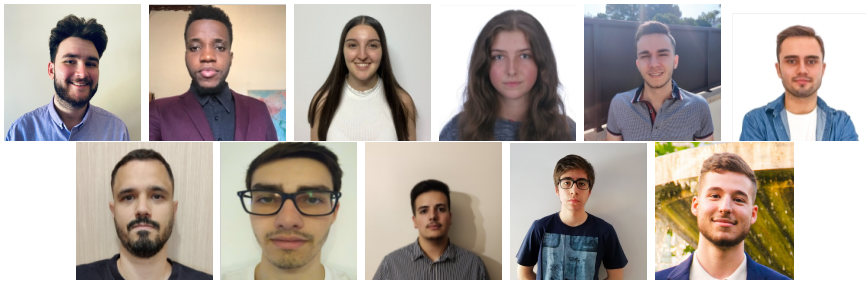
UMinho
Mestrado Engenharia Informática
Requisitos e Arquiteturas de Software (2023/24)

PROBUM

Grupo C, PL2 / Entrega 2

pg54465 - André da Luz Lopes Brandão
pg51634 - Elione Culeca Cossengue
pg53829 - Gabriela Santos Ferreira da Cunha
pg53879 - Inês Nogueira Ferreira
pg53905 - João António Redondo Martins
pg53985 - José Rafael Cruz Ferreira
pg52691 - Marcos Felipe Tonelli de Carvalho
pg54105 - Miguel Silva Pinto
pg52698 - Nuno Ricardo Oliveira Costa
pg54146 - Pedro Miguel Castilho Martins
pg54197 - Rodrigo José Teixeira Freitas



Braga, 1 de dezembro de 2023

Prefácio

No processo de analisar qual o padrão arquitetural mais adequado, o grupo começou por analisar todos os requisitos não funcionais, identificando os mais pertinentes e estabelecendo uma hierarquia de prioridades para os mesmos. Em seguida, partiu-se para a elaboração de um diagrama de blocos com a finalidade de identificar os serviços do sistema e os requisitos não funcionais associados aos mesmos. Após concluído, passou-se por efetuar uma análise detalhada dos padrões arquiteturais, realçando as características únicas dos modelos *Layered* e *Microservices*. Com o intuito de determinar o padrão arquitetural mais adequado, foram realizadas diversas iterações nos diagramas de sequência. Após a seleção do padrão a ser adotado, o grupo investigou também as ferramentas mais apropriadas para trabalhar com esse padrão específico. Para além disso, analisaram-se também várias formas de implementar a aplicação desenvolvida num ambiente real, identificando limitações e restrições potenciais em relação à solução alcançada.

Um dos grandes desafios ao qual estivemos expostos nesta fase foi a colaboração com um número elevado de membros por equipa, algo que nunca tínhamos experienciado em contexto académico. Para assegurar um desempenho eficaz no trabalho, fragmentamos o grupo em pequenas equipas, cada uma responsável por tarefas específicas. A utilização conjunta do Trello revelou-se um recurso fundamental para otimizar esta colaboração, sendo muito mais simples atribuir e rever as tarefas realizadas por cada grupo e estar a par da situação atual do projeto. Por último, importa salientar que as responsabilidades foram atribuídas às equipas após cada aula e, a cada fim de semana, foi realizada uma reunião onde era apresentado o progresso e trabalho de cada membro do grupo, juntamente com as questões e obstáculos encontrados. Na tabela seguinte, inclui-se a reflexão sobre o contributo de cada membro para o trabalho.

Número	Nome	Classificação
pg54465	André da Luz Lopes Brandão	+0.3
pg51634	Elione Culeca Cossengue	−0.7
pg53829	Gabriela Santos Ferreira da Cunha	+0.3
pg53879	Inês Nogueira Ferreira	−1
pg53905	João António Redondo Martins	+0.3
pg53985	José Rafael Cruz Ferreira	+0.1
pg52691	Marcos Felipe Tonelli de Carvalho	0
pg54105	Miguel Silva Pinto	+0.3
pg52698	Nuno Ricardo Oliveira Costa	+0.1
pg54146	Pedro Miguel Castilho Martins	+0.3
pg54197	Rodrigo José Teixeira Freitas	0

Conteúdo

Introdução e Objetivos

Neste trabalho prático da unidade curricular de Requisitos e Arquiteturas de Software, foi elaborado um documento de arquitetura para a aplicação Probum, um sistema para a realização de provas digitais. Esta aplicação tem como objetivo permitir a digitalização dos processos associados às provas de avaliação, englobando a sua criação, com a respetiva calendarização e reserva das salas, realização e correção.

Neste documento será abordada uma grande variedade de tópicos importantes para uma implementação bem-sucedida e estruturada da aplicação, mencionando restrições arquiteturais, requisitos de qualidade, modelação de blocos funcionais, decisões arquiteturais, estratégias de solução, entre outros.

Resumo dos Requisitos

Na primeira fase deste projeto foram definidos os requisitos funcionais do sistema. A compreensão detalhada de cada um destes não só nos fornece uma visão clara das funcionalidades disponíveis como desempenha o papel de bússola para as decisões críticas associadas à arquitetura. Deste modo, as decisões apresentadas neste documento, relativas à arquitetura do sistema, tiveram como base a especificação elaborada no documento de requisitos.

As principais interações entre os atores do sistema foram resumidas no seguinte diagrama, elaborado na primeira fase.

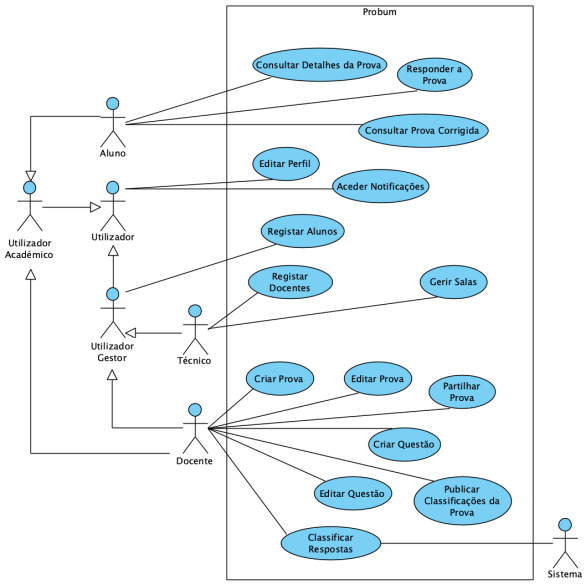


Figura 1: Diagrama de *use cases*.

A partir destas interações, foram elaborados os requisitos funcionais. A análise destes requisitos foi o ponto de partido desta fase, com vista a perceber quais representavam as funcionalidades base do sistema. Na tabela 1, destacam-se os requisitos que a equipa considerou serem de alta importância.

#REQ	Use Cases	Descrição
14	*Criar Prova*	*O Docente cria uma prova de avaliação*
17	*Criar Prova, Editar Prova*	*O Sistema faz uma calendarização inteligente, indicando as salas, horários e alunos alocados a cada sala*
19	*Criar Prova, Editar Prova*	*O Docente inscreve Alunos numa prova de avaliação*
21	*Criar Prova, Editar Prova*	*O Docente cria diferentes versões de uma prova de avaliação e associa cada versão às salas e horários calendarizados*
22	*Criar Prova, Editar Prova*	*O Docente adiciona Questões de escolha múltipla a uma prova de avaliação*
29	*Editar Prova, Consultar Detalhes da Prova, Partilhar Prova, Consultar Prova Corrigida*	*O Sistema disponibiliza a lista de provas a que um Utilizador tem acesso*
48	*Responder Prova*	*O Aluno tem uma opção para responder a Provas*
58	*Classificar Respostas*	*O Sistema corrige automaticamente perguntas com resposta pré-definida por um Docente*
59	*Classificar Respostas*	*O Docente corrige manualmente uma Prova, atribuindo pontuação a cada resposta*
63	*Consultar Prova Corrigida*	*O Aluno consulta a sua classificação de uma Prova*

Tabela 1: Requisitos funcionais.

Stakeholders

Função/Nome	Expectativas
Departamentos pedagógicos de IESs	*Disponibilização aos seus corpos docentes e discentes de um sistema informático que automatize a criação, calendarização, realização e correção de provas de avaliação*
Docentes do ensino superior	*Uma ferramenta informática que facilite a criação, distribuição e correção das provas de avaliação e que, ao mesmo tempo, os mantenha capacitados para a resolução de situações anómalas durante as realizações destas*
Alunos	*Um produto simples, eficiente, robusto, amigável e tolerante a falhas para que possam realizar as suas provas*
Técnicos	*Esperam que o produto seja fácil de instalar e configurar e que disponibilize métricas e logs que permitam antecipar e diagnosticar eventuais problemas*

Restrições

Restrições Arquiteturais

As restrições de arquitetura para o sistema Probum identificadas pela equipa de desenvolvimento foram:

- A aplicação deve executar na infraestrutura atual da respetiva IES;
- O computador disponibilizado a cada Aluno, no momento em que realiza uma prova de avaliação, apenas deve permitir acesso ao Probum.

Outras Restrições

O projeto possui também restrições temporais e orçamentais por forma a garantir que o desenvolvimento do sistema seja viável.

Restrições Temporais

O documento presente deverá ser entregue até ao dia 1 de dezembro de 2023, para que seja avaliado e consequentemente ajustado com vista a servir como base para a fase seguinte.

Restrições Orçamentais

O orçamento total para o desenvolvimento do projeto é de 20 000€ (vinte mil euros), durante um período de 4 meses. Este foi o valor acordado pelos *stakeholders* e todos os interessados pelo desenvolvimento do sistema.

Requisitos de Qualidade

A par dos requisitos funcionais, foram também definidos os requisitos não funcionais que moldam a experiência do utilizador, a confiabilidade, a segurança e a escalabilidade do sistema. Estes requisitos são fundamentais para assegurar não apenas a entrega de um produto que execute as funções desejadas, mas que o faça de maneira eficiente, segura e adaptável ao contexto real. Do conjunto de requisitos não funcionais, destacamos alguns destes e construímos uma árvore de qualidade devidamente ordenada pela prioridade atribuída a cada um dos requisitos e respetiva característica de qualidade.

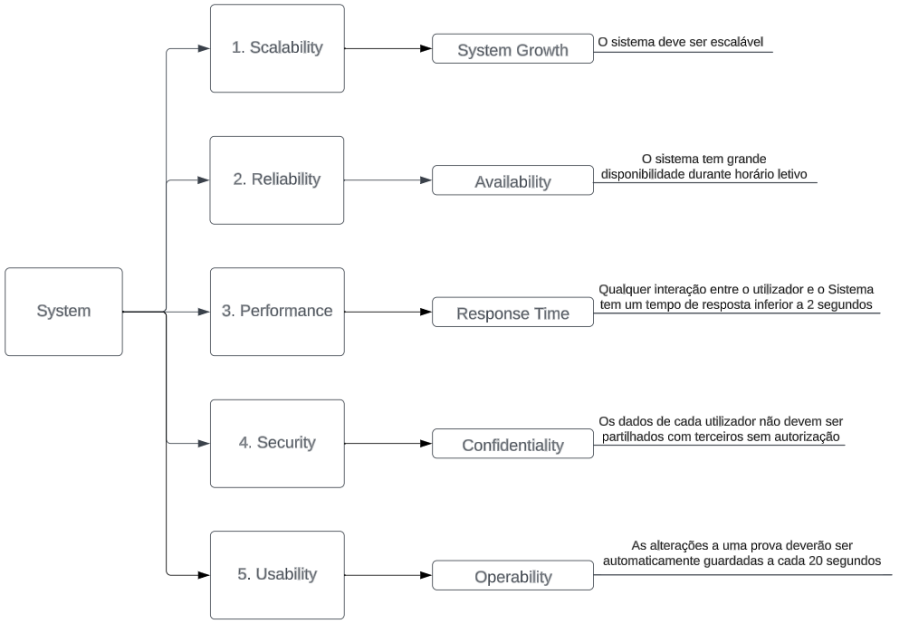


Figura 2: Árvore de qualidade.

RNF 13 - O sistema deve ser escalável

Um sistema de provas online é bastante suscetível a flutuações de demanda dos utilizadores, devido à pontualidade de uma prova de avaliação. Por outras palavras, uma prova traz vários alunos simultaneamente numa hora exata,

podendo acarretar problemas de desempenho para a aplicação, o que não é desejável num horário de realização de uma prova. O nosso principal objetivo é garantir que o sistema não sofre deste tipo de problemas, existindo a necessidade deste ser escalável, permitindo lidar com estes picos de carga sem comprometer o desempenho e garantindo que os utilizadores realizam as provas sem atrasos ou interrupções.

RNF 14 - O sistema tem grande disponibilidade durante horário letivo

O segundo requisito mais prioritário refere-se à disponibilidade horária do sistema. A grande disponibilidade durante o horário letivo é fundamental para garantir que as provas online possam ser realizadas de acordo com os horários e prazos académicos estabelecidos, mantendo a integridade do processo educacional. Durante o horário letivo, é esperado que haja um aumento significativo no número de utilizadores a usufruir do sistema. É, por isso, importante garantir que o mesmo está preparado para lidar com uma maior carga de pedidos concentrados nesse período.

RNF 9 - Qualquer interação entre o utilizador e o Sistema tem um tempo de resposta inferior a 2 segundos

Este requisito é classificado como um requisito de performance e atribuímos-lhe a terceira maior prioridade. O tempo de resposta rápido é fundamental para uma boa experiência com a aplicação e isso significa que o sistema deve responder rapidamente a qualquer interação do utilizador. Embora esta seja uma meta importante, não é tão crítica quanto a escalabilidade e a disponibilidade horária do sistema. Um tempo de resposta rápido é valioso para a experiência do utilizador, mas se o sistema não for escalável o suficiente para lidar com a demanda ou não estiver disponível quando necessário, a eficiência do tempo de resposta rápido é comprometida.

RNF 24 - Os dados de cada utilizador não devem ser partilhados com terceiros sem autorização

A proteção de dados é crucial para conformidade legal e confiança do utilizador. Este é um requisito que qualquer aplicação com persistência de dados deve definir. Neste caso particular, relativo à educação, os resultados de provas e avaliações muitas vezes contêm informações sensíveis sobre o desempenho académico de um estudante e é necessário garantir que esses dados não são partilhados sem autorização. A proteção da privacidade dos dados também ajuda a manter a integridade do processo de avaliação, na medida em que se os dados, como respostas a perguntas de prova, fossem partilhados indevidamente, surgiriam oportunidades de fraude e plágio. Por último, muitas regiões e países têm leis rigorosas acerca da proteção de dados e cumprir esse regulamento é crucial para evitar penalidades legais.

RNF 16 - As alterações a uma prova deverão ser automaticamente guardadas a cada 20 segundos

Ao salvar automaticamente as alterações feitas em cada prova a cada 20 segundos, o sistema reduz significativamente o risco de perda de dados em caso de falhas ou interrupções imprevistas. Os alunos podem continuar a responder à prova de onde pararam com base na última versão guardada automaticamente, constituindo uma experiência mais fluida e evitando frustrações associadas à perda de trabalho. Para além disso, durante a prova, os alunos podem se concentrar exclusivamente na resolução das questões, contribuindo, à partida, para um melhor desempenho académico.

Âmbito e Contexto do Sistema

Nesta secção apresentamos uma visão abrangente e organizada das fronteiras do sistema e as suas interações com as entidades externas que o rodeiam. Como tal, elaboramos o seguinte diagrama de contexto, representado na figura 3, onde observamos as interações entre o sistema Probum e os restantes elementos. Este diagrama ajuda a estabelecer claramente os limites do sistema, delineando o que está dentro e fora do contexto do projeto, evitando mal-entendidos sobre as responsabilidades e funcionalidades do sistema.

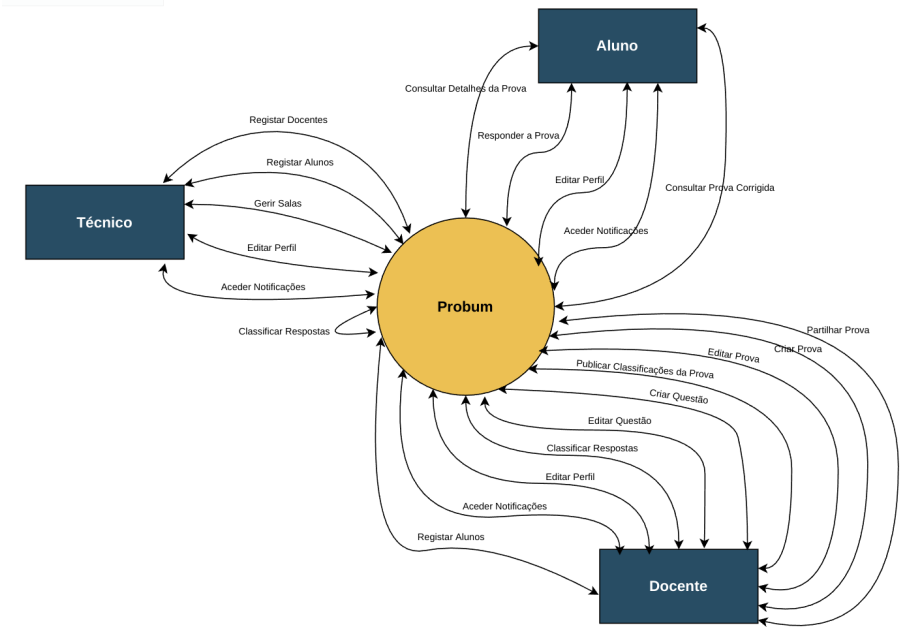


Figura 3: Diagrama de contexto.

Neste diagrama, podemos concluir que existem várias funcionalidades exclusivas a cada utilizador. Por exemplo, um docente não deverá poder responder a uma prova, assim como um aluno não deverá conseguir proceder à criação de uma prova. Por outro lado, os 3 atores principais (técnico, aluno e docente) acabam por estar englobados numa hierarquia, sendo cada um deles um utilizador, este que, por sua vez, poderá editar o seu perfil e aceder às notificações. Deste modo, podemos ainda concluir que o sistema deverá oferecer diferentes vistas de utilização conforme o tipo de utilizador.

Decomposição Funcional

Através do diagrama de contexto anterior, pudemos iniciar a construção de um diagrama de blocos funcionais, onde cada bloco representa uma parte do sistema responsável por realizar uma tarefa específica, neste caso, um conjunto de requisitos. Este processo acabou por ser iterativo, uma vez que à medida que atribuíamos os requisitos aos respetivos blocos funcionais, foram surgindo novos sub-blocos. Por último, efetuamos a conexão entre os vários blocos funcionais, despoletada por alguns requisitos que, numa primeira visualização, dado um par de blocos conectados, poderiam estar integrados em qualquer um destes blocos.

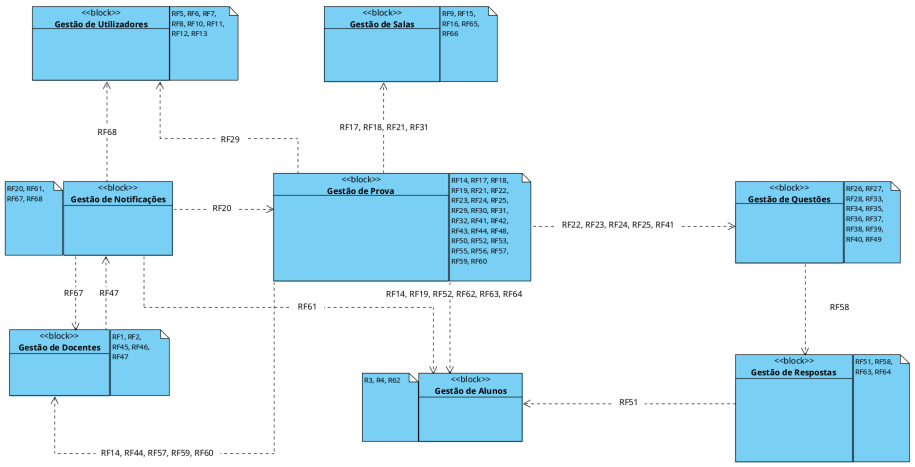


Figura 4: Diagrama de blocos funcionais.

Estratégia de Solução

Nesta secção, abordamos os diferentes modelos de arquitetura e todo o processo associado à escolha do modelo mais apropriado, do nosso ponto de vista, para este projeto. Para além disso, serão também apresentadas as decisões sobre as tecnologias adotadas para o desenvolvimento de cada microsserviço.

Modelos de Arquitetura

As arquiteturas de *software* podem ser categorizadas principalmente em duas grandes abordagens: arquiteturas monolíticas e arquiteturas distribuídas. Numa arquitetura monolítica, toda a aplicação é desenvolvida como um único sistema. Por outro lado, numa arquitetura distribuída, o sistema é dividido em vários componentes independentes que comunicam entre si através de uma rede.

Para cada uma destas grandes abordagens referidas, existem diversos modelos e padrões que podem ser seguidos. Numa arquitetura monolítica, utilizam-se modelos como a arquitetura em *pipeline*, a arquitetura de *microkernels* e a arquitetura em camadas. Já para a arquitetura distribuída, os modelos mais comuns são a arquitetura *space-based*, as orientadas a serviços, as orientadas a eventos e a de microsserviços.

De forma a analisar a melhor estratégia para desenvolver a aplicação, filtramos estes modelos de modo a termos apenas um correspondente a cada abordagem. Assim, foram somente consideradas duas arquiteturas: a arquitetura baseada em camadas, representando a abordagem monolítica, e a baseada em microsserviços, representando a abordagem distribuída.

Arquitetura em Camadas

A arquitetura em camadas organiza o sistema em camadas hierárquicas, onde cada camada desempenha um papel específico e comunica apenas com camadas adjacentes. Este modelo é geralmente dividido em 3 camadas - a interface com o utilizador, a lógica de negócios e a camada de dados. Esta estratégia é principalmente caracterizada por um controlo centralizado, facilidade na manutenção e escalabilidade e isolamento de responsabilidades.

Arquitetura de Microsserviços

A arquitetura assente em microsserviços divide o sistema em serviços independentes, pequenos e autónomos, que se comunicam por APIs. Cada serviço é dedicado a uma função específica e pode ser desenvolvido e implementado independentemente. Esta estratégia é fortemente caracterizada pelo seu desacoplamento de serviços, grande escalabilidade e flexibilidade e implementação independente.

Na figura 5, apresenta-se uma avaliação de ambas as arquiteturas perante cada característica arquitetural, de acordo com o livro *Fundamentals of Software Architecting*.

Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★
Elasticity	★
Evolutionary	★
Fault tolerance	★
Modularity	★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★

(a) Arquitetura em camadas.

Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★★★★
Evolutionary	★★★★★
Fault tolerance	★★★★
Modularity	★★★★★
Overall cost	★
Performance	★★
Reliability	★★★★
Scalability	★★★★★
Simplicity	★
Testability	★★★★

(b) Arquitetura de microsserviços.

Figura 5: *Ratings* das arquiteturas de acordo com a característica de arquitetura.

Escolha do Modelo

A escolha da arquitetura de solução a ser usada no projeto foi baseada nas necessidades apontadas pelos requisitos não-funcionais aos quais atribuímos maior importância, apresentados anteriormente no relatório. Desta forma, para cada requisito não-funcional, pontuamos cada uma das arquiteturas referidas, de acordo com as características de qualidade de cada requisito, como escalabilidade, simplicidade e custo.

Requisito Não Funcional		**Microsserviços**	**Layered**
13	O sistema deve ser escalável	⋆⋆⋆⋆⋆	⋆
14	O sistema tem grande disponibilidade durante horário letivo	⋆⋆⋆⋆⋆	⋆
9	Qualquer interação entre o utilizador e o Sistema tem um tempo de resposta inferior a 2 segundos	⋆⋆⋆	⋆⋆⋆⋆
24	Os dados de cada utilizador não devem ser partilhados com terceiros sem autorização	⋆⋆⋆⋆⋆	⋆
16	As alterações a uma prova deverão ser automaticamente guardadas a cada 20 segundos	⋆⋆⋆	⋆⋆⋆⋆

Tabela 3: *Star rating.*

RNF 13 - O sistema deve ser escalável

As pontuações atribuídas a cada arquitetura de acordo com este requisito acabam por ser óbvias, uma vez que os microsserviços possibilitam escalabilidade horizontal, permitindo a expansão independente de cada serviço para lidar com cargas específicas, enquanto que a arquitetura em camadas geralmente escala verticalmente, limitando a capacidade de expansão para um único servidor. A modularidade dos microsserviços permite uma adaptação mais rápida a mudanças de requisitos, enquanto que se for necessário atualizar algo numa arquitetura em camadas terá de se substituir a aplicação inteira, resultando em processos mais complexos e períodos de inatividade.

RNF 14 - O sistema tem grande disponibilidade durante horário letivo

Como foi referido no requisito anterior, é possível distribuir a carga de trabalho de forma eficiente pelos microsserviços, sendo mais fácil garantir alta disponibilidade durante o horário letivo, quando a demanda for significativa. Já em relação à arquitetura em camadas, esta pode encontrar limitações na capacidade de lidar com estes aumentos de pedidos durante o horário letivo. No caso dos microsserviços, se um microserviço falhar, isso não afeta necessariamente todo o sistema, contribuindo para uma maior disponibilidade, enquanto que, num sistema único, uma falha numa camada pode afetar diretamente todas as camadas subsequentes, resultando numa menor disponibilidade global. Para além disso, se necessitarmos de diminuir os recursos para a aplicação, podemos ter um controlo com uma granularidade mais fina dos recursos a serem usados conforme a carga da aplicação.

RNF 9 - Qualquer interação entre o utilizador e o Sistema tem um tempo de resposta inferior a 2 segundos

Numa arquitetura de camadas, a comunicação entre os diferentes componentes geralmente ocorre internamente, dentro da camada, resultando num menor *overhead* de comunicação em comparação com microsserviços, onde a comunicação entre os serviços é mais complexa. Contudo, esta discrepância não é tão acentuada como nos requisitos anteriores, uma vez que podemos configurar a arquitetura de maneira a alocar mais recursos para as partes do sistema específicas que precisam de lidar com a necessidade destes tempos de resposta rápidos.

RNF 24 - Os dados de cada utilizador não devem ser partilhados com terceiros sem autorização

Ao utilizarmos serviços, garantimos que há isolamento de dados, sendo que cada base de dados pertence a um serviço independente. Para além disso, cada microsserviço pode definir as suas próprias políticas de autorização, proporcionando uma maior flexibilidade e um maior controlo sobre os dados. Numa arquitetura em camadas, a informação está mais agrupada e, caso haja uma

fuga, o impacto será mais abrangente sendo que, à partida, cada camada engloba mais informação do que um microsserviço.

RNF 16 - As alterações a uma prova deverão ser automaticamente guardadas a cada 20 segundos

Numa arquitetura de microsserviços, esta funcionalidade estará sempre dependente da latência e número de comunicações entre microsserviços. Sendo que, numa primeira análise, esta funcionalidade não exigirá um número elevado de transações entre serviços, as pontuações atribuídas não foram muito diferentes. A arquitetura em camadas acabou por ter uma pontuação ligeiramente maior, devendo-se ao facto de esta funcionalidade ser um pouco mais direta perante o seu uso, pois o controlo sobre esta lógica é mais centralizado.

Deste modo, de acordo com a análise representada na tabela 3, a equipa de desenvolvimento estabeleceu que o padrão arquitetural a adotar seria o de microsserviços.

Implementação da Arquitetura

Com base na arquitetura escolhida, ou seja, a de microsserviços, tivemos de definir quais seriam os potenciais microsserviços da nossa aplicação. Começamos por atribuir um número de microsserviços igual ao número de blocos funcionais que havíamos anteriormente definido na secção IV.

Em seguida, foram realizados vários diagramas de sequência, com vista a definir a cooperação entre os microsserviços para cada funcionalidade. Assim como a decomposição funcional, a definição dos microsserviços trata-se de um processo iterativo, na medida em que, após a realização dos diagramas de sequência, deve ser efetuada uma devida análise destes para perceber se resultam em várias transações entre microsserviços, algo que não é desejável, sendo o objetivo de cada iteração minimizar estas transações.

Na seguinte tabela, apresentamos o número de transações identificadas para cada *use case* em cada iteração efetuada.

Use Case		1ª Iteração	2ª Iteração
1	Registar Docente	2	2
2	Registar Aluno	2	2
3	Autenticar	2	1
4	Editar Perfil	3	2
5	Criar Prova	10	8
6	Criar Questões	1	1
7	Editar Prova	9	8
8	Editar Questões	1	1
9	Consultar Detalhes da Prova	5	4
10	Partilhar Prova	4	4
11	Responder a Prova	8 + N	7 + N
12	Classificar Respostas	3 + 3*N	2 + 2*N
13	Publicar Classificações da Prova	8	7
14	Consultar Prova Corrigida	6	5
15	Gerir Salas	6	6
16	Aceder às Notificações	2	2

Tabela 4: Número de interações entre microsserviços para cada *use case*.

Nesta tabela, o N refere-se a um número que é dependente de fatores que são variáveis, em todos os casos o N corresponde ao número de respostas de uma dada prova, à exceção do UC-12 Classificar Respostas em que o N representa o número de provas respondidas por alunos que o docente pretende corrigir.

Torna-se também pertinente referir que num contexto real, e dependendo da complexidade do projeto, este processo iterativo contém, geralmente, um maior número de iterações de forma a alcançar uma solução ótima. Por exemplo, é comum e plausível a criação de iterações adicionais que, numa primeira análise, podem vir a fazer sentido mas, após a sua realização, podem não resultar nos microsserviços definidos, devido à agregação ou segregação excessiva dos potenciais microsserviços. Contudo, dado o contexto académico deste projeto e as consequentes limitações horárias, o grupo acabou por realizar apenas 2 iterações. Em seguida, explicaremos as mudanças efetuadas entre a 1ª e a 2ª iteração.

Agrupamento das provas, questões e respostas

No *use case* #12 identificamos várias interações entre o serviço de provas, questões e respostas que são repetidas conforme o número de provas a serem corrigidas. Para além disso, esta operação é recorrente em certos períodos de tempo, ou seja, os períodos de realização de provas em que vários docentes corrigem várias provas. Assim sendo, optámos por unir estes três microsserviços, formando só um microsserviço denominado de serviço de provas.

No *use case* #11 também identificamos o problema anteriormente referido, visto que, após o contacto ao serviço de provas para verificar se o aluno pode responder à mesma, necessitava de ir ao serviço de questões para obter as questões dessa prova. Assim sendo, como este mesmo pedido é feito numa quantidade significativa quando vários alunos vão responder à mesma prova no mesmo horário, a solução anteriormente apresentada resolve esta situação e este elevado número de interações entre microsserviços.

Por outro lado, nos *use cases* #13 e #15 verificamos 7 e 6 transações respetivamente, mas estas operações, publicar classificações e gerir as salas, são feitas menos frequentemente relativamente às referidas anteriormente e, assim sendo, não se justifica a alteração da arquitetura para otimizar estas operações.

Uniformização dos docentes e alunos

De seguida, também optamos por uniformizar o serviço de docentes e alunos, visto que são ambos utilizadores e maior parte dos seus atributos são comuns.

Orquestração

Na arquitetura de microsserviços, a equipa docente apresentou duas visões de os serviços comunicarem entre si: **orquestração** e **coreografia**. Para resolver o problema em causa, decidimos utilizar o modelo de **orquestração**.

Este modelo implica a utilização de um componente central responsável por coordenar e gerir as interações entre os microsserviços. Ao optar pela orquestração com a API Gateway como ponto central, os microsserviços deixam de ser obrigados a comunicar diretamente entre si. Basicamente, os microsserviços podem concentrar-se nas suas funcionalidades específicas e na exposição das suas próprias APIs, sem a necessidade de conhecer ou gerir diretamente a interface dos outros microsserviços.

Deste modo, garantimos um controlo mais refinado sobre as transações e falhas entre os microsserviços, possibilitando uma gestão mais eficiente das operações e a capacidade de lidar adequadamente com situações de falha neste ambiente distribuído. Esta foi a principal razão pela qual escolhemos a orquestração em vez da coreografia.

Por fim, a nossa aplicação vai ser constituída pelo serviço de provas, serviço de notificações, serviço de salas e serviço de utilizadores, para além da *API Gateway* e da *ProbumUI*.

Tecnologias de Desenvolvimento

A definição deste tipo de arquitetura permitiu uma maior flexibilidade na escolha das tecnologias a utilizar para o desenvolvimento do projeto, uma vez que cada microsserviço pode ser desenvolvido tendo por base uma tecnologia diferente. Assim, a escolha das tecnologias a utilizar assentou nas preferências e

experiência de cada sub-grupo responsável pelo desenvolvimento de um dado microsserviço.

A linguagem de programação escolhida foi *JavaScript*, sendo esta executada no ambiente *Node.js*. O modelo de I/O não bloqueante do *Node.js* permite operações assíncronas eficientes, o que é particularmente valioso em microsserviços, onde a eficiência em termos de desempenho e tratamento simultâneo de vários pedidos é crucial. Ademais, existe uma vasta gama de módulos e bibliotecas disponíveis no *npm*, facilitando a integração de funcionalidades e acelerando o desenvolvimento de cada microsserviço.

O ecossistema *JavaScript* oferece muitas ferramentas modernas e *frameworks* que são específicos para o desenvolvimento de microsserviços. Neste sentido, optamos pela utilização da *framework Express.js* devido à sua simplicidade e abordagem minimalista.

Para a implementação da *frontend*, decidimos utilizar a biblioteca *React.js*. A abordagem desta biblioteca é focada em componentes, o que permite dividir a interface do utilizador em partes reutilizáveis, tornando mais simples tanto a manutenção como o desenvolvimento da aplicação. Outra vantagem de utilizar *React.js* é o uso do *Virtual DOM*. Em vez de ser necessário atualizar diretamente o *DOM*, o *React* compara o *Virtual DOM* com o *DOM*, aplicando apenas as alterações necessárias.

Relativamente à persistência de dados, os microsserviços utilizam bases de dados *NoSQL*, à exceção do microsserviço das provas. Para as bases de dados *NoSQL* iremos utilizar *MongoDB*, conhecido pelo seu desempenho positivo em operações de leitura e gravação, sendo rápido e responsivo. Para a persistência de dados do microsserviço das provas, utilizamos uma base de dados *SQL*. Assim sendo, iremos utilizar *MySql*, devido ao grande número de operações de *JOIN* e *queries* complexas.

Building Block View

A *building block view* é essencial para a documentação de qualquer arquitetura e apresenta a decomposição estática do sistema em blocos. Nesta secção apresentaremos esta decomposição, sendo esta dividida por diferentes níveis de abstração.

Diagrama de Componentes

O primeiro nível de abstração consiste no diagrama de componentes geral do sistema, que nos demonstra todos os componentes que farão parte da conceção do sistema. Desta forma, conseguirmos ter uma visão melhor sobre as responsabilidade de cada componente. A nossa aplicação é constituída por 6 componentes gerais: a interface com o utilizador (*ProbumUI*), a API orquestradora (*API Gateway*) e os 4 microsserviços definidos, relacionados com os utilizadores, provas, salas e notificações.

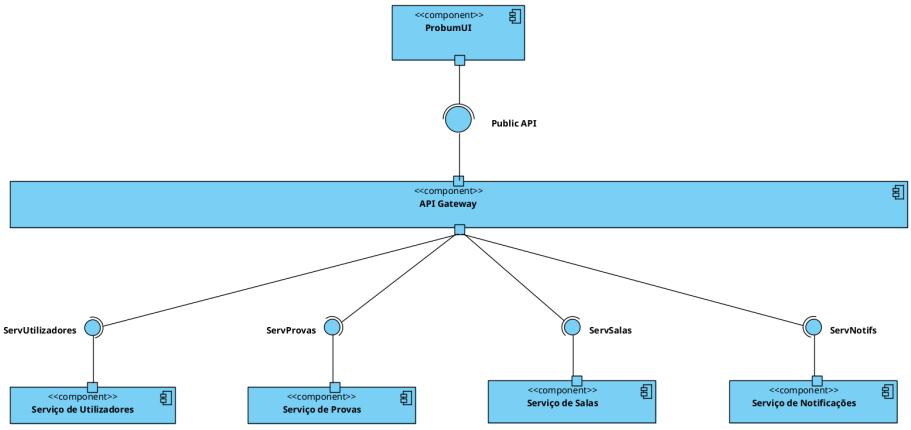


Figura 6: Diagrama de componentes da aplicação.

Componente	Responsabilidade
ProbumUI	Componente responsável por apresentar a interface com a qual os utilizadores interagem.
API Gateway	Componente que serve como porta de entrada para a lógica interna de uma aplicação, agindo como intermediário entre os clientes e os serviços de *backend*.
Serviço de Utilizadores	Componente responsável pela gestão dos utilizadores, incluindo registo, autenticação e gestão dos perfis.
Serviço de Provas	Componente responsável pela gestão das provas, incluindo a sua criação e edição.
Serviço de Salas	Componente responsável pela administração das salas, incluindo a sua criação e remoção.
Serviço de Notificações	Componente responsável pela gestão das notificações, incluindo a adição e remoção de notificações por utilizador.

Tabela 5: Resumo das responsabilidades dos componentes.

Especificação dos Componentes

O segundo nível de abstração foca-se em cada componente em particular. Para cada componente, apresentamos o seu diagrama de componentes interno, o seu modelo lógico e a definição das suas rotas, especificadas no Swagger.

ProbumUI

Este componente é o ponto de partida para qualquer interação no sistema. Globalmente, não acede diretamente a nenhum dos microsserviços, por isso, as suas ações podem não modificar os dados persistentes do sistema, tudo dependerá de como internamente cada serviço interpreta se estes pedidos podem ou não ter influência. Sempre que pretenda comunicar com qualquer serviço, este deverá

solicitar esse pedido à API que serve de intermedária entre este e os restantes componentes.

API Gateway

A *API Gateway* tem um papel crucial porque é através dela que passam todos os pedidos solicitados pelos utilizadores do sistema. Além disso, garante a autenticação destes utilizadores, retirando aos restantes componentes a tarefa de averiguar se, para cada utilizador e para cada pedido que este faz, é ou não legítimo.

A *API Gateway* é também responsável pela autorização de um pedido feito pelo *frontend*. Todos os *endpoints* que requerem o utilizador estar autenticado no sistema, vão utilizar um *middleware* de autorização que validará o *token* contido nos *headers* da *HTTP request*. Após validar o *token* recebido, cada *endpoint* irá conter informação acerca do utilizador e do seu tipo associado à ação em questão, validando se possuem a devida autorização para aceder *endpoint* em específico. Para além de validar o nível de autorização de cada utilizador, o mesmo também é responsável pela criação do *token* no momento em que o utilizador autentica-se no sistema.

Este componente tem também um papel de orquestrador. Ele é responsável por coordenar e gerir as interações entre os microsserviços. Portanto, deve ser capaz de contactar o serviço indicado para a realização de uma certa tarefa e deve estar pronto para realizar tarefas que envolvam vários serviços, devendo, nesses casos, esperar pela resposta dos serviços para poder avançar com o resto da tarefa.

Na seguinte figura, encontram-se as rotas definidas para a *API Gateway*.

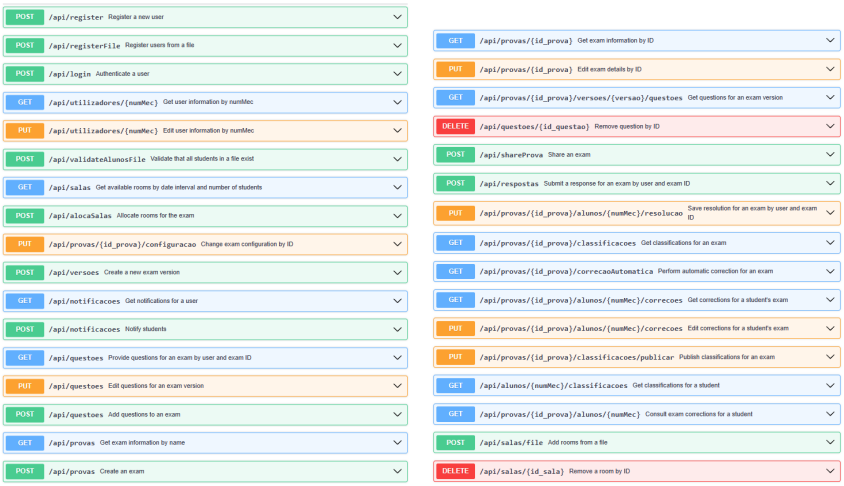


Figura 7: Rotas definidas para a *API Gateway*.

Serviço de Utilizadores

Este componente é responsável pela gestão de utilizadores. Por gestão entende-se, por exemplo, inserir e persistir dados para novos utilizadores, tratando do devido registo e autenticação de cada um, ou determinar os papéis de cada utilizador no sistema.

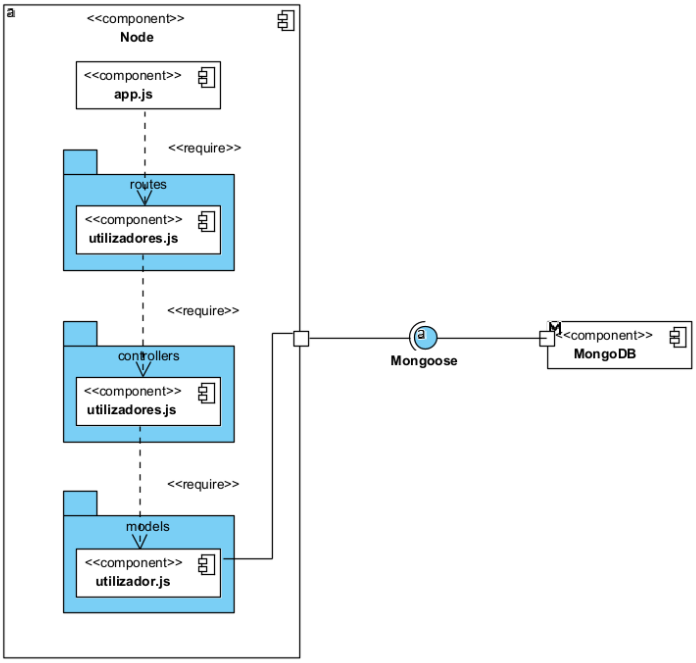


Figura 8: Diagrama de componentes do serviço de utilizadores.

Relativamente à persistência de dados, será necessária apenas uma coleção para representar os utilizadores. Em seguida, apresenta-se a estrutura de um documento desta coleção. Cada utilizador será identificado pelo seu `_id`, correspondente ao seu número mecanográfico, nome, email e tipo, podendo este ser um docente, aluno ou técnico.

```
{
    _id: Number,
    nome: String,
    email: String,
    type: Number
}
```

Na figura 9 encontram-se as rotas definidas para o serviço de utilizadores.

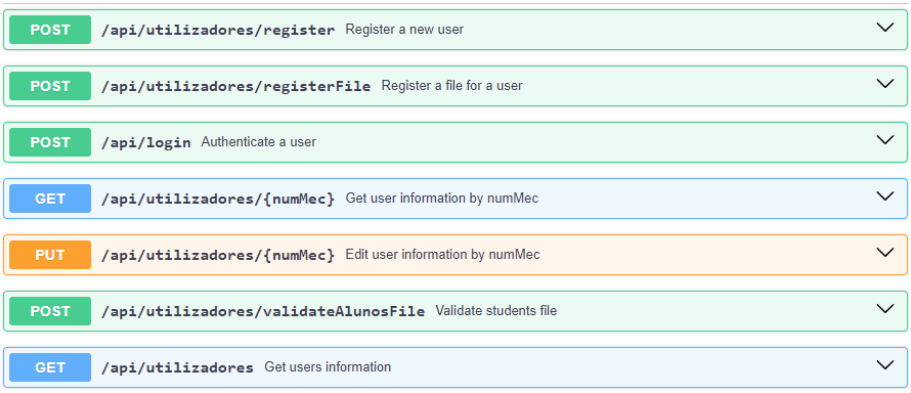


Figura 9: Rotas definidas para o serviço de utilizadores.

- **POST /api/utilizadores/register**: Regista um novo utilizador;
  O *body* do pedido deverá ter o seguinte formato:

```
{
    "nome": "José",
    "numMec": "a123",
    "email": "jose@gmail.com",
    "type": 1
}
```

- **POST /api/utilizadores/registerFile**: Regista novos utilizadores através de um ficheiro;
  No *body* do pedido pedido deverá ser passado o ficheiro e o tipo de utilizadores a registar.

- **POST /api/login**: Autentica o utilizador;
  O *body* do pedido deverá ter o seguinte formato:

```
{
    "numMec": "a123",
    "password": "teste"
}
```

- **GET /api/utilizadores/{numMec}**: Devolve a informação de um utilizador;

- **PUT /api/utilizadores/{numMec}**: Altera a informação de um utilizador;
  O *body* deverá conter as alterações efetuadas.

- **POST /api/utilizadores/validateAlunosFile**: Verifica se os alunos dos ficheiros são válidos;
  No *body* do pedido deverá ser passado o ficheiro.

- **GET /api/utilizadores**: Devolve a informação dos utilizadores;
  Neste caso, esta rota será geralmente utilizada para obter a informação de um dado grupo de utilizadores. Para isso, os números mecanográficos dos utilizadores devem ser passados em *query*.
  e.g.: `/api/utilizadores?numMec=a111&numMec=a222&numMec=a1333`

Serviço de Provas

Este componente é responsável pela gestão da informação que constitui as provas. Significa isto que, para cada prova, este é o componente que detém a informação sobre, por exemplo, os docentes com acesso à mesma, os alunos que terão acesso, as salas, horários associados, classificações, questões, respostas de cada aluno.

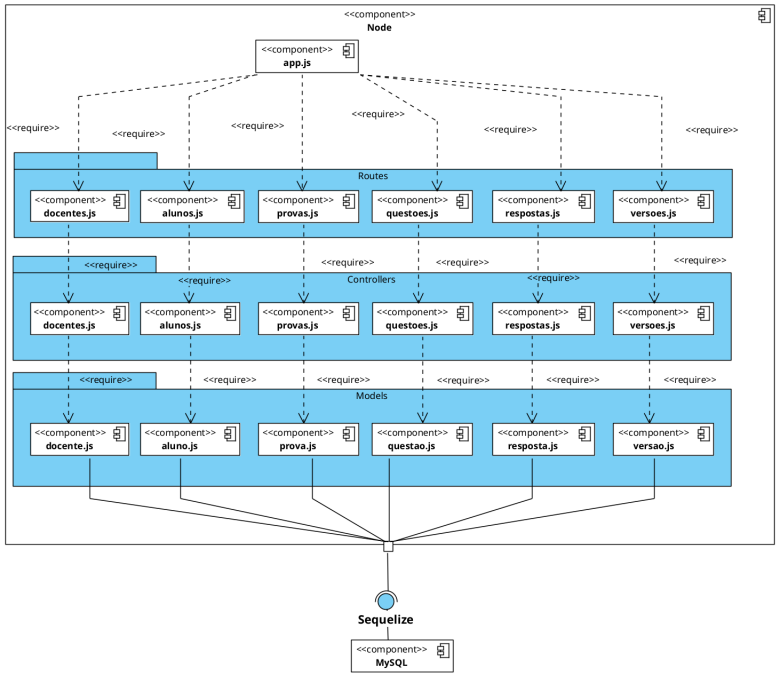


Figura 10: Diagrama de componentes do serviço de provas.

A base de dados do serviço de provas apresenta o seguinte modelo lógico:

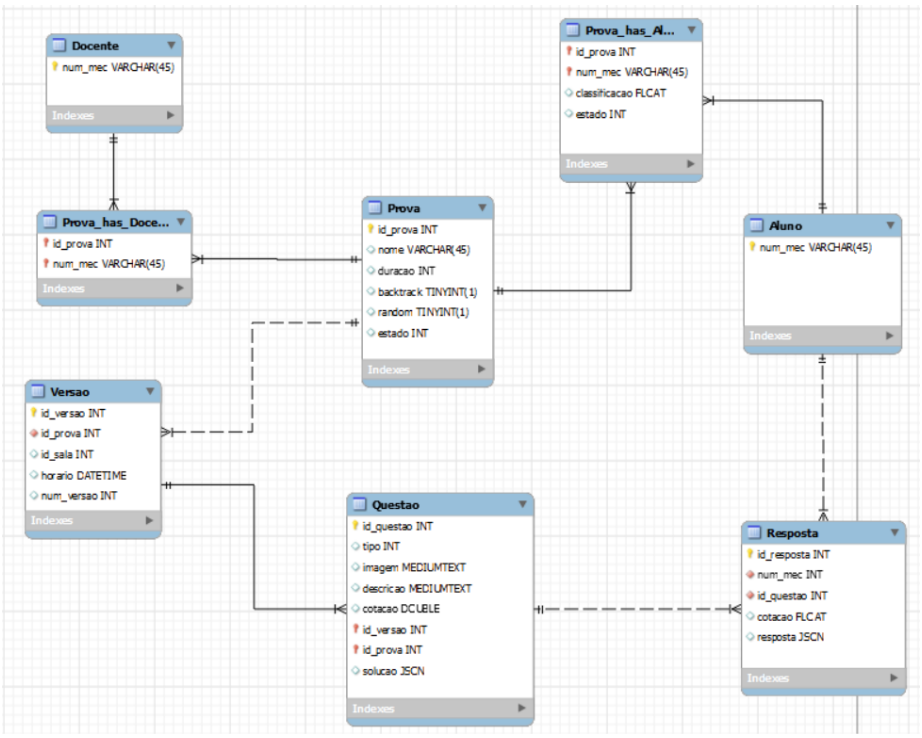


Figura 11: Modelo lógico da base de dados do serviço de provas.

O modelo lógico do serviço de provas é constituído pelas seguintes entidades base: Docente, Aluno, Prova, Versão, Questão e Resposta. Entre estas entidades destacamos os seguintes relacionamentos:

- Prova-Versão: Uma prova possui várias versões;
- Versão-Questão: Cada versão possui várias questões;
- Questão-Resposta: Cada questão pode ter várias respostas associadas.

Em seguida, apresentam-se as rotas definidas para o serviço de provas.

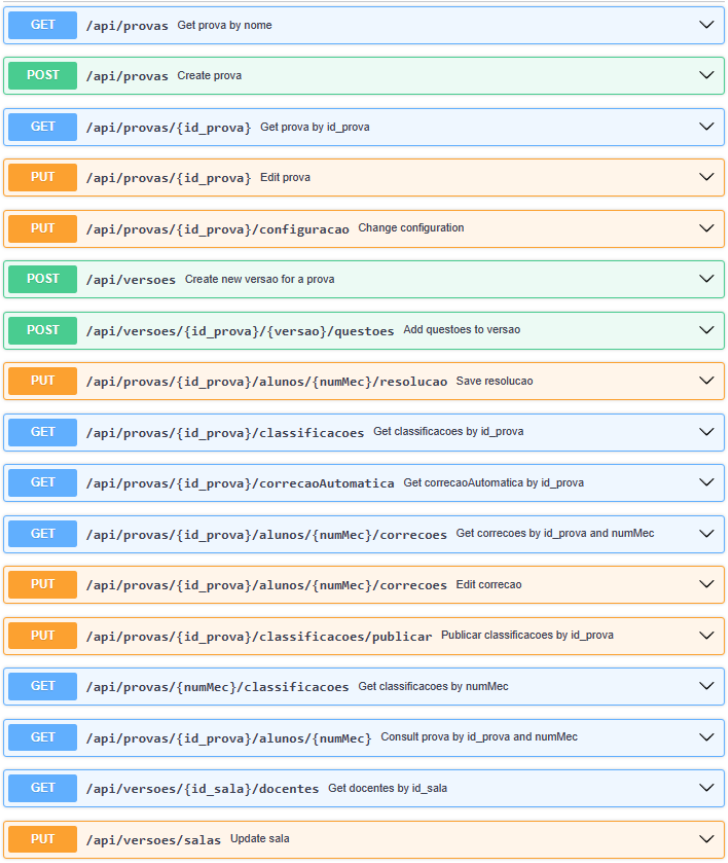


Figura 12: Rotas definidas para o serviço de provas.

- **GET /api/provas**: Devolve a prova com o nome introduzido;
  No pedido deverá ser passado o nome da prova em *query*.
  e.g.: `/api/provas/prova1_ras`

- **POST /api/provas**: Cria e devolve uma prova;
  O *body* do pedido deverá ter o seguinte formato:

```
[
    {
        "nome": "Prova1",
        "id_alunos": ["pg40350", "pg53678", "pg52543"],
        "id_salas": ["0.11","0.12"]
    }
]
```

- **GET /api/provas/{id_prova}**: Devolve a prova com o ID introduzido;
  No pedido deverá ser passado o ID da prova em *query*.
  e.g.: `/api/provas/1`

- **PUT /api/provas/{id_prova}**: Altera uma prova existente devolvendo essa prova;
  No pedido deverá ser passado o ID da prova em *query* e o *body* deverá conter a prova com as alterações feitas.
  e.g.: `/api/provas/1`

- **PUT /api/provas/{id_prova}/configuracao**: Altera a configuração da prova existente devolvendo essa prova;
  No pedido deverá ser passado o ID da prova em *query*.
  e.g.: /api/provas/1/configuracao
  O *body* deverá ter o seguinte formato:

```
[
    {
        "versao": 2,
        "randQ": 0,
        "backtrack": 1
    }
]
```

- **POST /api/versoes**: Cria uma nova versão para uma prova;
  O *body* do pedido deverá ter o seguinte formato:

```
[
    {
        "id_prova": 10,
        "versao": 2,
        "id_sala": "0.11",
        "horaInicio": 2024-1-20 15:30:00,
        "horaFim": 2024-1-20 17:00:00,
    }
]
```

- **POST /api/versoes/{id_prova}/{versao}/questoes**: Adiciona questões a uma prova e devolve um STATUS_CODE;
  No pedido deverá ser passado o ID e a versão da prova em *query* e *body* deverá a lista de questões a serem inseridas.
  e.g.: `/api/provas/1/2/questoes`

- **PUT /api/provas/{id_prova}/alunos/{numMec}/resolucao**: Altera o estado do aluno na prova e devolve um STATUS_CODE;
  No pedido deverá ser passado o ID da prova e o número mecanográfico do aluno em *query* e no *body* deverá ser passada a resolução em formato JSON.
  e.g.: `/api/provas/1/pg52345/resolucao`

- **GET /api/provas/{id_prova}/classificacoes**: Devolve as classificações dos alunos numa dada prova;
  No pedido deverá ser passado o ID da prova em *query*.
  e.g.: `/api/provas/1/classificacoes`

- **GET /api/provas/{id_prova}/correcaoAutomatica**: Corrige as questões da prova, automaticamente, sempre que for possível e devolve as classificações atualizadas;
  No pedido deverá ser passado o ID da prova em *query*.
  e.g.: `/api/provas/1/correcaoAutomatica`

- **GET /api/provas/{id_prova}/alunos/{numMec}/correcao**: Devolve a correção da prova realizada pelo aluno;
  No pedido deverá ser passado o ID da prova e o número mecanográfico do aluno em *query*.
  e.g.: `/api/provas/1/alunos/pg52345/correcoes`

- **PUT /api/provas/{id_prova}/alunos/{numMec}/correcao**: Altera a correção da prova realizada pelo aluno;
  No pedido deverá ser passado o ID da prova e o número mecanográfico do aluno em *query* e no *body* deverá ser enviada a correção, ou seja, uma lista de pares id_resposta com a cotação associada.
  e.g.: `/api/provas/1/alunos/pg52345/correcoes`

- **PUT /api/provas/{id_prova}/classificacoes/publicar**: Altera o estado da prova para que os alunos possam visualizar as suas classificações e devolve um STATUS_CODE;
  No pedido deverá ser passado o ID da prova em *query*.
  e.g.: `/api/provas/1/classificacoes/publicar`

- **GET /api/provas/{numMec}/classificacoes**: Devolve uma lista de classificações realizadas pelo aluno;
  No pedido deverá ser passado o número mecanográfico do aluno em *query*.
  e.g.: `/api/provas/pg52345/classificacoes`

- **GET /api/provas/{id_prova}/alunos/{numMec}**: Devolve a prova com as questões e respostas dadas pelo Aluno;
  No pedido deverá ser passado o ID da prova e o número mecanográfico do aluno em *query*.
  e.g.: `/api/provas/1/alunos/pg52345`

- **GET /api/versoes/{id_sala}/docentes**: Devolve os docentes associados às provas associadas à sala;
  No pedido deverá ser passado o ID da sala em *query*.
  e.g.: `/api/versoes/0.11/docentes`

- **PUT /api/versoes/salas**: Remove a sala associada à prova atribuindo uma nova e devolve um STATUS_CODE;
  O *body* do pedido deverá ter o seguinte formato:

```
[
    {
        "id_sala": "0.11",
        "new_id_sala": "0.13"
    }
]
```

Padrão *Observer*

Neste microsserviço é possível identificar o padrão *observer*. Neste caso, o *subject* é a prova e os *observers* são os alunos que estão inscritos na mesma, ou seja, a prova tem uma noção de estado (prova criada, prova disponível e prova corrigida) e quando ocorre uma mudança de estado na prova, cada um desses alunos é notificado, através do envio de um pedido ao microsserviço das notificações (*update*).

Serviço de Salas

Este componente é responsável pela gestão das infraestruturas físicas associadas ao Sistema. Significa isto que, é ele o responsável pelo escalonamento de provas tendo em conta as suas capacidades e limitações.

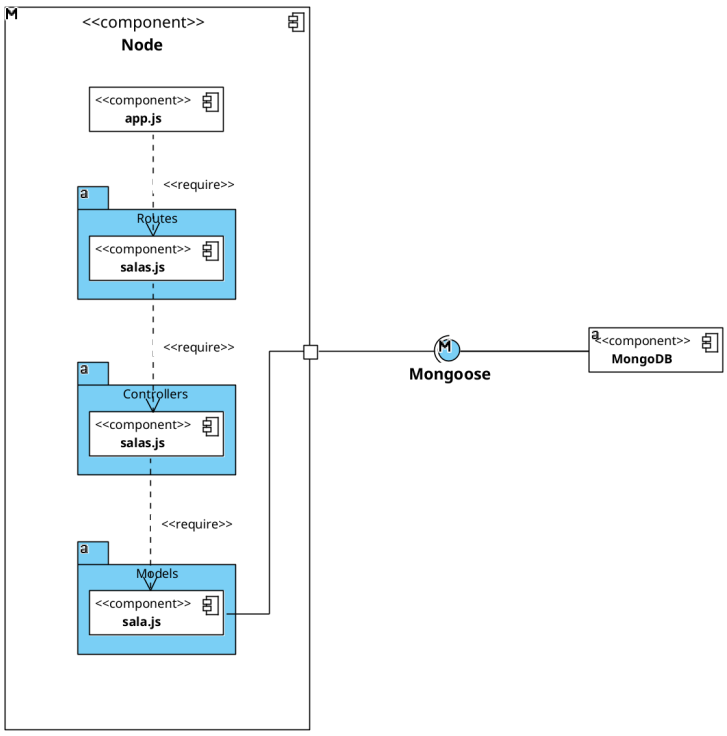


Figura 13: Diagrama de componentes do serviço de salas.

A base de dados do serviço de salas apresenta o seguinte *schema*:

```
{
    _id: String,
    building: String,
    floor: Number,
    room: String,
    capacity: Number,
    allocations: [{
        startDate: Date,
        endDate: Date,
    }]
}
```

Através do *schema* da base de dados podemos perceber os atributos de cada sala. Uma sala terá a informaçao sobre o seu *building*, *floor*, *room*, *capacity* e irá armazenar uma lista de alocações onde irá guardar a *startDate* e *endDate* da prova ao qual está alocada.

As rotas definidas para o serviço de salas são as seguintes:

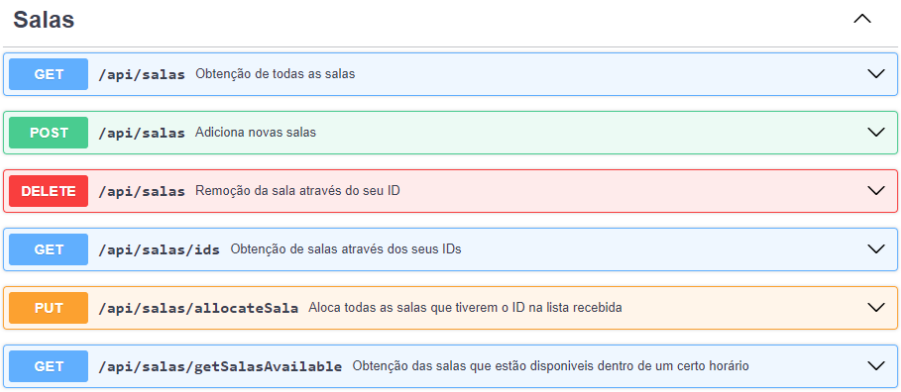


Figura 14: Rotas definidas para o serviço de salas.

- **GET /api/salas**: Devolve todas as salas no sistema;

- **POST /api/salas**: Introduz novas salas no sistema;
  O *body* do pedido deverá ter formato:

```
[
    {
        "building": "A",
        "floor": 1,
        "room": "A1",
        "capacity": 50
    }
]
```

- **DELETE /api/salas**: Remove uma sala do sistema;
  No pedido deverá ir o ID da sala as ser removida em *query*.
  e.g.: `/api/salas?id=1`

- **GET /api/salas/ids**: Devolve as salas que tiverem o ID na lista dos IDs;
  No pedido deverá ir os IDs das salas que deseja receber em *query*.
  e.g.: `/api/salas/ids?id=1&id=2&id=3`

- **PUT /api/salas/allocateSala**: Reserva salas nos respetivos *timeslots*; O *body* do pedido deverá ter formato:

```
[
    {
        "ids": 1,
        "startTime": "2024-1-20 15:30:00",
        "endTime": "2024-1-20 17:00:00"
    }
]
```

- **GET /api/salas/getSalasAvailable**: Obtenção das salas disponíveis dentro de um certo horário; No pedido deverá ir a data de inicio e fim do teste em *query*.
  e.g.: `/api/salas/getSalasAvailable?startTime=2024-1-20 15:30:00&endTime=2024-1-20 17:00:00`

Serviço de Notificações

Este é o componente responsável por fazer notar as informações pertinentes aos utilizadores relativas a si, quer seja de respostas a pedidos efetuados pelos mesmos na comunicação interna ao sistema quer indiretamente por qualquer fluxo que ocorra interno ao sistema, entre componentes, e que se relacione com os utilizadores afetados por estes fluxos.

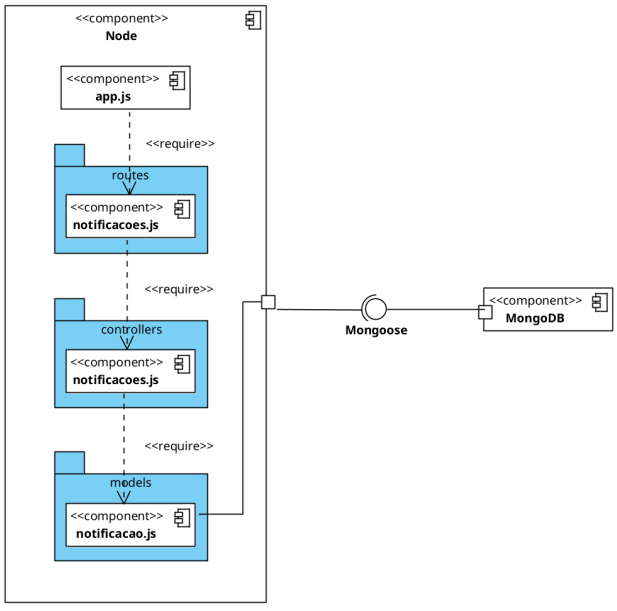


Figura 15: Diagrama de componentes do serviço de notificações.

Relativamente à base de dados NoSQL do serviço de notificações, cada notificação terá a informação sobre o seu *content*, *creationDate* e irá armazenar uma lista de destinatários que, por sua vez, irá guardar o número mecanográfico e o *readStatus* da notificação para cada utilizador. A estrutura de cada documento é a seguinte:

```
{
    receivers: [{
        mec: String,
        read: Boolean
    }],
    creationDate: Date,
    content: String
}
```

As rotas definidas para o serviço de notificações são as seguintes:

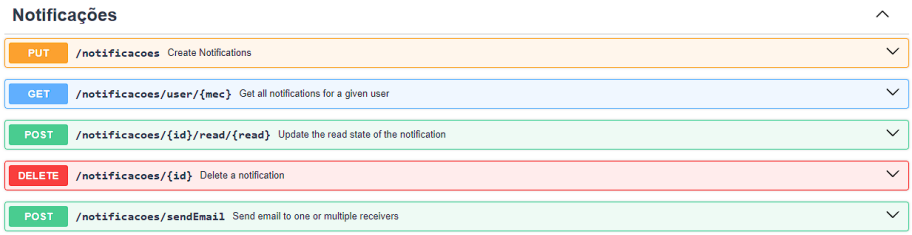


Figura 16: Rotas definidas para o serviço de notificações.

- **PUT /api/notificacoes**: Cria uma notificação
  O *body* do pedido deverá ter o seguinte formato:

  ```
  {
      "mecList": ["PG12345", "A987654"],
      "message": "ProbUM: Há uma nova prova disponível"
  }
  ```

- **GET /api/notificacoes/user/{mec}**: Devolve todas as notificações de um utilizador;
  O número mecanográfico do utilizador deve estar em *query*.
  e.g.: `/api/notificacoes/user/PG98765`

- **POST /api/notificacoes/{id}/read/{read}**: Marca uma notificação como lida/não-lida;
  No pedido deverá ir o ID da notificação a ser removida e o estado.
  e.g.: `/api/notificacoes/0fb48f5e-01ea-482e-b917-dbd61da0b05f/read/true`

- **DELETE /api/notificacoes/{id}**: Remove uma notificação do sistema;
  No pedido deverá ir o ID da notificação a ser removida.
  e.g.: `/api/notificacoes/0fb48f5e-01ea-482e-b917-dbd61da0b05f`

- **POST /api/notificacoes/sendEmail**: Envia emails;
  O *body* do pedido deverá ter o seguinte formato:

```
{
    "receiverList": [
        {
            "name": "José",
            "email": "jose@email.com"
        },
        {
            "name": "João",
            "email": "joao@email.com"
        },
    ],
    "email": {
        "subject": "ProbUM: Há uma nova prova disponível"
        "body": "<html><body>Uma nova prova está disponível.</body></html>"
    }
}
```

Runtime View

A *Runtime View* é essencial para analisar a interação da aplicação. Nesta secção, o objetivo é proporcionar uma compreensão de como os componentes do sistema realizam as suas funções, sendo uma ferramenta valiosa aos menos familiarizados para compreenderem o funcionamento da aplicação.

Máquina de Estados

A nível de *frontend*, elaboramos um diagrama de máquina de estados para representar as possíveis transições entre as páginas do nosso sistema. A adoção de uma máquina de estados oferece uma perspetiva visual e estruturada das funcionalidades e cenários inerentes do sistema. Ao mapear cada decisão tomada na aplicação por parte do utilizador, a máquina de estados emerge como uma ferramenta indispensável para compreender a dinâmica operacional do ProbUM de maneira abrangente e intuitiva. O diagrama facilita, assim, a compreensão do fluxo de estados e transições.

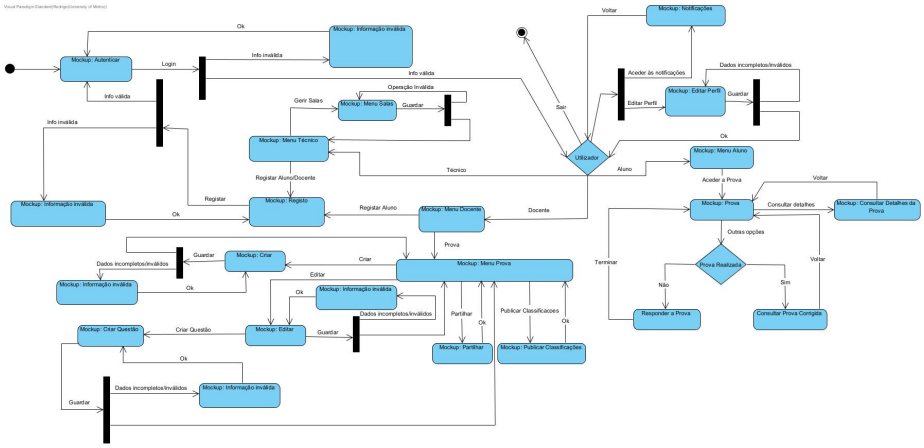


Figura 17: Transições entre as várias páginas da aplicação.

Diagramas de Sequência

Para consolidar os passos seguintes à modelação e especificação do ProbUM, foram criados diagramas de sequência que permitem uma melhor perceção do funcionamento das ações principais e das trocas de mensagens necessárias para a sua execução. Em seguida, apresentaremos alguns dos diagramas de sequência relativos às ações que consideramos mais críticas sobre a aplicação. Os restantes diagramas elaborados encontram-se em anexo.

Cenário 1: Registar Docentes

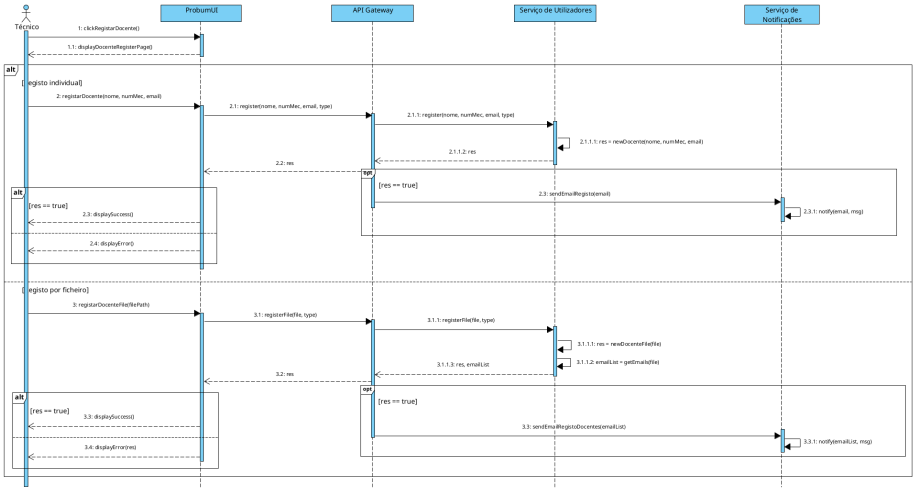


Figura 18: Diagrama de sequência do UC “Registar Docentes“.

O diagrama da figura 18 representa os dois principais sub-cenários do registo de docentes: individual e múltiplo através de ficheiro. Se não ocorrer nenhum erro no registo dos docentes, o serviço de notificações será invocado pela *API Gateway* para enviar email ao(s) novo(s) docente(s) criados, notificando-os da criação da sua conta com as respetivas informações necessárias para acederem à sua nova conta.

Cenário 5: Criar Prova

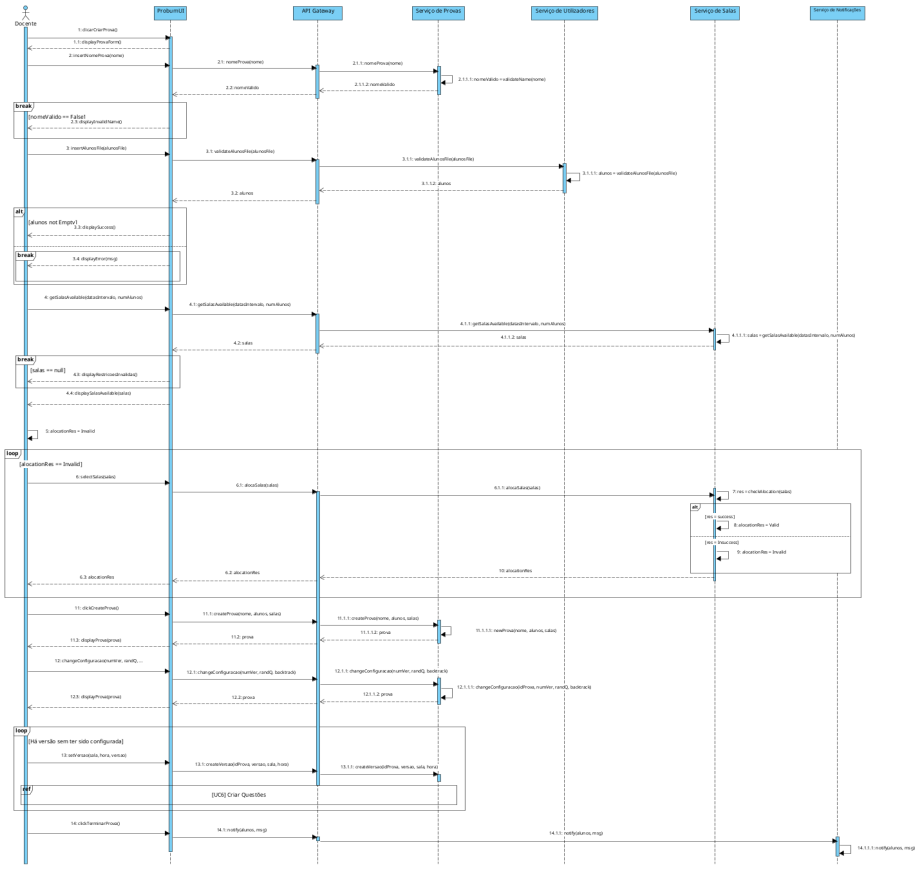


Figura 19: Diagrama de sequência do UC "Criar Prova".

A criação da prova é um caso de uso crítico ao nosso sistema, sendo uma prova o conceito principal sobre o qual o sistema está centrado. Todos os serviços existentes estão envolvidos neste *use case*, o que demonstra a sua importância e complexidade. Para criação de uma prova é necessário um nome, um ficheiro com todos os alunos a ser incluídos, a atribuição de uma sala, alguns parâmetros importantes como a hora e versões, e por fim adicionar as questões. Por fim,

todos os alunos associados à prova são informados da criação da prova, através de uma notificação.

Cenário 11: Responder a Prova

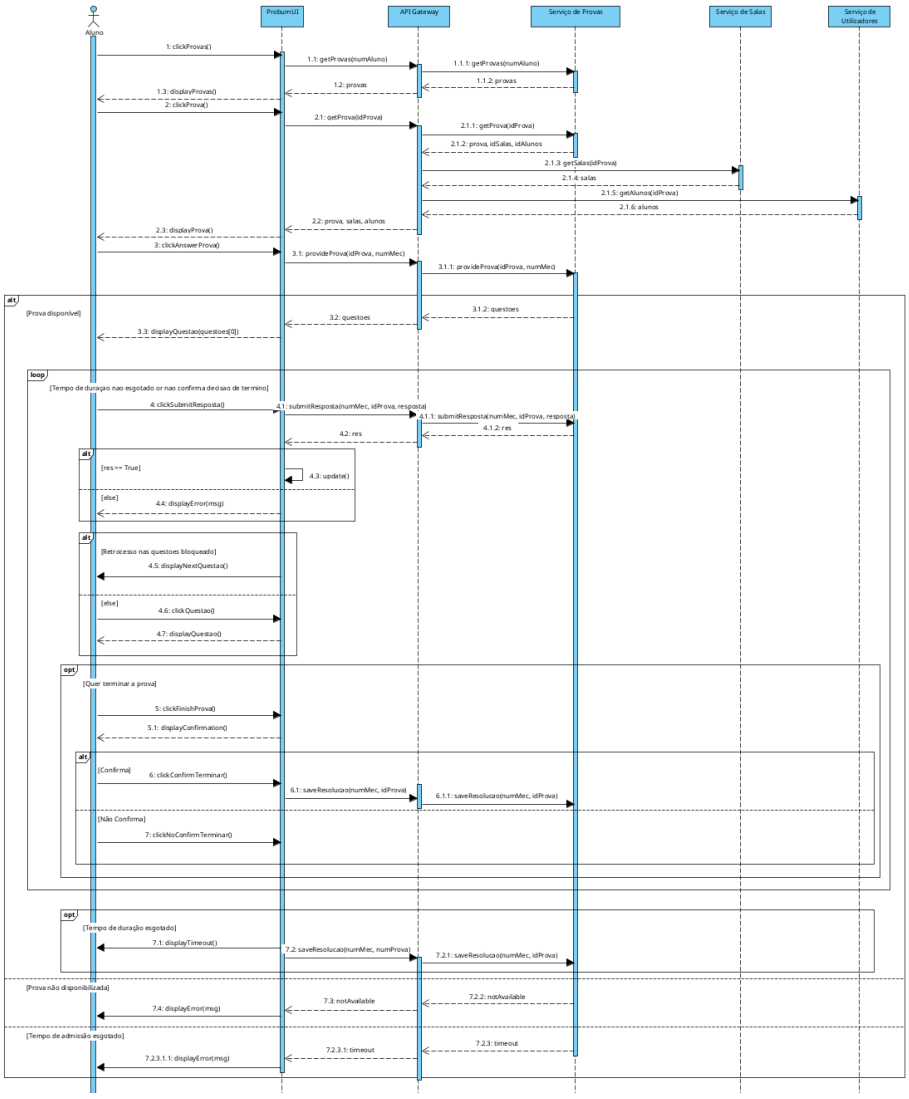


Figura 20: Diagrama de sequência do UC "Responder a Prova".

No que toca ao *use case* "Responder a Prova", os intervenientes são o serviço de provas, o de salas e o de utilizadores. Inicialmente, a prova é escolhida pelo aluno e dá-se início ao processo de resposta às questões. Existem duas

maneiras de o aluno terminar a prova - por escolha própria ou por esgotamento do tempo estipulado. Enquanto a mesma decorre normalmente, o aluno submete a resposta a uma pergunta e o sistema regista-a. Dependendo de como a prova foi configurada, o aluno pode retroceder as respostas ou apenas avançar para a seguinte.

Face à primeira iteração, este diagrama de sequência sofreu algumas alterações, não só na introdução da *API Gateway*, como também na junção dos serviços de questões e respostas no serviço de provas. Esta alteração permitiu reduzir algumas interações que existiam entre serviços, diminuindo o potencial **overhead** da rede previamente imposto pela separação destes microsserviços.

Cenário 12: Classificar Respostas

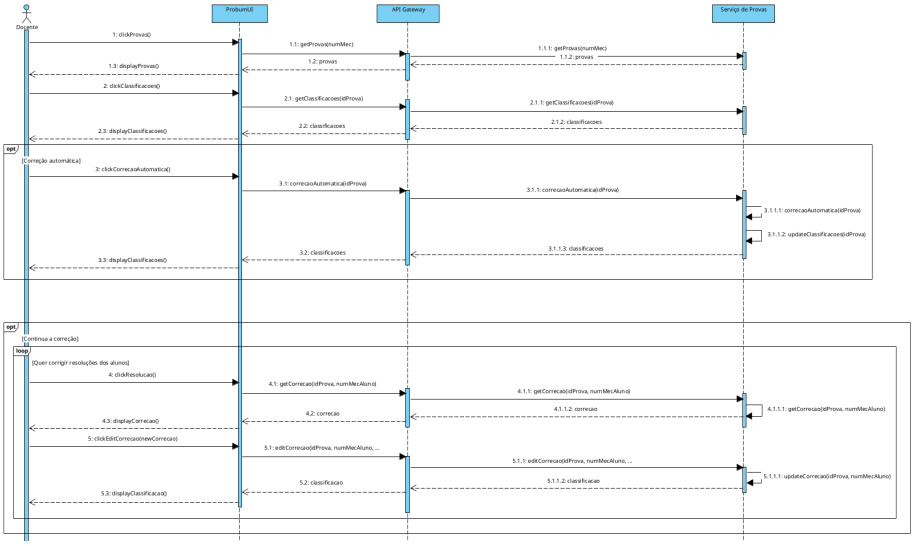


Figura 21: Diagrama de sequência do UC “Classificar Respostas“.

Para a classificação de respostas dispomos de 2 cenários: a correção automática, em que o serviço de provas fica encarregue de atribuir as classificações às respostas dadas pelos alunos e a correção manual, que permite ao docente corrigir manualmente a classificação de cada aluno. Tal como no *use case* “Responder a Prova“, a junção dos microsserviços permitiu colmatar o potencial *overhead* de rede que existiria caso os serviços de provas, questões e respostas fossem separados.

Cenário 15: Gerir Salas

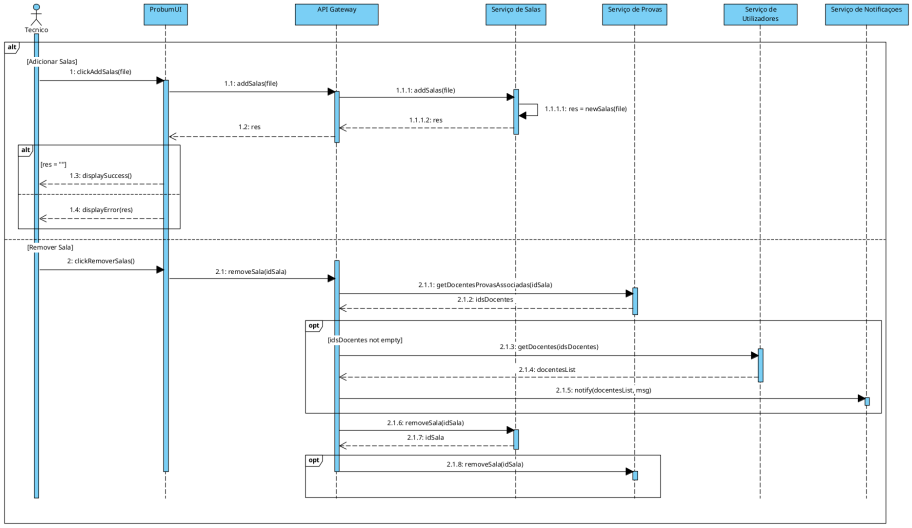


Figura 22: Diagrama de sequência do UC "Gerir Salas".

O diagrama que diz respeito ao *use case* "Gerir Salas" representa os 2 principais sub-cenários de Gerir Salas: adição de salas no sistema e remoção de uma sala do sistema.

No caso de adicionar salas, o ator envia um ficheiro com a informação das salas que deseja adicionar ao sistema. Essa adição decorrerá no serviço de salas, onde a informação contida no ficheiro enviado será validada e, se estiver de acordo com o esperado, o serviço cria as salas.

No caso de remover uma sala, o sistema precisa, primeiramente, de verificar se existem docentes associados a essas salas, fazendo um pedido ao serviço de provas. Após obter os docentes associados à sala, a API irá fazer um pedido ao serviço de utilizadores para obter a informação dos docentes, necessária para notificar os docentes sobre a remoção da sala. Depois, um pedido é enviado ao serviço de salas e provas para que a sala seja removida do sistema e para que as provas com essa sala alocada a removam da alocação.

Deployment View

A secção de *Deployment View* desempenha um papel crucial na especificação do sistema de *software*, proporcionando uma visão abrangente da implementação física da aplicação.

Este segmento da documentação oferece uma representação visual através de um diagrama de *deployment* UML, que ilustra as interações entre os diversos componentes do sistema, incluindo *hardware* e *software*, dando informação importante para equipas de DevOps. Na figura 23 expõe-se o diagrama de *deployment* concebido.

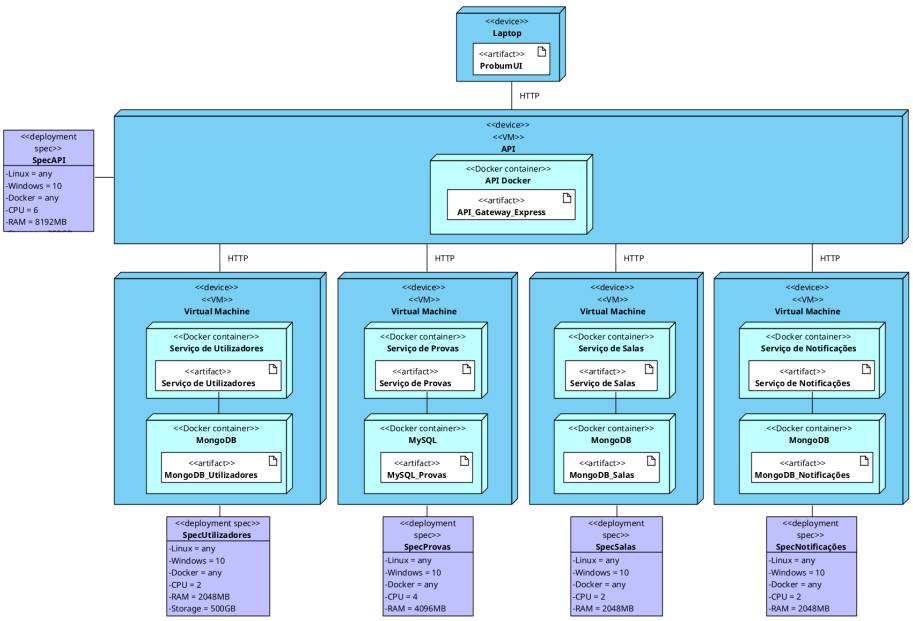


Figura 23: Diagrama de *deployment*.

A nossa aplicação segue o padrão de microsserviços e, desta forma, para aproveitar a divisão entre os serviços, providenciada pelo padrão, decidimos ter 1 máquina virtual para cada serviço e para a *API Gateway*. Na máquina virtual da *API Gateway*, estará a correr um *docker container* com a aplicação desenvolvida para servir os pedidos dos clientes. Já nas máquinas virtuais dos serviços, estarão a correr *containers* com a lógica do serviço e um outro *container* com a base de dados para esse serviço. A comunicação entre a API e os serviços e os clientes e a API é feita através do protocolo HTTP.

Anexos

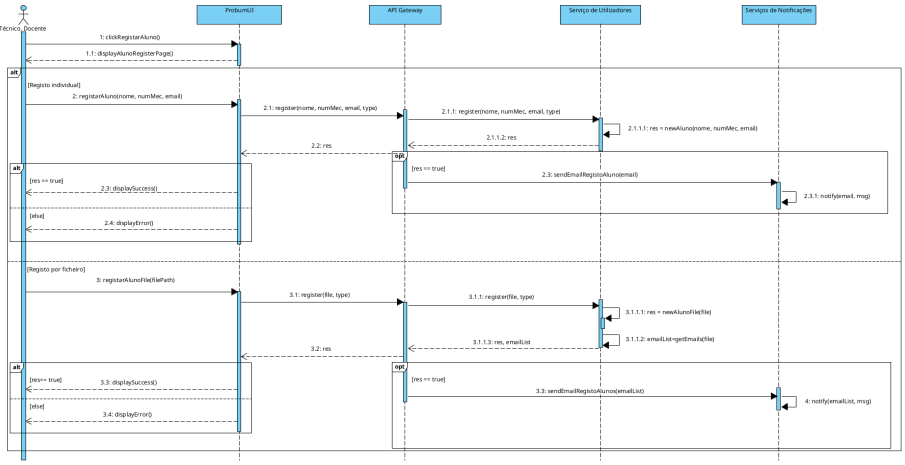


Figura 24: Diagrama de sequência do UC “Registar Alunos“.

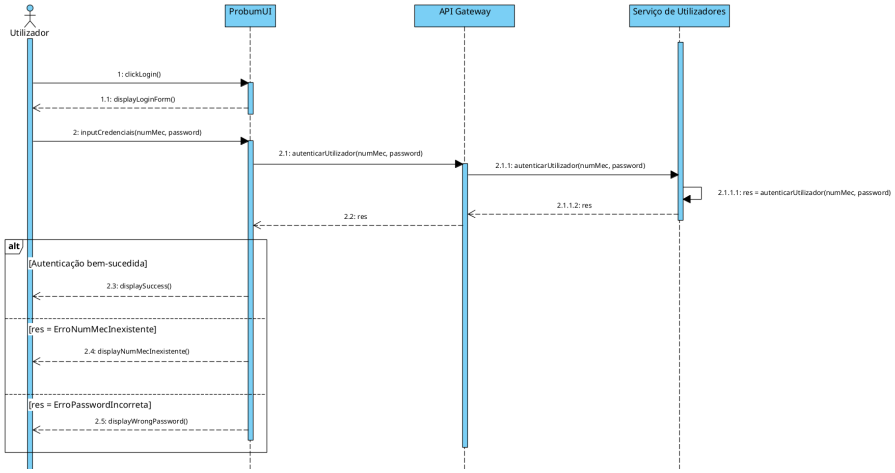


Figura 25: Diagrama de sequência do UC “Autenticar“.

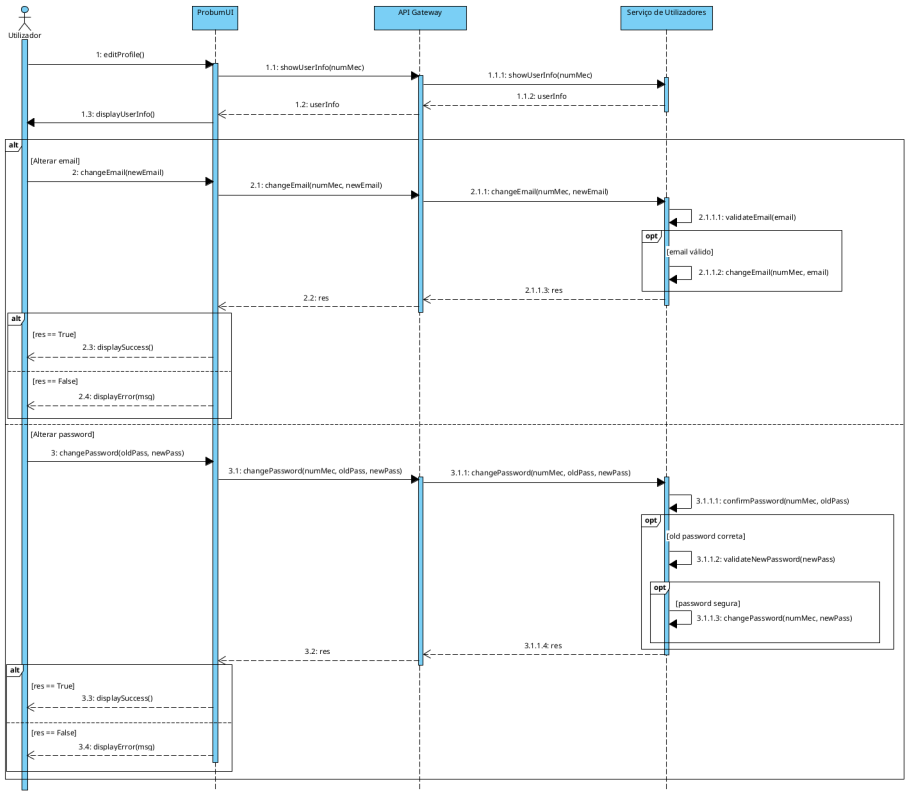


Figura 26: Diagrama de sequência do UC “Editar Perfil“.

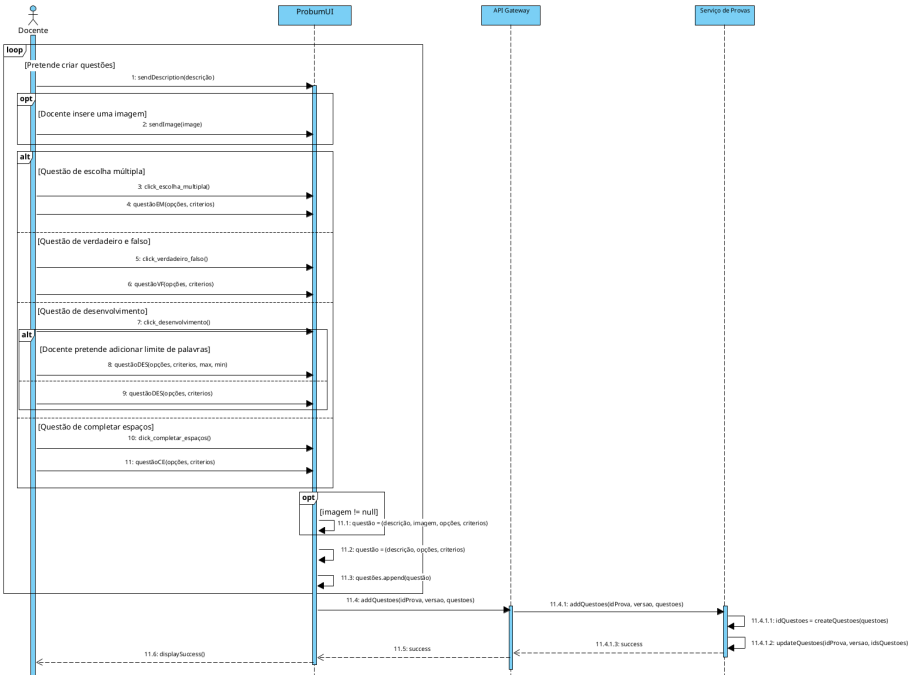


Figura 27: Diagrama de sequência do UC “Criar Questões“.

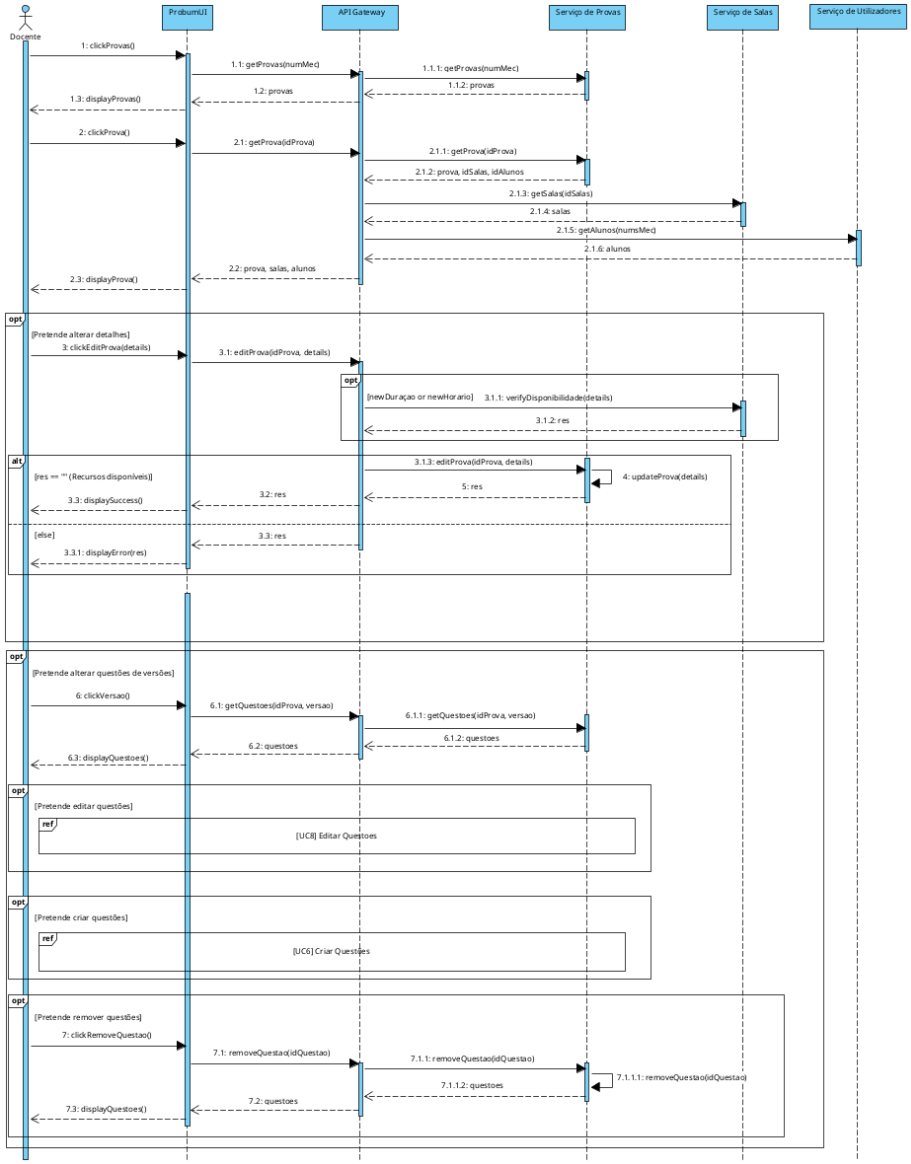


Figura 28: Diagrama de sequência do UC “Editar Prova“.

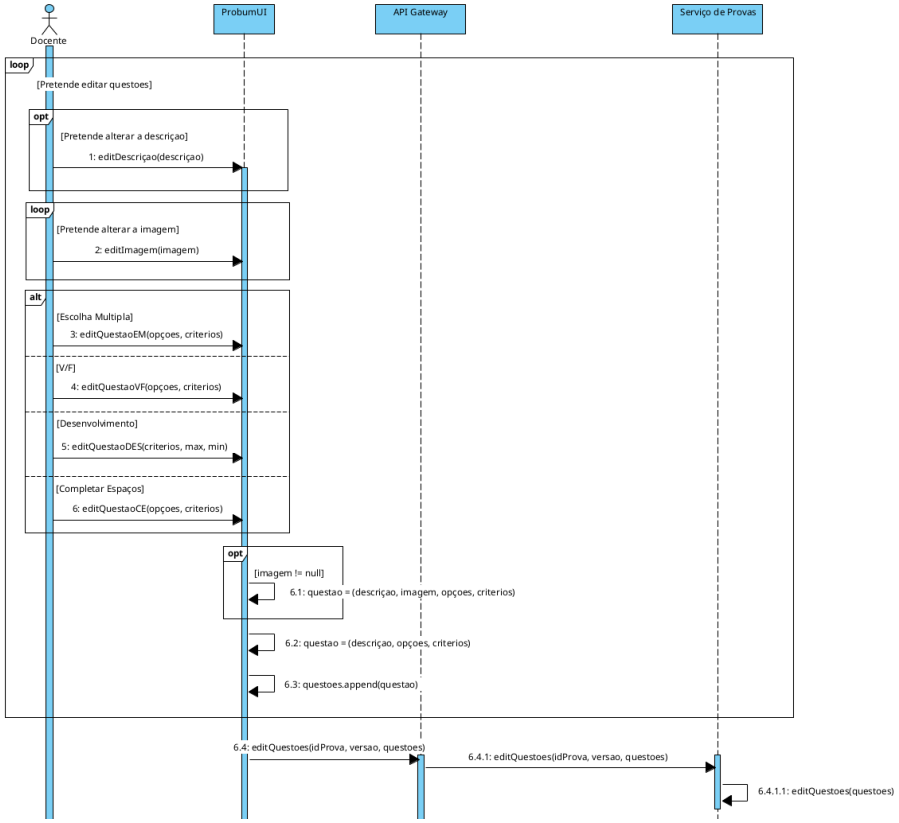


Figura 29: Diagrama de sequência do UC “Editar Questões“.

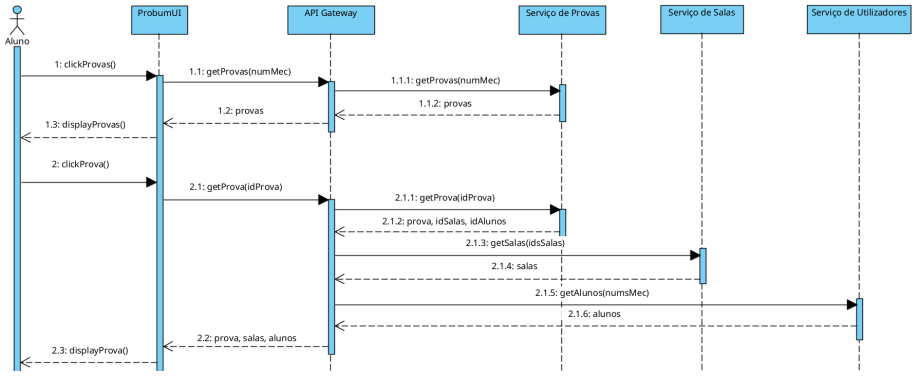


Figura 30: Diagrama de sequência do UC “Consultar detalhes da Prova“.

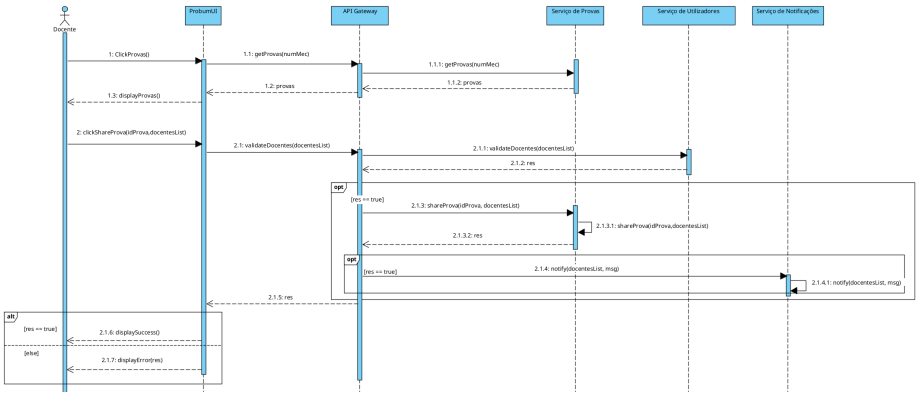


Figura 31: Diagrama de sequência do UC “Partilhar Prova“.

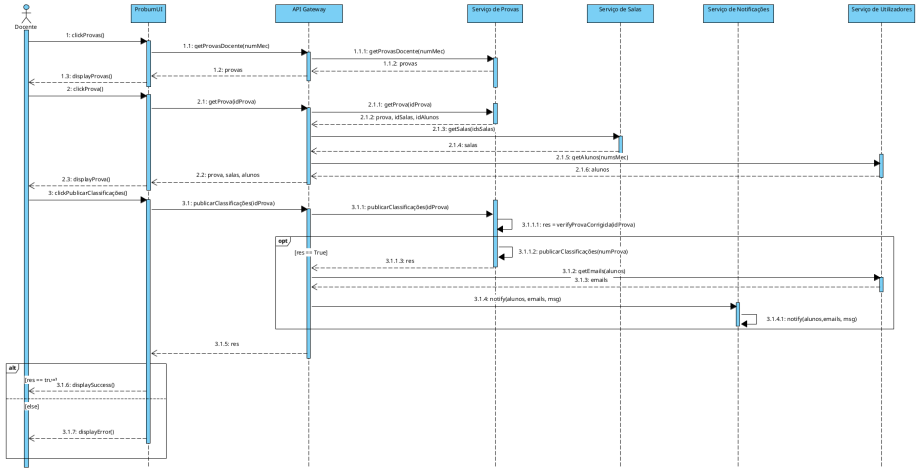


Figura 32: Diagrama de sequência do UC “Publicar Classificações da Prova“.

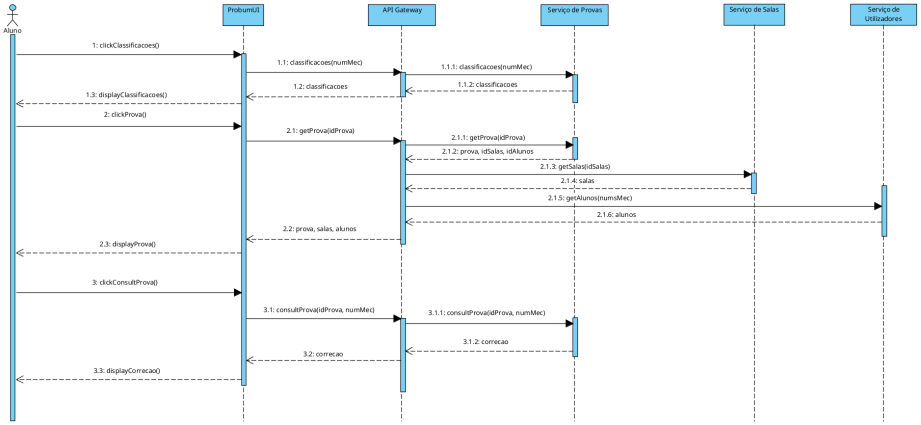


Figura 33: Diagrama de sequência do UC "Consultar Prova Corrigida".

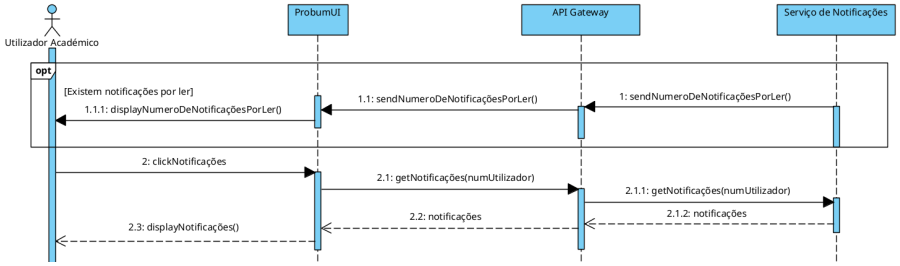


Figura 34: Diagrama de sequência do UC "Aceder às Notificações".