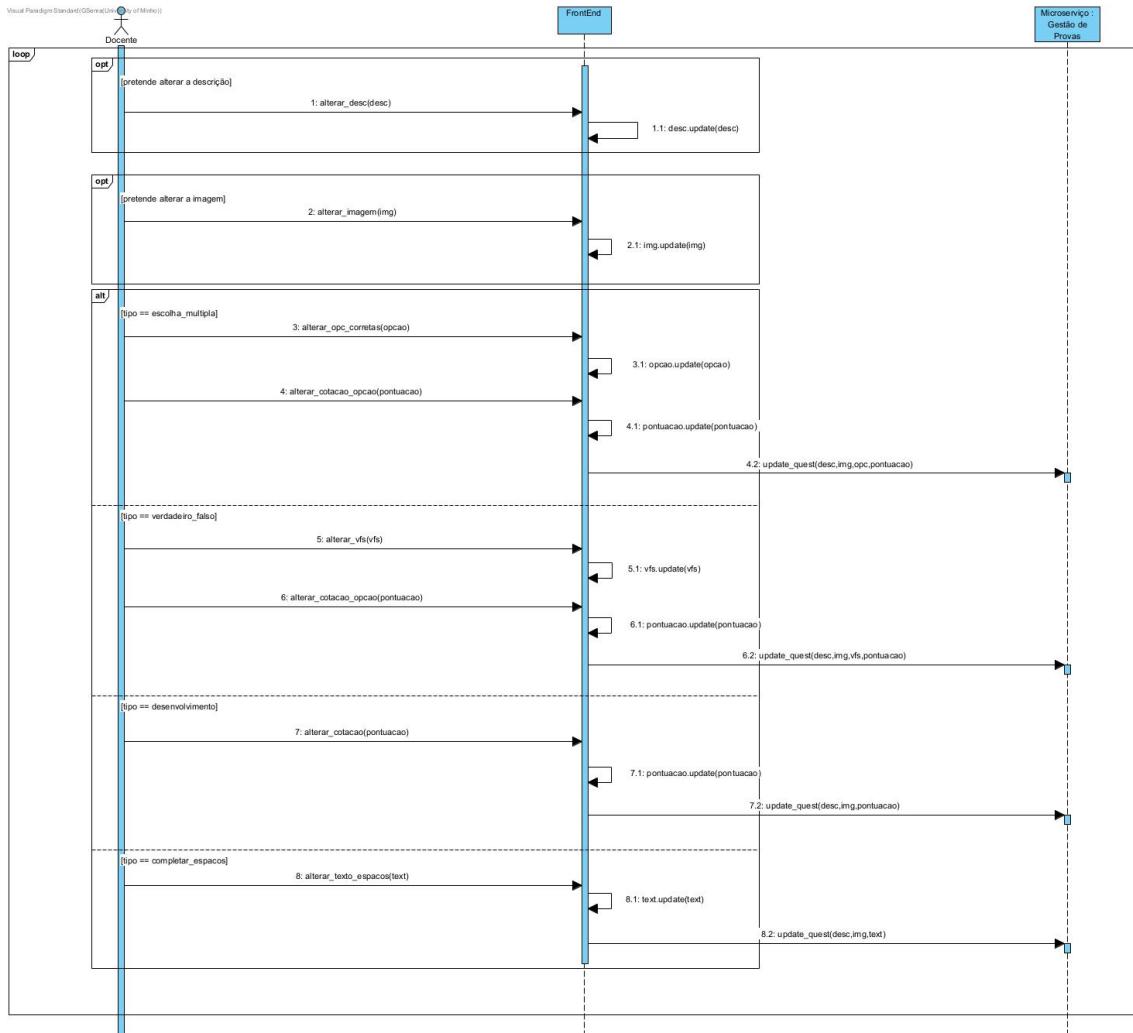


**Figura 7.7:** Diagrama de sequência do use case “Editar Prova”.

## 7.8 Editar Questões

Neste diagrama, apresentamos o fluxo de interações para o use case “Editar Questões” que se baseia na interação dos atores do tipo "Docente" com o *FrontEnd* com vista a editar perguntas de uma prova com base num formulário disponibilizado pelo mesmo.

Após a conclusão da edição de cada questão de qualquer tipo (V/F, escolha múltipla, desenvolvimento e completar espaços) é chamado o método *update\_quest* do micro-serviço *Gestão de Provas* para que a alteração seja registada, repetindo-se este processo até que o Docente deseje terminar a edição de perguntas para a prova.

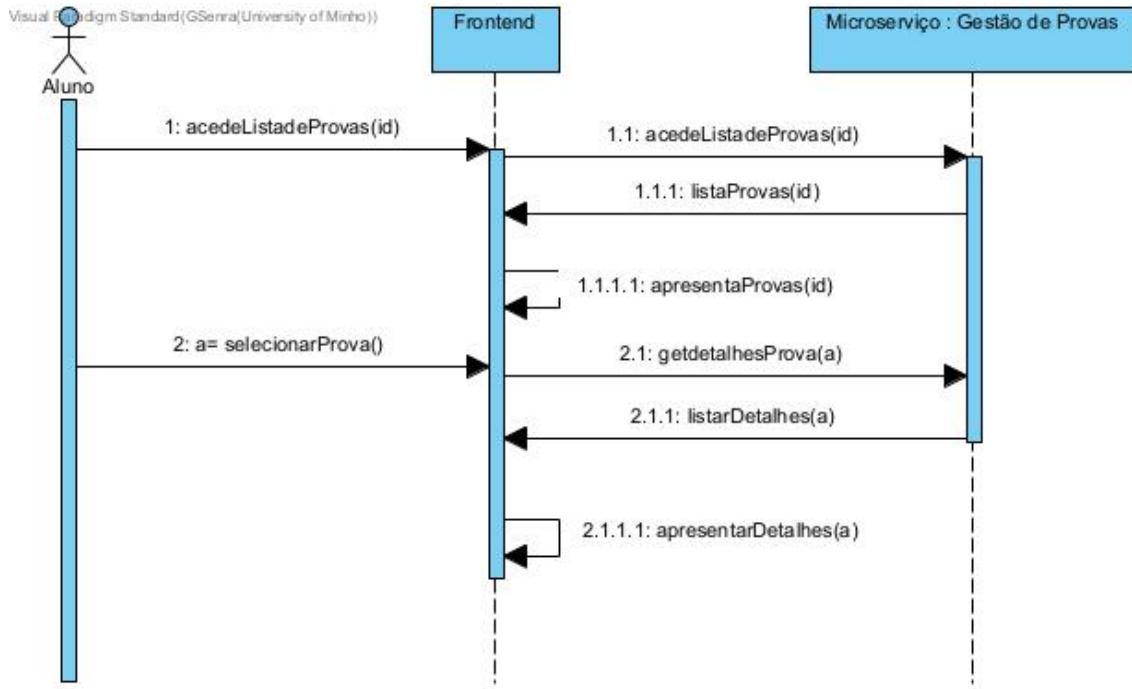


**Figura 7.8:** Diagrama de sequência do use case “Editar Questões”.

## 7.9 Consultar detalhes da Prova

Neste diagrama, apresentamos o fluxo de interações para o use case “Consultar detalhes da Prova” que permite os atores do tipo "Aluno" verificar os detalhes relativos a uma Prova a que têm acesso.

O *FrontEnd* é capaz de listar e apresentar os detalhes das provas a que um está associado através da invocação de métodos presentes no micro-serviço *Gestão de Provas*.

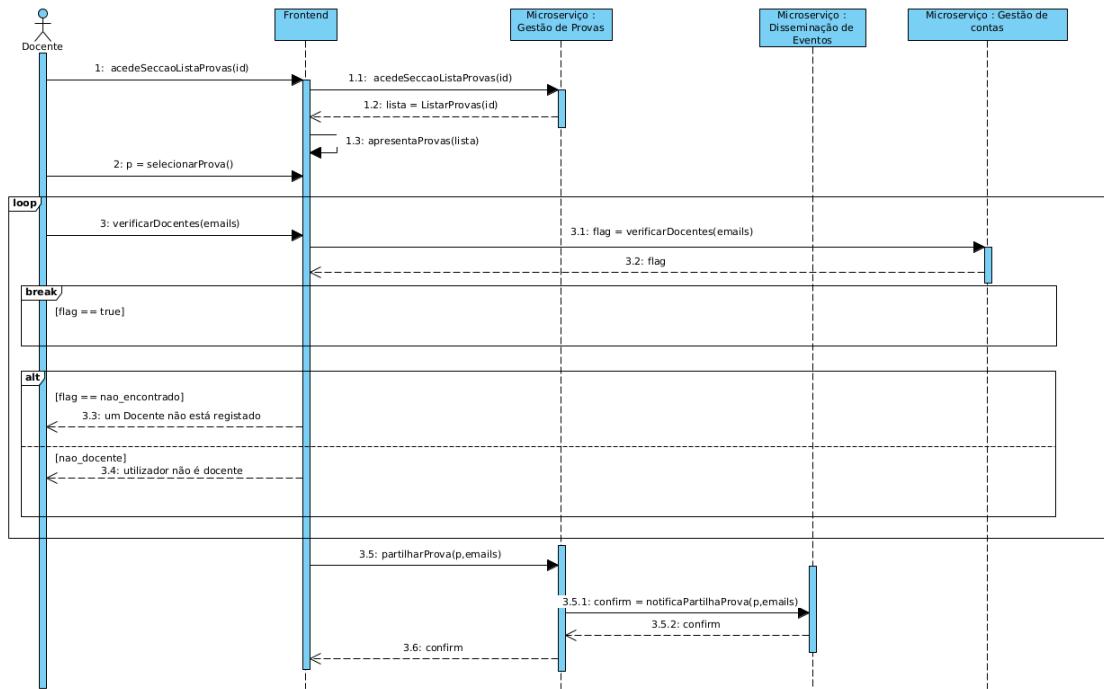


**Figura 7.9:** Diagrama de sequência do use case “Consultar detalhes da Prova”.

## 7.10 Partilhar Prova

Neste diagrama, apresentamos o fluxo de interações para o use case “Partilhar Prova” com o objetivo de partilhar uma prova com outros docentes.

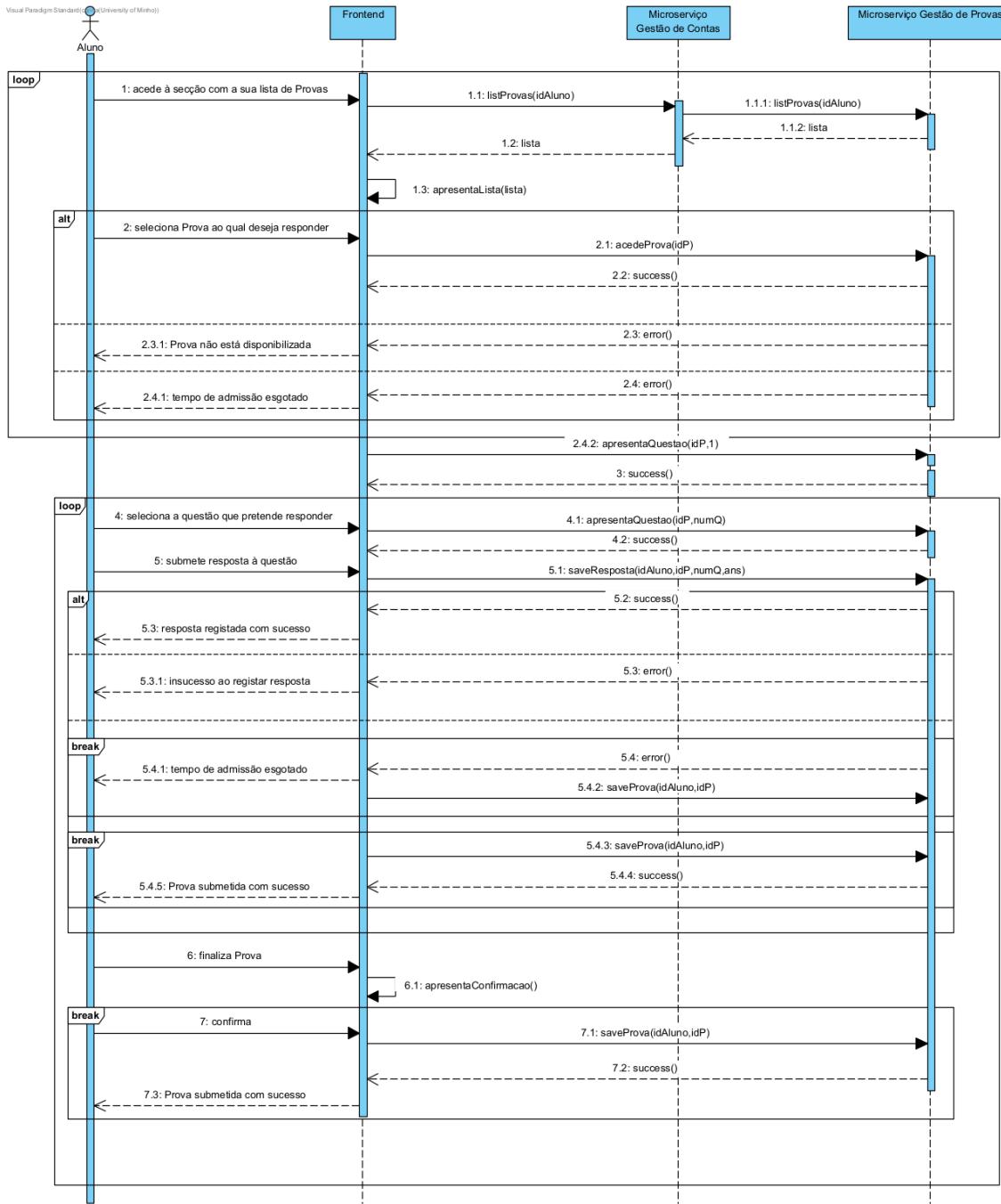
Olhando a esta premissa, de um modo geral, existe invocação do método *verificarDocentes* para verificar se o Docente indicado existe. O método *partilharProva* do micro-serviço *Gestão de Prova* torna a prova pública para os "docentes" que foram introduzidos anteriormente, gerando uma notificação que é enviada para os "docentes" em questão.



**Figura 7.10:** Diagrama de sequência do use case “Partilhar Prova”.

## 7.11 Responder a Prova

Neste diagrama, apresentamos o fluxo de interações para o use case “Responder a Prova” que se baseia na interação dos atores do tipo "Aluno" com o *FrontEnd* com vista a responder a uma das provas ao qual este tem acesso. Após a conclusão de cada resposta ou da prova no seu total, é chamado o método *saveResposta* e *saveProva* respetivamente do micro-serviço *Gestão de Provas* para que a submissão seja registada.

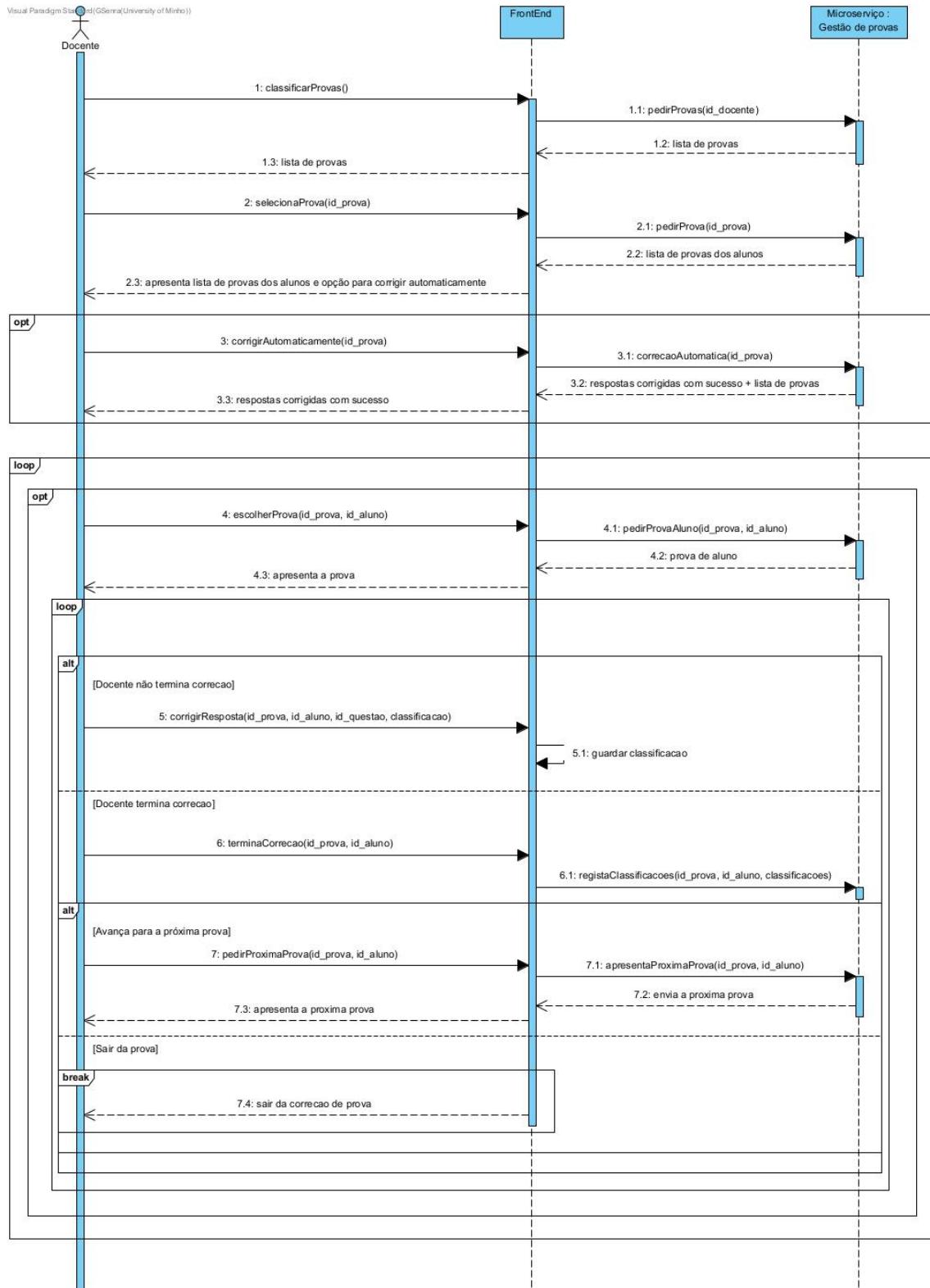


**Figura 7.11:** Diagrama de sequência do use case “Responder a Prova”.

## 7.12 Classificar respostas

Neste diagrama, apresentamos o fluxo de interações para o use case “Classificar respostas” de modo que um ator do tipo “Docente” tenha o poder de publicar as classificações dos alunos quando as provas já estão todas corrigidas.

De um modo geral, são invocados os métodos *pedirProvas*, *pedirProva*, *correcaoAutomatica*, *pedirProvaAluno*, *registaClassificacoes* e *apresentaProximaProva* do micro-serviço *Gestão de provas* por parte do *FrontEnd* com base na necessidade ocorrente do Docente, visando que existe condições opcionais para o Docente nas correções e ordens de execução obrigatórias.

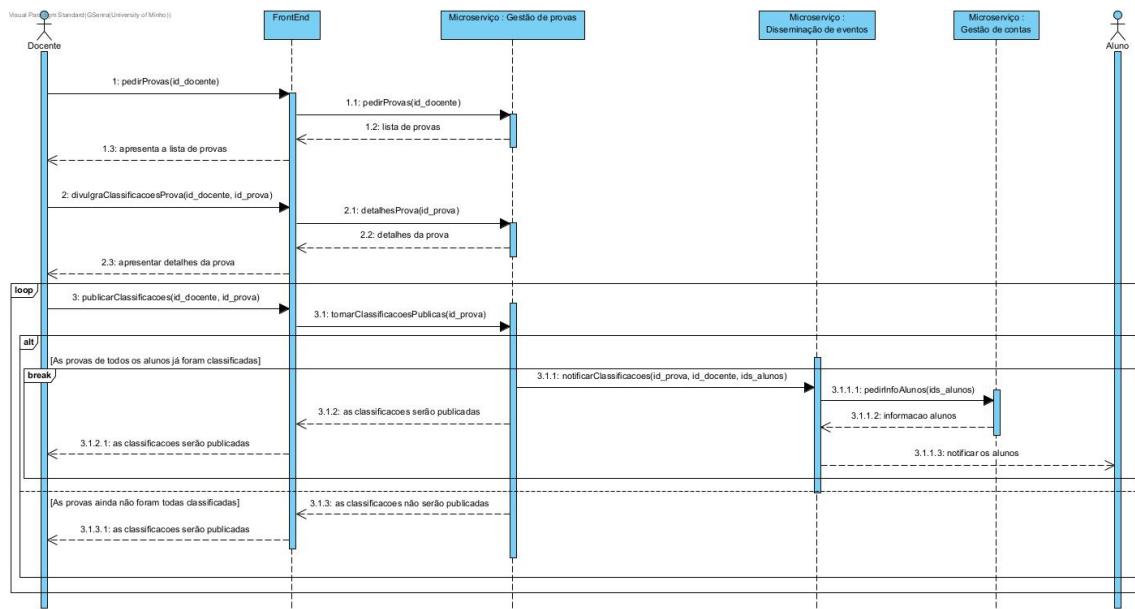


**Figura 7.12:** Diagrama de sequência do use case “Classificar respostas”.

## 7.13 Publicar classificações da prova

Neste diagrama, apresentamos o fluxo de interações para o use case "Publicar classificações da prova" de modo que um ator do tipo "Docente" consiga publicar as classificações dos alunos quando as provas já estão todas corrigidas.

O *FrontEnd* permite ao Docente visualizar os pedidos por si efetuados que serão satisfeitos através da invocação de métodos do micro-serviço *Gestão de provas*. No caso, de não haver condições que inviabilizem a publicação das classificações, este micro-serviço invoca o método *notificarClassificacoes* da *Disseminação de eventos* e, consecutivamente este micro-serviço invoca o método *pedirInfoAlunos* da *Gestão de provas*, com vista a conseguir obter todas as informações necessárias para serem despoletadas notificações com as classificações aos Alunos que realizaram a prova em questão.

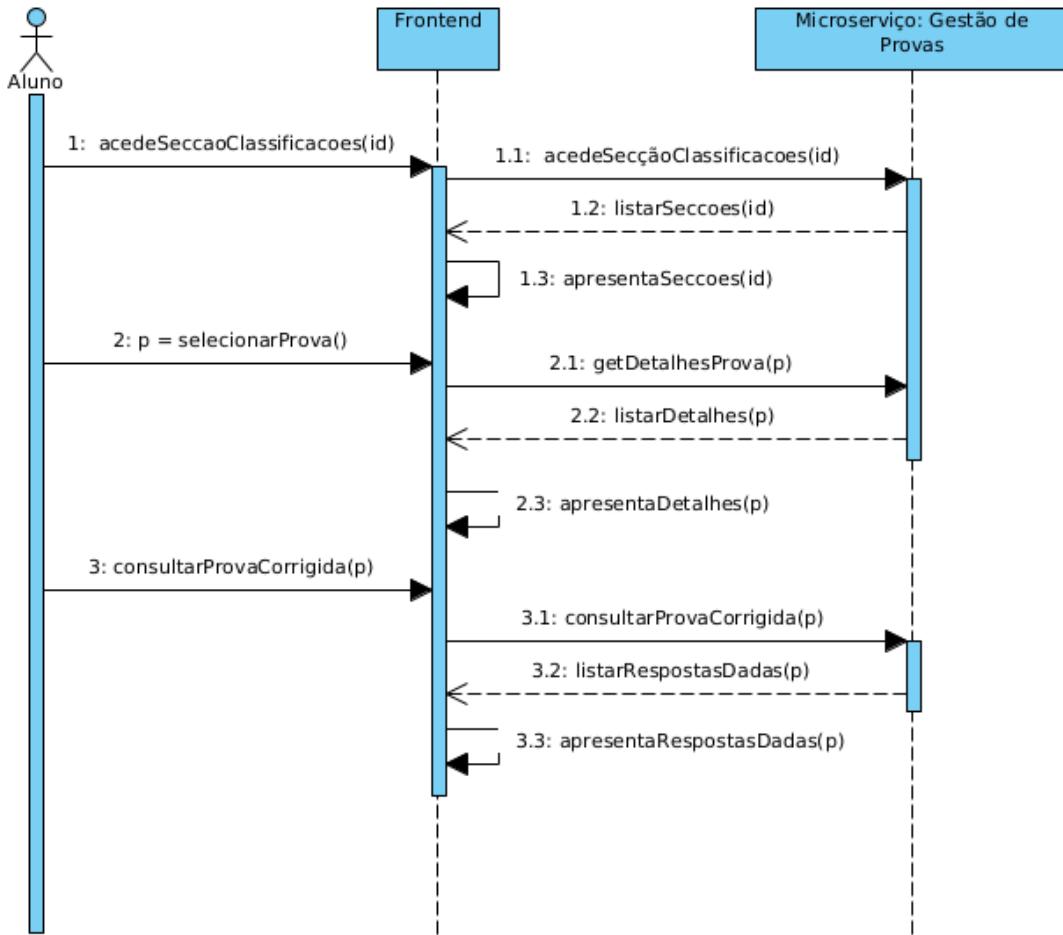


**Figura 7.13:** Diagrama de sequência do use case “Publicar classificações da prova”.

## 7.14 Consultar prova corrigida

Neste diagrama, apresentamos o fluxo de interações para o use case “Consultar prova corrigida” com vista a viabilizar por parte dos atores do tipo "Aluno" a consulta da classificação final e respostas dadas a uma prova por si efetuada.

Com base nisto, o *FrontEnd* chama os métodos do micro-serviço *Gestão de Provas*, *getDetalhesProva* e *ConsultarProvaCorrigida* para obter a classificação e verificar a prova corrigida, respectivamente.

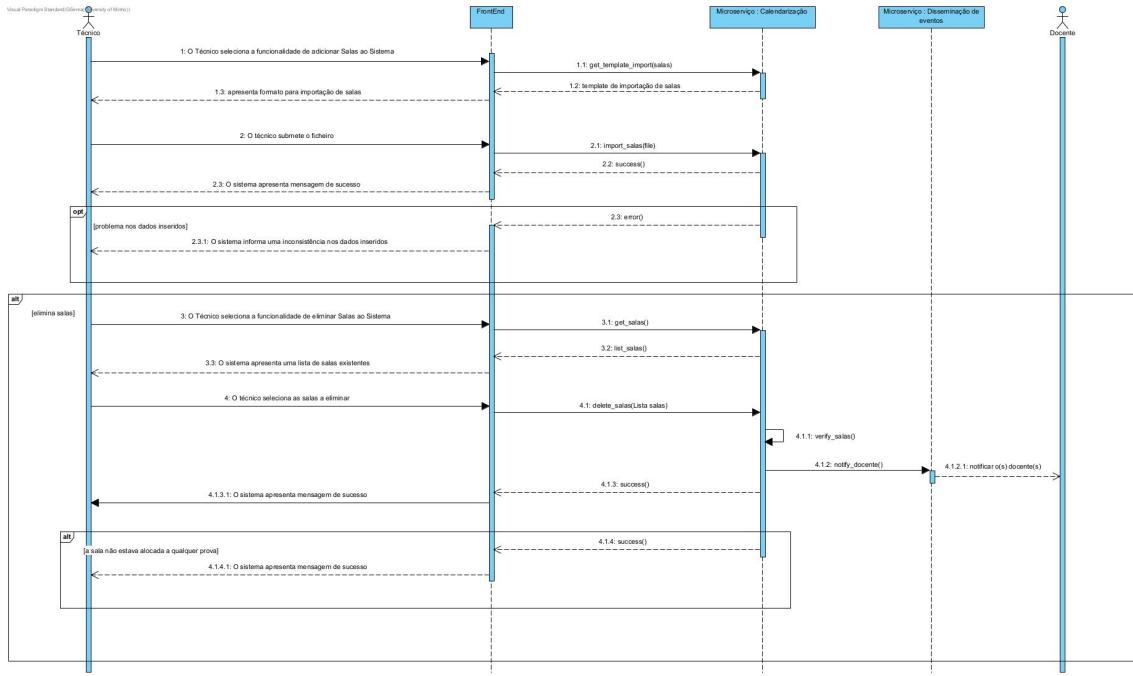


**Figura 7.14:** Diagrama de sequência do use case “Consultar prova corrigida”.

## 7.15 Gerir Salas

Neste diagrama, apresentamos o fluxo de interações para o use case “Gerir Salas” que possibilita os atores do tipo "Técnico" registar ou eliminar salas para a realização de provas.

De um modo geral, *FrontEnd* invoca métodos do micro-serviço *Calendarização* para satisfazer pedidos do ator. No caso, de o ator eliminar uma sala em que iria ocorrer uma prova, este micro-serviço, chama o método *notify\_docente* do micro-serviço *Disseminação de eventos* para que o ator do tipo "Docente" seja notificado do sucedido.

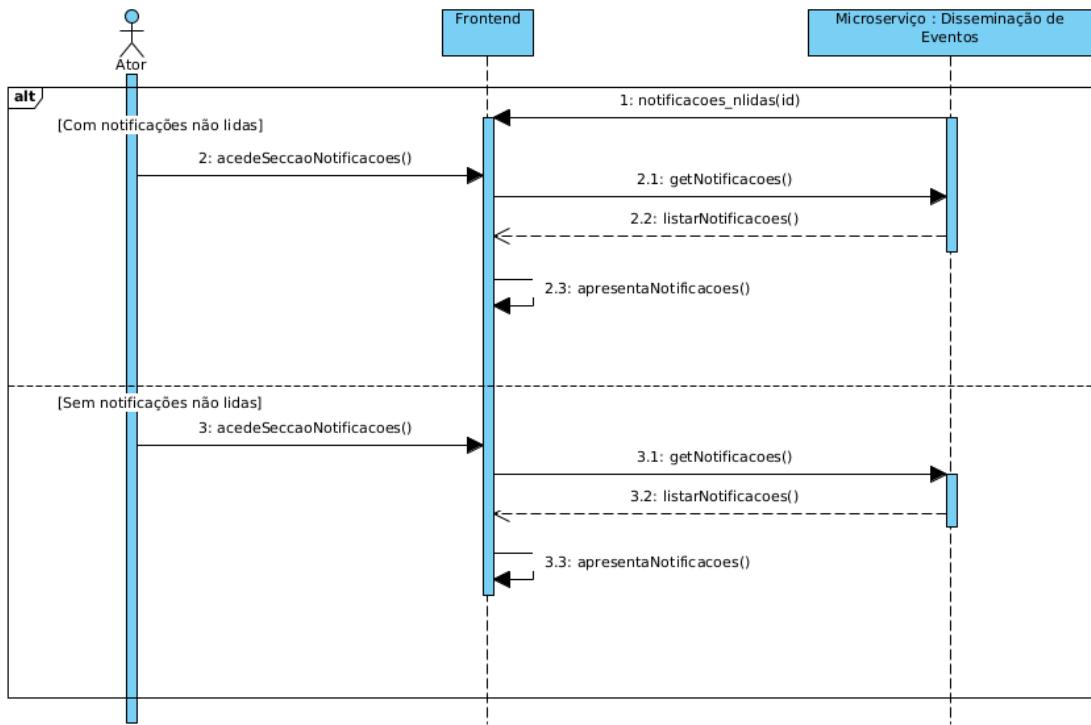


**Figura 7.15:** Diagrama de sequência do use case “Gerir Salas”.

## 7.16 Aceder às notificações

Neste diagrama, apresentamos o fluxo de interações para o use case “Aceder às notificações” com vista a permitir que um ator do tipo "Docente" ou "Aluno" consiga obter todas a suas notificações lidas e não lidas.

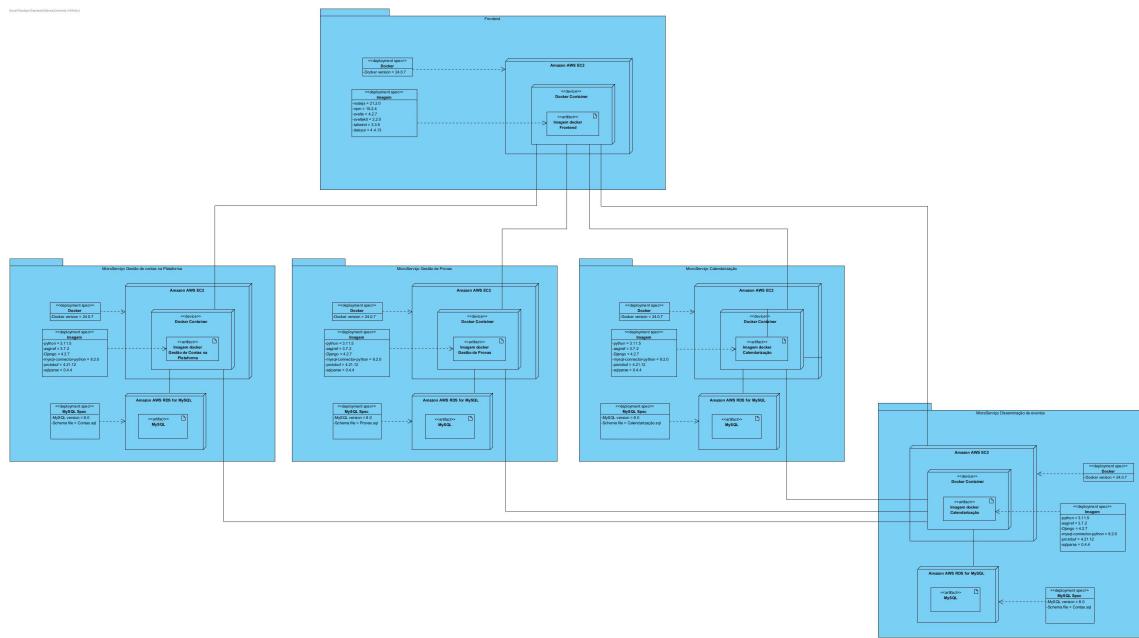
O *FrontEnd* através da invocação do método *getNotificações* do micro-serviço *Disseminação de eventos* obtém uma lista de todas as notificações destinadas ao ator que as requisitou, apresentando-as ao mesmo. No caso da geração de uma nova notificação, a *Disseminação de eventos* chama o método *notificacoes\_nlidas* do *FrontEnd* para que o ator seja notificado da ocorrência.



**Figura 7.16:** Diagrama de sequência do use case “Aceder às notificações”.

## 8. Vista *Deployment*

A fase de *deployment* é crucial para assegurar que a arquitetura proposta seja implementada eficientemente, garantindo uma transição suave do ambiente de desenvolvimento para o ambiente de produção. Nesta secção, detalharemos o processo de *deployment*, abordando os principais componentes com o apoio do seguinte diagrama de *deployment*.



**Figura 8.1:** Diagrama de *deployment*.

### 8.1 *Docker Containers*

Os *Docker Containers* são essenciais para a portabilidade e consistência do ambiente de *deployment*. Cada microserviço e o frontend são empacotados em ficheiros de imagens *Docker*, que incluem todas as dependências necessárias para a execução da aplicação. Esta abordagem garante que a aplicação seja executada de maneira consistente, independentemente do ambiente de produção.

### 8.2 *Amazon AWS EC2*

As instâncias *EC2* da *Amazon AWS* servem como o ambiente de produção para os *Docker Containers*. Estas substituem a necessidade de uma máquina virtual ou servidor de produção.

### **8.3 AMAZON AWS RDS for MySqQL**

Como cada serviço precisa de uma base de dados para persistir os dados, foi escolhido o serviço *RDS for MySQL* da *Amazon AWS* para tal efeito. Escolhemos esta opção em vez de uma instância *EC2* devido à otimização do *RDS* para motores de bases de dados. São usados ficheiros de *schema* (descritos nos respetivos *deployment specifications*) para a inicialização de cada base de dados.