



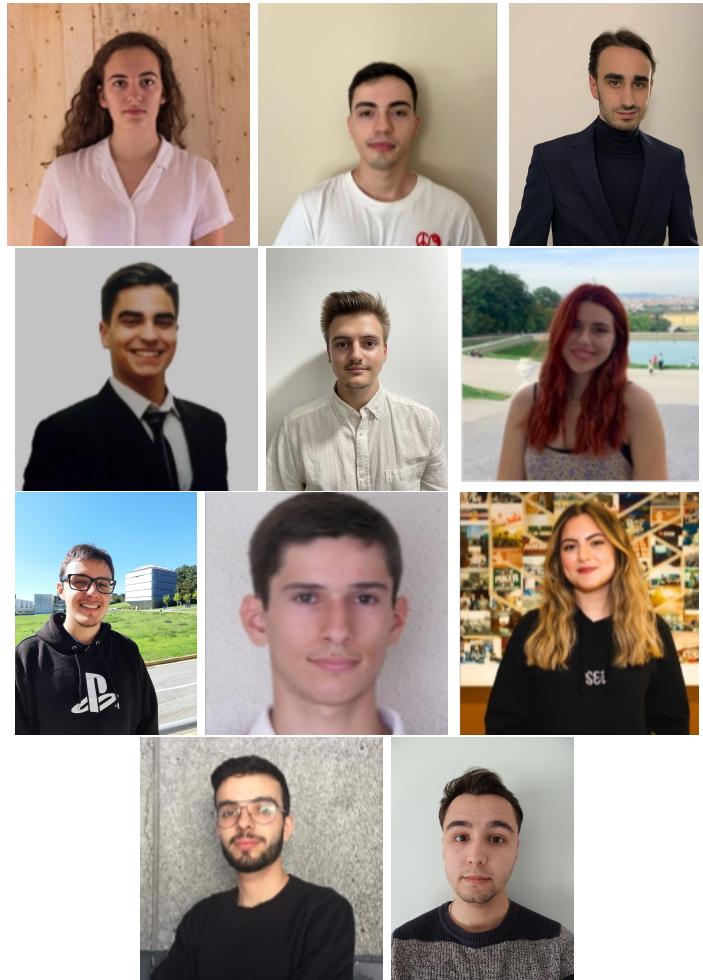
Mestrado em Engenharia Informática

Requisitos e Arquiteturas de Software (2023/24)

PROBUM

Grupo 2-B / Entrega 2

Marta Isabel da Silva e Sá - PG54084
Nuno Guilherme Cruz Varela - PG54117
Bernardo Amado Pereira da Costa - PG53699
Afonso Xavier Cardoso Marques - PG53601
André Castro Alves - PG53651
Inês Daniela Alves Ferreira - PG53870
Tomás Cardoso Francisco - PG54263
Miguel de Sousa Braga - PG54095
Millena de Freitas Santos - PG54107
Pedro Rafael Bernardo Medeiros - A80580
Renato André Machado Gomes - PG54174



Braga, 1 de dezembro de 2023

Contents

Prefácio	6
Introdução e Objetivos	7
Visão geral dos requisitos	7
Stakeholder	9
Restrições	10
Restrições à Solução	10
Restrições Temporais	11
Restrições Orçamentais	11
Requisitos de qualidade	12
Requisito 14	12
Requisito 9	12
Requisito 13	12
Requisito Novo	13
Âmbito e Contexto do Sistema	13
Contexto do negócio	13
Estratégia para a solução	14
Decisões arquiteturais	14
Decisões relativas à arquitetura de microsserviços	15
Microsserviços e API Gateway	17
FrontEnd	18
Building Block View	18
Visão geral do sistema	18
API Gateway	19
Microsserviços	20
Microsserviço das provas	20
Microsserviço dos utilizadores	26
Microsserviço das notificações	30
Microsserviço das salas	32
Runtime View	35
Diagramas de Sequência	35
Registar Alunos	35
Criar Prova	36
Responder à Prova	38
Atualização do estado de uma prova	39
Deployment View	40

Vista “Frontend” **41**

Anexos **47**

List of Figures

1	Diagrama de <i>Use Cases</i>	7
2	Diagrama de contexto	13
3	Vantagens e desvantagens das arquiteturas.	14
4	Decomposição em blocos funcionais	16
5	Diagrama de componentes	18
6	Pedidos suportados pela API Gateway.	20
7	Diagrama de componentes do microsserviço das provas	21
8	Funções a serem disponibilizadas pela TestsAPI	23
9	Representação das rotas disponíveis para o microsserviço de provas utilizando a ferramenta <i>Swagger</i> - 1	24
10	Representação das rotas disponíveis para o microsserviço de provas utilizando a ferramenta <i>Swagger</i> - 2	24
11	Diagrama de componentes para o microsserviço de utilizadores.	27
12	API para o microsserviço de utilizadores	28
13	Representação das rotas disponíveis para o microsserviço dos utilizadores utilizando a ferramenta <i>Swagger</i>	28
14	Diagrama de componentes para o microsserviço de utilizadores.	30
15	Funções a serem disponibilizadas pela Notifications API	30
16	Representação das rotas disponíveis para o microsserviço de notificações utilizando a ferramenta <i>Swagger</i>	31
17	Diagrama de componentes para o microsserviço das salas.	33
18	Funções a serem disponibilizadas pela Classrooms API	33
19	Representação das rotas disponíveis para o microsserviço das salas utilizando a ferramenta <i>Swagger</i>	34
20	Diagrama de sequência para registar alunos	36
21	Diagrama de sequência para criar uma prova	37
22	Diagrama de sequência para responder a uma prova	39
23	Máquina de estados de uma prova	40
24	Diagrama de ‘deployment’	40
25	Navegação em algumas páginas iniciais	42
26	Navegação das páginas durante o registo de alunos	43
27	Navegação das páginas de realização de prova	44
28	Navegação em páginas de gestão de salas	44
29	Navegação em páginas de edição e partilha de provas	45
30	Navegação em páginas criação de provas	46
31	Diagrama de sequência para register alunos	47
32	Diagrama de sequência para autenticar	47
33	Diagrama de sequência para editar perfil	48
34	Diagrama de sequência para criar uma prova	49
35	Diagrama de sequência para criar questões	50
36	Diagrama de sequência para editar uma prova	51
37	Diagrama de sequência para editar questões	52
38	Diagrama de sequência para consultar detalhes de uma prova	53
39	Diagrama de sequência para partilhar prova	53

40	Diagrama de sequência para responder a prova	54
41	Diagrama de sequência para classificar respostas	55
42	Diagrama de sequência para publicar classificações	55
43	Diagrama de sequência para consultar prova corrigida	56
44	Diagrama de sequência para gerir salas	56
45	Diagrama de sequência para aceder a notificações	57

List of Tables

1	Análise da funcionalidade dos requisitos	8
2	Expectativas de cada utilizador	9
3	Restrição quanto à infraestrutura informática	10
4	Restrição quanto ao isolamento da aplicação de resposta às provas	11
5	Análise dos requisitos de qualidade	12
6	Pontuação de cada requisito não funcional	15
7	Análise das responsabilidades de cada componente	19
8	Estrutura da prova	21
9	Estrutura do objeto “student”	22
10	Estrutura do objeto “classroom”	22
11	Estrutura do objeto “version”	22
12	Estrutura do objeto “question”	22
13	Estrutura do objeto “option”	22
14	Estrutura da resposta	23
15	Estrutura do utilizador	27
16	Estrutura da notificação	32
17	Representações do tipo	32
18	Estrutura da sala	35
19	Estrutura da reserva	35

Prefácio

Este relatório tem por objetivo documentar as decisões tomadas no desenvolvimento da segunda fase do projeto de desenvolvimento de software da Unidade Curricular “Requisitos e Arquiteturas de Software”. Nesta fase, procedemos à modelação de aspectos arquiteturais do sistema probum, cujo levantamento de requisitos tinha sido feito na primeira fase. Em relação aos objetivos para esta fase, estes foram devidamente cumpridos, tendo contudo sido necessário um esforço extra na modelação comportamental do sistema, devido às várias iterações do trabalho desenvolvido. Comparativamente com a primeira fase, o número de elementos de cada grupo aumentou de 5 para 11 elementos, o que veio criar dificuldades adicionais na gestão do projeto. A maior dificuldade foi a organização do grupo extra-aula, dado que foi impossível reunir todos os elementos do grupo num horário comum. Em relação ao trabalho desenvolvido por cada elemento do grupo, a distribuição foi a seguinte:

Número	Nome	Classificação
PG54084	Marta Isabel da Silva e Sá	-0.25
PG54117	Nuno Guilherme Cruz Varela	+0.4
PG53699	Bernardo Amado Pereira da Costa	-0.5
PG53601	Afonso Xavier Cardoso Marques	+0.4
PG53651	André Castro Alves	0
PG53870	Inês Daniela Alves Ferreira	-0.5
PG54263	Tomás Cardoso Francisco	+0.4
PG54095	Miguel de Sousa Braga	+0.4
PG54107	Millena de Freitas Santos	+0.4
A80580	Pedro Rafael Bernardo Medeiros	-0.5
PG54174	Renato André Machado Gomes	-0.25

Introdução e Objetivos

O objetivo geral do sistema a desenvolver é fornecer uma plataforma de gestão de provas de avaliação, onde docentes criam provas de avaliação e alunos as realizam. Este deverá ser um sistema diferenciador, uma vez que deverá permitir a correção automática das provas, facilitando o trabalho do docente. Para além disso, deverá contribuir para uma gestão eficiente de espaços e recursos, devendo contar, por isso, com um sistema inteligente de calendarização de salas.

Visão geral dos requisitos

Este sistema terá por base os seguintes casos de uso, detalhados de forma mais completa no documento de requisitos:

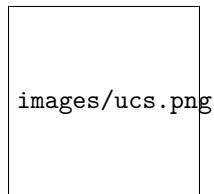


Figure 1: Diagrama de *Use Cases*

Podemos resumir as funcionalidades mais importantes nos seguintes requisitos:

ID	Prioridade	Use Case	Descrição
14	<i>Must</i>	Criar Prova	O Docente cria uma prova de avaliação
19	<i>Must</i>	Criar Prova Editar Prova	O Docente inscreve alunos numa prova de avaliação
20	<i>Must</i>	Criar Prova Editar Prova	O Sistema notifica alunos inscritos numa prova
21	<i>Must</i>	Criar Prova Editar Prova	O Docente cria versões de uma prova de avaliação
22	<i>Must</i>	Criar Prova Editar Prova	O Docente adiciona questões a uma prova de avaliação
44	<i>Must</i>	Partilhar Prova	O Docente partilha uma prova com outros Docentes
48	<i>Must</i>	Responder a Prova	O Aluno responde a uma prova
58	<i>Must</i>	Classificar Respostas	O Sistema corrige automaticamente perguntas com critérios pré-definidos por um Docente
60	<i>Must</i>	Publicar Classificações	O Docente publica as classificações de uma prova
63	<i>Should</i>	Publicar Classificações	O Aluno consulta a sua classificação de uma prova
65	<i>Must</i>	Gerir Salas	O Técnico adiciona salas

Table 1: Análise da funcionalidade dos requisitos

A análise destes requisitos permite-nos já perceber a existência de diversos tipos de dados na aplicação, como provas, respostas e salas, bem como utilizadores com papéis distintos. Será a partir destes conceitos base da aplicação que iremos realizar a separação do sistema em blocos funcionais.

É também notória a preponderância do conceito de “prova” no âmbito do problema. Esta será a entidade central da nossa aplicação e irá agregar múltiplas outras entidades como alunos, docentes e salas. Será essencial garantir a correta separação de responsabilidades entre os componentes do sistema de modo a manter sempre coerente as relações entre as provas e as outras entidades.

Outro conceito fundamental da nossa aplicação é o de “notificação”. Devido à existência de diversos tipos de notificação e ao esperado crescimento de utilizadores, será essencial garantir a escalabilidade desta funcionalidade.

Foram ainda introduzidas duas pequenas alterações relativamente aos requisitos apresentados no documento de requisitos:

- O requisito 9 foi separado em: “O Sistema valida as informações relativas

aos identificadores, nomes e emails“ e “O Sistema valida as informações relativas às salas fornecidas.“ pelo facto de mais adiante pertencerem a unidades funcionais distintas.

- O requisito 29 foi separado em: “O Sistema disponibiliza a lista de provas a que um aluno tem acesso.“ e “O Sistema disponibiliza a lista de provas a que um docente tem acesso.“ por, mais uma vez, representarem funcionalidades distintas (diferentes atores).

Stakeholder

É também importante relembrar quais as expectativas específicas de cada utilizador:

Stakeholder	Expectativas
Departamentos pedagógicos da Universidade	O Sistema proporcione uma melhor experiência de avaliação a Docentes e Alunos
Docentes	O Sistema seja fácil de usar e facilite os seus processos, ao mesmo tempo que lhes permite corrigir situações anómalas
Alunos	O Sistema seja fácil de usar e tolerante a falhas
Técnicos	O Sistema seja fácil de instalar

Table 2: Expectativas de cada utilizador

Restrições

Restrições à Solução

Requisito #:	Rest1	Tipo:	Restrição	<i>Use cases #:</i>	n.a.
Descrição			A aplicação deve executar na infraestrutura atual da respetiva IES		
<i>Rationale</i>			Para que não seja necessário investir em novo equipamento		
Origem			Cliente		
<i>Fit criterion</i>			Todos os componentes de software devem estar instalados em máquinas da IES e todas as funcionalidades da plataforma para os Alunos devem executar em pleno nas máquinas que forem disponibilizadas para as provas de avaliação		
Prioridade					

Table 3: Restrição quanto à infraestrutura informática

	Requisito #: Rest2	Tipo: Restrição	<i>Use cases #:</i> n.a.
Descrição	O computador disponibilizado a cada Aluno, no momento em que realiza uma prova de avaliação, apenas deve permitir acesso ao Probum		
<i>Rationale</i>	Para evitar que o Aluno recorra a outras aplicações (email, navegadores web, WhatsApp, Skype, etc.) durante a realização da sua prova de avaliação; O produto tem que estar preparado para funcionar em computadores instalados em salas da IES		
Origem	Cliente		
<i>Fit criterion</i>	Enquanto uma prova de avaliação estiver a decorrer não deve ser possível aceder a nenhuma outra aplicação que não o Probum		
Prioridade			

Table 4: Restrição quanto ao isolamento da aplicação de resposta às provas

Restrições Temporais

- **Descrição:** O documento presente terá de ser entregue até 1 de dezembro de 2023.

Justificação: De forma a poder ser avaliado o estado do projeto na fase da conceção da solução, é necessário que seja feita uma entrega que contenha a segunda fase deste projeto, que abrange a solução arquitetural do sistema.

- **Descrição:** A implementação parcial do sistema deverá ser entregue até 22 de dezembro de 2023.

Justificação: De forma a fornecer uma versão do PROBUM ainda sem a totalidade das funcionalidades.

Restrições Orçamentais

- **Descrição:** O orçamento total para o desenvolvimento do projeto é de 20 000€ (vinte mil euros), durante um período de 4 meses.

Justificação: A equipa responsável pelo desenvolvimento do projeto é constituída por onze engenheiros de *software*. Para além de ter em conta

os salários dos elementos, é preciso também a compra de um domínio, bem como de um computador para hospedar todos os dados da aplicação.

Requisitos de qualidade

Os requisitos de qualidade têm uma grande influência na escolha da arquitetura usada para conceber uma solução. Através de uma análise do documento de requisitos escolhemos os seguintes requisitos ordenados pela sua prioridade:

Prioridade	Tipo	Requisito
1	Performance	14 - O sistema deve ter grande disponibilidade durante horário letivo
2	Performance	9 - Qualquer interação entre o utilizador e o Sistema tem um tempo de resposta inferior a 2 segundos
3	Performance	13 - O sistema deve ser escalável
4	Manutenção e Suporte	Novo - O sistema deve ser fácil de manter

Table 5: Análise dos requisitos de qualidade

Requisito 14

O sistema deverá ter uma grande disponibilidade durante o horário letivo para que os seus utilizadores consigam utilizá-lo sempre que necessitarem. Consideramos este requisito fundamental tendo-lhe sido atribuída prioridade máxima, uma vez que é essencial garantir que o sistema possa ser acedido pelos seus utilizadores com a melhor experiência possível.

Requisito 9

O sistema deverá ser responsivo e fornecer *feedback* num intervalo de tempo muito curto, mais especificamente 2 segundos. O objetivo é proporcionar uma experiência suave e eficiente ao utilizador, pelo que lhe foi atribuída uma prioridade alta.

Requisito 13

Este requisito refere-se à capacidade do sistema ao lidar com um aumento de utilizadores, quantidades de dados ou até mesmo picos de afluência. Esta característica é crucial para que o sistema possa crescer e evoluir de modo a atender às necessidades em constante mudança. Deste modo, consideramos

também um requisito com bastante importância na escolha da arquitetura da solução.

Requisito Novo

O último requisito não se encontrava no documento de requisitos original, no entanto, é uma característica necessária à boa evolutividade do nosso sistema. Este deve ser desenhado e implementado de modo a facilitar o eventual processo de manutenção e de adição de novas funcionalidades no futuro.

Âmbito e Contexto do Sistema

Contexto do negócio

Tendo em conta o documento de requisitos, o sistema Probum terá como atores principais os Docentes, Alunos e Técnicos de uma IES. É importante perceber quais os papéis que cada utilizador pode tomar no Sistema, uma vez que o conjunto de funcionalidades a que cada um tem acesso diverge. Por exemplo, o Docente poderá realizar operações de gestão de provas, enquanto que o aluno apenas as poderá consultar e realizar. Já o técnico será responsável pela gestão dos espaços disponíveis assim como pela inscrição de alunos e docentes.

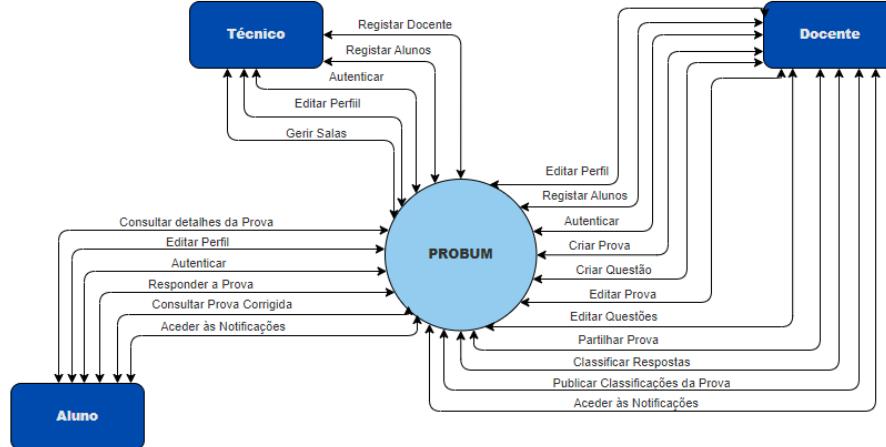


Figure 2: Diagrama de contexto

Do diagrama de contexto acima, podemos concluir que o sistema terá de fornecer vistas de utilização diferentes consoante o tipo de utilizador que utiliza o sistema. Para além disso, cada utilizador tem necessidades específicas (por exemplo, o aluno será um ator mais crítico do sistema dado o seu contexto de utilização), o que leva à necessidade de adequar o sistema às necessidades do utilizador.

Estratégia para a solução

Nesta secção são apresentadas as decisões mais importantes que levaram à escolha da solução para o problema, quer em termos arquiteturais quer em termos de tecnologias.

Decisões arquiteturais

Relativamente à arquitetura base do sistema, foram discutidas duas abordagens bastante distintas: microsserviços e arquitetura por camadas. A primeira enquadra-se nas arquiteturas distribuídas e a segunda nas arquiteturas monolíticas. As vantagens e desvantagens de cada arquitetura são apresentadas em seguida na forma de tabela:

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 por 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 em microsserviços.

Figure 3: Vantagens e desvantagens das arquiteturas.

De forma resumida, uma arquitetura em microsserviços favorece do nível de desagregação que pode ser concebida à solução, o que leva a um grande nível de escalabilidade, modularidade, tolerância a faltas e elasticidade. Por outro lado, o alto custo e performance desta arquitetura não são fatores a favor. A arquitetura monolítica por camadas tem a vantagem de ser mais simples, barata e de ser potencialmente mais eficiente. A escalabilidade e a manutenção são pontos menos fortes deste tipo de arquitetura.

Após fazer uma comparação direta entre as duas possíveis abordagens, tendo por base os requisitos não funcionais acima referidos, decidimos optar pela arquitetura em microsserviços, visto que oferece maiores garantias de escalabilidade e de evolução a longo prazo. Esta decisão foi tomada com base na seguinte tabela,

em que com base nas características de cada requisito não funcional, atribuímos a sua devida pontuação para ambas as arquiteturas discutidas.

Requisito	Microsserviços	Layered
9 - Qualquer interação entre o utilizador e o Sistema tem um tempo de resposta inferior a 2 segundos	**	**
13 - O sistema deve ser escalável	*****	*
14 - O sistema deve ter grande disponibilidade durante o horário letivo	****	*
Novo - O sistema deve ser fácil de manter	*****	*

Table 6: Pontuação de cada requisito não funcional

Desta forma, optamos por escolher uma arquitetura orientada aos microsserviços, visto que serve com melhor qualidade os requisitos não funcionais que achamos mais preponderantes na escolha da arquitetura.

Decisões relativas à arquitetura de microsserviços

A arquitetura de microsserviços é um estilo arquitetural, no qual uma aplicação é dividida em vários microsserviços independentes. Desta forma, é necessário segregar o sistema em vários componentes ou microsserviços, tendo em conta que o grau de desagregação provoca consequências.

Uma grande quantidade de microsserviços aumenta o desacoplamento (que favorece uma maior capacidade de manutenção e uma maior escalabilidade), mas provoca perdas no desempenho. Por outro lado, uma quantidade reduzida de microsserviços prejudica a escalabilidade, mas favorece o desempenho, uma vez que há menos interações entre microsserviços.

A quantidade de microsserviços dependerá também da quantidade de blocos funcionais (*bounded contexts*) identificados numa análise mais detalhada dos requisitos. Numa primeira análise, esses blocos funcionais foram levantados com base nos diferentes tipos de dados necessários para o funcionamento da aplicação (docentes, alunos, provas, salas, etc), tendo-se chegado à seguinte divisão funcional:

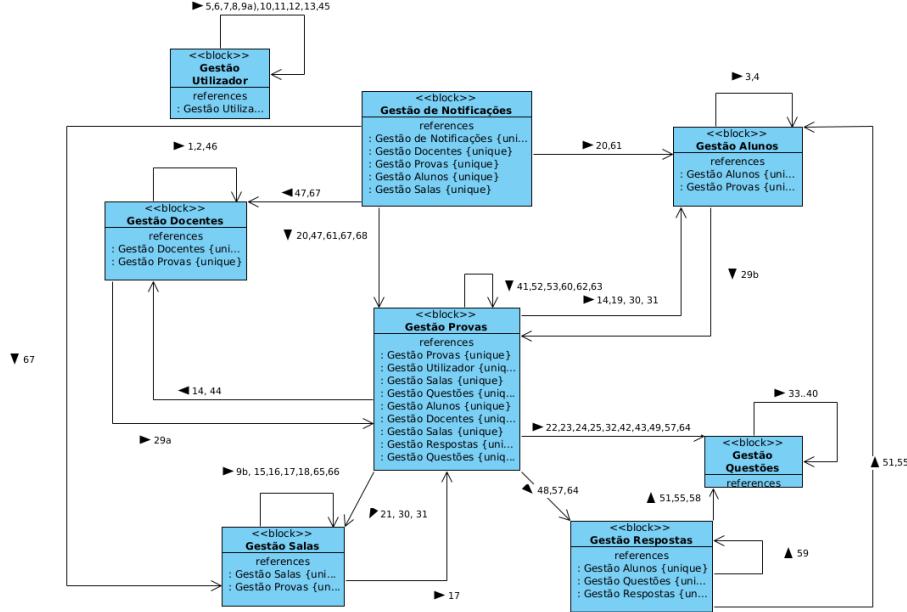


Figure 4: Decomposição em blocos funcionais

No diagrama, as setas representam o fluxo da informação dentro do sistema em cada requisito. Por exemplo, no requisito 22 (Criação de questões de escolha múltipla), a criação da questão parte da gestão de provas mas tem uma dependência com o bloco de gestão de questões.

De um modo geral, esta separação resultou numa grande quantidade de dependências entre blocos funcionais, algo que tendencialmente deve ser minimizado. Esta pareceu-nos, por isso, uma visão demasiado segregada do sistema, pelo que optamos por agrregar alguns destes blocos funcionais.

Com vista a agrregar os blocos funcionais inicialmente feitos na decomposição apresentada acima, recorremos à modelação de vários diagramas de sequência para visualizar as transações. Após esse processo, optamos por juntar o bloco de “Gestão Utilizadores”, “Gestão Docentes” e “Gestão Alunos”, visto que são blocos intrinsecamente ligados aos utilizadores, formando assim o microsserviço de utilizadores.

Da mesma forma, os blocos funcionais “Gestão Provas”, “Gestão Questões” e “Gestão Respostas” são blocos que apresentam um grande número de dependências entre si, pelo que darão origem ao microsserviço das provas. Dando um exemplo concreto, nos *use cases* “Consultar Prova Corrigida” e “Classificar

Respostas“ são consultados detalhes gerais relativos ao teste, são consultadas as questões desse teste e as respostas de um dado aluno. Ao separar estes três componentes estaremos a criar a necessidade de enviar 3 pedidos diferentes que, no fundo estão intrinsecamente relacionados. Ao escalar esta interação para um número elevado de clientes (alunos e docentes) rapidamente vemos que, mesmo este pequeno aumento no número de interações, poderá ser problemático, ao aumentar o tempo de resposta ao pedido do cliente.

De forma semelhante, tinhamos optado por juntar a “Gestão de Notificações“ com a “Gestão de Utilizadores“ anteriormente referida, dado que uma notificação está intrinsecamente ligada a um utilizador. Após alguma discussão, decidimos criar um microsserviço independente para as notificações, dado que pode ser um serviço que pode escalar muito rapidamente devido ao elevado volume de dados que vai ter de suportar. Por fim, criamos um microsserviço para as salas, visto que diminuímos algumas dependências entre microsserviços e acaba por ser um sistema independente de todos os outros.

Resumidamente a nossa arquitetura de microsserviços suportará 4 microsserviços distintos: utilizadores, provas, notificações e salas.

O *front-end* da aplicação irá interagir com uma camada de *software* denominada de API Gateway através de uma API REST. A API Gateway surge como forma de gerir todos os pedidos da aplicação e como forma de segurança, porque é ela que vai validar e gerir todos os pedidos.

Decisões tecnológicas

Microsserviços e API Gateway

A decisão sobre as tecnologias adotadas para os microsserviços foi uniforme, resultando na utilização de um conjunto padronizado de tecnologias por todos eles.

A linguagem de programação **JavaScript** será utilizada juntamente com o ambiente de execução **Node.js** devido à sua eficiência e à natureza baseada em eventos e não bloqueante, ideal para microsserviços que necessitam de alta performance e escalabilidade. Juntamente, optamos pelo **Express**, uma *framework* minimalista e flexível. O Express facilita a criação de APIs REST robustas, permitindo-nos desenvolver funcionalidades complexas com simplicidade e eficiência.

Para a persistência dos dados, escolhemos o **MongoDB**, considerando a sua compatibilidade com **Node.js** e a flexibilidade do seu modelo de dados baseado em documentos. Esta escolha permite-nos uma fácil escalabilidade vertical, essencial para lidar com o crescimento do volume de dados gerados por um

sistema de microsserviços. Além disso, a estrutura de dados sem esquema do **MongoDB** simplifica o processo de manutenção e adaptação às mudanças de requisitos, uma vez que não há a necessidade de reestruturar tabelas como em base de dados relacionais. Isto não se aplica à API *Gateway* visto que não há a necessidade de persistência de dados.

FrontEnd

O desenvolvimento do *frontend* vai ser realizado em **React** devido à sua eficiência e capacidade de criar interfaces dinâmicas e responsivas. A sua abordagem baseada em componentes permite a reutilização de código, facilitando a manutenção e o desenvolvimento acelerado. Além disso, a vasta comunidade e o ecossistema em torno do React proporcionam acesso a uma ampla gama de ferramentas e bibliotecas, aumentando a produtividade e possibilitando a implementação de práticas modernas de *design*.

Building Block View

Visão geral do sistema

Com base na divisão em blocos funcionais do capítulo anterior, definimos a versão final do diagrama de componentes que espelha a 1^a camada de abstração da aplicação. A nossa aplicação será constituída por 6 componentes principais: a interface com o utilizador (Probum FrontEnd), a Gateway API e os microsserviços relacionados com os utilizadores, provas, notificações e salas.

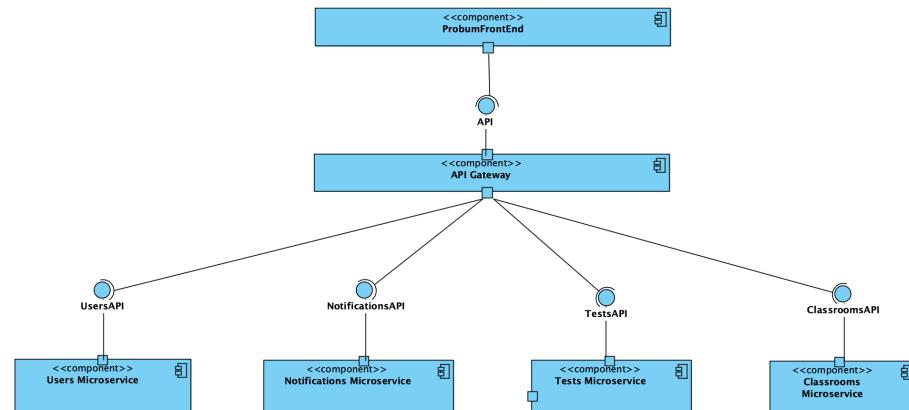


Figure 5: Diagrama de componentes

A divisão em 4 microsserviços teve por base o objetivo da minimização do número de transações entre microsserviços, mantendo também em mente a necessidade

de escalabilidade do sistema. As responsabilidades de cada componente são as descritas na tabela abaixo:

Componente	Responsabilidade
ProbumFrontEnd	Interagir com o cliente por meio de uma interface web
APILayer	Servir como um proxy para o FrontEnd; Interagir com os vários microsserviços; Assegurar autenticação
Microsserviço dos Utilizadores	Fazer a gestão dos utilizadores (alunos e docentes) e fornecer um meio de autenticação
Microsserviço das Notificações	Gerir as notificações
Microsserviço dos Testes	Fazer a gestão das provas dos docentes e das respostas dos alunos
Microsserviço das Salas	Permitir gerir salas e fazer a calendarização

Table 7: Análise das responsabilidades de cada componente

API Gateway

A API Gateway tem um papel fundamental no contexto do sistema, visto que é uma camada por onde vão chegar todos os pedidos. É também responsável por controlar toda a segurança e permissões de cada pedido. De maneira a ser comunicada, o *FrontEnd* fará uso de uma API REST para enviar pedidos a esta camada de *software*.

No contexto do sistema Probum, a API Gateway terá um papel de “orquestrador” quando está a atender pedidos. Desta forma, será responsável por tratar pedidos que envolvam o contacto de vários microsserviços, pelo que terá de aguardar pelas diversas respostas provenientes destes para poder contactar o próximo serviço. Desta forma, a API Gateway retira a responsabilidade aos microsserviços de resolverem os pedidos e utilizará as APIs REST exportadas por serviços para interagir com eles.

Com base na análise dos *use cases*, a API Gateway terá de suportar os seguintes pedidos, que posteriormente serão materializados em pedidos HTTP, visto que estamos perante uma API REST.

<code><<Interface>></code>
APIGateway
<code>+getUnreadNotifications(userId) : number</code>
<code>+getNotifications(userId) : List<Notification></code>
<code>+getTeacherTests(teacherId) : List<Test></code>
<code>+getTestTeacher(testId) : Test</code>
<code>+correctAutomatically(testId) : void</code>
<code>+getStudentsTest(testId, studentId) : Test</code>
<code>+updateCorrection(testId, studentId, questionId, newCorrection)</code>
<code>+getStudentTests(studentId) : List<Test></code>
<code>+getTestStudent(testId) : Test</code>
<code>+consultCorrection(studentId, testId) : Correction</code>
<code>+validateStudents(file) : success</code>
<code>+scheduleClassroomsRequest(date, hour, duration, admissionTime) : Map<Sala, Horário></code>
<code>+createTest(name, students, classrooms)</code>
<code>+addTestDetails(versions, randomness, goBack, versionClassroomAssociations)</code>
<code>+createVersionRequest(version, questions)</code>
<code>+sendTestRegistrationNotifications(emails)</code>
<code>+getProfile(userId) : Profile</code>
<code>+editProfile(userId, password, email) : success</code>
<code>+registerStudent(name, number, email) : success</code>
<code>+registerStudents(file) : success</code>
<code>+registerTeacher(name, number, email) : success</code>
<code>+registerTeachers(file) : success</code>
<code>+login(email, password) : success</code>
<code>+startTest(testId, studentId) : List<Question></code>
<code>+submitTest(answers, testId, studentId) : success</code>
<code>+publishClassifications(testId) : success</code>
<code>+shareTest(testId, teacherIds) : success</code>
<code>+addClassrooms(file) : success</code>
<code>+getClassrooms() : List<Classroom></code>
<code>+removeClassrooms(classroomId) : success</code>
<code>+editTest(testId, details) : success</code>
<code>+getVersionQuestions(testId, version) : List<Question></code>

Figure 6: Pedidos suportados pela API Gateway.

Microsserviços

Tal como foi anteriormente referido nas decisões tecnológicas tomadas, os microsserviços serão desenvolvidos com recurso à tecnologia “Node.js” recorrendo à framework “Express.js”, pelo que recorremos a diagramas de componentes para descrever cada microsserviço.

Os microsserviços possuem todos uma mesma estrutura em que existe um componente “app.js” que serve como “motor de arranque” para a inicialização de cada serviço. Os “routers” são uma peça de software fundamental que vão atender aos pedidos HTTP e acabam por exportar a API de cada microsserviço que será utilizada posteriormente pela API Layer. Com vista a persistir os dados provenientes do *FrontEnd*, vão ser utilizados “models”, responsáveis por fazer o contacto com a base de dados MongoDB com recurso à biblioteca Mongoose. Os “controllers” são módulos que recorrem aos “models” com o objetivo de ir buscar informação à base de dados.

Microsserviço das provas

O microsserviço relativo às provas materializa-se no seguinte diagrama de componentes:

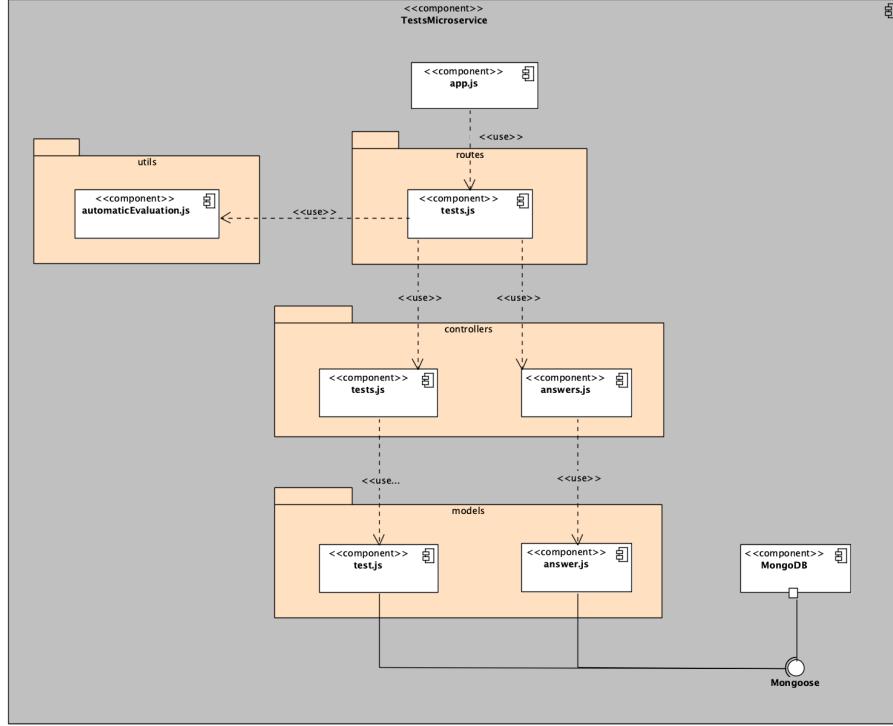


Figure 7: Diagrama de componentes do microserviço das provas

Relativamente à informação das provas que necessita de ser persistida na base de dados, optamos por dividir em duas coleções: uma para os detalhes relacionados com as provas e outra para guardar as respostas dadas pelos alunos às questões. Na primeira coleção será necessário guardar os professores, alunos, salas e versões para cada prova. Na coleção das respostas serão guardadas todas as respostas efetuadas nos testes. Desta forma foi definida a seguinte estrutura para guardar a informação anteriormente referida:

Nome do campo	Tipo	Descrição
_id	String	Identificador da prova
name	String	Nome da prova
throwback	Boolean	Retrocesso nas questões
randomness	Boolean	Aleatoriedade das questões
teachers	[String]	Lista de professores
students	[student]	Lista de objetos “student”
classrooms	[classroom]	Lista de objetos “classroom”
versions	[version]	Lista de objetos “version”

Table 8: Estrutura da prova

A estrutura dos documentos “student”, “classroom” e “version” consiste nos seguintes esquemas:

Nome do campo	Tipo	Descrição
_id	String	Identificação do aluno
classroom	String	Sala atribuída ao aluno

Table 9: Estrutura do objeto “student”

Nome do campo	Tipo	Descrição
_id	String	Identificador da sala
date	Date	Data
version	Number	Identificador da versão

Table 10: Estrutura do objeto “classroom”

Nome do campo	Tipo	Descrição
_id	Number	Identificação da versão
questions	[question]	Lista de objetos “question”

Table 11: Estrutura do objeto “version”

Nome do campo	Tipo	Descrição
_id	Number	Identificação da questão
description	String	Descrição da questão
type	Number	Tipo da questão
grade	Number	Cotação da questão
options	[option]	Lista de objetos "option"

Table 12: Estrutura do objeto “question”

Nome do campo	Tipo	Descrição
description	String	Descrição da alínea
grade	Number	Cotação da alínea
solution	String	Solução da alínea

Table 13: Estrutura do objeto “option”

Nome do campo	Tipo	Descrição
questionId	Number	Identificação da questão
studentId	String	Identificação do aluno
testId	String	Identificação da prova
text	String	Resposta de texto livre
options	[String]	Respostas selecionadas pelo aluno
score	Number	Cotação atribuída à resposta

Table 14: Estrutura da resposta

Foram identificadas as seguintes funções a serem disponibilizadas pela API:

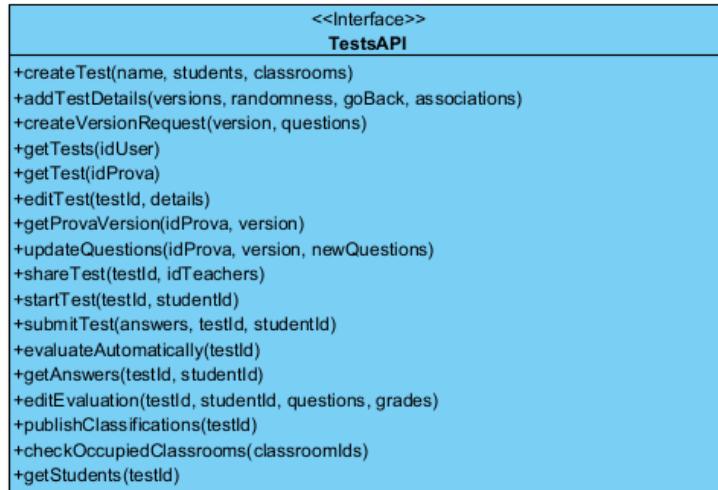


Figure 8: Funções a serem disponibilizadas pela TestsAPI

Com base no levantamento dos pedidos que a API dos testes terá de servir, foram identificadas as seguintes rotas para este microsserviço:

The screenshot shows the 'Test API' documentation page. At the top, it says '1.0.0 OAS 3.0'. Below that is a 'Servers' section with a dropdown menu set to 'https://localhost:3000 - Te...'. The main content is titled 'Tests' and lists several API endpoints:

- POST /tests** Creates a new test
- PUT /tests** Add details to a test
- GET /tests/{idTest}** Gets test
- PUT /tests/{idTest}** Saves all answers from a student to a test
- GET /tests/{idUser}** Gets all tests from a student
- PUT /tests/{idUser}** Share test with other professors
- PUT /editTest/{idTest}** Edits the details of this test

Figure 9: Representação das rotas disponíveis para o microsserviço de provas utilizando a ferramenta *Swagger* - 1

This screenshot is identical to Figure 9, showing the 'Tests' endpoint with the same seven available routes.

Figure 10: Representação das rotas disponíveis para o microsserviço de provas utilizando a ferramenta *Swagger* - 2

Relativamente aos pedidos acima, é importante referir a estrutura do campo body dos pedidos:

POST /tests

```
{
  "name": String,
  "students": [String],
  "classrooms": [String]
}
```

```

PUT /tests
{
    "versions": Number,
    "randomness": Boolean,
    "goBack": Boolean,
    "associations": [
        {
            "classroomId": String,
            "version": Number,
            "time": String
        }
    ]
}

PUT /tests/{idTest}
{
    "studentID": String,
    "answers": [
        {
            "questionID": String,
            "answer": String
        }
    ]
}

PUT /tests/{idUser}
{
    "idTeachers": [String]
}

PUT /editTest/{idTest}
{
    "name": String,
    "duration": String,
    "dates": [String],
    "times": [String],
    "classrooms": [String]
}

POST /tests/{idTest}/{version}
{
    "version": 1,
    "questions": [
        {
            "description": String,

```

```

        "type": String,
        "grade": Number,
        "options": {
            "description": String,
            "grade": Number,
            "solution": String
        }
    }
]
}
}

PUT /tests/{idTest}/{version}
{
    "version": 1,
    "questions": [
        {
            "description": String,
            "type": String,
            "grade": Number,
            "options": {
                "description": String,
                "grade": Number,
                "solution": String
            }
        }
    ]
}
}

PUT /evaluateTest/{idTest}/{studentId}
{
    "questionId": String,
    "grade": Number
}

```

Neste microsserviço, podemos ainda observar a presença do padrão observer na funcionalidade de publicação das notas de uma prova. Na verdade, cada prova terá a noção de estado (i.e foi criada, está disponível e já foi corrigida) e será por isso o *Subject*. As entidades interessadas na mudança da prova serão os alunos inscritos na prova (*observers*), pelo que 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*).

Microsserviço dos utilizadores

O microsserviço dedicado à gestão dos utilizadores será constituído por quatro diretórias diferentes *authentication*, *routes*, *controllers* e *models*. A diretoria dos

controladores tem a função de gerir os acessos à base de dados. Os roteadores atendem os pedidos que irão ser realizados nesta API. A diretoria relativa aos modelos guarda todos os *schemas* da informação que pretendemos persistir na base de dados. E por fim a diretoria da autenticação trata de todo o processo de autenticar ou registar um utilizador no sistema.

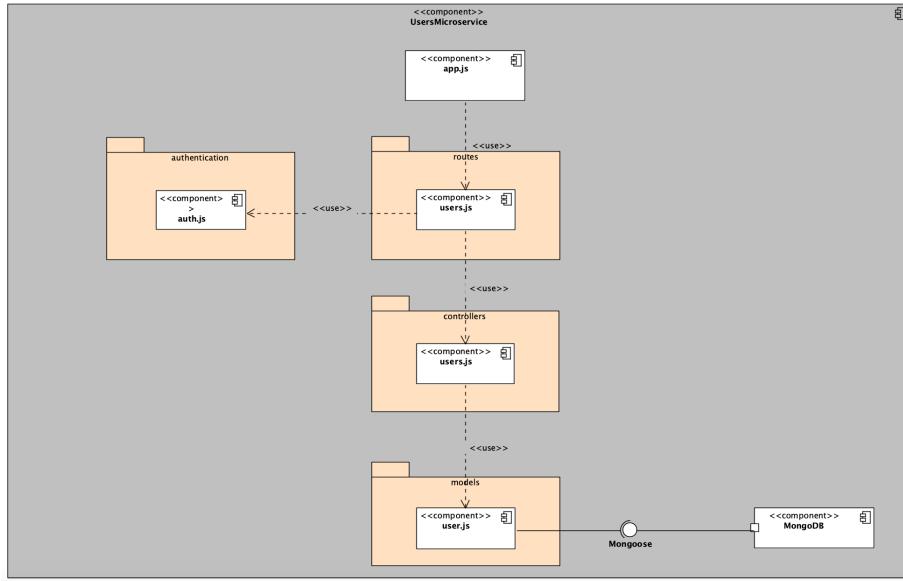


Figure 11: Diagrama de componentes para o microsserviço de utilizadores.

A base de dados do utilizador terá apenas uma coleção que irá persistir os utilizadores do sistema. A estrutura do utilizador a ser persistido possui todas as informações relativas a este como o seu identificador, email, nome e *hash* da sua *password*.

Nome do campo	Tipo	Descrição
<code>_id</code>	String	Número mecanográfico do utilizador
<code>email</code>	String	Endereço email do utilizador
<code>name</code>	String	Nome do utilizador
<code>password</code>	String	Hash da password do utilizador
<code>role</code>	String	Papel do utilizador no Sistema (Cliente, Técnico ou Aluno)

Table 15: Estrutura do utilizador

mudamos o tipo de role para String

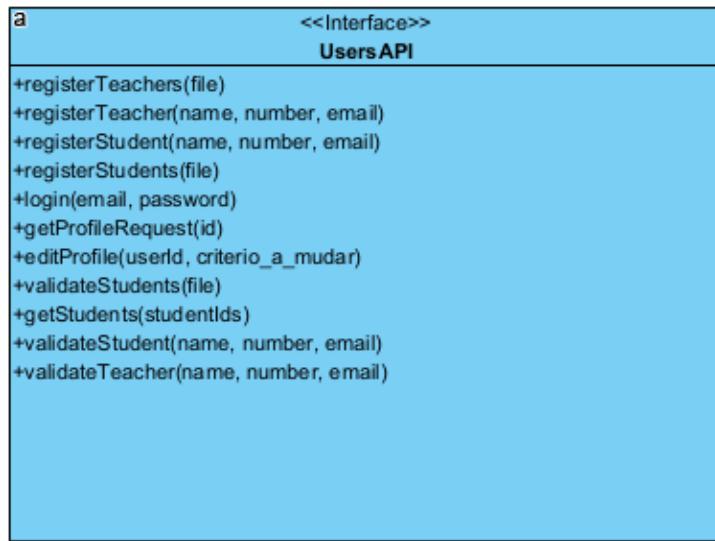


Figure 12: API para o microsserviço de utilizadores

A interface apresentada representa, de forma abstrata, a API dos utilizadores sendo que os métodos que contém vão ser mapeados em pedidos HTTP, dando origem às seguintes rotas.

Users API 1.0.0 OAS 3.0

Users

^

GET /users/{id} Get users.

PUT /users/{id} Edit profile

POST /users Create a list of new users.

PUT /users/validate Validate users.

POST /users/login Perform the Login.

^

Figure 13: Representação das rotas disponíveis para o microsserviço dos utilizadores utilizando a ferramenta *Swagger*

Relativamente às rotas acima especificadas, é importante referir a estrutura do campo body dos pedidos:

```

GET /users/{id}
{
    "_id": String,
    "name": String,
    "role": String
}

PUT /users/{id}
{
    "name": String,
    "_id": String,
    "email": String,
    "role": String
}

POST /users
[
    {
        "name": String,
        "_id": String,
        "email": String,
        "role": String
    }
]
PUT /users/validate
[
    {
        "name": String,
        "_id": String,
        "email": String,
        "role": String
    }
]
POST /users/login
[
    {
        "name": String,
        "_id": String,
        "email": String,
        "role": String,
        "password": String
    }
]

```

Microsserviço das notificações

O microsserviço das notificações, responsável por guardar as notificações de cada utilizador terá uma estrutura semelhante à dos outros microsserviços, subdividindo-se num módulo de resposta de pedidos (*router*), num módulo de funcionalidade CRUD (*controller*) e num módulo de interface com a base de dados (*model*). Para além das funcionalidades básicas de acesso a notificações, deverá ser também implementada a funcionalidade de envio de emails. Optamos, assim, por criar um módulo independente para essa funcionalidade, que será implementada através da biblioteca **nodemailer** do *express*.

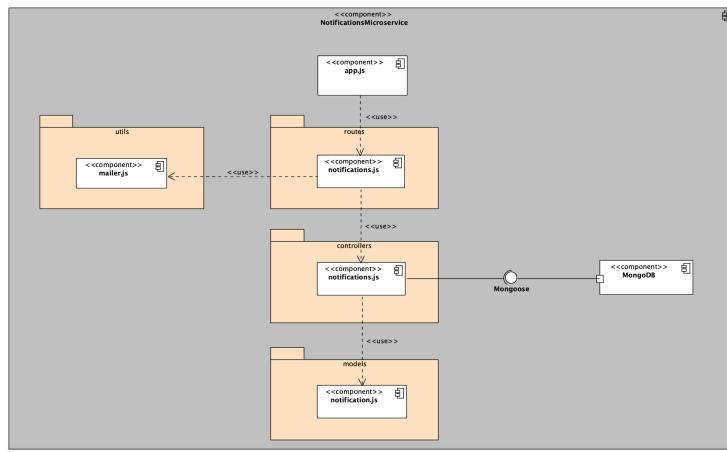


Figure 14: Diagrama de componentes para o microsserviço de utilizadores.

Foram identificadas as seguintes funções a serem disponibilizadas pela API:

<<Interface>> NotificationsAPI
+notifyRegistration(emails) +notifyTestRegistration(emails) +notifyTestSharing(teacherIds) +notifyClassificationsPublication(studentId, testId, mark) +notifyClassroomsRemoval(idDocentes, idProvas, classroomsIds) +getUnreadNotifications(userId) +getNotifications(userId)

Figure 15: Funções a serem disponibilizadas pela Notifications API

O conjunto de pedidos que deve ser respondido é o seguinte:

Notifications API 1.0.0 OAS 3.0

Notifications	
GET	/notifications/idUser Gets notifications from user
GET	/notifications/unread/idUser Gets unread notifications from user
POST	/notifications/classroomRemoval Notifies teacher of removal of a classroom
POST	/notifications/testSharing Notifies teacher of sharing of a test
POST	/notifications/gradesAvailable Notifies availability of grades
POST	/notifications/registration Notifies registration of user

Figure 16: Representação das rotas disponíveis para o microsserviço de notificações utilizando a ferramenta *Swagger*

É também importante mencionar a estrutura do campo body dos seguintes pedidos:

POST /notifications/classroomRemoval

```
{  
  "classrooms": [  
    {"id": Number,  
     "tests": [  
       {"id": Number,  
        "teachers": [Number]  
      ]  
    ]  
  ]  
}
```

POST /notifications/testSharing

```
{  
  "testId": Number,  
  "teachers": [Number]  
}
```

POST /notifications/gradesAvailable

```
{  
  "testId": Number,  
  "studentId": Number,  
  "grade": Number  
}
```

POST /notifications/registration

```
{
  "emails": [String]
}
```

Para persistir a informação das notificações na base de dados, serão criados os seguintes campos:

Nome do campo	Tipo	Descrição
user	String	Identificador do utilizador associado à notificação
date	Date	Data em que foi emitida a notificação
text	String	Texto da notificação
status	Boolean	Lida ou não lida
type	Number	O tipo de notificação (partilha de prova, classificações disponíveis, etc)

Table 16: Estrutura da notificação

Tipo	Representação
<i>classroomRemoval</i>	0
<i>testSharing</i>	1
<i>gradesAvailable</i>	2
<i>registration</i>	3

Table 17: Representações do tipo

Microsserviço das salas

O microsserviço das notificações seguirá também a mesma estrutura, sendo contudo necessário fazer referência ao componente *scheduling.js* que deverá tratar de questões ligadas ao escalonamento das salas.

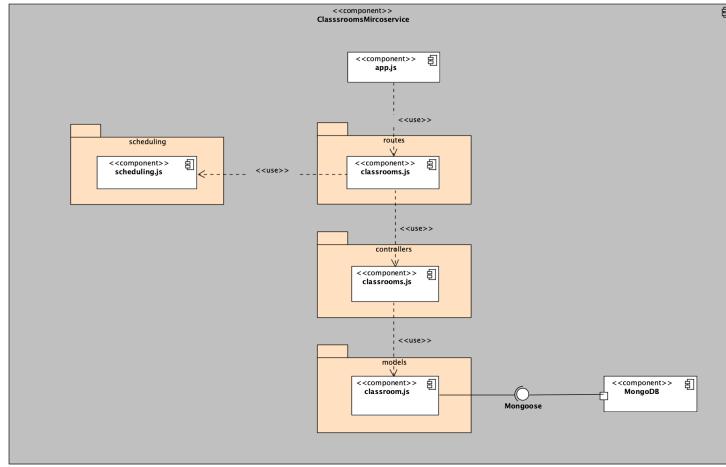


Figure 17: Diagrama de componentes para o microsserviço das salas.

Foram identificadas as seguintes funções a serem disponibilizadas pela API:

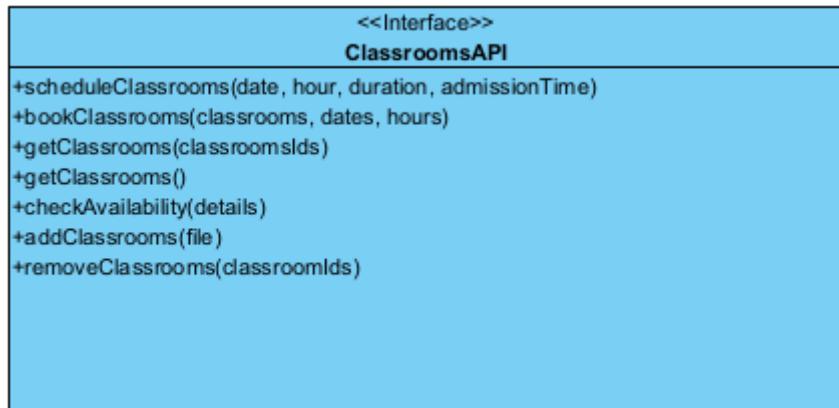


Figure 18: Funções a serem disponibilizadas pela Classrooms API

Este microsserviço procurará assim dar resposta aos seguintes pedidos:

The screenshot shows the 'Classrooms' section of a Swagger API documentation. At the top, there are two status indicators: '1.0.0' and 'OAS 3.0'. Below this, the 'Classrooms' section title is followed by a list of five API endpoints:

- GET** /classrooms/<id> Gets the information from a classroom with a given id
- GET** /classrooms/schedule/{date}/{duration} Gets all classrooms available in that schedule
- PUT** /classrooms/reserve Reserves a set of rooms in a given timetable (highlighted in orange)
- POST** /classrooms/create Adds new classrooms
- DELETE** /classrooms/delete/{classroomId} Delete a given classroom

Figure 19: Representação das rotas disponíveis para o microsserviço das salas utilizando a ferramenta *Swagger*

Relativamente aos pedidos acima, fazemos novamente referência à estrutura do campo body dos mesmos:

PUT /classrooms/reserve

```
[  
  {  
    "id": Number,  
    "start": String,  
    "end": String  
  }  
]
```

POST /classrooms/create

```
[  
  {  
    "floor": "1",  
    "building": "CP1",  
    "capacity": 30  
  }  
]
```

Cada sala deverá assim ser guardada na Base de Dados com campos que permitem a sua futura calendarização, entre eles a capacidade da sala e as reservas da mesma.

Nome do campo	Tipo	Descrição
_id	String	Identificador da sala
floor	String	Piso no edifício
building	String	Edifício
capacity	Number	Capacidade da sala
reservations	[reservation]	Lista de objetos “reservation”

Table 18: Estrutura da sala

Nome do campo	Tipo	Descrição
startDate	Date	Início da reserva
endDate	Date	Fim da reserva

Table 19: Estrutura da reserva

Runtimes View

Diagramas de Sequência

Nesta secção iremos modelar através de diagramas de sequência a forma como os microsserviços interagem entre si nos casos de uso mais relevantes do sistema Probum.

Registrar Alunos

No caso de uso “Registrar Alunos” o Docente interage com o *Frontend*, indicando que quer aceder à opção de registar alunos. O *Frontend* então disponibiliza ao Docente a página para o registo de Alunos. Esta página disponibiliza duas opções diferentes de registo de alunos: registo de um único aluno e registo de múltiplos alunos simultaneamente.

Caso o Docente opte por apenas registrar um aluno, o *Frontend* irá disponibilizar a página específica contendo o formulário a ser preenchido pelo Docente. Este deverá então introduzir o nome, número mecanográfico e email do estudante. O *Frontend* então de seguida um pedido à APIGateway com esta informação para registrar o aluno. A APIGateway reencaminhará este pedido para o microsserviço de Utilizadores que será responsável por efetuar o registo do aluno.

Neste ponto, a informação recebida é validada. Caso a validação seja positiva, o microsserviço de Utilizadores enviará um pedido ao microsserviço de Notificações para notificar o aluno registrado. Caso seja negativa, o microsserviço de Utilizadores irá enviar uma mensagem de erro à APIGateway que será reencaminhada até o *Frontend* para que o Docente seja devidamente informado.

Caso o Docente opte por introduzir múltiplos alunos no sistema simultaneamente, o *Frontend* irá requisitar que introduza um ficheiro com a informação dos alunos em questão. Ao submeter o ficheiro, o *Frontend* envia um pedido, contendo o ficheiro, à APIGateway para registar os alunos. Este pedido é então reencaminhado ao microsserviço de utilizadores.

Neste ponto, a informação do ficheiro é validada. Caso a validação seja positiva, os alunos são devidamente registados no sistema e o microsserviço de Utilizadores enviará um pedido ao microsserviço de Notificações para notificar todos os alunos registados. Caso seja negativa, o microsserviço de Utilizadores irá enviar uma mensagem de erro à ApiGateway que será reencaminhada até ao Frontend para que o Docente seja devidamente informado que o ficheiro que introduziu foi considerado inválido.

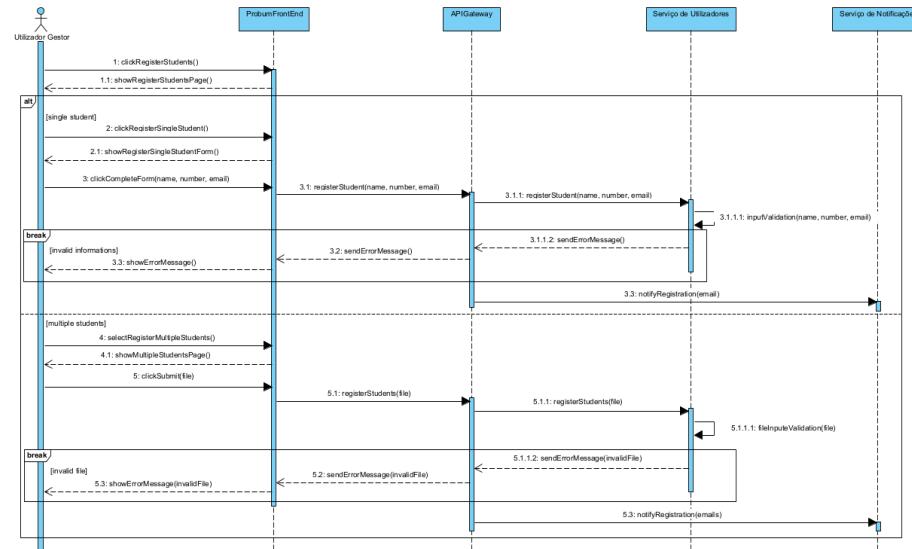


Figure 20: Diagrama de sequência para registar alunos

Criar Prova

No caso de uso “Criar Prova“, o Docente começa por submeter um ficheiro com informação relativa a todos os utilizadores, pelo que o *Front End* terá de enviar um pedido à API Gateway com a informação do ficheiro para validar a existência dos alunos no sistema. A API Gateway devolverá uma resposta de confirmação.

De maneira a ser realizada a calendarização, o *Front End* realizará um pedido, com a data, duração, tempo de admissão e número de alunos fornecidos pelo Docente, à API Gateway que posteriormente será retransmitido ao serviço das

salas, visando verificar a disponibilidade das salas nos parâmetros fornecidos. O microserviço das salas devolverá uma resposta com as salas disponíveis de acordo com a informação recebida. Numa situação de indisponibilidade das salas, responderá com uma mensagem de erro.

Quando o Docente aceitar as salas propostas, irá ser realizado um pedido à API Gateway para criar a prova com as salas, datas e alunos já fornecidos, mas ainda sem as questões. Como referenciado na secção anterior, a API Gateway assume um papel de “orquestrador”, algo que se pode verificar no seu comportamento face a este pedido. Este componente de *software* começa por contactar o serviço das salas para reservar as salas nas datas fornecidas e, de seguida, contacta o serviço das provas para criá-la sem questões.

Antes de definir as versões da prova, o Docente pode ainda definir alguns detalhes como o número de versões, opção de retroceder nas perguntas, aleatoriedade nas perguntas e ainda associar para cada sala uma versão do teste. Desta forma, será realizado um pedido para adicionar estes detalhes à prova já previamente criada.

Para cada versão da prova, o Docente definirá as suas questões e, posteriormente, será feito um pedido com o número da versão e questões, com o objetivo de adicionar essa versão à prova.

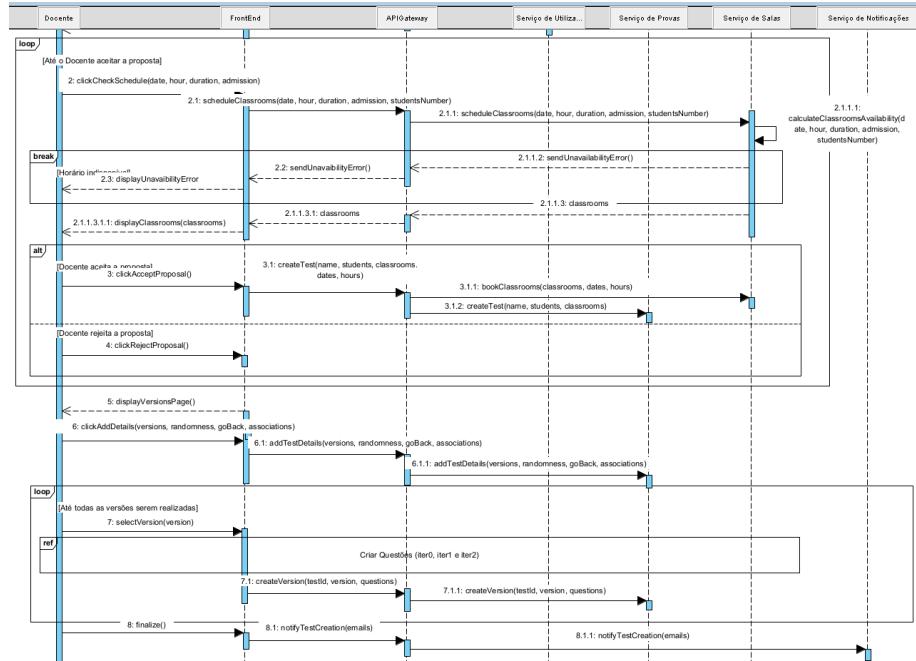


Figure 21: Diagrama de sequência para criar uma prova

Responder à Prova

No caso de uso “Responder à Prova”, o Aluno envia um pedido ao *Frontend* para iniciar a Prova. É acrescentado então o id da prova e o id do aluno ao pedido e este é reencaminhado ao microsserviço de Provas. O microsserviço de Provas então responde devolvendo as questões, que são reencaminhadas de volta até ao *Frontend*. Caso não seja possível recolher as questões, uma mensagem de erro é encaminhada para que o Aluno seja devidamente informado.

Para cada questão, o *Frontend* disponibiliza todo o seu conteúdo. Caso a possibilidade de escolher questão esteja disponível, o Aluno envia um pedido ao *Frontend* de modo a requisitar a próxima questão e este disponibiliza o respetivo conteúdo.

O aluno então envia um pedido, para cada questão, ao *Frontend* para submeter a sua resposta à respetiva questão. Caso a questão seja salva, o *Frontend* disponibiliza uma mensagem de confirmação ao Aluno. Caso contrário, disponibiliza uma mensagem de erro para que seja devidamente informado.

Ao desejar terminar de responder a prova, o Aluno então envia um pedido ao *Frontend* para finalizar a prova. Este agrupa todas as respostas dadas, o id da prova e o id do aluno e envia um pedido contendo esta informação à APIGateway que trata de encaminhar o pedido para o microsserviço de provas para que a informação seja devidamente tratada e guardada. Por fim, o microsserviço de provas envia uma mensagem de confirmação à APIGateway e esta reencaminhada até ao o *Frontend* de modo a ser disponibilizada ao Aluno.

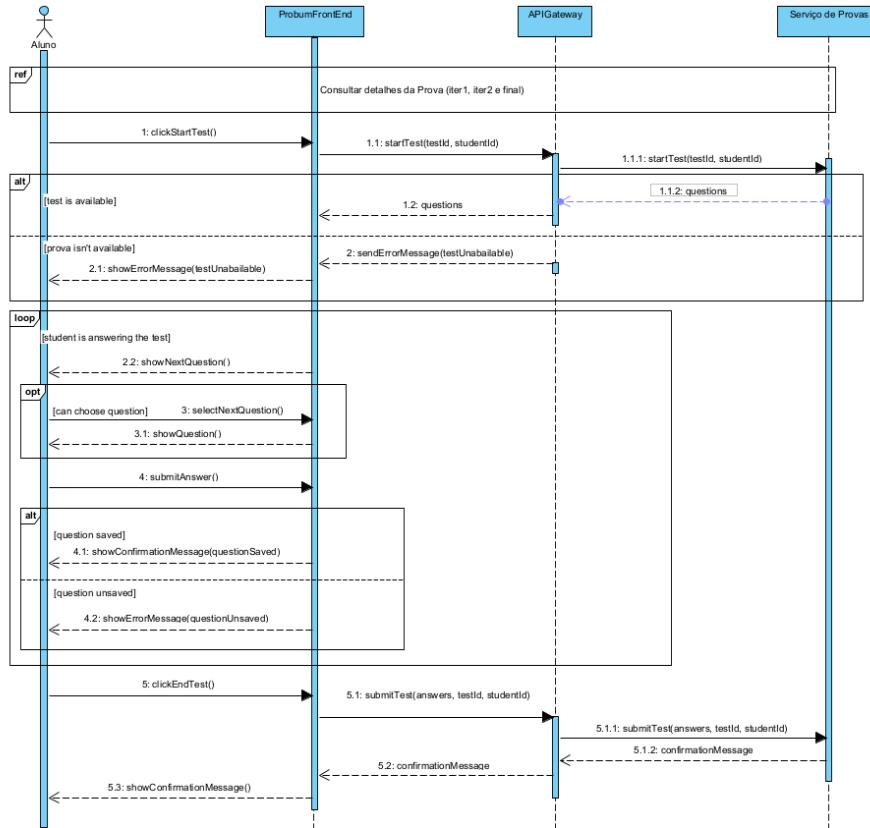


Figure 22: Diagrama de sequência para responder a uma prova

Atualização do estado de uma prova

Relativamente ao estado de uma prova, será necessário entender os mecanismos de transição entre estados e as consequências de cada transição. A criação da prova, como vimos, é espoletada pelo docente e permite o registo da mesma no microsserviço das provas, juntamente com os seus alunos. Depois, a prova deverá ser disponibilizada no horário definido pelo docente, por meio de um evento que desencadeia a mudança de estado de uma prova. Depois, o ato de publicação das notas por parte do docente origina a execução do método de notificação dos alunos inscritos na prova. Por fim, outro evento gerado pelo sistema deverá ser o da remoção da prova, acontecendo 2 anos depois da publicação das notas.

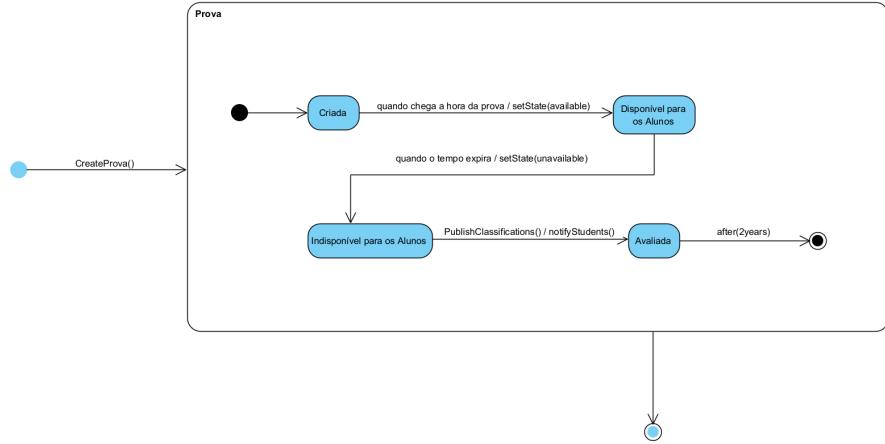


Figure 23: Máquina de estados de uma prova

Deployment View

Para melhor entendermos como os vários componentes do sistema Probum devem ser instalados, desenhamos o seguinte diagrama de *deployment*, que nos mostra as características e o tipo de dispositivos que vão suportar a nossa aplicação.

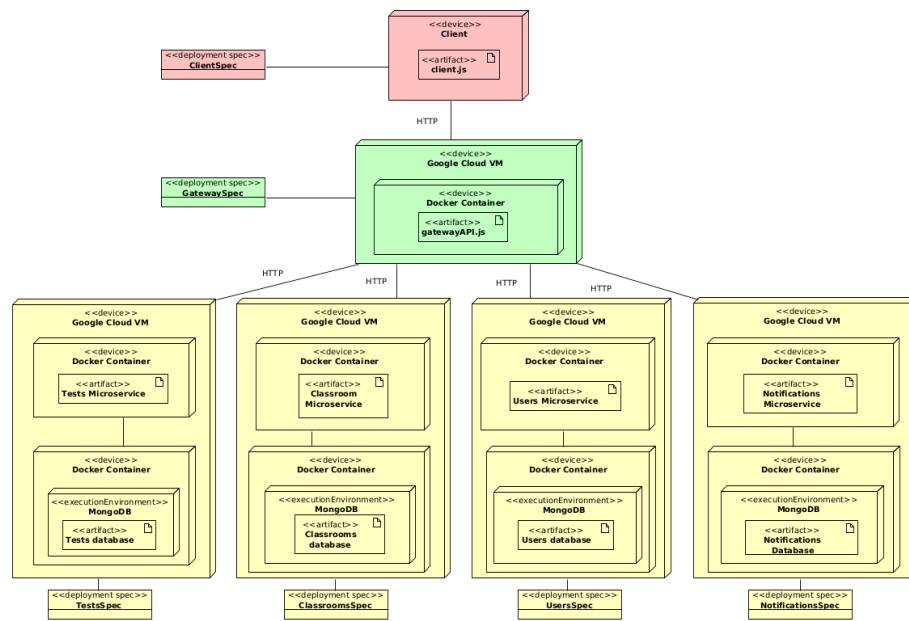


Figure 24: Diagrama de ‘deployment’

A opção por uma instalação distribuída, fazendo uso dos serviços de computação em nuvem da google tem por base os requisitos não funcionais de escalabilidade referidos anteriormente. Da mesma forma, cada microsserviço deverá executar de forma independente da base de dados mongo respetiva, pelo que executarão em “docker containers” diferentes. A opção pelo docker dá-nos também uma maior facilidade na instalação da mesma aplicação em múltiplas máquinas.

As especificações de cada *device* são as seguintes:

- **ClientSpec** - Nenhuma restrição em particular. Apenas é preciso um browser para correr a aplicação web.
- **GatewaySpec** - Componente crítico da aplicação. Precisa de ter o docker instalado. Mínimo de 8Gb de RAM. Deve poder ser escalado verticalmente.
- **TestsSpec** - Componente crítico da aplicação. Precisa de ter o docker instalado. Mínimo de 8Gb de RAM e 128Gb de disco.
- **ClassroomsSpec** - Componente menos crítico da aplicação. Um mínimo de 32Gb de RAM.
- **UsersSpec** - Componente crítico da aplicação. Precisa de ter o docker instalado. Mínimo de 8Gb de RAM e 128Gb de disco.
- **NotificationsSpec** - Componente crítico da aplicação. Precisa de ter o docker instalado. Mínimo de 8Gb de RAM e 128Gb de disco.

Vista “Frontend”

Nesta secção, apresentamos alguns mapas de navegação da interface do sistema. Com isto os podemos compreender melhor a disposição da aplicação, facilitando a navegação e a procura de informações. Esta estrutura será a seguida na implementação da aplicação cliente.

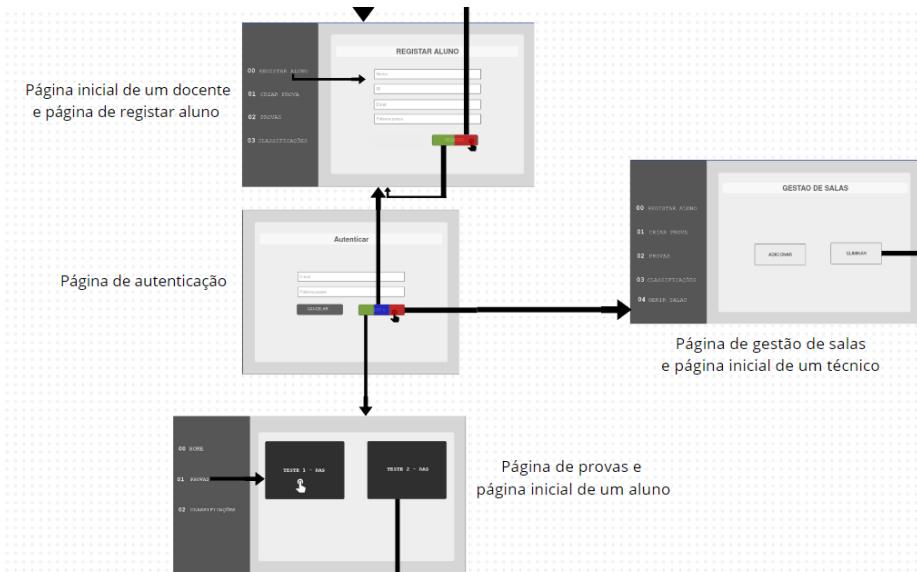


Figure 25: Navegação em algumas páginas iniciais

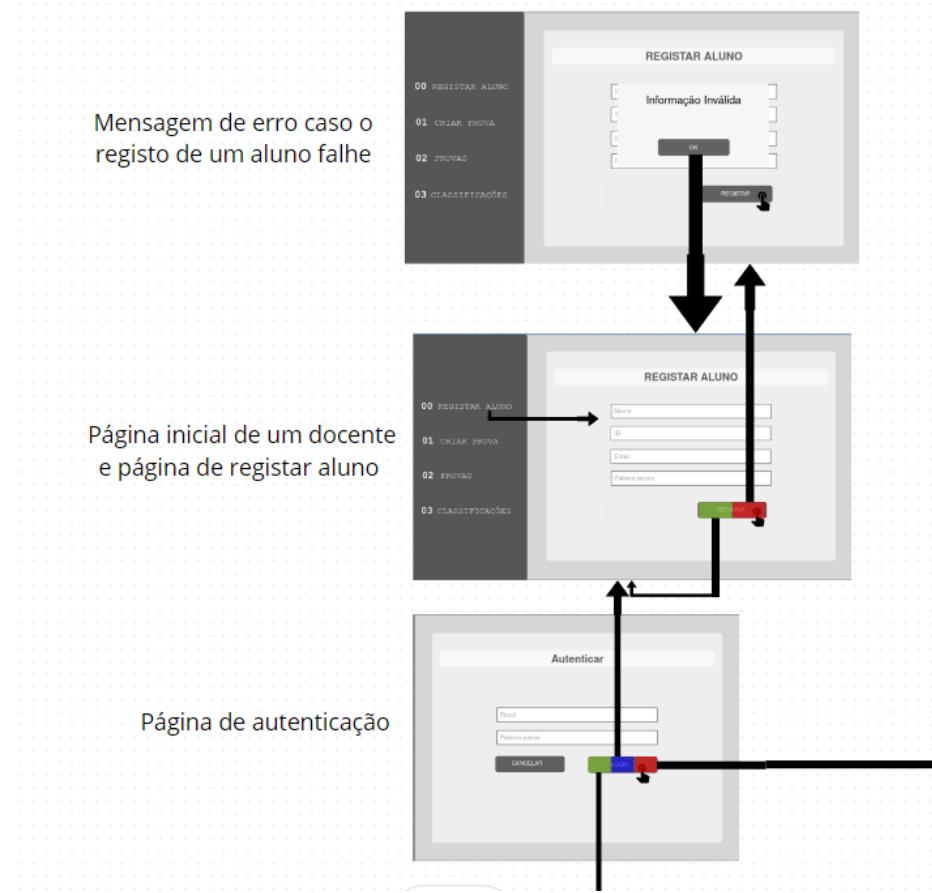


Figure 26: Navegação das paginas durante o registo de alunos

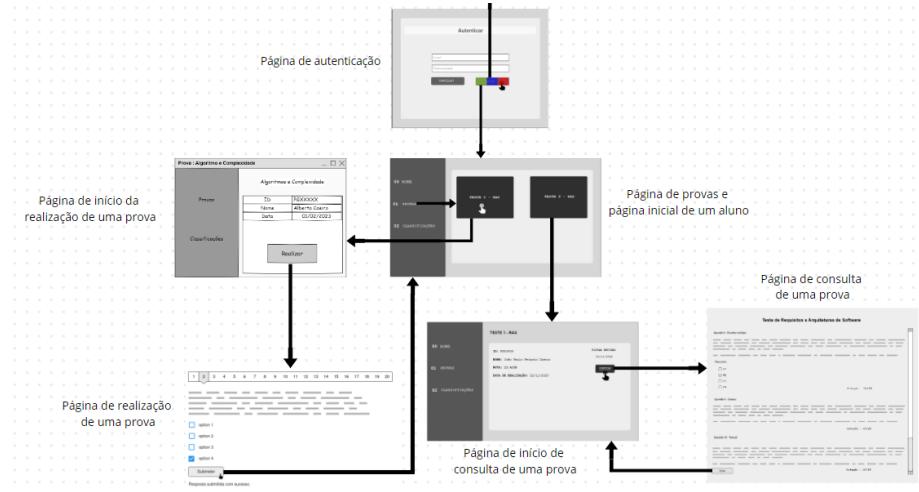


Figure 27: Navegação das paginas de realização de prova

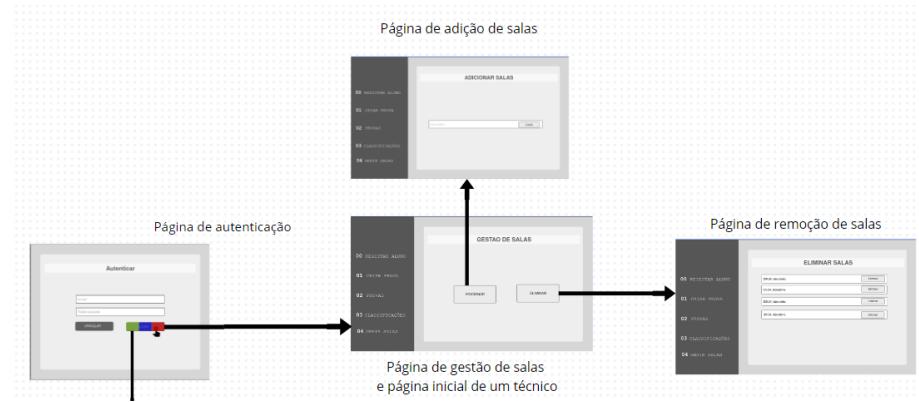


Figure 28: Navegação em paginas de gestão de salas

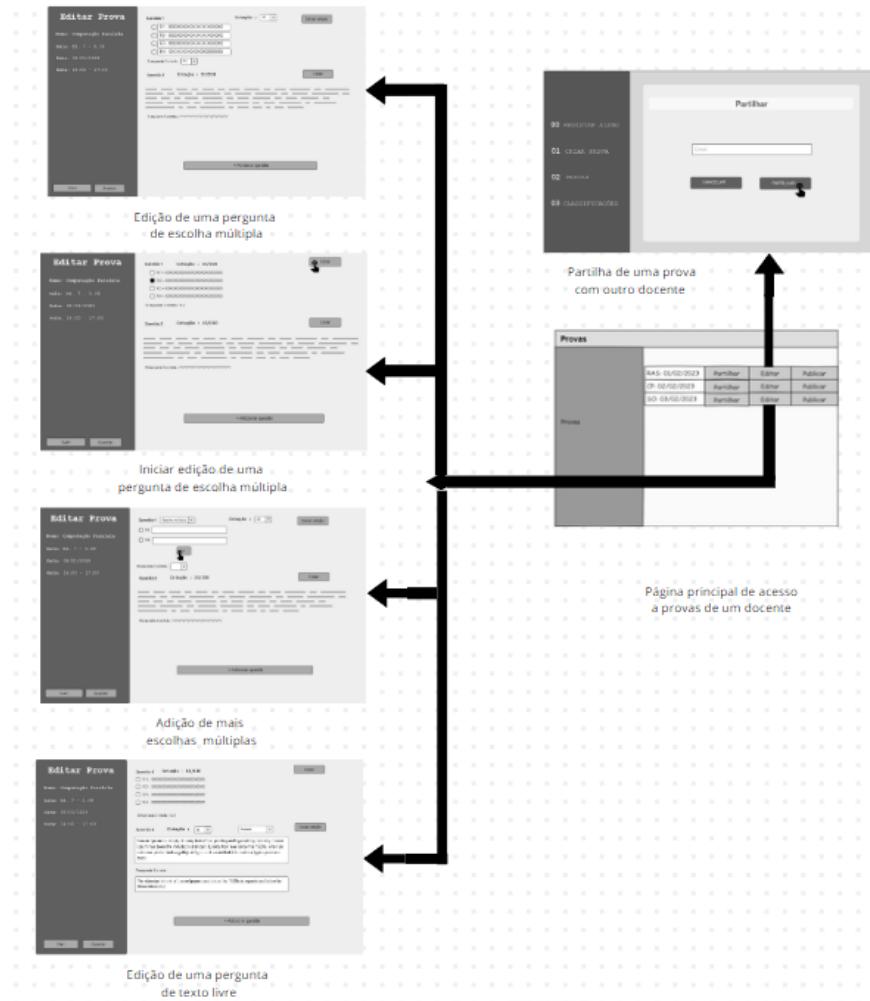


Figure 29: Navegação em páginas de edição e partilha de provas

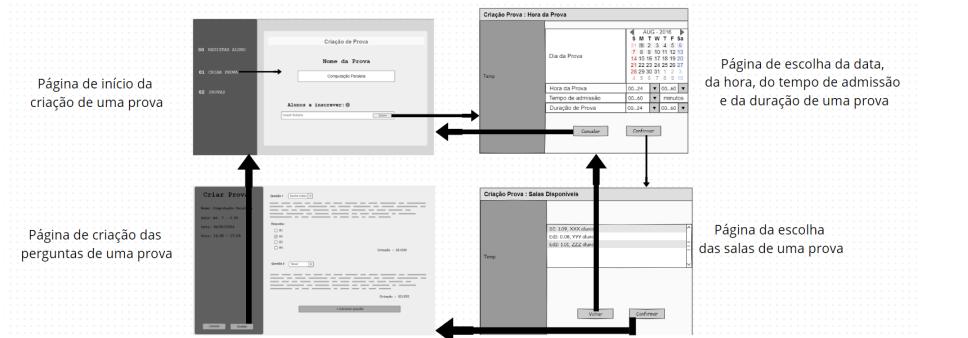


Figure 30: Navegação em paginas criação de provas

Anexos

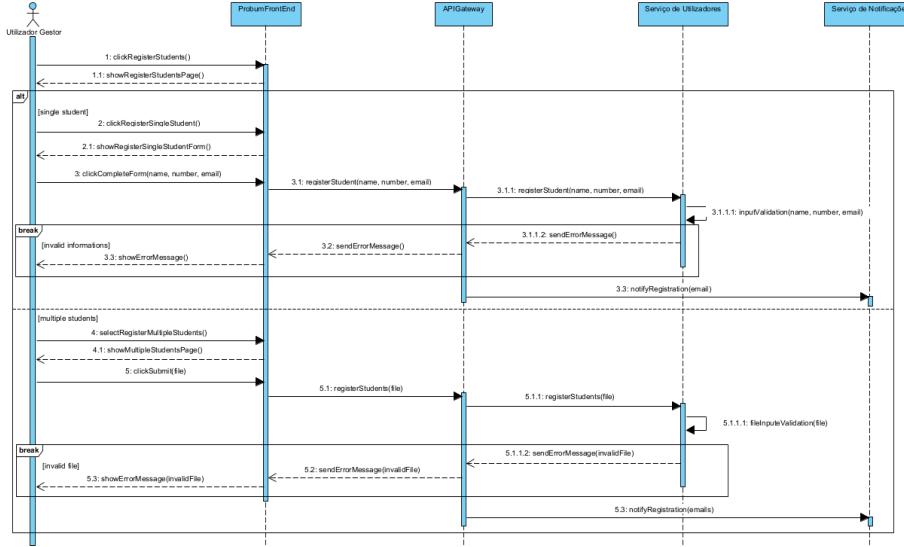


Figure 31: Diagrama de sequência para register alunos

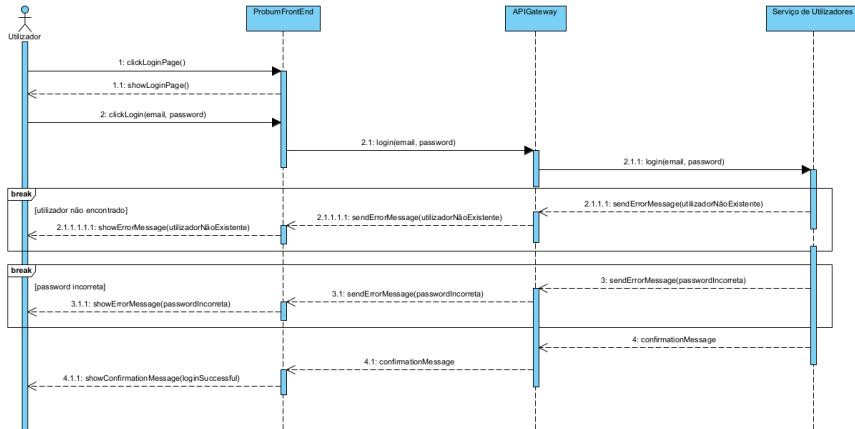


Figure 32: Diagrama de sequência para autenticar

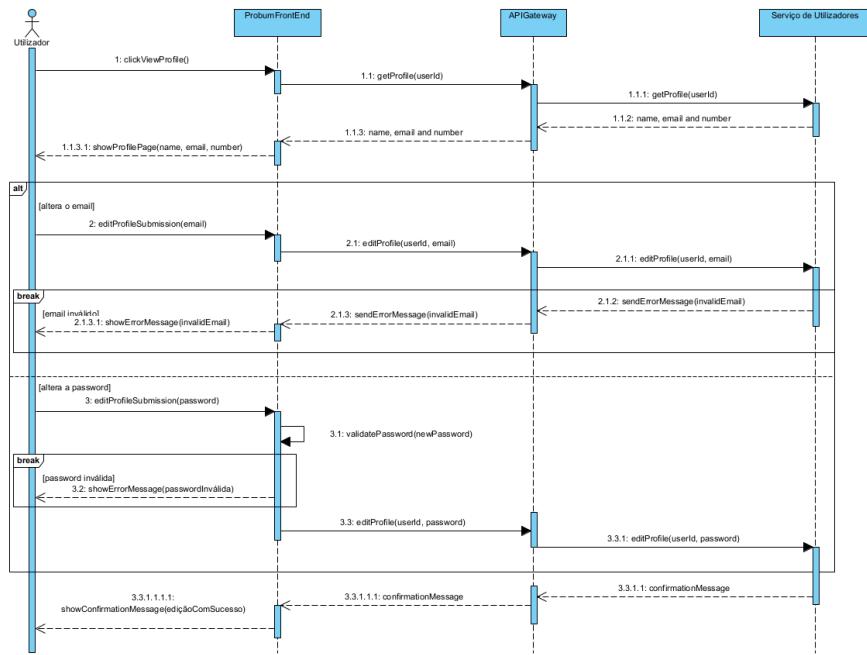


Figure 33: Diagrama de sequência para editar perfil

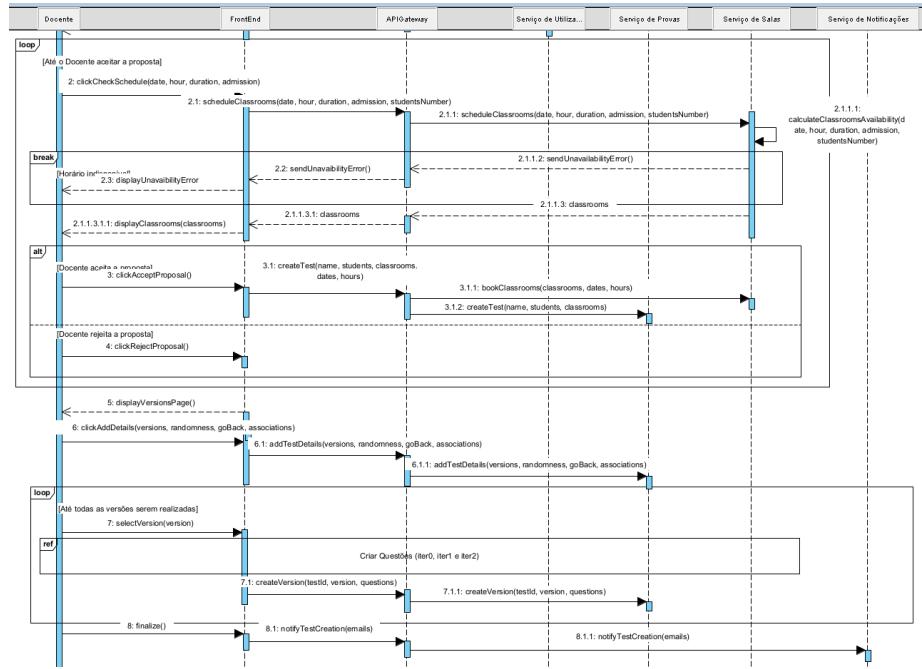


Figure 34: Diagrama de sequência para criar uma prova

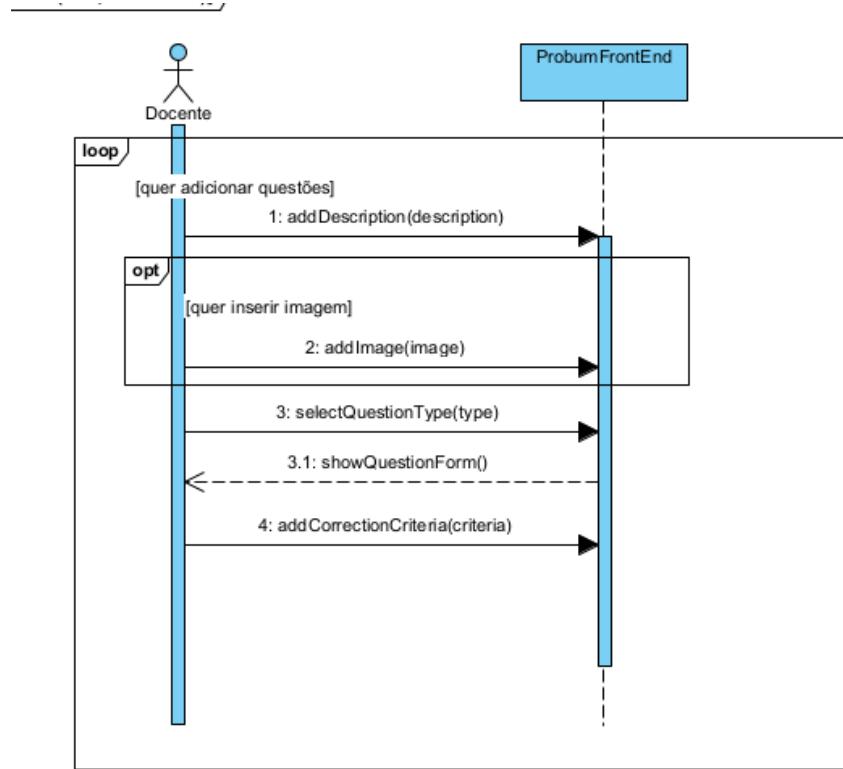


Figure 35: Diagrama de sequênci para criar questões

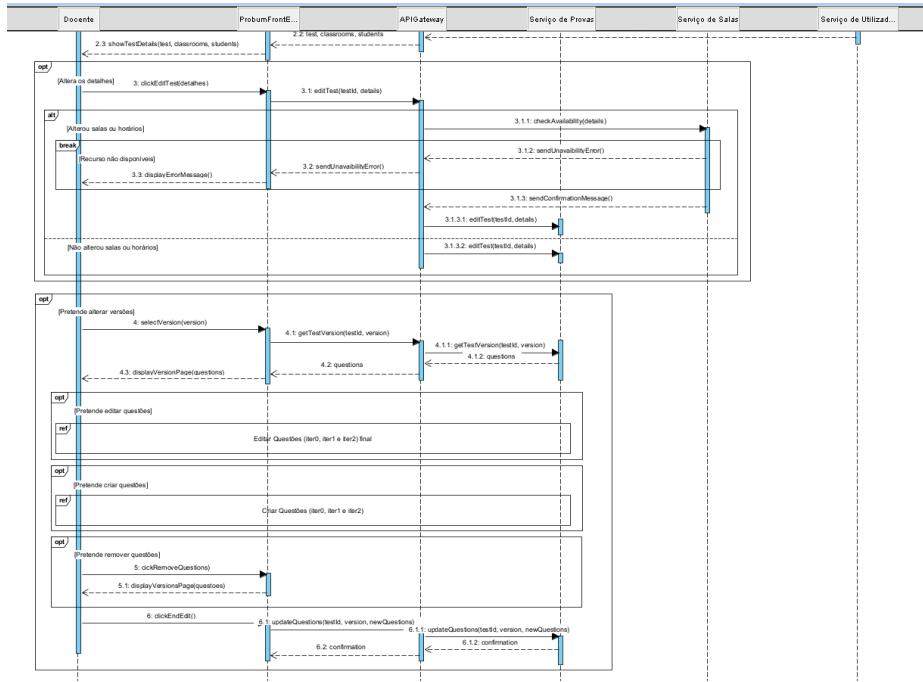


Figure 36: Diagrama de sequência para editar uma prova

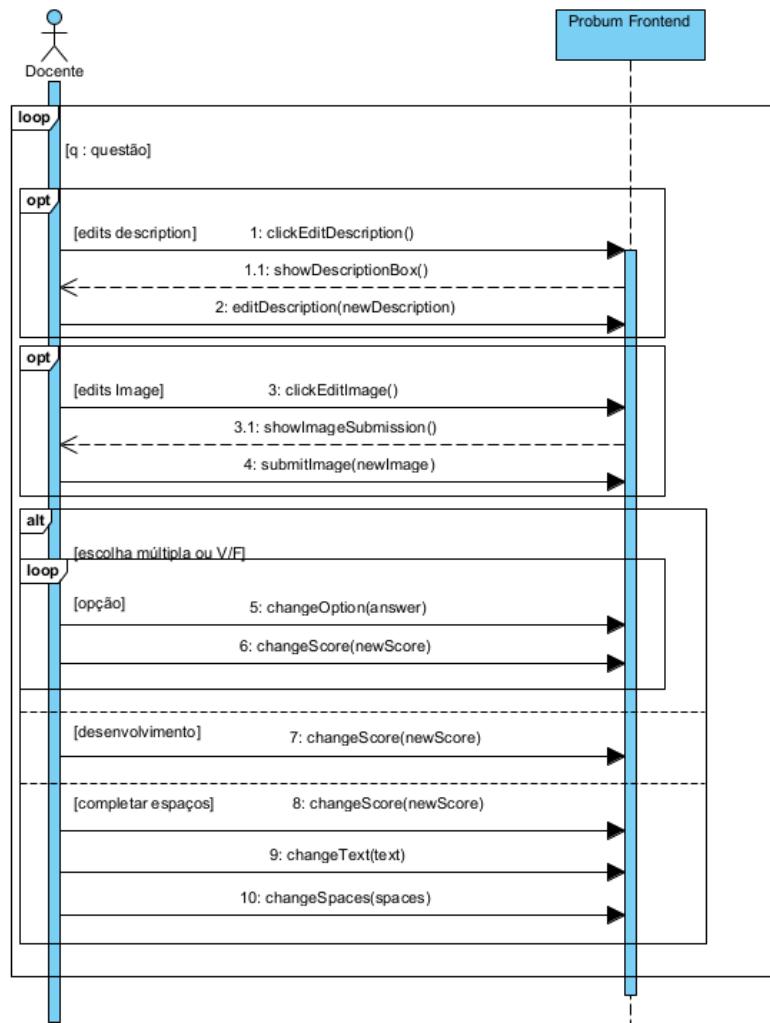


Figure 37: Diagrama de sequência para editar questões

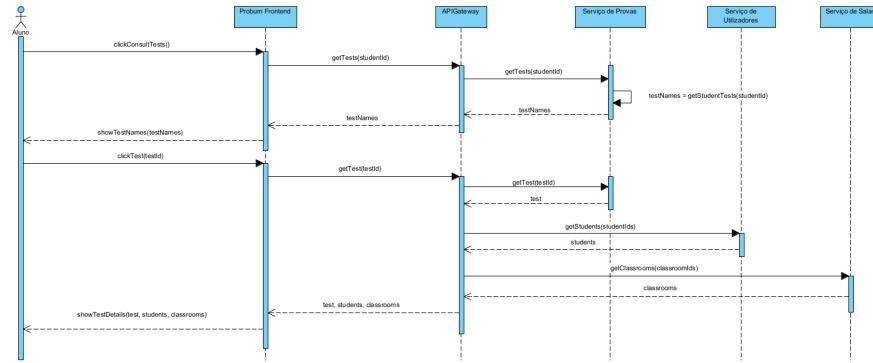


Figure 38: Diagrama de sequência para consultar detalhes de uma prova

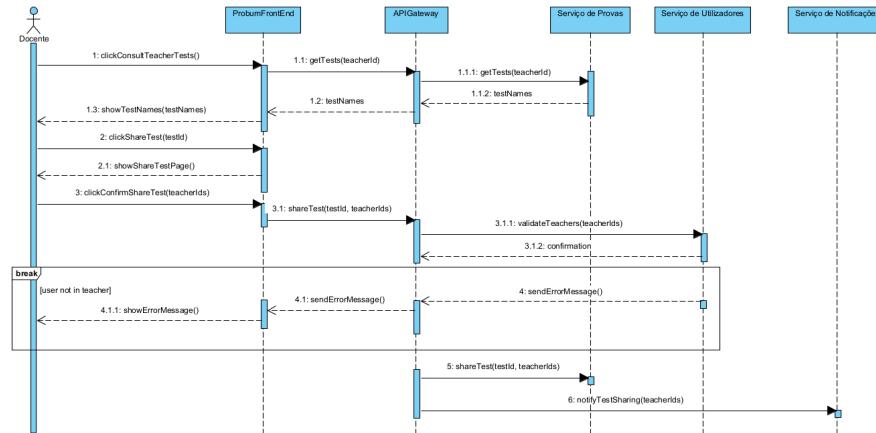


Figure 39: Diagrama de sequência para partilhar prova

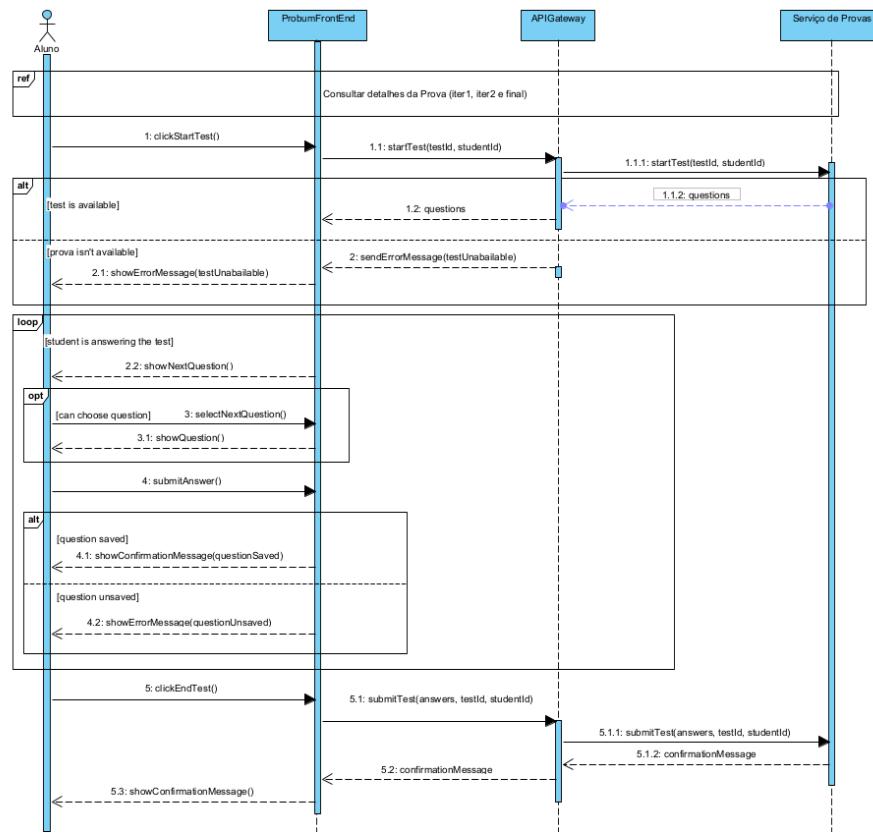


Figure 40: Diagrama de sequência para responder a prova

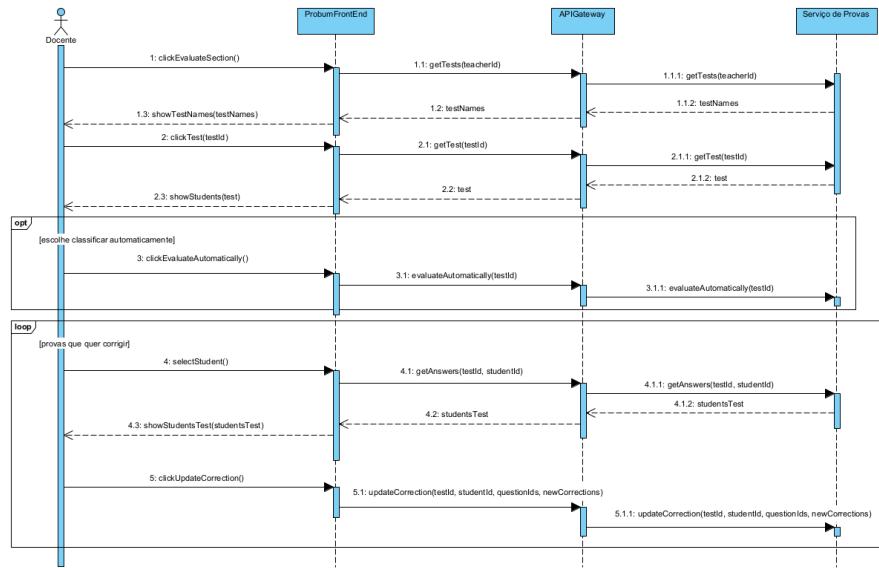


Figure 41: Diagrama de sequência para classificar respostas

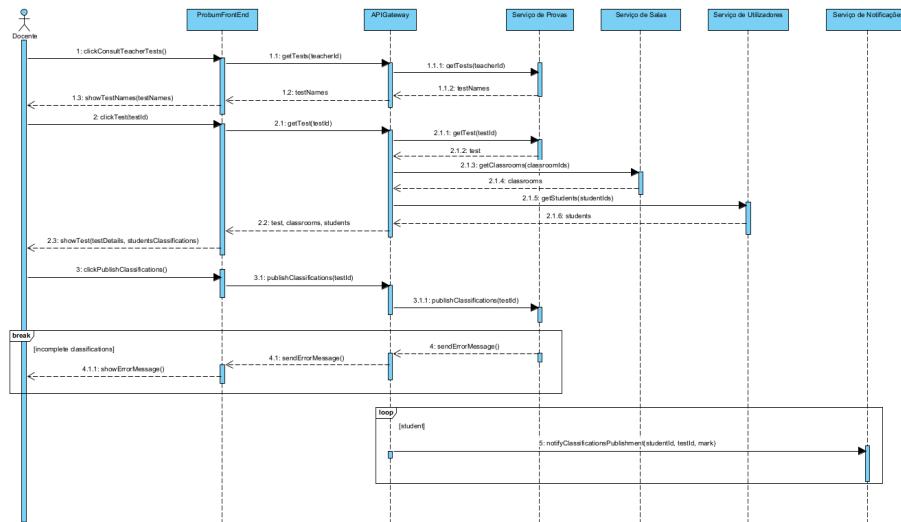


Figure 42: Diagrama de sequência para publicar classificações

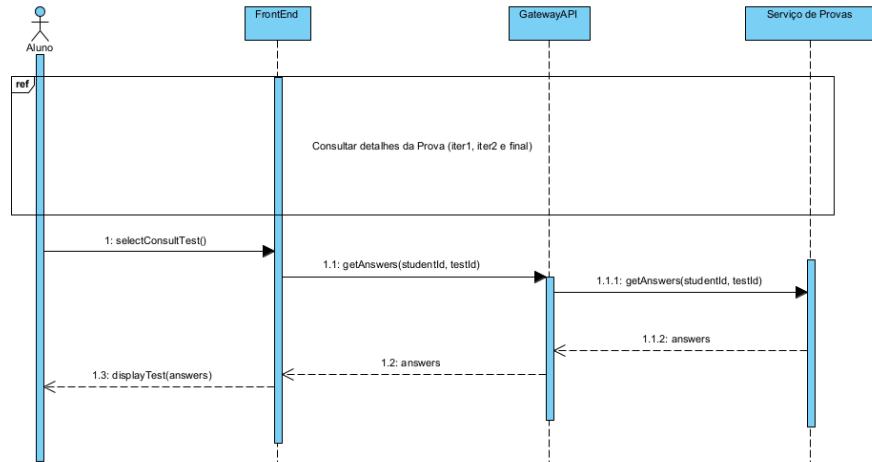


Figure 43: Diagrama de sequência para consultar prova corrigida

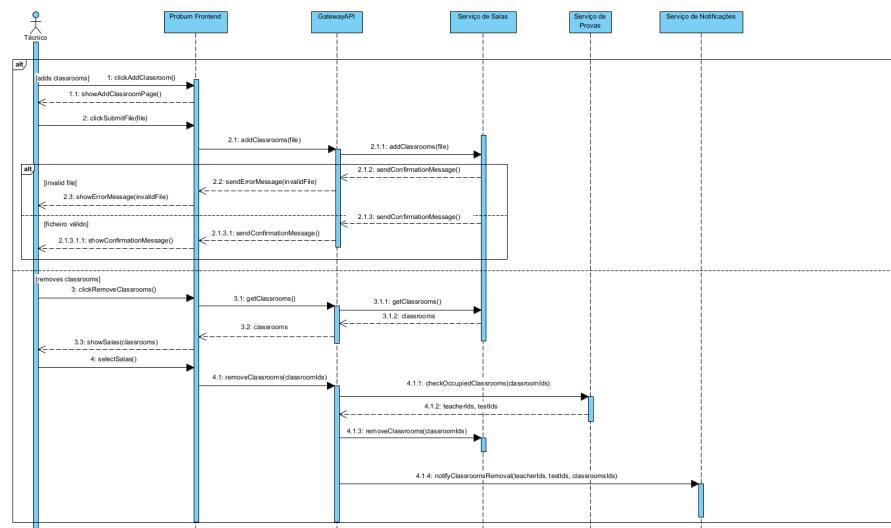


Figure 44: Diagrama de sequência para gerir salas

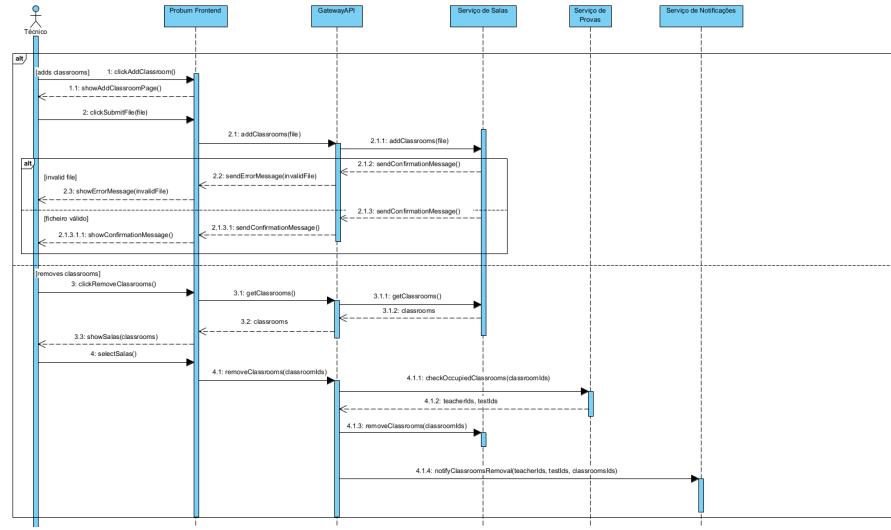


Figure 45: Diagrama de sequência para aceder a notificações