



Universidade do Minho

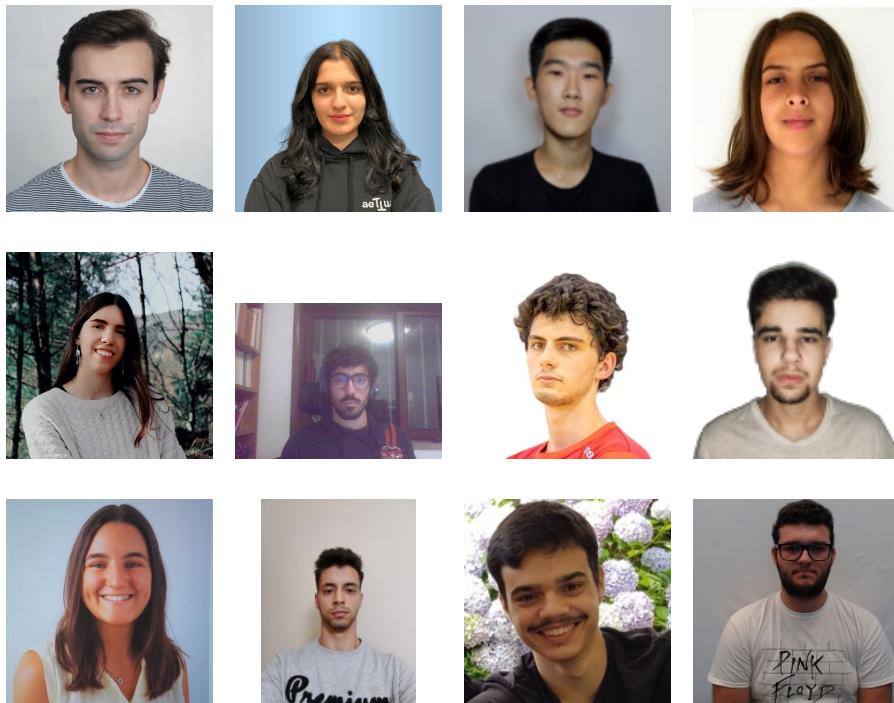
UMinho

**Mestrado Engenharia Informática
Requisitos e Arquiteturas de
Software (2023/24)**

PROBUM v.1

Grupo 1-A, PL1 / Entrega 2

Afonso Ferreira - pg52669
Catarina Costa - pg52676
Daniel Du - pg53751
Francisca Barros - pg53816
Joana Pereira - pg53895
João Alvim - pg53902
José Martins - pg53968
José Pereira - a89596
Marta Aguiar - pg52469
Telmo Oliveira - pg54247
Ricardo Araújo - pg54190
Xavier Mota - pg54276



Braga, March 22, 2024

Prefácio

Durante a elaboração desta segunda fase do projeto, deparamo-nos com alguns desafios, nomeadamente na análise e compreensão dos requisitos funcionais e não funcionais da aplicação. Esta tarefa, por não termos sido os responsáveis pela definição inicial desses requisitos, revelou-se demorada, exigindo uma compreensão profunda de cada detalhe para garantir a continuidade consistente do projeto.

A escolha da arquitetura também suscitou algumas dúvidas, dado ser uma decisão subjetiva que requer uma análise cuidada. No entanto, conseguimos superar rapidamente essas incertezas.

Adicionalmente, exploramos a elaboração de diagramas, como o de blocos e de *deployment*, que não tínhamos ainda abordado em outras fases do curso, pelo que surgiram algumas dificuldades.

Por fim, analisamos o contributo de cada elemento do grupo para o desenvolvimento do trabalho. Assim, cada elemento é avaliado com 0 caso o contributo seja próximo da média, com + caso o contributo seja acima da média e - caso contrário.

Afonso Ferreira	0
Catarina Costa	0
Daniel Du	0
Francisca Barros	0
Joana Pereira	0
João Alvim	0
José Martins	0
José Pereira	0
Marta Aguiar	0
Telmo Oliveira	0
Ricardo Araújo	0
Xavier Mota	0

Contents

1	Introdução e Objetivos	4
1.1	Visão Geral dos Requisitos	4
1.2	Metas de Qualidade	5
1.3	Stakeholders	5
2	Restrições de Arquitetura	6
3	Âmbito e Contexto	7
4	Estratégias de Solução	9
5	Visualização do Bloco de Construção	10
5.1	Nível 1	10
5.1.1	Diagrama de Blocos	10
5.1.2	Diagrama de Componentes	10
5.2	Nível 2 - Diagrama de Classes	12
5.2.1	Diagrama de classes da Sala	12
5.2.2	Diagrama de classes da Notificação	13
5.2.3	Diagrama de classes da Prova	14
5.2.4	Diagrama de classes do Utilizador	15
6	Visualização da execução	16
6.1	Transações entre Microserviços	16
6.2	Registar Docentes	17
6.3	Registar Alunos	19
6.4	Autenticar	21
6.5	Editar Perfil	22
6.6	Criar Prova	23
6.7	Criar Questões	25
6.8	Editar Prova	27
6.9	Editar Questões	29
6.10	Consultar Detalhes da Prova	31
6.11	Partilhar Prova	32
6.12	Responder a Prova	33
6.13	Classificar Respostas	35
6.14	Publicar Classificações da Prova	37
6.15	Consultar Prova Corrigida	38
6.16	Gerir Salas	39
6.17	Aceder às Notificações	41
7	Modelação de Desenvolvimento	42
7.1	Infraestrutura Nível 1	42
8	Conceitos Transversais	43
8.1	Conceitos de Domínio	43

8.2	Experiência do Utilizador	44
8.3	Proteção e Segurança	44
9	Requisitos de Qualidade	45
9.1	Árvore de Qualidade	45
9.2	Cenários de Qualidade	46
10	Riscos e Custos	47
11	Implementação	48
11.1	Provas	48
12	Conclusão	51
13	Glossário	52

1 Introdução e Objetivos

Nesta 2^a fase do trabalho foi elaborado um documento de arquitetura para a aplicação Probum. Este documento aborda diversas questões relevantes para a implementação futura da aplicação, tais como restrições da arquitetura, estratégias da solução, modelação do bloco de construção e da execução, e ainda conceitos transversais, decisões arquiteturais e requisitos de qualidade.

O Probum tem como principal objetivo otimizar processos e melhorar a experiência de Alunos e Docentes. Ou seja, a ideia é permitir que alunos de uma dada unidade curricular de um curso universitário ou politécnico 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. O Probum deverá permitir que os docentes criem provas de avaliação e que as calendarizem, que os alunos as realizem e que essas provas sejam corrigidas, tendencialmente de forma automática.

De modo a clarificar o âmbito da aplicação, apresentamos uma série de exemplos de funcionalidades:

- Criação de provas de avaliação digitais
- Calendarização inteligente das provas
- Realização das provas
- Correção automática das provas

1.1 Visão Geral dos Requisitos

Existem 68 requisitos funcionais e 24 requisitos não funcionais. Estes requisitos foram identificados e explicitados no relatório da Fase 1 nos capítulos denominados Requisitos Funcionais e Requisitos Não Funcionais.

1.2 Metas de Qualidade

No contexto deste trabalho, consideramos importante realçar os seguintes objetivos de qualidade, tendo em conta os requisitos não funcionais:

#	Qualidade	Motivação
1	<i>Operabilidade</i>	<i>O sistema deve ser compreendido, aprendido e utilizado pelos utilizadores facilmente.</i>
2	<i>Segurança</i>	<i>Proteção do sistema contra acessos, usos ou alterações indevidos.</i>
3	<i>Confiabilidade</i>	<i>O sistema deve suportar um nível específico de desempenho quando usado sob condições específicas.</i>
4	<i>Escalabilidade</i>	<i>O sistema deve poder ser modificado, corrigido, adaptado ou melhorado devido a alterações no âmbito dos requisitos.</i>

1.3 Stakeholders

No relatório da Fase 1 no capítulo Cliente, Consumidor e Stakeholders estão especificados os anteriores em maior detalhe. Abaixo temos uma tabela que os descreve brevemente.

Papel	Expectativas
<i>Clientes</i>	<i>Sistema eficiente para automatizar processos de provas. Participação ativa no planeamento e validação da solução.</i>
<i>Consumidores</i>	<i>Procuram uma ferramenta informática que facilite a criação, distribuição e correção de provas, capacitando-os para lidar com situações anômalas. Participam na definição da dinâmica do processo.</i>
<i>Alunos</i>	<i>Produto simples, eficiente, robusto, amigável e tolerante a falhas para realizar provas de avaliação.</i>
<i>Técnicos</i>	<i>Facilidade de instalação e configuração, além de métricas e logs para antecipar e diagnosticar problemas nos equipamentos informáticos.</i>

2 Restrições de Arquitetura

Em relação às restrições de arquitetura, foram identificadas e explicitadas no relatório da fase 1, no capítulo de Restrições do Projeto. Aí, encontram-se as restrições à solução, restrições temporais e restrições orçamentais.

Abaixo temos as duas restrições à solução apresentadas no documento.

Restrições	Descrição
<i>Rest1</i>	<i>A aplicação deve executar na infraestrutura atual da respetiva IES.</i>
<i>Rest2</i>	<i>O computador disponibilizado a cada Aluno, no momento em que realiza uma prova de avaliação, apenas deve permitir acesso ao Probum.</i>

Além destas restrições identificadas na fase anterior de desenvolvimento, nascem outras graças às escolhas feitas para a concepção desta aplicação. Estas restrições existem graças ao tipo de arquitetura de software escolhido - microsserviços. Esta escolha será justificada no capítulo 4, **Estratégias de Solução**.

Restrições	Descrição
<i>Rest3</i>	<i>Cada micro serviço deve ter o seu próprio armazenamento de dados. Qualquer alteração a um serviço não deve impactar os outros.</i>
<i>Rest4</i>	<i>A comunicação entre micro serviços deve ser feita através de APIs RESTful ou protocolos de mensagens como o MQTT, AMQP. Esta comunicação deve ser assíncrona de modo a evitar gargalos.</i>
<i>Rest5</i>	<i>Devem ser evitadas transações distribuídas que envolvam vários microsserviços, dado que essas podem levar a problemas de desempenho e complexidade.</i>

3 Âmbito e Contexto

Neste capítulo será abordado o âmbito e o contexto do sistema. Possibilitando assim, a definição clara dos limites do sistema e o seu âmbito em relação a todos os seus parceiros de comunicação, ou seja sistemas vizinhos.

Tal como foi exposto no documento da fase 1, a plataforma em desenvolvimento tem como propósito final a disponibilização de um serviço sustentável para a criação, calendarização, realização, correção e consulta de provas online.

Tratando-se de um serviço disponibilizado via online, que por sua vez promete ser acessível e benéfico para múltiplos utilizadores em simultâneo, é necessário investir em certas características como disponibilidade constante, recuperação imediata de falhas no funcionamento e grande escalabilidade do sistema, garantindo assim um acesso consistente à plataforma para todos os utilizadores.

O diagrama apresentado abaixo que destaca as interações da plataforma a ser desenvolvida com as entidades descritas. No caso prático em perspetiva, a aplicação com a API da plataforma *Probum*, que por sua vez interage com as base de dados de cada microserviço respetivamente. Relembmando que a abordagem de microserviços irá ser elaborada com mais detalhe no capítulo seguinte (Capítulo 4 - **Estratégias de Solução**).

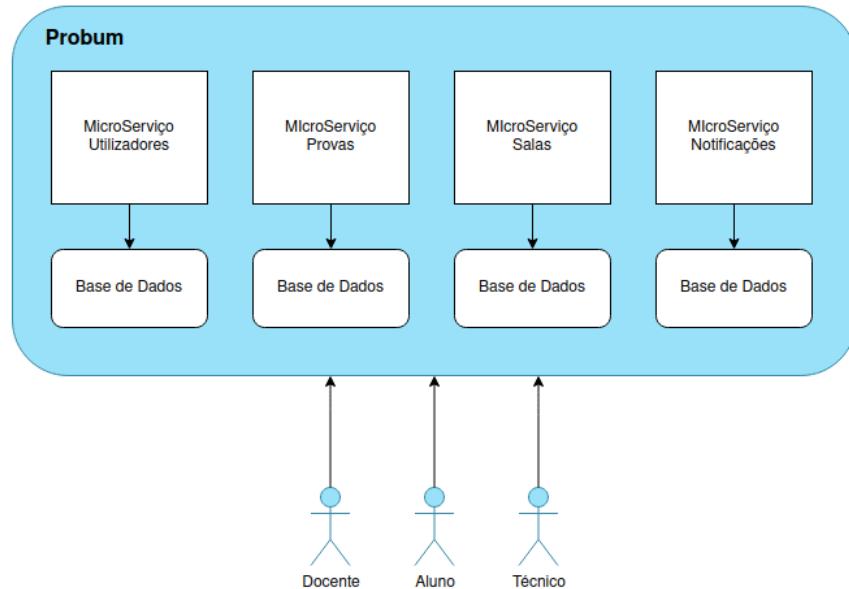


Figure 1: Diagrama de contexto

Na tabela seguinte podem-se observar as funcionalidades de cada componente do diagrama de contexto.

Componente	Descrição
<i>Probum</i>	<i>Produto simples, eficiente, robusto, amigável e tolerante a falhas para realizar provas de avaliação.</i>
<i>MicroServiço</i>	<i>Abordagem de arquitetura de software em que o Probum é construído como um conjunto de serviços independentes, cada um executando uma função específica. Cada microserviço é autônomo, escalável e comunica-se com outros microserviços através de APIs.</i>
<i>Base de Dados</i>	<i>Contém toda a informação de dados associada a cada microserviço da plataforma.</i>

Apesar de existir uma alta complexidade na implementação desta arquitetura e do possível overhead na comunicação com a API, isso pode ser controlado através da formulação de um bom protocolo de comunicação ou mesmo do uso de caching em certos pontos. É também de considerar a possibilidade de haver atrasos na recolha de dados, caso a API receba muitos pedidos e forme uma espécie de fila de espera.

4 Estratégias de Solução

Após a análise dos requisitos não funcionais (RNFs) associados ao desenvolvimento da aplicação, o grupo identificou um conjunto de critérios essenciais que orientaram a escolha da arquitetura do sistema.

Dentre esses requisitos, destacamos, em particular, a necessidade de tempos de resposta rápidos (RNF9), a capacidade de operar em modo local em caso de perda de conexão com o servidor (RNF10), a compatibilidade com computadores de gama baixa (RNF12), a escalabilidade (RNF13 e RNF14), e a facilidade de manutenção e suporte (RNF17).

Após analisar os requisitos críticos, optámos por uma arquitetura baseada em microserviços devido à sua escalabilidade, elasticidade e adaptabilidade. Cada microserviço opera independentemente, favorecendo tempos de resposta rápidos (RNF9) e oferecendo compatibilidade com computadores de gama baixa (RNF12). Além disso, a arquitetura suporta facilidade de manutenção (RNF17), minimizando impactos ao atualizar serviços individualmente.

Contudo é importante reconhecer que, embora os microserviços ofereçam vantagens significativas, a performance pode ser um desafio. Para isso o grupo tentou ao máximo minimizar transações entre microserviços de forma a mitigar esse desafio.

5 Visualização do Bloco de Construção

Num contexto de visualização *top-down* da aplicação, elaboraram-se diagramas de nível 1, nomeadamente o **Diagrama de Blocos** e o **Diagrama de Componentes**, e de nível 2, designadamente os **Diagramas de Classes**.

5.1 Nível 1

5.1.1 Diagrama de Blocos

Para a realização deste diagrama, foi indispensável realizar uma análise rigorosa dos requisitos funcionais da aplicação. Este processo envolveu a compreensão aprofundada de cada requisito funcional, destacando as interações e dependências entre as diferentes funcionalidades do sistema.

O Diagrama de Blocos proporciona uma visão global da arquitetura da aplicação, permitindo uma compreensão rápida e intuitiva da estrutura do sistema. Além disso, destaca as interconexões essenciais entre os componentes, orientando a subsequente elaboração de diagramas mais detalhados, como os Diagramas de Componentes e de Classes, que exploram cada bloco em maior profundidade. Esta abordagem modular promove a clareza e a manutenção eficiente do sistema ao longo do desenvolvimento.

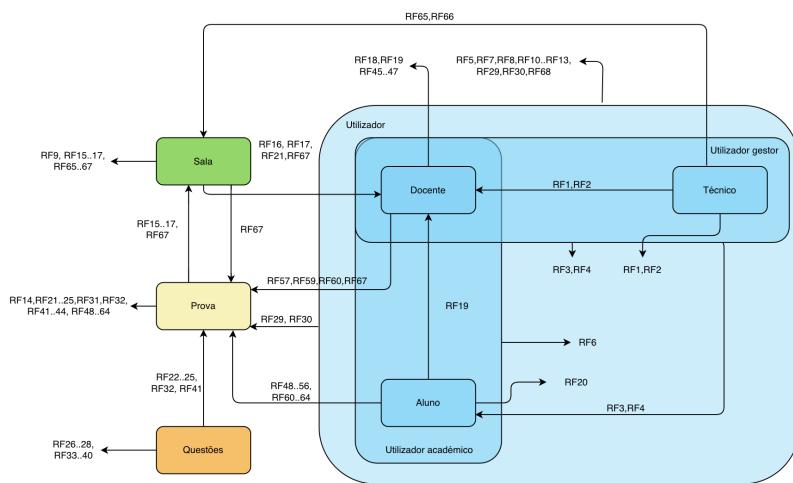


Figure 2: Diagrama de Blocos.

5.1.2 Diagrama de Componentes

De forma a perceber melhor como o nosso sistema iria estar organizado, foi criado um diagrama de componentes que subdivide a nossa aplicação em subsistemas.

Cada um deles dispõe de uma interface para ser usado externamente, mas de resto é completamente isolado do exterior.

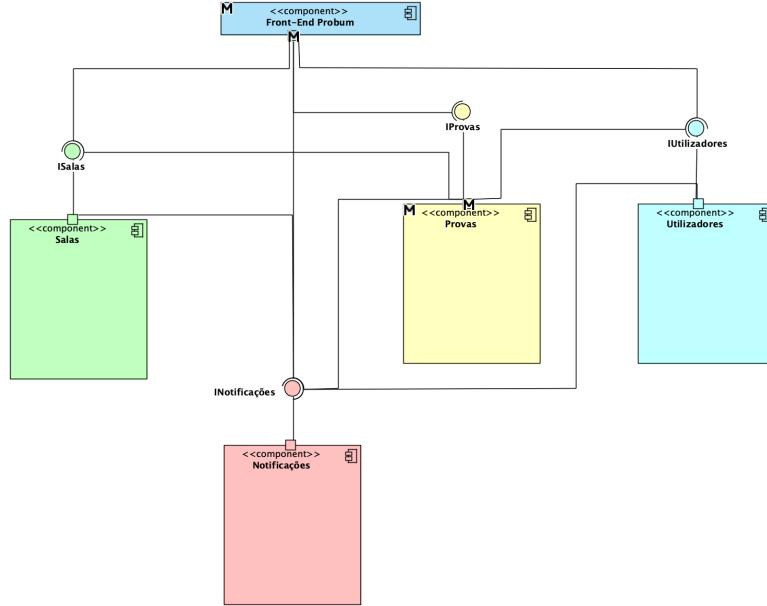


Figure 3: Diagrama de Componentes.

Descrição dos componentes

Name	Responsibility
Salas	Responsável por toda a gestão das salas
Notificações	Responsável por enviar notificações para os utilizadores
Provas	Responsável pela gestão, correção e realização das provas (principal componente do sistema)
Utilizadores	Responsável para gestão dos utilizadores

É de ressaltar a decisão de não criar um microserviço específico para "Questões". Esta decisão baseou-se na observação de que as questões estão sempre ligadas a uma única prova. Não há um "Banco de Questões" centralizado para reutilização das mesmas. Por isso, integrar a gestão de questões diretamente no microserviço associado à prova simplifica a arquitetura, evitando complexidade desnecessária. Dessa forma, as questões são tratadas como parte integrante do serviço responsável pela gestão da prova em que estão inseridas, sem a necessidade de um serviço separado para gerir exclusivamente questões.

5.2 Nível 2 - Diagrama de Classes

Nesta secção eram ser descritos os Diagramas de Classe, que são fundamentais para proporcionar uma visão mais aprofundada e detalhada da organização interna de cada componente identificado no Diagrama de Componentes.

Os Diagramas de Classes descrevem a estrutura e as relações entre as classes, destacando atributos e métodos específicos de cada uma. Esta abordagem permite uma compreensão minuciosa das entidades e dos objetos no sistema, assim como as interações entre eles.

5.2.1 Diagrama de classes da Sala

Neste diagrama, utilizamos a classe *ISala* como interface das salas, sendo esta responsável por todas as operações internas e externas referentes ao microsserviço.

Para simplificar a interação com as diversas classes dentro do microsserviço, optamos pelo design pattern Facade. Desta forma é possível que o **Gestor de Salas** sirva como uma ponte direta, eliminando a necessidade de acessar diretamente às classes internas, centralizando essa responsabilidade no gestor.

Neste caso específico, a função principal do gestor é estabelecer a comunicação entre as salas e a interface. Cada instância da Sala possui atributos específicos.

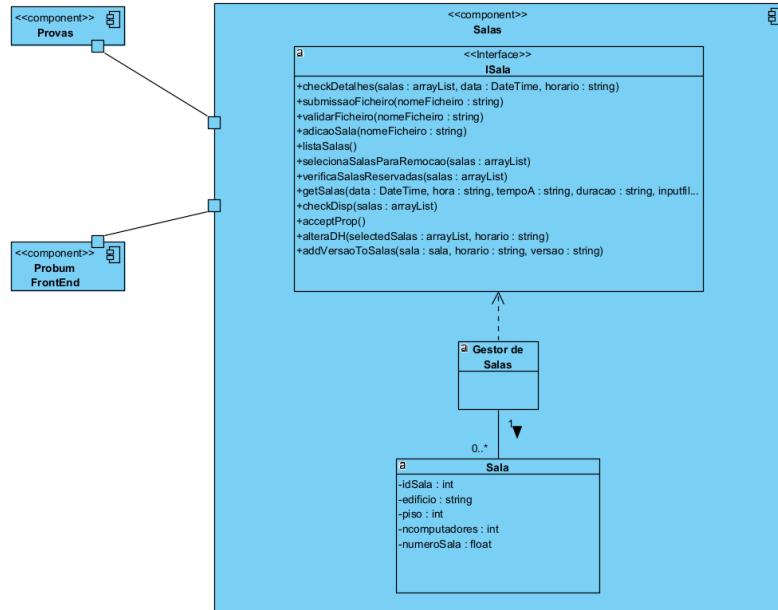


Figure 4: Diagrama de Classes.

5.2.2 Diagrama de classes da Notificação

Dentro deste diagrama, utilizamos a classe *INotificacao* como interface da notificação, sendo esta responsável por todas as operações internas e externas ao microsserviço. Para isso, recorremos ao *pattern facade*, o qual permite estabelecer uma ligação direta com todas as classes dentro deste microsserviço, sem necessidade de aceder a elas diretamente, delegando essa tarefa para o **Gestor de Notificações**. Gestor de Notificações é quem estabelece a ligação direta entre a classe e a interface, focando-se, neste caso, apenas na relação entre as notificações e a interface. Quanto à ligação do componente de notificações com o resto do programa foi usado o design pattern **Observer**. O padrão comportamental Observer foi usado de forma a implementar as notificações ligadas aos outros componentes, tais como; provas, salas, utilizadores e front-end. Deste modo, quando uma ação dos outros componentes é realizada e esta mesma precisa de uma notificação, é invocado o método *notificaObserver()* que irá notificar o componente de notificações, e através do método *update()*, este consegue obter a informação atualizada e realizar as devidas notificações.

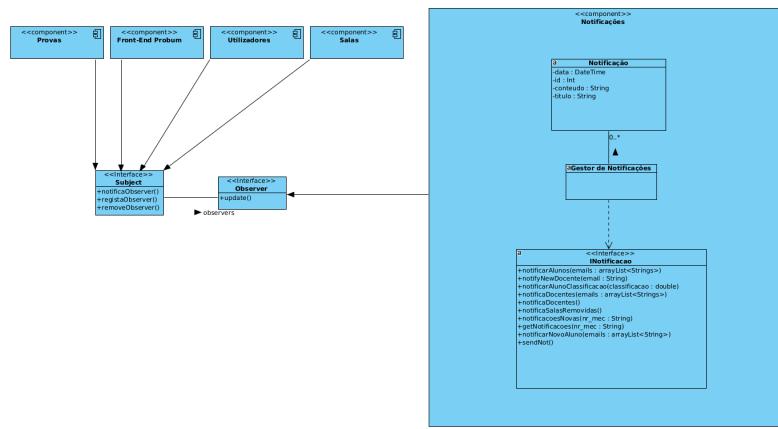


Figure 5: Diagrama de Classes.

5.2.3 Diagrama de classes da Prova

A interface IProva atua como a porta de entrada para operações externas (API). A classe GestorDeProvas desempenha um papel crucial, implementando a interface IProva e gerenciando todas as operações, tanto internas quanto externas.

Para simplificar a interação com as diversas classes dentro do microserviço, optamos pelo design pattern Facade. Essa escolha permite que o GestorDeProvas sirva como uma ponte direta, eliminando a necessidade de acessar diretamente as classes internas, centralizando essa responsabilidade no gestor.

No contexto específico deste microserviço, função principal do gestor é estabelecer a comunicação entre as provas e a interface. Cada instância de prova possui atributos específicos e está associada a uma ProvaComVersao, que, por sua vez, mantém uma lista de questões e uma referência a uma sala específica.

No que diz respeito às questões, optamos pelo design pattern Strategy. Essa escolha permite a implementação de vários tipos de questões de forma flexível e extensível. Cada tipo de questão é tratado como uma estratégia distinta, proporcionando uma abordagem modular para lidar com a diversidade de questões que podem surgir.

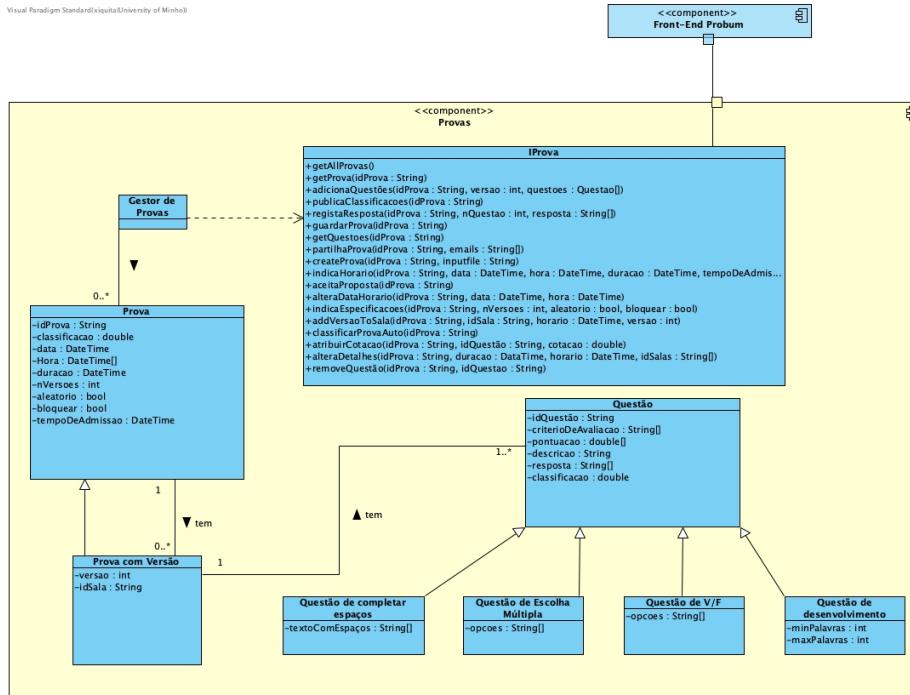


Figure 6: Diagrama de Classes da prova

5.2.4 Diagrama de classes do Utilizador

A interface IUtilizador atua como a porta de entrada para operações externas (API). A classe GestordeUtilizadores desempenha um papel crucial, implementando a interface IProlva e gerenciando todas as operações, tanto internas quanto externas.

Para simplificar a interação com as diversas classes dentro do microserviço, optamos pelo design pattern Facade. Essa escolha permite que o GestorDeUtilizadores sirva como uma ponte direta, eliminando a necessidade de acessar diretamente as classes internas, centralizando essa responsabilidade no gestor.

No contexto específico deste microserviço, a função principal do gestor é estabelecer a comunicação entre os utilizadores e a interface. Cada instância de Utilizador possui atributos específicos.

No que diz respeito aos utilizadores, optamos pelo design pattern Strategy. Essa escolha permite a implementação de vários tipos de utilizadores, quer sejam eles utilizadores gestores e utilizadores académico ou futuramente um outro tipo de utilizador, de forma flexível e extensível. Cada tipo de utilizador é tratado como uma estratégia distinta, proporcionando uma abordagem modular para lidar com a diversidade de utilizador que podem surgir.

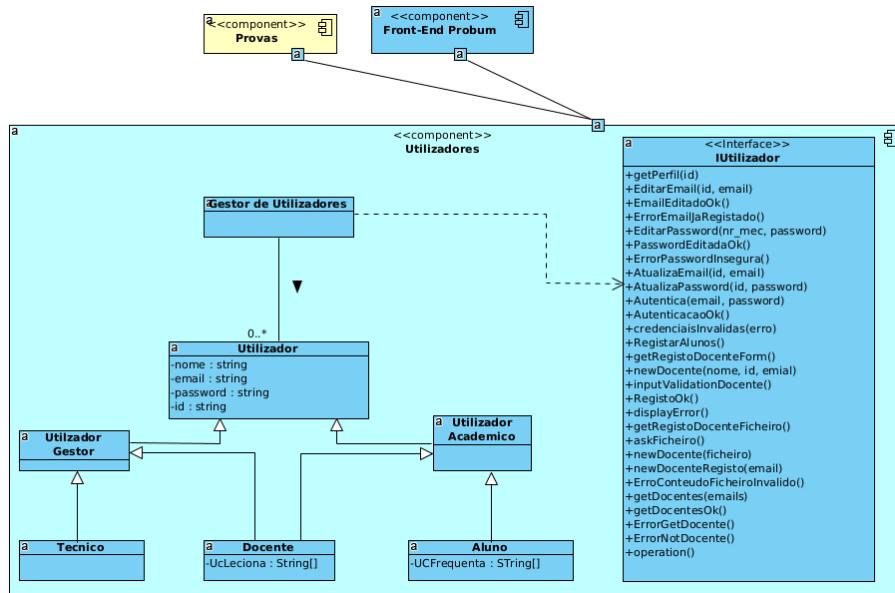


Figure 7: Diagrama de Classes do utilizador

6 Visualização da execução

6.1 Transações entre Microserviços

A seguir, apresentaremos e explicaremos os diagramas de sequência desenvolvidos, considerando a arquitetura baseada em microserviços.

Após a elaboração de todos os diagramas de sequência, foram analisadas as transações entre os microserviços, conforme resumido na tabela abaixo:

Use Case	Nº Transações
Registrar Docente	1
Registrar Alunos	1
Autenticar	0
Editar Perfil	0
Criar Prova	min 5
Criar Questões	0
Editar Prova	1
Editar Questões	0
Consultar Detalhes da Prova	0
Partilhar Prova	max 3
Responder Prova	0
Classificar Respostas	1
Publicar Classificações Prova	1
Consultar Prova Corrigida	0
Gerir Salas	1
Aceder às notificações	0

Destaca-se que, em geral, o número de transações entre os microserviços permaneceu baixo, sendo a exceção no caso de "Criar Prova" com um mínimo de 5 transações. Contudo, este cenário é justificável devido à complexidade do mesmo.

Assim sendo, a análise revelou que a divisão dos microserviços foi eficaz, contribuindo para uma arquitetura bem estruturada.

6.2 Registar Docentes

No diagrama representado abaixo, é possível visualizar as comunicações entre o Técnico, o frontend (Probum) e os microserviços de Gestão de Utilizadores e Notificações.

Na sequencia do diagrama conseguimos, observar que o Técnico tem duas opções para efetuar o registo do Docente.

Na primeira opção envolve um registo individual, onde o Técnico começa por interagir com o frontend, clicando num botão que lhe permite obter um formulário onde irá inserir as informações sobre o Docente a inscrever. Em seguida, o frontend encaminha as informações ao microserviço de Gestão de Utilizadores, que por sua vez, valida as informações fornecidas e faz o registo do novo Docente. Caso, o registo tenha sucesso o microserviço Gestão de Utilizadores interage com o microserviço notificações para mandar um email ao Docente, de modo a notifica-lo sobre o seu registo. Entretanto, caso as informações sejam inválidas, o microserviço notifica o frontend sobre a invalidade.

Na segunda opção, o registo individual é feito por meio de um ficheiro, seguindo a mesma lógica da primeira opção. No entanto, ao invés de um formulário, é pedido ao Técnico um ficheiro contendo as informações do Docente a ser inscrito. Se o conteúdo do ficheiro não for válido, o microserviço de Notificações informa o frontend sobre a sua invalidade.

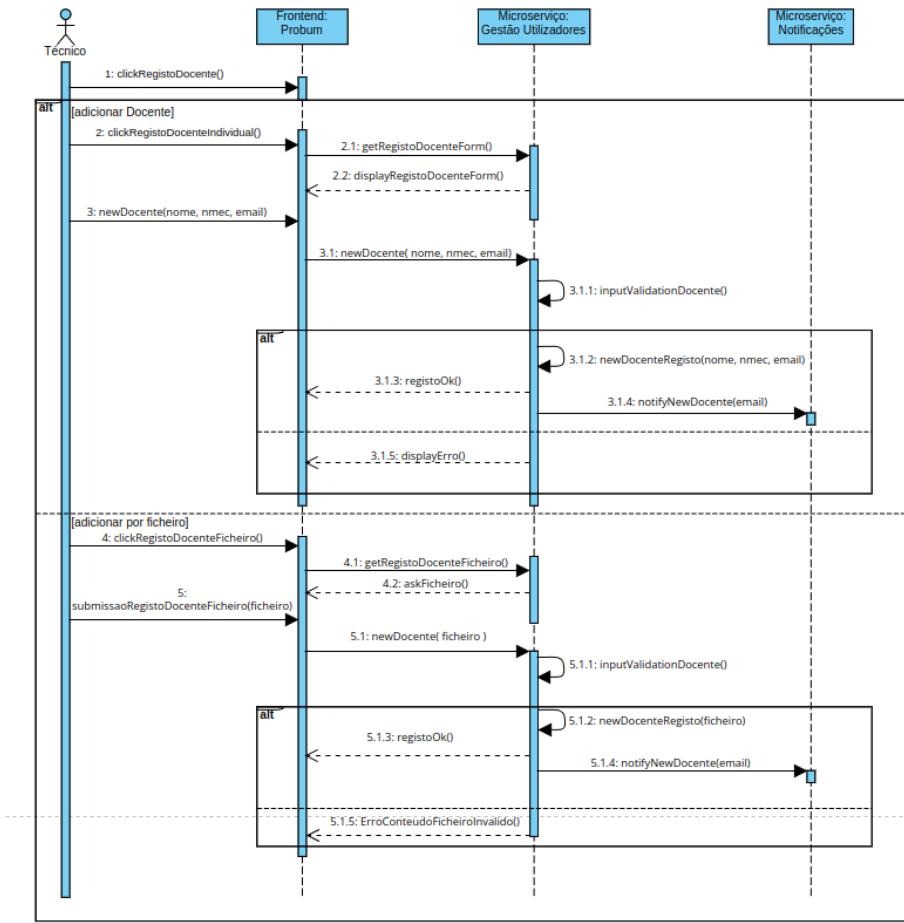


Figure 8: Diagrama de Sequência - Use Case Registar Docentes

6.3 Registar Alunos

Neste *use case* é possível visualizar as comunicações entre os atores principais - Técnico ou Docente -, o FrontEnd (Probum) e os microserviços de Utilizadores e Notificações.

Num primeiro momento, o Técnico ou Docente deve selecionar a secção de registar alunos. Interage assim com o FrontEnd que, por sua vez, interage com o microserviço de Utilizadores. Na sequência desta ação, conseguimos observar que o ator tem duas possíveis formas de registar alunos.

A primeira opção é um registo individual, onde o ator começa por interagir com o FrontEnd, selecionando num botão que lhe permite obter um formulário onde irá preencher os dados do aluno (nome completo, número mecanográfico e e-mail). Em seguida, o FrontEnd encaminha as informações ao microserviço de Utilizadores que valida as informações fornecidas e faz o registo do novo Aluno. Havendo um registo com sucesso, o microserviço de Utilizadores interage com o microserviço de Notificações para mandar um e-mail ao Aluno registado, de modo a notificá-lo sobre o sucedido.

A segunda opção é um registo múltiplo, através de um ficheiro. Este processo segue a mesma lógica da primeira opção, no entanto, ao invés de um formulário, é pedido ao ator um ficheiro contendo as informações de um ou mais alunos a serem registados. Este ficheiro é passado para o microserviço de Utilizadores e é validado. Sendo o conteúdo do ficheiro válido, o microserviço de Utilizadores comunica com o microserviço de Notificações utilizando o mesmo método da primeira opção, diminuindo assim a redundância, enviando uma lista de e-mails de modo a informar os alunos que foram registados. De notar que esta lista de e-mails pode conter um ou mais e-mails, pois o registo pode ser de um ou mais alunos.

No primeiro caso, se o aluno já tiver sido previamente registado ou no segundo caso, se o ficheiro for inválido, o microserviço de Utilizadores notifica o FrontEnd sobre a invalidade.

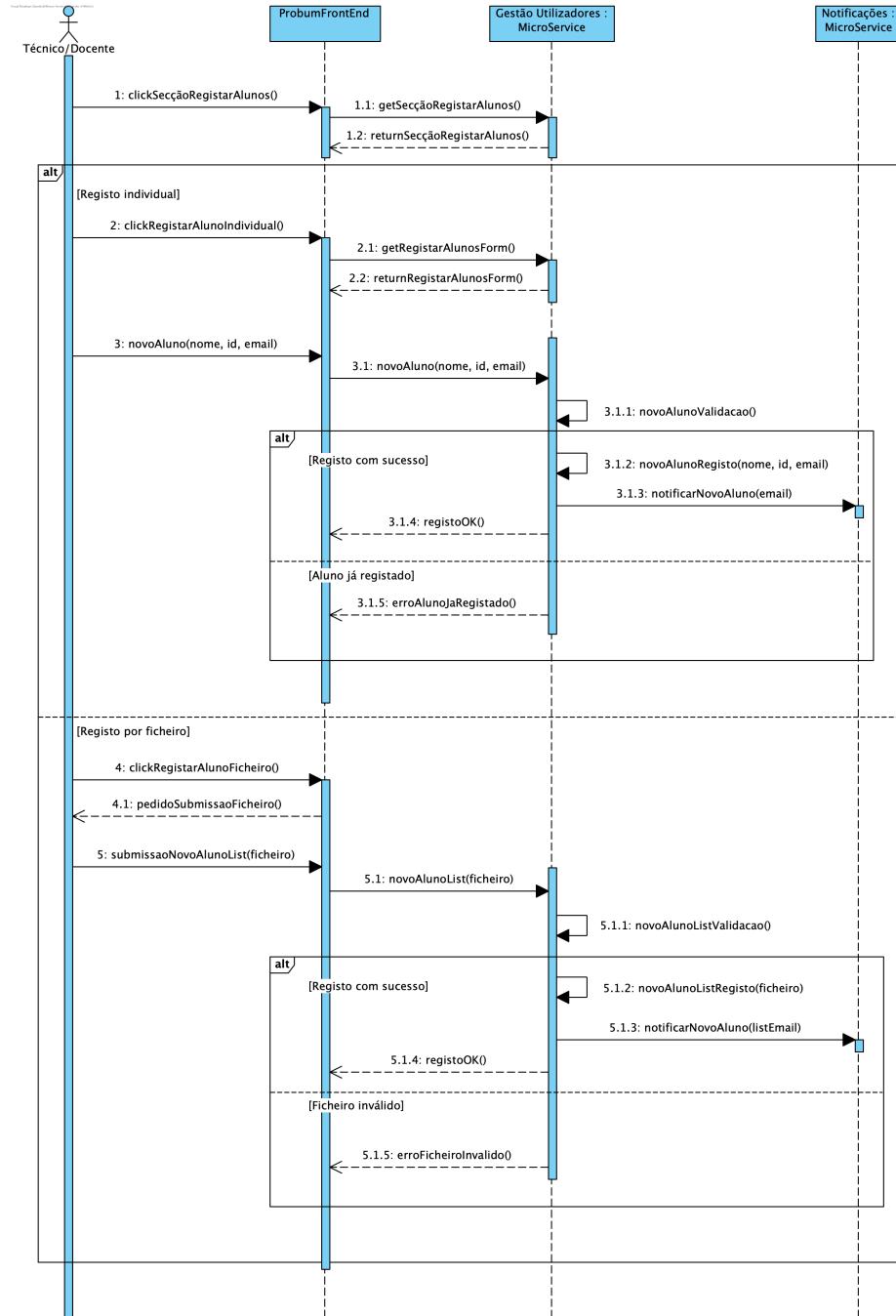


Figure 9: Diagrama de Sequência - Use Case Registrar Alunos

6.4 Autenticar

No seguinte diagrama é possível verificar melhor as comunicações feitas entre o *frontend* (*Probum*) e o microserviço Utilizador.

O Utilizador começa por interagir com o *frontend*, inserindo seu email e password. Em seguida, o *frontend* encaminha essas informações ao microserviço de utilizadores.

O microserviço, por sua vez, valida as credenciais fornecidas. Se forem consideradas válidas, o microserviço confirma a autenticação e estabelece uma sessão para o Utilizador. Entretanto, caso as credenciais sejam inválidas, o microserviço notifica o *frontend* sobre a invalidade.

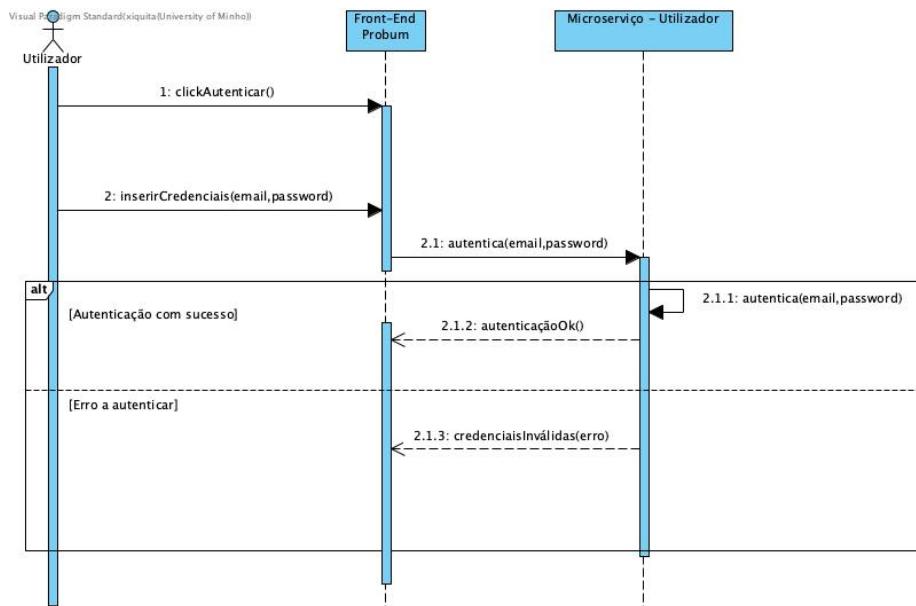


Figure 10: Diagrama de Sequência - Use Case Autenticar

6.5 Editar Perfil

No use case abaixo apresentado, encontra-se uma comunicação entre o utilizador, o *frontend(Probum)* e o microsserviço dos utilizadores. Neste é representado o flow da funcionalidade **Editar Perfil** pode-se ver que o utilizador acede ao seu perfil com o intuito de o modificar. Para tal, este deve clicar num botão que lhe permite obter o seu perfil com as suas características. A nível do sistema isto deve representar um **getPerfil** com o número mecanográfico do mesmo utilizador que está a realizar a ação. Este get irá aceder ao microsserviço dos utilizadores e apresentar o perfil com todas as suas características atuais.

Dentro da opção de editar perfil o utilizador tem a possibilidade de poder alterar tanto o seu email como a sua password, tendo estas comportamentos bastante parecidos a nível do sistema. Este deve então selecionar o que pretende editar e preencher o campo com o novo valor que ele pretende. De seguida o sistema deve aceder ao microsserviço dos utilizadores e mudar o campo que o utilizador pretender, respondendo sempre com uma mensagem de sucesso ou insucesso.

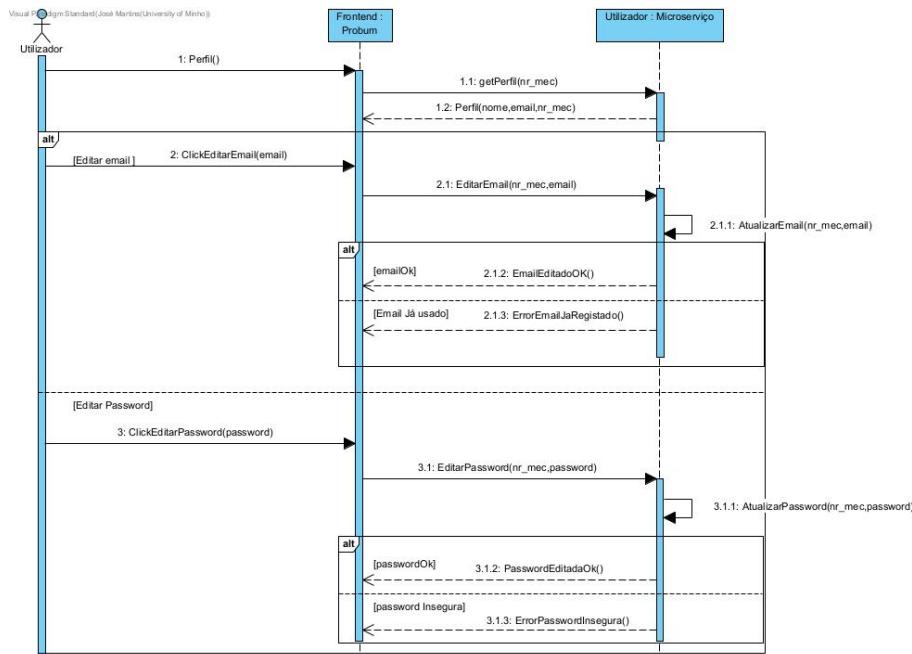


Figure 11: Diagrama de Sequência - Use Case Editar Perfil

6.6 Criar Prova

De maneira a criar uma prova o Docente seleciona a opção *Criar Prova* indicando o nome da mesma e o nome do ficheiro com a lista dos alunos inscritos. Em seguida o sistema valida o input introduzido ou lança um erro caso contrário.

Após efetuada a avaliação o sistema executa a função *pedeHorario()*, esta efetua um print no ecrã para o docente indicar a data, hora, tempo de admissão da prova e duração da mesma. O Docente fornece então os dados solicitados e o sistema pede em seguida ao microserviço Salas as salas disponíveis considerando os dados introduzidos, juntamente com o nome do inputfile, de maneira a obter o número de alunos que irão realizar a prova. O sistema verifica então se existe disponibilidade considerando os dados introduzidos e lança uma mensagem de validação juntamente com um print das salas selecionadas para o efeito em caso afirmativo ou uma mensagem de erro acompanhada do porquê de não haverem salas disponíveis, caso contrário. Caso o docente aceite a proposta do sistema, este indica que aceita a mesma e as salas são então reservadas. Por outro lado, caso o mesmo pretenda solicitar outras salas, pode indicar as salas que pretende reservar dentre as disponíveis e também os respetivos horários.

De seguida, o Docente deve indicar o número de versões, indicar se pretende ou não aleatorizar a ordem das questões e ainda se pretende bloquear o retrocesso das questões (estas duas últimas variáveis serão booleanos). Finalmente irá então indicar a versão da prova que cada sala irá receber e o respetivo horário.

Feitas estas especificações irá então proceder à criação das questões para cada versão da prova (isto remete ao use case *Criar Questões*).

Ao concluir a criação de todas as versões da prova é então dado o término da criação por parte do Docente, a prova é guardada automaticamente e o sistema notifica os alunos inscritos na mesma.

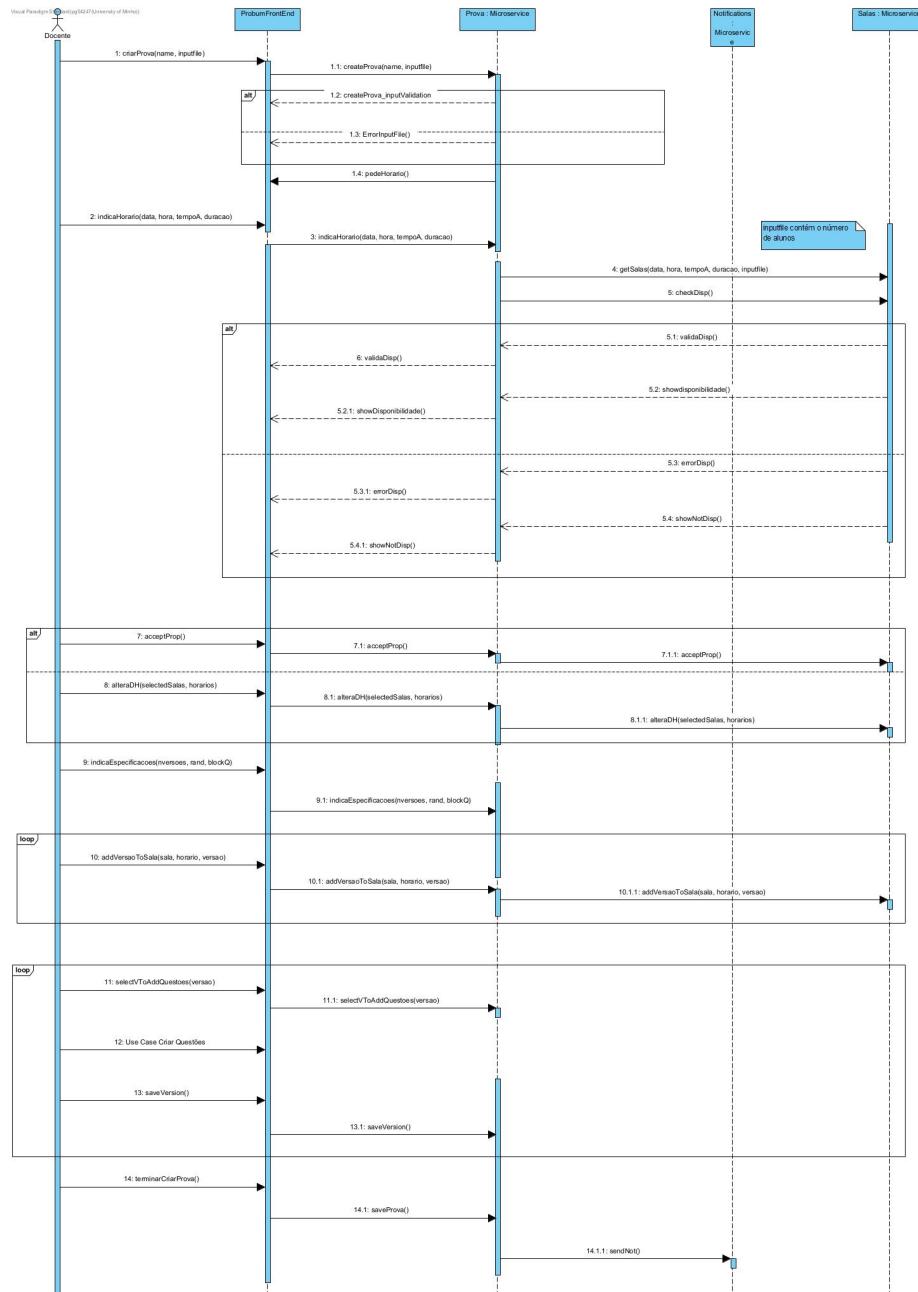


Figure 12: Diagrama de Sequência - Use Case Criar Prova

6.7 Criar Questões

Através do seguinte diagrama é possível vizualizar as iterações feitas entre o Docente, o *frontend* (*Probum*) e o microserviço Prova.

O Docente começa por inserir a descrição da questão e caso queira uma imagem, informações que serão guardadas no frontend e em seguida escolhe o tipo da questão.

Para questões do tipo "Escolha Múltipla" e "Verdadeiro ou Falso," o Docente pode criar diferentes opções. Para cada opção é preciso dar a correção e pontuação da mesma, e de seguida essas informações são temporariamente armazenadas no front-end. Somente quando todas as opções são adicionadas, é enviado para o microserviço de prova as informações da questão, incluindo descrição, imagem e opções.

Para questões do tipo "Desenvolvimento," o Docente insere limites de palavras, se desejar e define o critério de avaliação, enviando essas informações para o frontend, seguindo de seguida para o microserviço de prova.

Para questões do tipo "Completar Espaços," o Docente envia o texto que desejar e os espaços para completar. De seguida, é feito um loop, enviando a avaliação para cada espaço ao frontend. O texto da questão e as avaliações dos espaços são depois enviados ao microserviço da Prova.

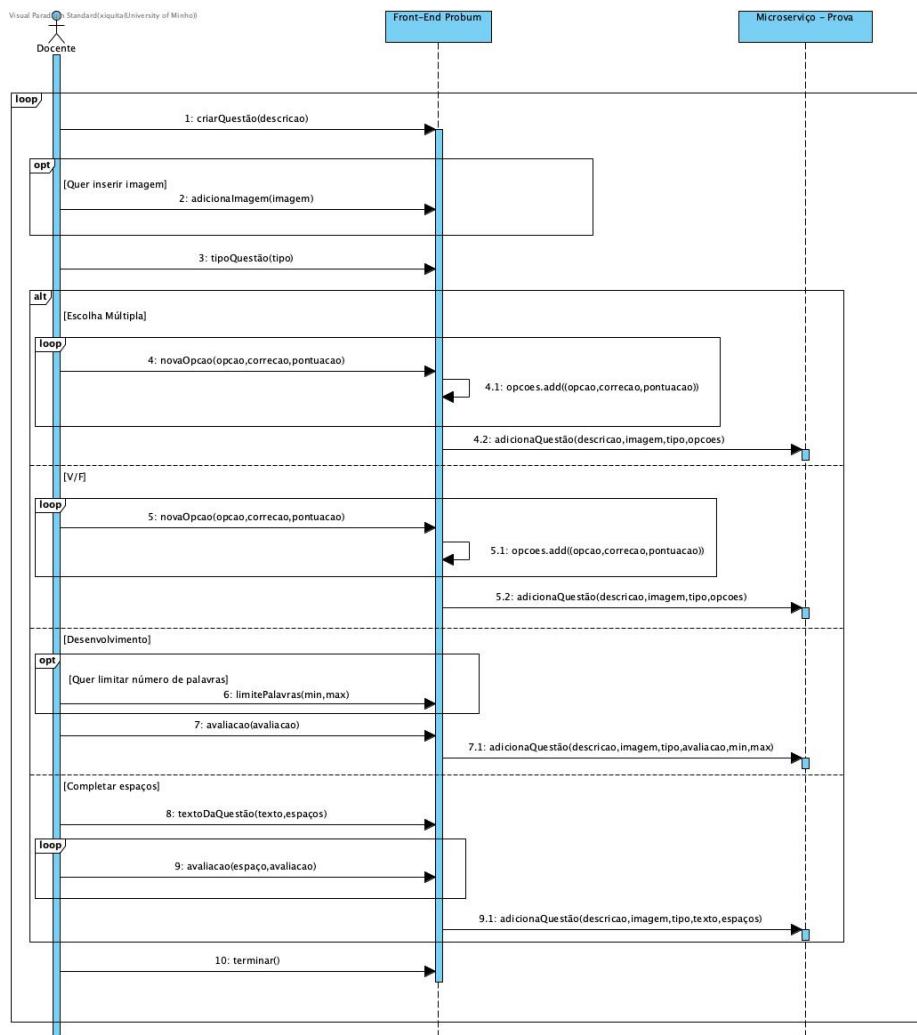


Figure 13: Diagrama de Sequência - Use Case Criar Questões

6.8 Editar Prova

Neste use case pretende-se editar uma prova. Neste, o único utilizador que pode ser protagonista é o Docente, sendo que este vai aceder diretamente ao *frontend(Probum)*, e nas suas acções vão estar associados os **microsserviços** das **provas** e das **salas**. O docente começa por aceder à lista de provas pelo *frontend*, este vai ao microsserviço das provas para obter todas as provas associadas ao docente.

Após lhe serem dadas as provas o docente escolhe uma prova para editar, ao que o sistema, recorrendo novamente ao microsserviço mostra a prova em si com todos os seus detalhes. Os detalhes podem ser ou não editados, sendo que esta decisão cabe ao docente. Se este decidir editar, este deve mudar os detalhes que pretende e o sistema deve fazer uma verificação dos detalhes para garantir que estes são válidos. Se o docente quiser alterar a sala, hora ou duração da prova, o sistema deve aceder ao microsserviço das salas, para confirmar que as salas não vão estar ocupadas naquele data durante aquele tempo.

Passa então a editar as questões propriamente ditas, sendo que antes tem de escolher a versão da prova a editar para o sistema fazer uma coleta de todas as questões associadas à versão escolhida.

Chegando a esta parte o docente pode escolher editar uma certa questão, podendo mudar os detalhes da mesma, sendo passados depois ao sistema para este inserir a questão de novo no microsserviço das provas com os seus detalhes editados. Criar uma questão em que o sistema disponibiliza a mesma página que apresentaria ao criar uma questão, tendo por isso o mesmo *workflow*. Por fim pode eliminar uma questão, sendo que este só tem de escolher a questão a eliminar e o sistema irá processar essa ação no microsserviço das provas.

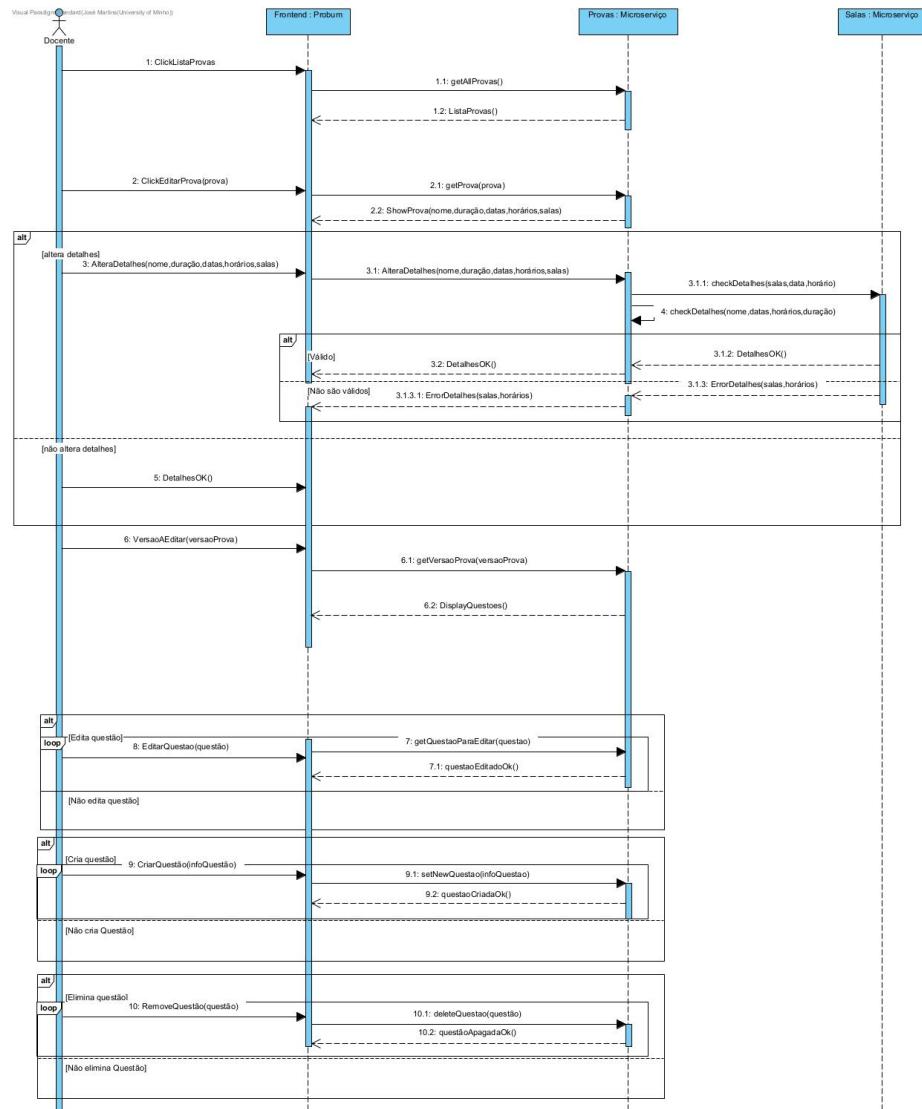


Figure 14: Diagrama de Sequência - Use Case Editar Prova

6.9 Editar Questões

Neste *use case* pretende-se a edição de questões de uma prova. O único ator possível deste *use case* é o Docente, uma vez que apenas ele tem a autorização de editar provas e, por conseguinte, é logicamente o único com a capacidade de editar questões. O processo começa quando o Docente acede diretamente ao FrontEnd (Probum), estando as suas ações associadas aos microserviços das provas. Inicialmente, o docente começa por aceder através do FrontEnd à lista de questões de uma certa prova. Para tal, este vai ao microserviço das provas de modo a obter todas as questões associadas à prova anteriormente selecionada. Após serem listadas as questões, o Docente entra num processo repetitivo, em que escolhe uma questão para editar, o FrontEnd apresenta-a, o Docente edita-a e recomeça.

Dentro da edição da questão, o Docente é confrontado com a possibilidade de editar a descrição da questão ou a imagem da questão, quer estas existam ou não. De seguida, dependendo do tipo da questão, pode editar as opções corretas e editar pontuação (questões de escolha múltipla ou verdadeiros e falsos), apenas editar a pontuação (questão de desenvolvimento) ou então alterar o texto e espaços e a pontuação (questão de completar espaços). Depois de qualquer uma destas edições, o FrontEnd atualiza os dados editados.

Finalizadas todas as alterações, e editadas todas as questões pretendidas pelo Docente, este finaliza o processo, comunicando com o FrontEnd, que por sua vez, comunica com o microserviço das Provas e guarda as correções feitas às questões.

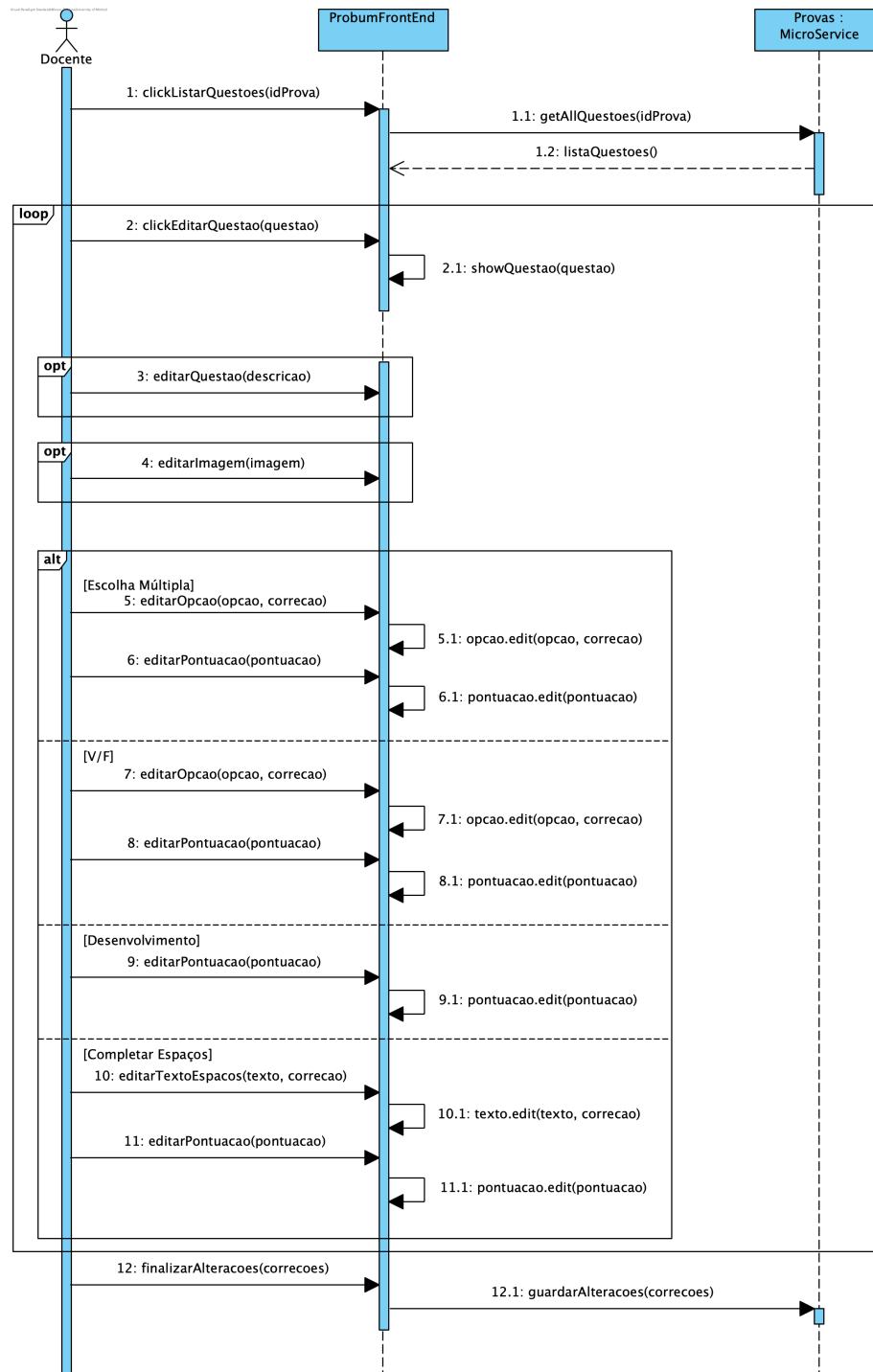


Figure 15: Diagrama de Sequência - Use Case Editar Questões

6.10 Consultar Detalhes da Prova

No seguinte diagrama, é possível visualizar as comunicações entre o Aluno, o frontend (Probum) e o microserviço Provas.

Inicialmente, o Aluno interage com o frontend, clicando num botão que lhe permite obter a lista de todas as provas disponíveis. Em seguida, seleciona a prova para a qual deseja consultar, e esta lhe é apresentada.

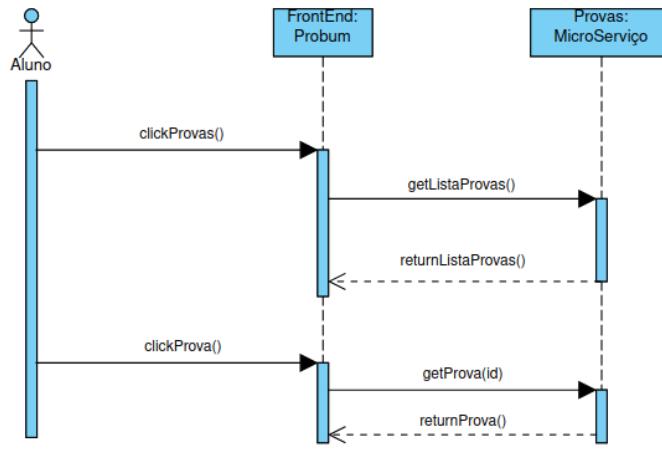


Figure 16: Diagrama de Sequência - Use Case Consultar Detalhes da Prova

6.11 Partilhar Prova

Este diagrama de sequência é relativo ao Use Case Partilhar Prova. Primeiramente, o Docente seleciona a secção lista de provas, apresentada ao utilizador pelo Frontend. De seguida é apresentado ao docente as provas disponíveis e este seleciona a prova que pretende partilhar. Depois o Docente fornece ao Frontend os emails dos docentes com quem pretende partilhar a prova. A seguir o sistema partilha a prova com os Docentes, e notifica-os por email. No caso de algum dos emails fornecidos, corresponder a um email de Docente não registado, o sistema informa do erro(Docente não registrado) e o Docente poderá voltar a fornecer os emails. E ainda, caso algum dos emails dados, não coincidir com um email de um Docente, o sistema avisa do erro>Email fornecido não é de um Docente), e o Docente também poderá voltar a fornecer os emails.

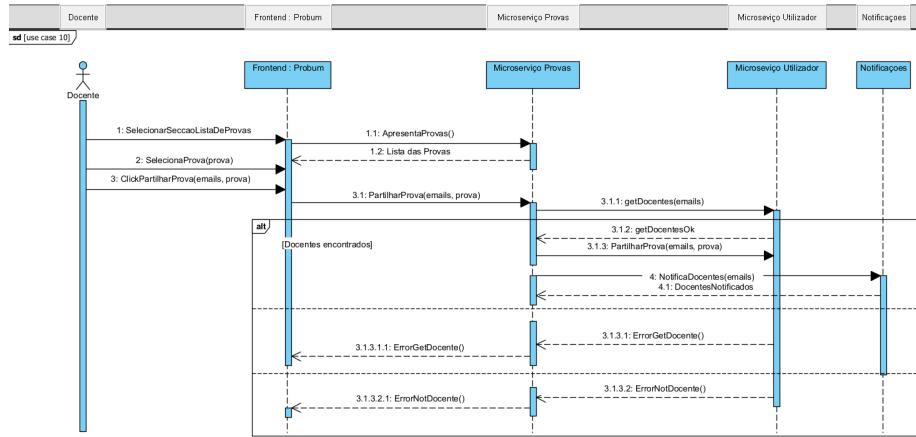


Figure 17: Diagrama de Sequência - Use Case Partilhar Prova

6.12 Responder a Prova

Estamos presente um cenário em que um aluno (ator) responde a uma prova. Ele inicia o cenário fazendo um pedido ao *FrontEnd* uma lista de Provas na qual tem está inscrito/tem acesso. Por sua vez, o *FrontEnd* faz um pedido ao Micro-serviço *GestãoProva* a lista das provas à qual o aluno tem acesso.

De seguida, após o *FrontEnd* ter apresentado a lista o aluno seleciona uma prova da lista à qual pretende responder. Antes do aluno poder responder à prova, o microserviço *GestãoProva* fornece primeiramente os detalhes da prova e somente após a apresentação é que o aluno é capaz de responder à prova.

Se aluno não conseguir aceder à prova a tempo ou a prova ainda não estiver disponível, por exemplo quando um aluno tenta responder à prova mesmo esta ainda não ter sido inicializada, é lhe apresentado um erro. Caso o aluno for felizardo de não ter ocorrido nenhum dos erros acima descritos, o *FrontEnd* pede ao microserviço *GestãoProva* todas as questões e apresenta ao aluno a primeira questão.

Enquanto que o aluno ainda não terminou de responder à prova este pode ou não, caso a regressão de respostas não for permitida, de selecionar a questão à qual quer responder. Caso seja permitida, o frontend apresenta a questão solicitada pelo aluno para responder. Dada a resposta à pergunta, caso a resposta tenha sido registada com sucesso ou insucesso é apresentada uma mensagem de feedback.

Dada a finalização da prova, o frontend lança um termo da prova ao microserviço e este lança uma mensagem de confirmação ao aluno para o termo da prova.

Quanto às interações entre micro serviços procuramos diminuir ao máximo o número de interações buscando a prova na sua integrada e guardando a lista de questões no frontend em vez de ir buscar cada questão ao microserviço durante o ciclo. Isso faz com que as interações entre o microserviço seja apenas quando o *FrontEnd* pede a lista de provas à qual o aluno tem acesso, as informações da prova que o aluno selecionou, a prova com as questões, o registo de cada resposta e a confirmação de término de prova e a sua submissão.

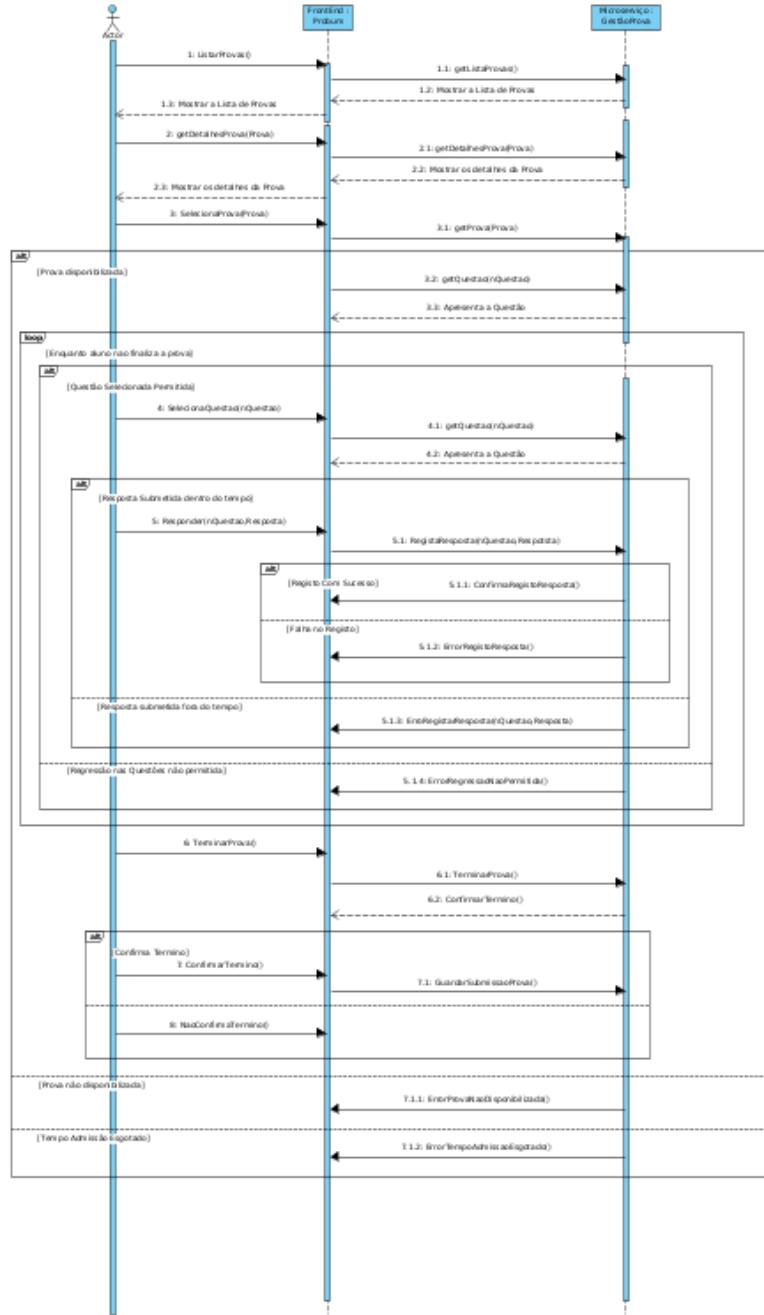


Figure 18: Diagrama de Sequência - Use Case Responder a Prova

6.13 Classificar Respostas

No diagrama representado abaixo, é possível visualizar as comunicações entre o Docente, o frontend (Probum) e os microserviços Provas e Notificações.

O Docente começa por interagir com o frontend, selecionando da lista de provas que o sistema apresenta, as provas que deseja que sejam classificadas. Na sequencia do diagrama conseguimos observar que o Docente tem duas opções para efetuar a classificação das Provas.

Na primeira opção envolve, a classificação automática das provas, onde o Docente clica num botão que lhe permite classificar as provas automaticamente, o frontend encaminha o pedido para o microserviço Prova e este por sua vez entra num ciclo loop de correção destas. Após isso, retorna as classificações para o frontend.

A segunda opção envolve a classificação manual das provas, em que cada questão pode ser classificada através de um parametro introduzido pelo Docente. Sempre que esse valor é introduzido no frontend, este é enviado para o microserviço que é responsável pelas Provas e é atribuida a classificação à respetiva questão. Esta ação pode ser realizada de forma repetitiva para cada questão.

O docente tem a opção de avançar para a correção manual da seguinte prova quando terminar a correção das questões da prova atual.

Por fim, o Docente clica num botão que lhe efetua a conclusão da classificação das provas. Esta ação, responsável pelo microserviço Provas, consiste em guardar todas alterações efetuadas na base de dados correspondente ao microserviço referido anteriormente. Seguidamente, o microserviço Provas faz uma comunicação com o microserviço Notificações e este fica responsabilizado por notificar o Aluno que a sua prova já foi corrigida e envia o valor da classificação total.

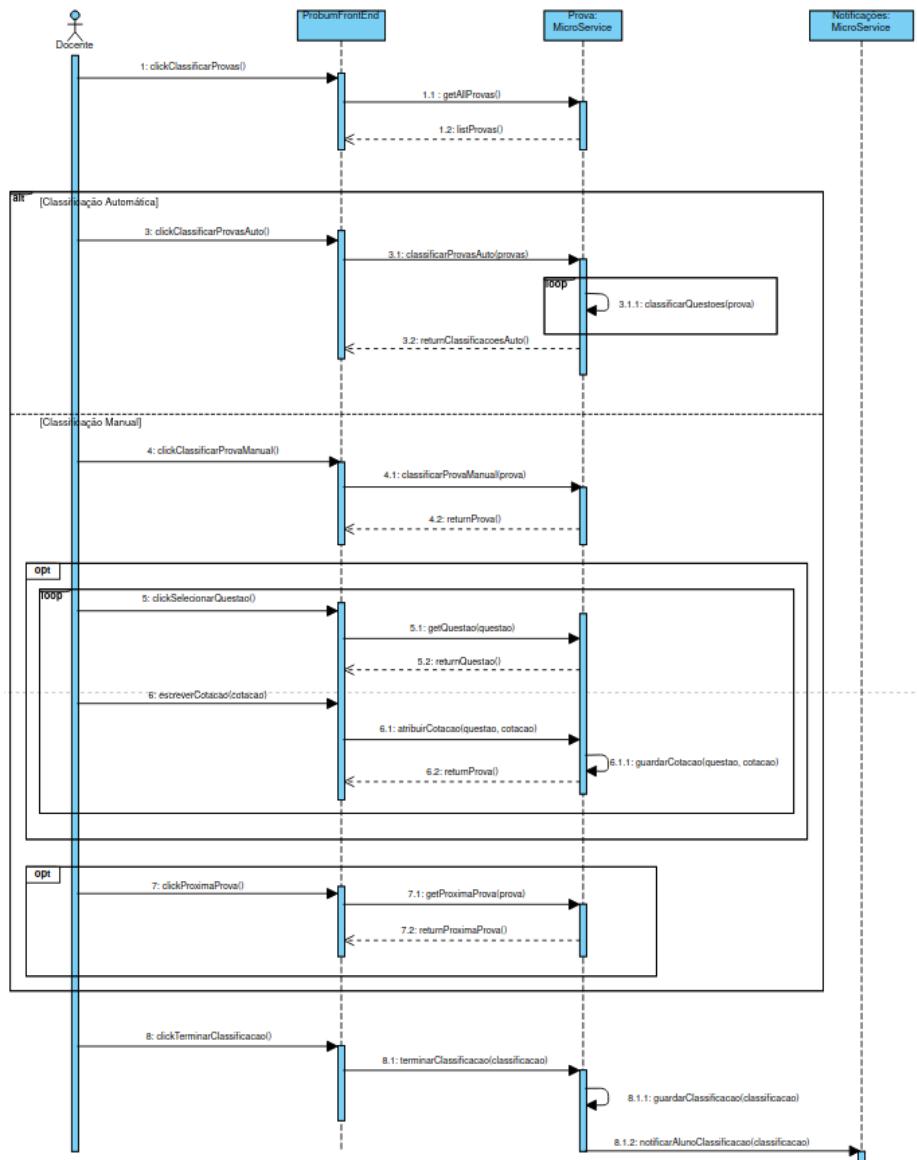


Figure 19: Diagrama de Sequência - Use Case Classificar Respostas

6.14 Publicar Classificações da Prova

No diagrama representado abaixo, é possível visualizar as comunicações entre o Docente, o frontend (Probum) e os microserviços de Gestão de Utilizadores e Notificações.

Inicialmente, o Docente interage com o frontend, clicando num botão que lhe permite obter a lista de todas as provas disponíveis. Em seguida, seleciona a prova para a qual deseja publicar as classificações e são exibidas algumas informações sobre essa prova, incluindo a sua classificação. O Docente, então, escolhe a opção de tornar a classificação dessa prova pública.

O frontend encaminha essas informações para o microserviço Provas, que, por sua vez, torna a classificação pública.

Se a publicação seja bem-sucedida, o microserviço Provas interage com o microserviço Notificações para enviar um email aos Alunos, notificando-os sobre a publicação da classificação da prova. No entanto, se as informações forem inválidas, o microserviço Notificações avisa o frontend sobre o erro ocorrido.

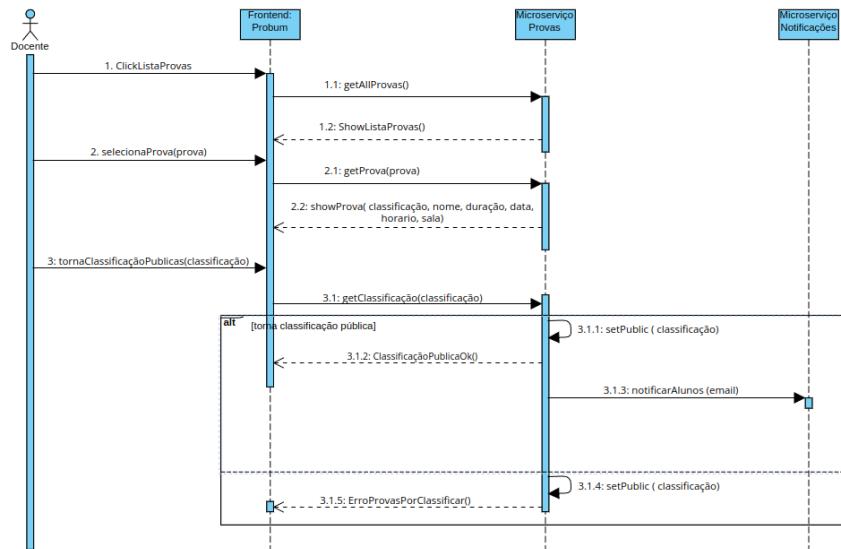


Figure 20: Diagrama de Sequência - Use Case Publicar Classificações da Prova

6.15 Consultar Prova Corrigida

No use case abaixo apresentado, encontra-se uma comunicação entre o utilizador, o *frontend(Probum)* e o microsserviço das provas. Neste é representado o flow da funcionalidade **Consultar Prova Corrigida**. Para o utilizador consultar a sua prova corrigida, este deve clicar no botão de visualização de provas corrigidas, que deverá, posteriormente apresentar, a lista de provas corrigidas para consulta, onde, o utilizador terá que selecionar o teste que deseja consultar, após tal, é apresentada a prova.

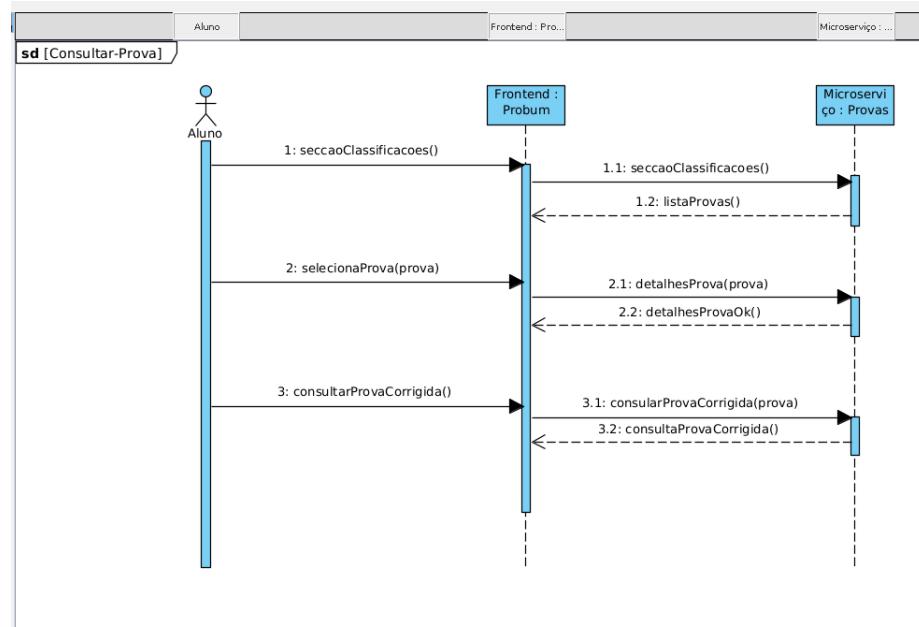


Figure 21: Diagrama de Sequência - Use Case Consultar Prova Corrigida

6.16 Gerir Salas

No use case abaixo apresentado, encontra-se uma comunicação entre o utilizador, o *frontend(Probum)*, o microsserviço das provas e o microsserviço das notificações. Neste é representado o flow da funcionalidade **Gerir Salas**.

Por um lado, o técnico seleciona a opção de adicionar salas. De seguida, o frontend requisita um ficheiro ao utilizador com os detalhes da sala, subsequentemente o técnico realiza a submissão do ficheiro onde o frontend reencaminha-o para o *microsserviço salas* onde este valida o ficheiro. Se o ficheiro fornecido incluir dados inválidos, o microsserviço informa o técnico, senão o microsserviço realiza a adição das salas e, através do frontend, notifica a adição das mesmas.

Por outro lado, o técnico pode remover salas. Para tal, deve clicar no botão de remoção de salas, subsequentemente é-lhe apresentado uma lista das salas que podem ser removidas, onde deve selecionar a(s) sala(s) que pretende remover. De seguida, o sistema notifica todos os docentes que lecionam nessa sala da remoção da mesma, finalmente, o sistema informa o técnico, através do mesmo front-end, que a remoção foi bem sucedida.

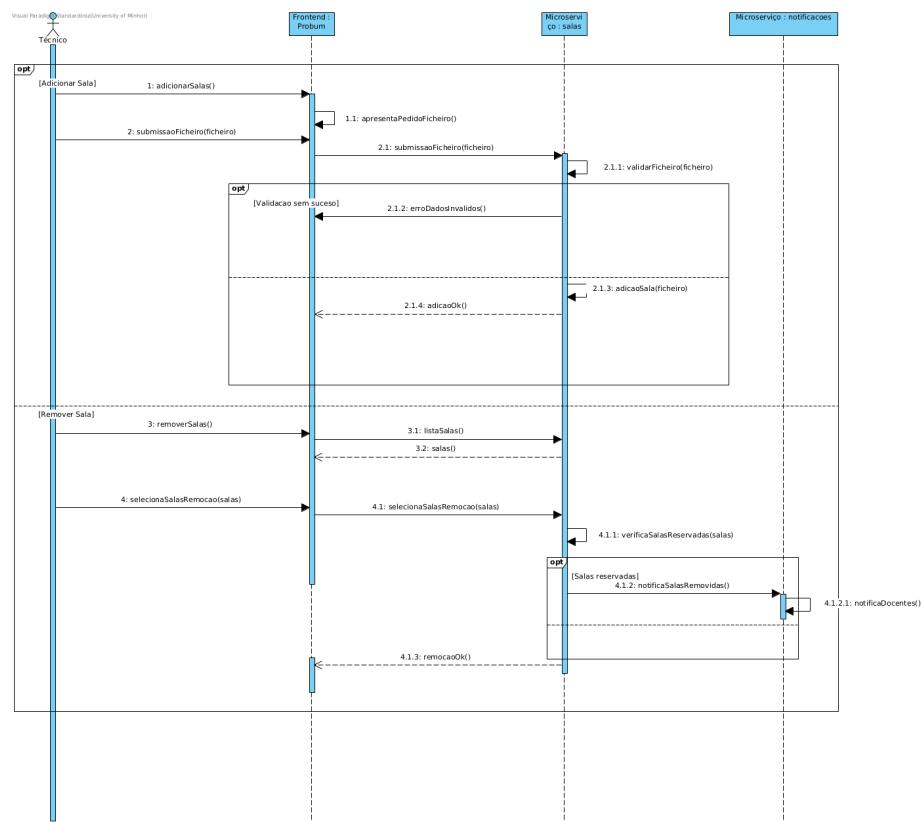


Figure 22: Diagrama de Sequência - Use Case Gerir Salas

6.17 Aceder às Notificações

No seguinte use case, descreve-se a comunicação entre o *utilizador*, o *frontend(Probum)* e o *microserviço de notificações*. Este cenário ilustra o processo da funcionalidade **Aceder às Notificações**, no qual o microserviço de notificações disponibiliza, através do método *notificacoesNovas*, novas notificações para o frontend (Probum) com base no número mecanográfico do utilizador. Posteriormente, o utilizador abre as notificações e visualiza-as.

No âmbito do sistema, o frontend (Probum) utiliza o método *getNotificacoes* para solicitar ao microserviço de notificações a lista de notificações associadas ao número mecanográfico do utilizador. Como resultado, as notificações são apresentadas ao utilizador. No entanto, se o utilizador não tiver novas notificações, o microserviço de notificações não executa o método "notificacoesNovas".

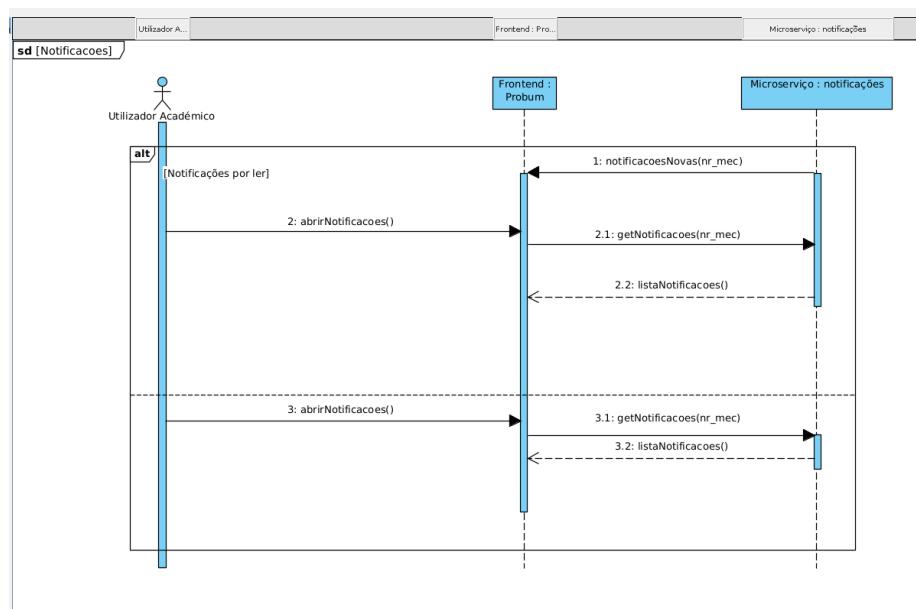


Figure 23: Diagrama de Sequência - Use Case Aceder às Notificações

7 Modelação de Desenvolvimento

Conteúdo

A vista de implementação descreve:

1. a infraestrutura técnica utilizada para executar o sistema, com elementos de infraestrutura como ambientes, computadores, processadores, canais e topologias de rede, bem como outros elementos de infraestrutura e
2. mapeamento de blocos de construção (software) para esses elementos de infraestrutura.

Frequentemente, os sistemas são executados em ambientes diferentes; Este tipo de diagramas fornece uma vista de implementação, sendo ainda mais importante se o software for executado como um sistema distribuído com mais de um computador, processador, servidor ou container.

Do ponto de vista do software, é suficiente capturar apenas aqueles elementos de uma infraestrutura que são necessários para mostrar uma implementação dos blocos de construção.

Motivação

O software não funciona sem hardware. Esta infraestrutura subjacente pode e irá influenciar um sistema e/ou alguns conceitos transversais. Portanto, há uma necessidade de conhecer a infraestrutura.

7.1 Infraestrutura Nível 1

No diagrama de *deployment* mostram-se os microsserviços e a forma como estes "comunicam" com o cliente. Assumindo que o próprio computador vai ter a capacidade de correr tudo sozinho sem a necessidade de recorrer a serviços externos (p.e. AWS), todos os componentes vão ser corridos na máquina local. Começou-se então por definir o frontend, criou-se então um componente **Probum App** e um artifact que representa o frontend propriamente dito. Para este ter uma fácil comunicação com os microsserviços criou-se um **APIServer**, a gateway desta API está dentro de um docker container de maneira a termos um ambiente isolado. A comunicação entre esta e o frontend é feita a partir de uma *REST API* sendo que depois as informações passadas pelo frontend irão levar o seu rumo para cada um dos microsserviços.

A composição dos microsserviços é muito semelhante entre elas, tendo por isso dois containers docker, um com a interface do microsserviço onde se encontram as operações que irão ser realizadas no microsserviço e o segundo com a base de dados relativa a este mesmo microsserviço que guarda todas as informações relevantes em relação a ele. No diagrama abaixo representado podem também ser observadas algumas ligações entre microsserviços, estas estão associadas a algumas operações que podem envolver mais do que um microsserviço e que precisem de informações exteriores ao mesmo.

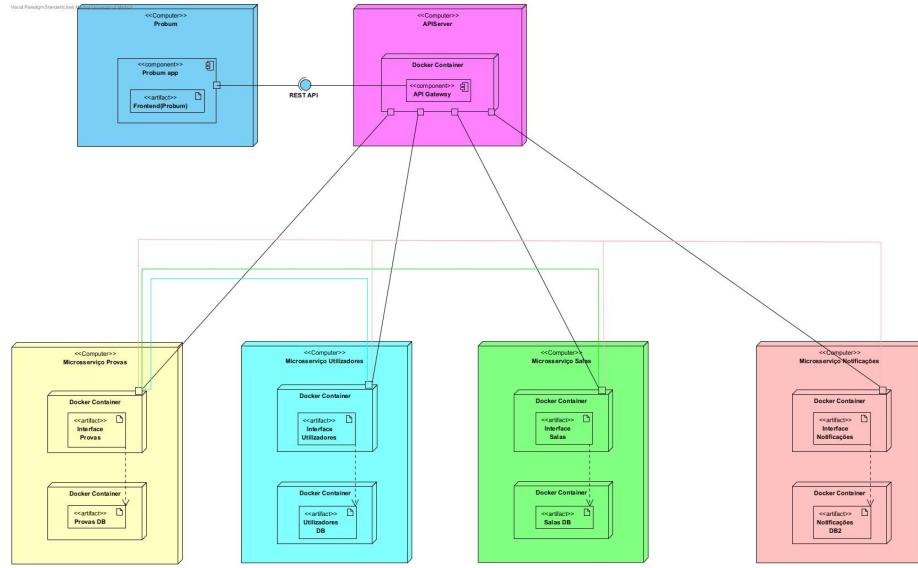


Figure 24: Diagrama de *Deployment*

8 Conceitos Transversais

Neste capítulo, serão discutidos diversos tópicos pertinentes relacionados com as diferentes partes do sistema, com o objetivo de apresentar uma visão abrangente da aplicação.

8.1 Conceitos de Domínio

Como já foi discutido anteriormente em maior detalhe, no capítulo 4 - **Estratégias de Solução** -, este sistema está baseado em microsserviços. Desta forma, cada serviço consegue funcionar independentemente dos outros, resultando numa redução significativa da interdependência entre os diferentes módulos. Essa autonomia não apenas promove uma maior flexibilidade no desenvolvimento e manutenção, mas também contribui para a criação de um sistema menos suscetível a comprometimentos.

Ao mitigar as interligações, fortalecemos a resiliência do sistema, proporcionando uma base sólida para a estabilidade e segurança contínua. Para mais informação relativamente aos conceitos de domínio, estes foram identificados e explicitados no relatório da **Fase 1**, no capítulo de **Taxinomia e definições**, na secção **Modelo de Domínio**.

8.2 Experiência do Utilizador

A interface com o utilizador tem por base o GUI apresentado anteriormente, nos Mockups presentes na Fase 1, no capítulo **Anexos**. O utilizador interage com a plataforma via aplicação web e, dependendo de quem for, vai poder registar utilizadores na plataforma, criar e editar provas ou responder a provas.

8.3 Proteção e Segurança

Para assegurar uma política eficaz de proteção de dados dos utilizadores, todos os dados deverão ser guardados segundo um protocolo de segurança, que aplica encriptação ao nível de dados pessoais, especialmente informações pessoais como palavras-passe, são armazenados com criptografia. Esta medida visa proteger os dados sensíveis contra acessos não autorizados, garantindo a integridade dos dados.

Os utilizadores só conseguem aceder à plataforma via login, sendo por isso necessário uma rotina de autenticação sempre que o utilizador pretenda entrar na aplicação. Desta forma o serviço garante a proteção dos dados do utilizador.

9 Requisitos de Qualidade

Nesta seção, foi elaborada uma **Árvore de Qualidade** referente aos Requisitos Não Funcionais da aplicação. Os requisitos de qualidade estão devidamente identificados e elucidados no relatório anterior, no capítulo referente aos Requisitos Não Funcionais, distribuídos em distintas secções.

Também foi elaborada uma tabela referente aos requisitos de qualidade provenientes da Árvore de Qualidade.

9.1 Árvore de Qualidade

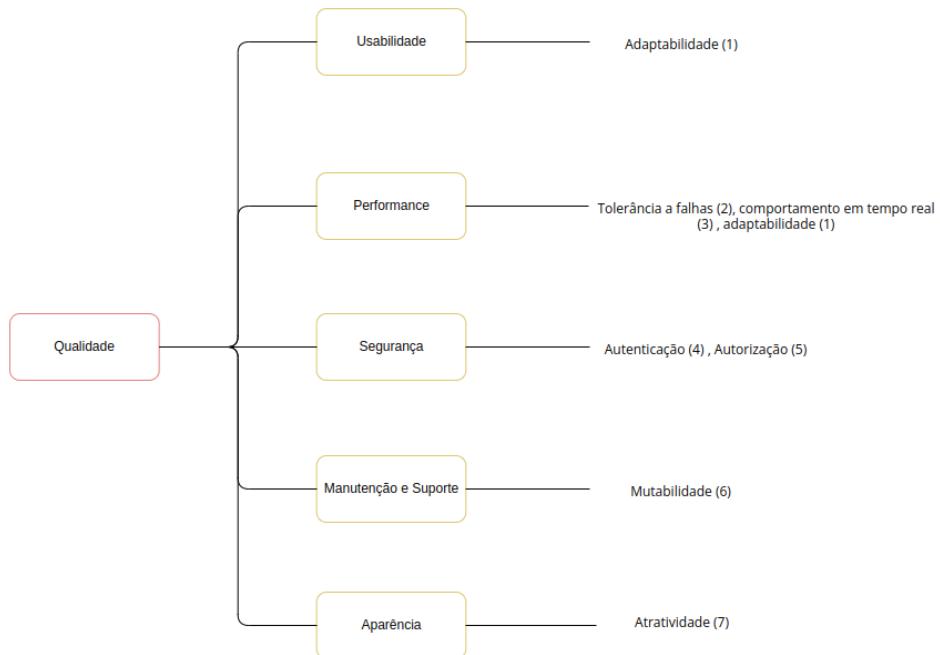


Figure 25: Árvore de Qualidade

9.2 Cenários de Qualidade

Número	Descrição
1	A aplicação será adaptável a vários tipos de dispositivos diferentes, principalmente computadores de gama baixa. A aplicação também será projetada de modo a ser de fácil utilização, acessível para utilizadores com deficiências visuais e apresentará os dados por ordem cronológica, de forma a ser mais intuitiva.
2	A aplicação estará preparada para processar input incorreto inserido pelo utilizador. Este tipo de input não irá influenciar negativamente a aplicação. Também estará preparada para situações de falhas de conexão com os servidores.
3	Depois da implementação de vários servidores e de vários testes os analistas do projeto poderão analisar o comportamento da aplicação em tempo real, assim como o seu tempo de resposta.
4	A aplicação estará pronta para permitir que apenas utilizadores registados no sistema poderão interagir com a mesma.
5	A aplicação estará preparada para restringir o acesso a funcionalidades de acordo com o utilizador.
6	O código deverá ser fácil de manter e modificar, uma vez que há um suporte quando à necessidade de manter o código desenvolvido documentado.
7	A aplicação possuirá uma interface amigável e de fácil uso para o utilizador geral. A aplicação deverá apresentar um cronometro e um aluno com visão corrigida 20/20 deve ser capaz de ler todas as questões.

10 Riscos e Custos

No desenvolvimento de uma aplicação para alunos e professores universitários, é essencial identificar e gerir riscos técnicos. A integração de diversas tecnologias, as mudanças rápidas nos requisitos devido a metodologias educacionais em evolução e desafios de escalabilidade são áreas críticas.

Medidas proativas incluem atualizações regulares para tecnologias emergentes, protocolos de segurança robustos e uma infraestrutura escalável. A comunicação eficiente com os utilizadores finais e a incorporação de feedback são práticas valiosas para identificar potenciais desafios desde o início do desenvolvimento.

Ao abordar essas preocupações, o desenvolvimento da aplicação se torna mais resiliente, garantindo uma solução robusta que atende às exigências diversas do ambiente académico.

11 Implementação

Nesta etapa, o grupo foi organizado em 5 subgrupos especializados. Cada subgrupo concentrou-se no desenvolvimento de um microserviço específico, contribuindo assim para a criação de uma arquitetura robusta e altamente modular para a aplicação. Essa estratégia permitiu uma clara divisão de responsabilidades, promovendo o progresso simultâneo em várias partes do sistema. Em seguida serão exploradas em mais detalhe as implementações feitas para cada microserviço.

11.1 Provas

O microserviço de provas desempenha um papel central na aplicação Probum, sendo responsável pela criação, realização, correção e consulta de provas acadêmicas.

Para a implementação deste microserviço, optamos pelo uso do Node.js e do MySQL. A escolha do Node.js foi motivada pela familiaridade com JavaScript, eficiência no desenvolvimento e integração facilitada com o frontend. Quanto ao MySQL, escolhemos este banco de dados devido à sua natureza relacional e transacional, proporcionando uma escalabilidade robusta.

Para assegurar uma implementação coesa e eficiente do microserviço, foi realizado um modelo de base de dados estruturado. Com a realização deste modelo é possível ter os dados melhor organizados, minimizando incoerências e simplificando o processo de implementação. A realização do mesmo é também importante para criar uma estrutura robusta, reduzindo a probabilidade de alterações significativas no futuro, proporcionando uma base estável e confiável para o desenvolvimento contínuo do microserviço.

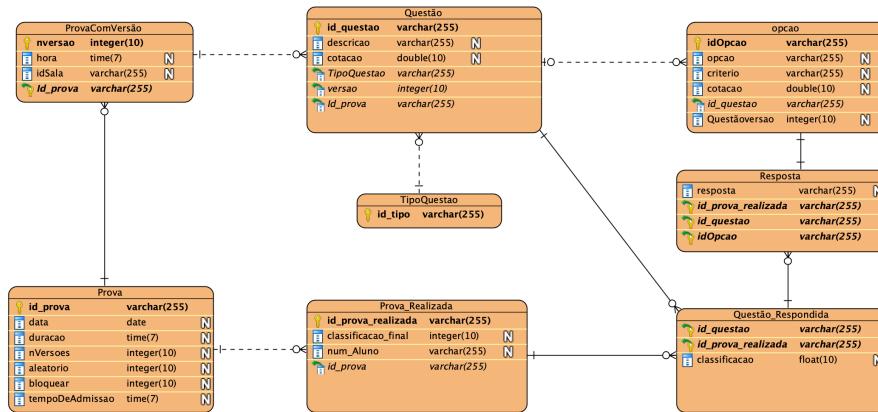


Figure 26:

Primeiro o grupo começou por criar a tabela **Prova** que serve como a entidade principal para registrar os detalhes gerais de cada prova. Esta armazena informações fundamentais como o seu ID, a data, a duração, o número de versões, além de indicadores como se a prova é aleatória e se é bloqueada (não poder voltar atrás nas perguntas).

A tabela **ProvaComVersão** foi criada para lidar com diferentes versões das provas, uma vez que cada versão terá as suas próprias questões. Esta tabela está relacionada à tabela *Prova* por meio do identificador da prova. (hora ??mudaconsoante versao??)

A tabela **Questão** é responsável por armazenar detalhes das questões associadas às provas. Cada questão é identificada exclusivamente pelo *id_questao* e está vinculada à tabela **ProvaComVersao** para indicar a versão específica à qual pertence. Tem também informação quanto à descrição e o valor da questão. (nao faz mt sentido a questao e a opcao terem os dois valores ??)

A tabela **Opção** trata das opções disponíveis para as questões. Ela contém informações sobre cada opção, incluindo identificador único (*idopcao*), o texto da opção, o critério e quanto vale. O critério é uma String de modo a funcionar para qualquer tipo de questão. Quando é o caso da escolha múltipla uma opção selecionada conta como *True* e uma não seleciona conta como *False*.

A tabela **Prova_realizada** regista uma prova realizada por um dado aluno, contendo o número do mesmo, a classificação final e referência à prova original. Uma prova *prova_realizada* só é inserida na tabela após o aluno responder a uma dada prova. Esta é inserida com a classificação final a NULL, e só posteriormente, após a correção da prova é que é alterado esse valor. A classificação final é calculado através da soma das cotações de cada questão. A chave estrangeira *id_prova* está vinculada à tabela *Prova*

A tabela **Questão_Respondida** armazena as questões específicas respondidas em uma prova realizada, juntamente com a classificação obtida, inicialmente a NULL, calculado só posteriormente após a correção da prova. O cálculo é feito através da soma das cotações obtidas em cada opção. As chaves estrangeiras **id_prova_realizada** e **id_questao** garantem a ligação com as tabelas **Prova_realizada** e **Questao**.

A tabela **Resposta** regista as respostas dadas pelos alunos para cada opção de uma dada questão. É através da comparação da resposta guardada nesta tabela com o critério da opção que é calculado a classificação de uma dada questão. Esta tabela conecta as tabelas **prova_realizada**, **Questao** e **Opcão** através de chaves estrangeiras.

Falar das funcoes mais importantes???

Método	Rota	Descrição
GET	/provas/docente/<idD>	Lista das provas de um docente
GET	/provas/<idP>/versao/<nV>	Informações sobre versão de prova
GET	/provas/<idP>/versao/<nV>/questoes	Lista de questões de uma versão
GET	/provas/<idP>/versao/<nV>/questoes/<idQ>	Informação de uma questão específica
GET	/provas/prova_realizada/<idP>	Prova realizada para consulta
GET	/provas/aluno/<idA>	Lista de provas e o seu estado
GET	/provas/<idP>	Informações de prova
POST	/provas/<idP>/responder	Respostas do aluno
POST	/provas/	Adicionar provas à base de dados
POST	/provas/<idP>/versao/<nV>/questoes	Adicionar questões a uma versão da prova
PUT	/provas/<idP>/corrigir	Corrige uma prova automaticamente
PUT	/provas/docente/<idD>/corrigir	Corrige todas as provas de um docente automaticamente
DELETE	/provas/<idP>	Apaga um prova
DELETE	/provas/<idP>/versao/<nV>	Apaga uma versão da prova
DELETE	/provas/<idP>/versao/<nV>/questoes/<idQ>	Apaga uma questão da prova
DELETE	/provas/<idP>/versao/<nV>/opcoes/<idO>	Apaga uma opção de uma questão

12 Conclusão

Esta segunda fase do trabalho destaca o aprofundamento do planeamento arquitetural, formulando uma estrutura e objetivos para a implementação subsequente.

A terceira fase concentrar-se-á na aplicação do plano desenvolvido, garantindo um desenvolvimento coeso e detalhado. A coesão entre as partes do projeto, considerando sua natureza iterativa será crucial para a evolução bem sucedida. Além disso, a avaliação crítica do documento de arquitetura destaca a conformidade com o template fornecido, sugerindo melhorias potenciais para implementações futuras.

Em resumo, o balanço do trabalho até ao momento é positivo, superando desafios e cumprindo requisitos, proporcionando uma base sólida para a próxima fase de implementação.

13 Glossário

Termos	Definições
Probum	Nome da plataforma e-learning a implementar
RAS	Requisitos e Arquiteturas de Software
IES	Instituição de Ensino Superior
REST	Representational state transfer
API	Application Programming Interface
