Universidade do Minho
Departamento de Informática

Mestrado Engenharia Informática
Requisitos e Arquiteturas de Software

**Grupo 4-B / Entrega 3**

PG53975 José Carvalho
PG54133 Paulo Oliveira
PG52667 David Matos
PG54177 Ricardo Oliveira
PG53597 Afonso Bessa
PG54708 Sérgio Ribeiro
PG53924 João Loureiro
A95323 Henrique Fernandes
PG52702 Ricardo Fonseca
PG54022 Luís Fernandes
PG50709 Ricardo Gama

# Prefácio

Acreditamos que o trabalho poderia ter corrido melhor. Sentimos que a descoordenação é o ponto a destacar e talvez o principal motivo para que a nossa solução não esteja completa. No entanto, achamos que se tivéssemos mais algum tempo, iríamos resolver as lacunas do nosso projeto. Do ponto de vista arquitetural, achamos que desenvolvemos uma arquitetura bastante boa, do ponto de vista da eficiência.

| **Membro da Equipa** | **Componente de Desenvolvimento** | **Contribuição (% da implementação)** | **Avaliação de Pares Global** |
|---|---|---|---|
| PG53975 | Correção-Prova+API-Gateway | 80%+20% | 0 |
| PG54133 | Utilizadores | 100% | 0 |
| PG52667 | Recursos | 100% | 0 |
| PG54177 | WebApp/Front-End | 100% | 0 |
| PG53597 | Realizar-Prova | 50% | 0 |
| PG54708 | Correção-Prova | 20% | 0 |
| PG53924 | Provas | 50% | 0 |
| A95323 | API-Gateway | 80% | 0 |
| PG52702 | Notificações | 100% | 0 |
| PG54022 | Provas | 50% | 0 |
| PG50709 | Realizar-Prova | 50% | 0 |

# Conteúdo

# Lista de Figuras

# 1 Introdução

Neste relatório, iremos detalhar as alterações efetuadas à arquitetura proposta na segunda fase, embora não haja nada de significativo a assinalar.

Recordamos que optámos por utilizar micro-serviços como base da arquitetura. Na fase 2, identificámos seis micro-serviços e continuamos a mantê-los, sendo eles:

- **Micro-Serviço Utilizadores**
  Elaborado pelo aluno PG54133
- **Micro-Serviço Recursos**
  Elaborado pelo aluno PG52677
- **Micro-Serviço Correção Prova**
  Elaborado pelos alunos PG53975 e PG54708
- **Micro-Serviço Notificações**
  Elaborado pelo aluno PG52702
- **Micro-Serviço Provas**
  Elaborado pelos alunos PG53924 e PG54022
- **Micro-Serviço Realização de Provas**
  Elaborado pelos alunos PG53597 e PG50709

# 2 Tecnologias

As tecnologias finais que adotámos foram:

- ***MySQL***
- ***Python***
- ***JavaScript***
- ***Docker***
- ***Postman***

O ***MySQL*** foi usado como motor de base de dados em todos os micro-serviços. A escolha do ***MySQL*** foi fundamentada na familiaridade prévia de todos os elementos do grupo com esta tecnologia.

Adicionalmente, o ***Python*** foi adotado em todos os micro-serviços, com a exceção do micro-serviço Recursos que utilizou ***JavaScript***. A opção pelo *Python* deve-se à sua simplicidade e fácil acesso à documentação.

Cada micro-serviço foi encapsulado em *containers* ***Docker***, facilitando assim o processo de inicialização da aplicação e proporcionando um ambiente isolado para cada micro-serviço.

O ***Postman*** foi utilizado de maneira abrangente por todos os micro-serviços, com o objetivo de verificar a correta operacionalidade dos mesmos.

# 3 Arquitetura Final

## 3.1 Front-End

A Front-End do nosso sistema foi implementada na forma de uma WebApp desenvolvida em Python. Para disponibilizar as rotas e estruturar o código de melhor forma foi utilizada a biblioteca Flask. Quanto às views foram criados templates com recurso a Jinja2. Esta foi desenvolvida por Ricardo Oliveira (PG54177).

### 3.1.1 Arquitetura

A arquitetura da aplicação web é notavelmente simples, composta por uma única classe responsável pela comunicação entre o utilizador e a gateway, para que estes possam aceder as funções do backend de forma controlada. Esta classe desempenha um papel central na entrega das views aos utilizadores, proporcionando uma interação com o sistema desenhada para focar num design simples e intuitivo. Essa abordagem promove uma comunicação clara e direta entre a aplicação e a gateway, facilitando a manutenção e compreensão do código, mantendo ofuscados os micro-serviços do exterior.

A simplicidade da arquitetura também contribui para uma implementação ágil de novas funcionalidades, permitindo que a sua escalabilidade seja mais eficiente e possibilitando replicação.

### 3.1.2 Controlo de Permissões

O controlo de permissões na aplicação é crucial para garantir a segurança e a privacidade dos utilizadores. Isto é especialmente crítico para garantir que os estudantes não têm acesso às provas de antemão. Foram implementados mecanismos robustos de autenticação e autorização, utilizando decoradores nas rotas específicas para verificar o tipo de utilizador sempre que este acede através de um token gerado durante o login. Esta abordagem garante uma separação forte entre as rotas do técnico, estudante e docente.

Isto não só reforça a segurança, mas também proporciona uma experiência personalizada, direcionada às necessidades específicas de cada utilizador. Isto significa que cada um dos utilizadores tem uma "landing page"personalizada para as suas necessidades. O cuidado na implementação do controlo de permissões contribui para a confiabilidade e robustez do sistema como um todo.

### 3.1.3 Rotas

Além das rotas planeadas inicialmente, o nosso sistema apresenta ainda rotas de erro, oferecendo uma abordagem abrangente para lidar com situações inesperadas, mesmo para utilizadores sem um conhecimento técnico elevado.

Assim as rotas implementadas nesta componente correspondem às mesmas do API-Gateway com adição das mencionadas acima.

A documentação das várias rotas através do Swagger simplificou o processo de implementação, compreensão e a integração com a aplicação. Essa abordagem facilita o desenvolvimento futuro, permitindo uma adaptação mais rápida a novos requisitos e mudanças na lógica de negócios.

A flexibilidade na gestão de rotas contribui para uma aplicação mais resistente a situações adversas, proporcionando uma experiência do utilizador mais consistente. As rotas de erro demonstram uma consideração cuidadosa para lidar com situações imprevistas, garantindo uma experiência mais robusta mesmo em condições menos ideais.

### 3.1.4 Views

As *views* desempenham um papel central na interação dos utilizadores com a aplicação, sendo implementadas de maneira eficaz para apresentar informações de forma clara e concisa. De modo a responder às constantes mudanças de informação foram criados cerca de 20 *templates* de *Jinja2* que são utilizados dinamicamente aquando do render da rota desejada. Para complementar o HTML foi ainda criado um ficheiro de *CSS* de modo a uniformizar o design da aplicação.

A implementação cuidadosa das *views* contribui para a usabilidade geral da aplicação, minimizando a complexidade e focando na entrega eficaz de informações. A atenção aos detalhes na apresentação visual e na disposição dos elementos cria uma experiência coesa e agradável para os utilizadores, principalmente em ambientes de teste onde o foco é a componente principal e por isso foram removidos todos os componentes acessórios.

As imagens de algumas das *views* implementadas encontram-se na secção 7.1.

### 3.1.5 Problemas

Durante o desenvolvimento, surgiram algumas dificuldades, principalmente a nível de testes já que se trata de uma componente altamente dependente do resto do sistema. A abordagem para ultrapassar este problema surgiu na forma de um micro-servidor de dados estático que simula rotas estáticas a fim de testes. Esta atitude pro-ativa resultou numa aplicação mais estável e robusta, permitindo testar casos que de outra forma não seria possível.

Um outro problema, ainda que menor, veio na forma do alinhamento de mensagens entre o Front e BackEnd. Isto e, com mudanças ao longo do desenvolvimento foi necessário alterar em varias instâncias as variáveis utilizadas quer

na aplicação web, quer nos templates criados. Esta dificuldade foi no entanto ultrapassada com sucesso graças a uma boa comunicação com os membros da equipa.

Ainda assim, este debito de funcionalidades de micro-serviços impede o funcionamento de varias rotas num sistema que de facto utilize a API-Gateway e os respetivos micro-serviços. Esta foi uma dificuldade que apesar do esforço não foi ultrapassada a 100% tendo as suas funcionalidades limitadas.

### 3.1.6 Análise Crítica

Apesar da implementação eficaz, há oportunidades para melhorias na aplicação, principalmente, como referido anteriormente, no que toca as rotas que acabam por não estar disponível por défices de data provenientes de funções do backend.

Quanto a implementação isolada da interface, foi possível implementar todas as *views* necessárias para o utilizador, bem como a gestão/correção de testes e *landing page* dos docentes e ainda a gestão dos utilizadores por parte do técnico. Em termos gerais a interface conseguiu expandir em áreas inicialmente não previstas, como as paginas iniciais de cada tipo de utilizador, *display* de erros, etc. mas apresenta défices em rotas que foram consideradas como de menor importância nesta fase (como e o caso das Views do utilizador Técnico).

Conclui-se assim que a FrontEnd foi implementada com um nível de sucesso esperado a quando do inicio do desenvolvimento e cumpriu, em grande parte, os requisitos não funcionais propostos.

## 3.2 API Gateway

O **API Gateway** é responsável por gerir as várias operações a que os utilizadores da aplicação, incluindo alunos, docentes e técnicos, tem acesso. A Gateway interage com todos os Micro-Serviços da aplicação.

### 3.2.1 API

A API nesta fase do trabalho sofreu leves alterações devido a pequenas alterações em outros micro serviços e adição de funcionalidades.

#### Utilizador

No user separou-se as rotas de criação de users em duas, uma para criação de alunos e outra de Docentes.


Figura 1: API Utilizadores

**Recursos**

Nos recursos não houve necessidade de alteração.

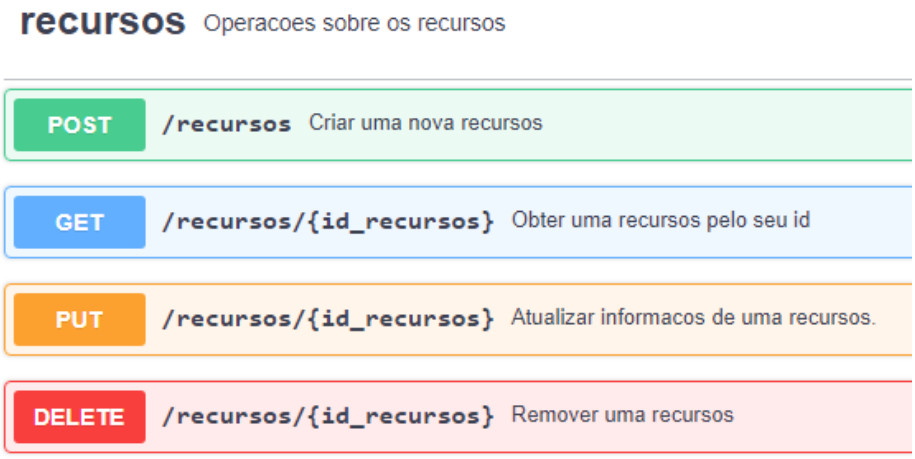


Figura 2: API Recursos

**Prova**

Acrescentamos novas rotas que achamos interessantes,uma delas obtém todos os resultados(notas) de um aluno outra para obter todos os testes futuros de um aluno e para alem destas temos uma para obter todos os testes de um aluno e outra para um docente.

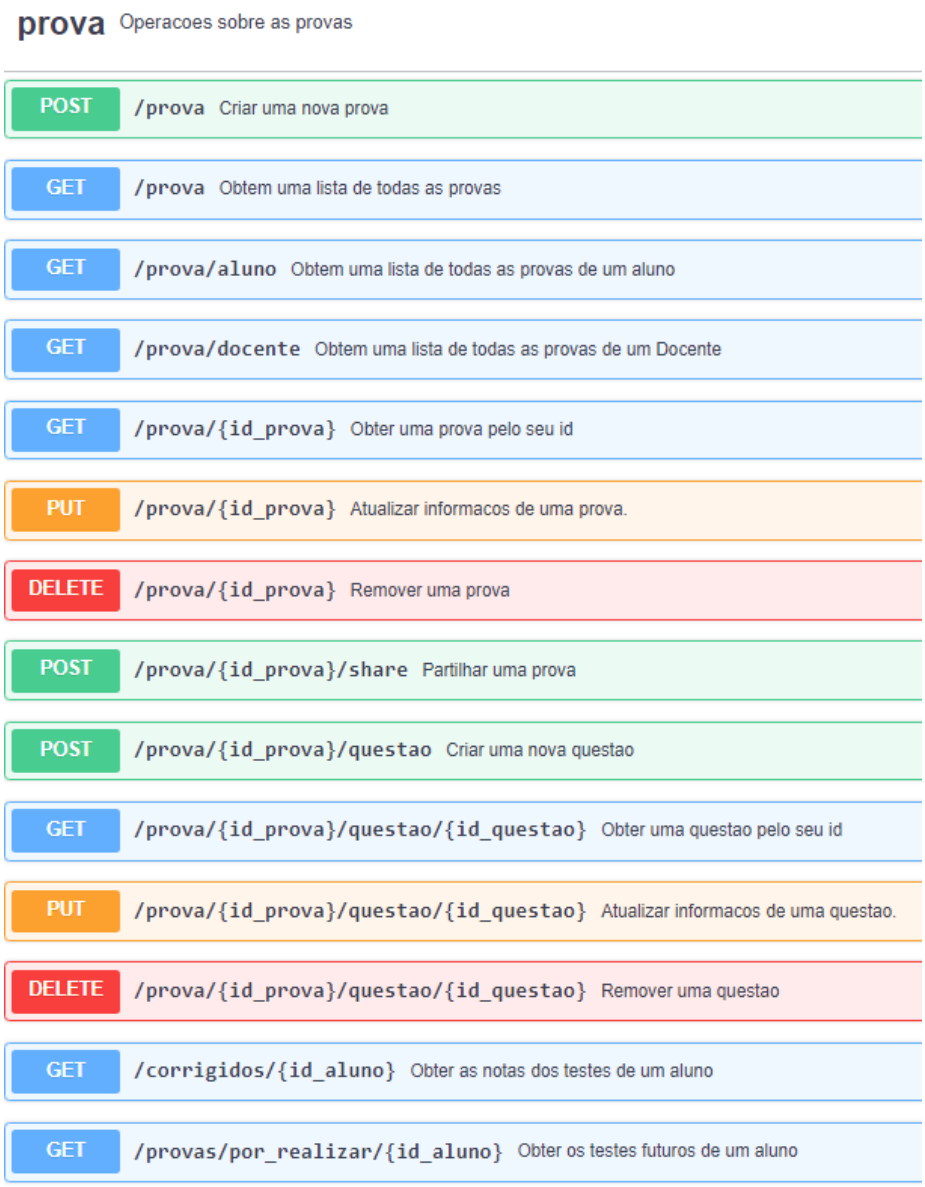


Figura 3: API Prova

### Realização de Provas

Para a realização de provas não foi preciso fazer modificações.

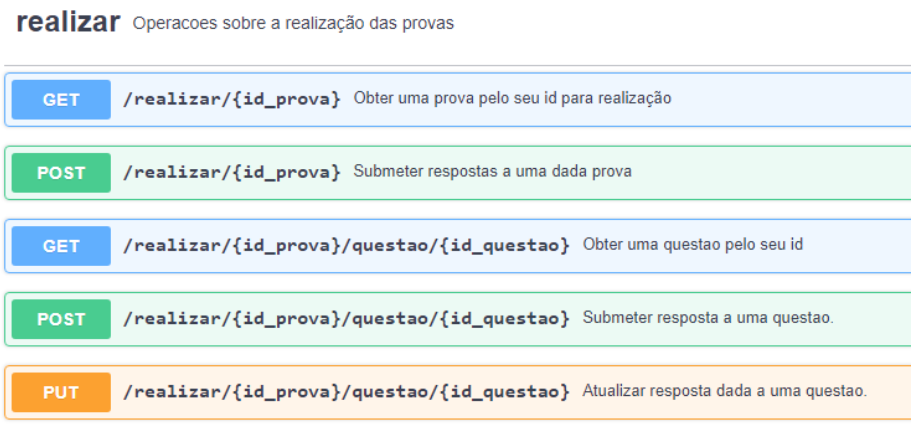


Figura 4: API Realização de Provas

### Correção de Provas

Na correção de provas foi preciso mudar o nome de algumas rotas mas as funcionalidades continuam as mesmas.

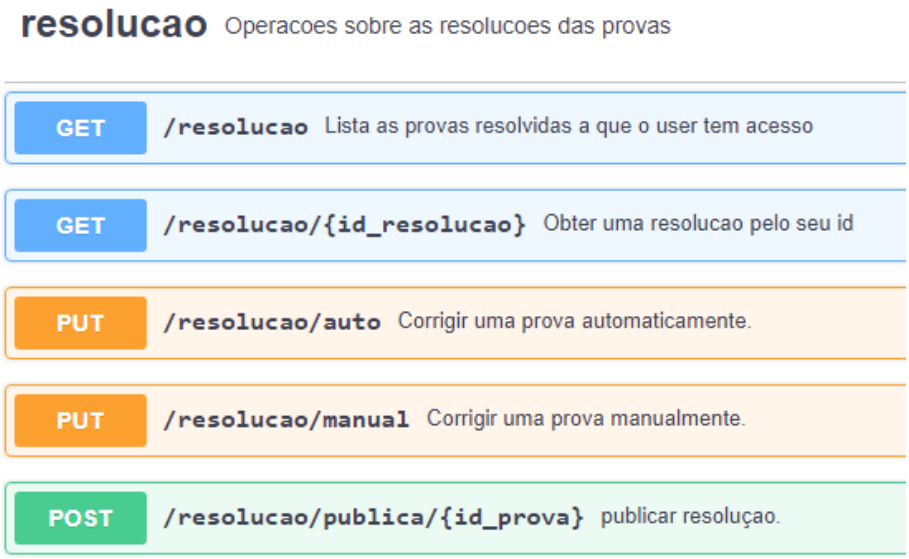


Figura 5: API Correção de Provas

**Notificações**

Nas notificações foi os preciso mudar o nome da rota mas a funcionalidade continuam a mesma.

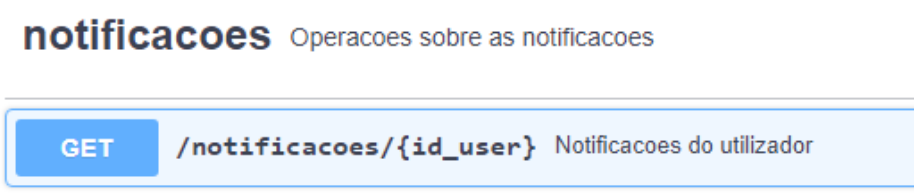


Figura 6: API Notificações

### 3.2.2 Arquitetura

Desde a última fase, a arquitetura experimentou mudanças subtis, concentrando-se em ajustes específicos em determinadas funções que ainda faltavam ou que tiveram de ser modificada devido a alterações de outros micro serviços. No final, este serviços contém apenas 3 ficheiros, sendo que o _init.py é para inicializar o serviço, a routes.py é o ficheiro que contém as rotas e por último o ficheiro script.py é usado para contactar os micro-serviços.

## 3.3 Micro-Serviço Utilizadores

Este micro-serviço não foi muito alterado ao longo do desenvolvimento do projeto pela sua estruturação inicial não ter sido modificada, tendo ficado as estruturas da API, da Arquitetura e da Base de Dados inalteradas.

### 3.3.1 API

A API deste micro-serviço não sofreu assim nenhuma alteração tendo-se mantidas as rotas pretendidas para o seu funcionamento.

A versão final da API do **Micro-Serviço Users** é a seguinte:

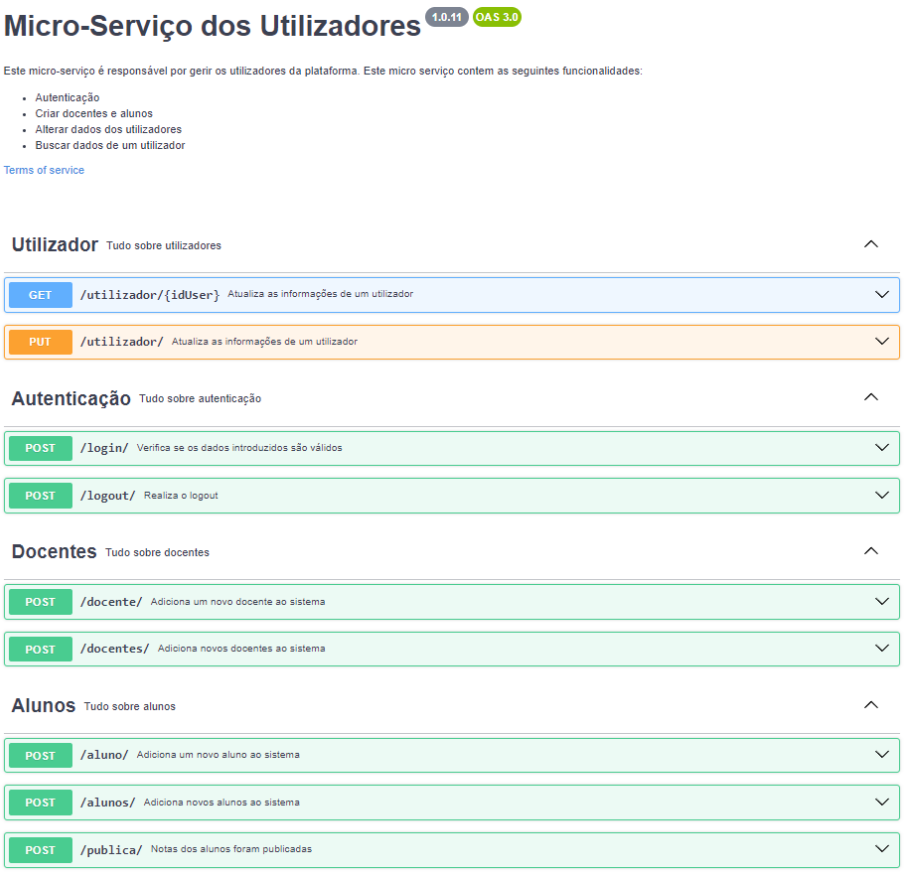


Figura 7: API - Users

### 3.3.2 Arquitetura

Quanto à arquitetura, esta também não sofreu significativas alterações neste micro-serviço, tendo sido acrescentados uma função para estabelecer conexão en-

tre serviços como "relayNotificationsMicroservice(students_emails)"e o atributo "user_type"por forma a distinguir os utilizadores de aluno e docente. Foram alterados os nomes das funções e de alguns atributos por forma a facilitar a sua compreensão. Com isto, ficamos então com as classes dos users e do sistema que tratam de todas as operações necessárias para a realização dos requisitos funcionais apresentados previamente. O módulo ContactDB é um módulo que pode ser visto como um DBO cujo objetivo é o acesso à base de dados deste micro-serviço.

A versão final da Arquitetura do **Micro-Serviço Users** é a seguinte:

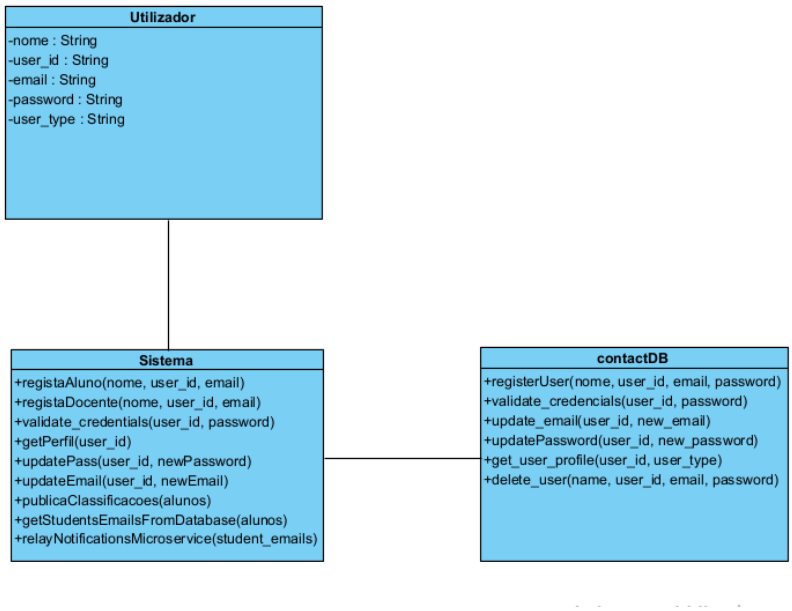


Figura 8: Arquitetura - Users

### 3.3.3 Problemas

Este micro-serviço não encontrou qualquer tipo de problemas.

### 3.3.4 Análise Crítica

Este micro-serviço ficou então operacional desempenhando as funcionalidades esperadas.

## 3.4 Micro-Serviço Recursos

Este micro-serviço sofreu poucas alterações, no entanto, a única alteração relevante ocorreu na API.

### 3.4.1 API

A alteração foi feita no método GET no contexto das salas; a rota anterior */recursos/salas/{id}/reservas* agora passou a ser */recursos/salas/{id}*. Isto simplificou consideravelmente a implementação desse método em JavaScript.

A versão final da API do **Micro-Serviço Recursos** é a seguinte:


Figura 9: API - Recursos

### 3.4.2 Arquitetura

A arquitetura mantém-se igual à exceção do nome de um atributo que mudou de "id_salas"para "id_sala". Uma reserva corresponde à alocação de uma ou várias salas para a realização de uma prova. Um docente que pretenda reservar duas salas para a realização de uma prova deverá efetuar duas reservas distintas, uma para cada sala.

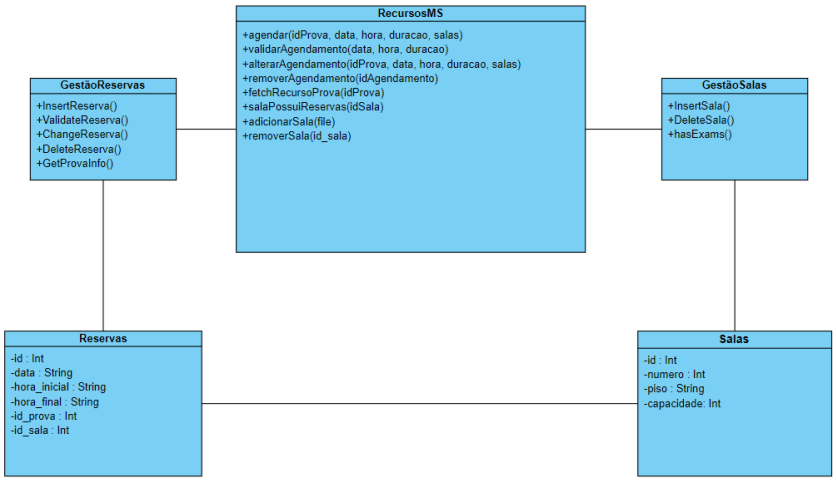


Figura 10: Diagrama Estrutural - Recursos

Para uma melhor organização do micro-serviço, procedeu-se à sua subdivisão em três subconjuntos, cada um contendo código em JavaScript associado aos contextos das salas e das reservas. Estes contextos podem ser considerados análogos entre si.

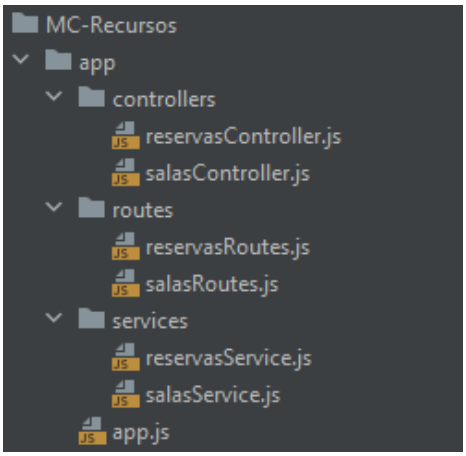


Figura 11: Estrutura do Projeto - Recursos

**Routes**

Encarregues de receber pedidos (testado com Postman), e de os mapear aos correspondentes controladores.

**Controllers**

Os controladores verificam a integridade da informação fornecida pelo utilizador e encaminham essa informação para os serviços. Em caso de erro, tanto por parte do utilizador como do serviço, são os controladores que fornecem informações sobre esse erro.

**Services**

Os serviços utilizam *Prisma* para realizar todas as transações necessárias com as bases de dados.

### 3.4.3 Problemas

As dificuldades previstas na implementação do micro-serviço estavam primariamente associadas ao uso da linguagem SQL para fazer operações nas bases de dados. No entanto, a utilização de Prisma mitigou essas dificuldades ao permitir a utilização de JavaScript para realizar essas operações.

### 3.4.4 Análise Crítica

Este micro-serviço cumpre os requisitos #9, #15, #18, #30, #31, #65, #66, #67 mas não trata de listar as salas e horários indisponíveis (#16) e a calendarização das provas é feita pelo Docente e não pelo Sistema (#17). Com isso em consideração, o micro-serviço cumpre 80% dos requisitos a ele associados.

## 3.5 Micro-Serviço Correção Prova

Em relação ao **Micro-Serviço Correção Prova**, houve algumas alterações, tanto a nível da arquitetura, como da própria API, mas nada muito significante.

### 3.5.1 API

Em relação à API, a versão final deste micro-serviço conta com a API presente na seguinte imagem:

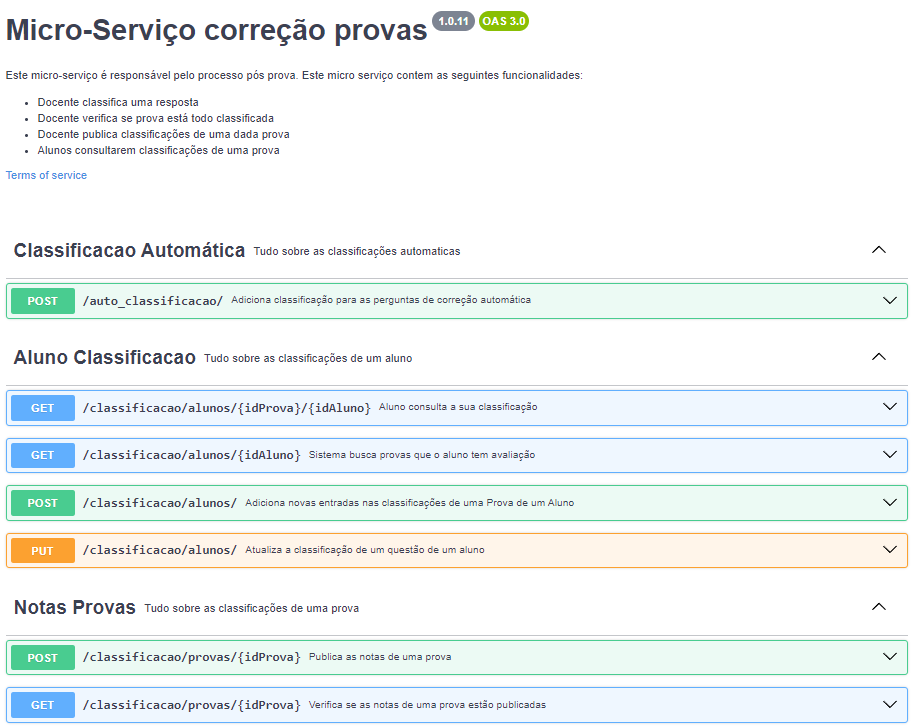


Figura 12: API - Correção Prova

### 3.5.2 Arquitetura

Em relação à arquitetura, o diagrama ilustrado na figura 13, representa a arquitetura final completa deste micro-serviço. Para simplificar aqui estão algumas notas da arquitetura:

- *ContactDB*, *ClassificacoesQuestoes*, *ClassificacoesProvas* e *app* são módulos e não classes.
- *ClassificacaoProva*, *CriaInstancias*, *ClassificacaoQuestao*, *Classificacao-Automatica*, *Questoes_Resposta* e *AlunoResposta* são classes e não módulos.

- Todas as classes são inicializadas retirando a informação *json* que é recebida pelo micro-serviço.
- O módulo *app* pode ser visto como o *facade* do micro-serviço.
- O módulo *ContactDB* é um módulo que generaliza os acessos à base de dados deste micro-serviço, podendo ser visto como sendo um *DBO*.
- Os módulos *ClassificacoesQuestoes* e *ClassificacoesProvas* utilizam o módulo *ContactDB* para aceder à base de dados.

A nível de alterações, adicionamos a classe *CriaInstancias* e detalhamos melhor quais as funções e atributos que cada módulo / classe tem.

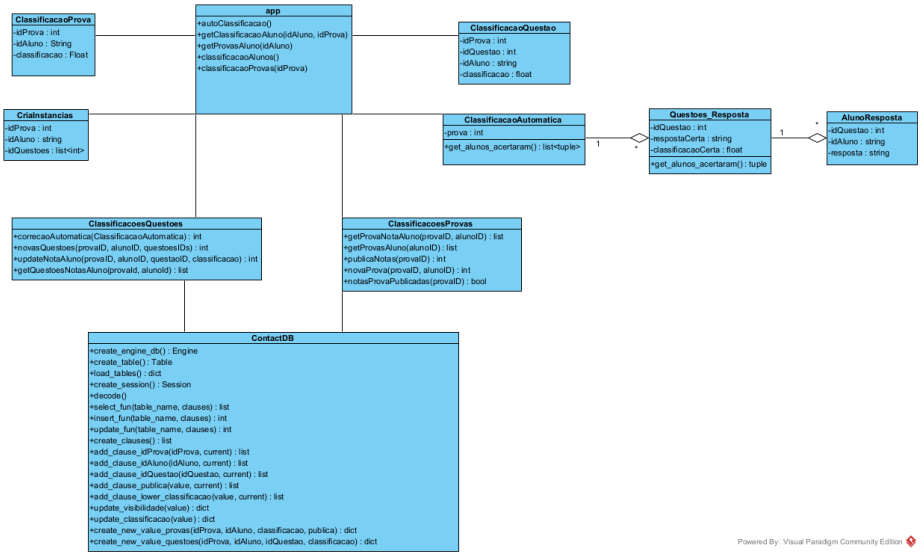


Figura 13: Arquitetura - Correção Prova

### 3.5.3 Base de dados

Em relação à base de dados, as tabelas mantém-se. O único aspeto a apontar é a criação de um *trigger*, que sempre que a tabela de classificação de questão é atualizada, a respetiva classificação na tabela prova também é atualizada adequadamente. Por exemplo, o Docente ao classificar a resposta de um aluno a uma questão de uma prova, a respetiva classificação da prova desse mesmo aluno irá alterar conforme a classificação do docente.

### 3.5.4 Problemas

No geral, o desenvolvimento deste micro-serviço correu sem qualquer dificuldades associadas. Isto adveio-se à facilidade com que encontramos a resolução

de erros que enfrentamos, bem como a facilidade de encontrar bibliotecas *Python* que podíamos usar para o projeto e que cuja utilização é bastante simples, sendo *sqlalchemy* um exemplo disso.

### 3.5.5 Análise Crítica

Todos os requisitos funcionais que necessitavam deste micro-serviço ficaram concluídos. Achamos que este micro-serviço esteja 100% completo e funcional tal como planeado.

## 3.6 Micro-Serviço Notificações

Este micro-serviço sofreu algumas alterações para uma melhor alteração tanto na API como ao nível da arquitetura.

### 3.6.1 API

A versão final da API do micro-serviço é a seguinte:


Figura 14: API - Notificações

Esta API sofreu alterações em relação com a anterior. O método *GET* manteve-se mas ao *POST* da rota **/notificacoes/** foi retirado o parâmetro **"userid"** pois não é necessário. Também foi adicionado o *GET* para as rotas **/notificacoes/read** e **/notificacoes/idNotificacao**, sendo a primeira rota para obter notificações que foram ou não lidas e a segunda rota sendo para mostrar apenas uma notificação buscando pelo seu ID.

### 3.6.2 Arquitetura

A arquitetura deste micro-serviço é estruturada da seguinte forma:

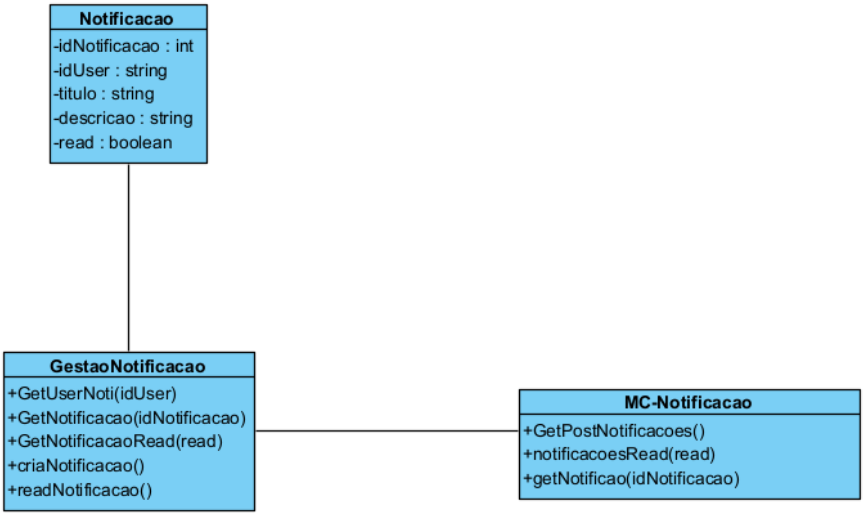


Figura 15: Arquitetura - Notificações

- Temos a classe *notificacoes* ao qual adicionamos o atributo titulo de tipo string
- Depois temos o módulo para obter a informação sobre as notificações, onde alterações foram feitas mas as funcionalidades continuam como descritas previamente.
- Por ultimo, para ficar semelhante à API mostrada na 14, houve alterações no aspeto dos métodos mas mantêm as mesmas funções.

### 3.6.3 Análise Crítica

Apesar das alterações no projeto e as suas dificuldades na implementação, o micro-serviço de notificações ficou assim concluído.

## 3.7 Micro-Serviço Provas

Em relação ao **Micro-Serviço Prova**, efetuaram-se algumas alterações a nível de arquitetura e da sua API, sendo a maioria destas para retificar alguns lapsos de lógica da fase anterior que foram notados aquando o desenvolvimento, não tendo por isso havido alterações significantes que alteram o seu funcionamento.

### 3.7.1 API

Quanto à API, foram acrescentadas algumas rotas chave que faltavam na versão anterior, como, por exemplo, uma rota para obter as respostas de perguntas de correção automática de uma prova, bem como a eliminação das rotas **DELETE**, que revelaram-se desnecessárias.

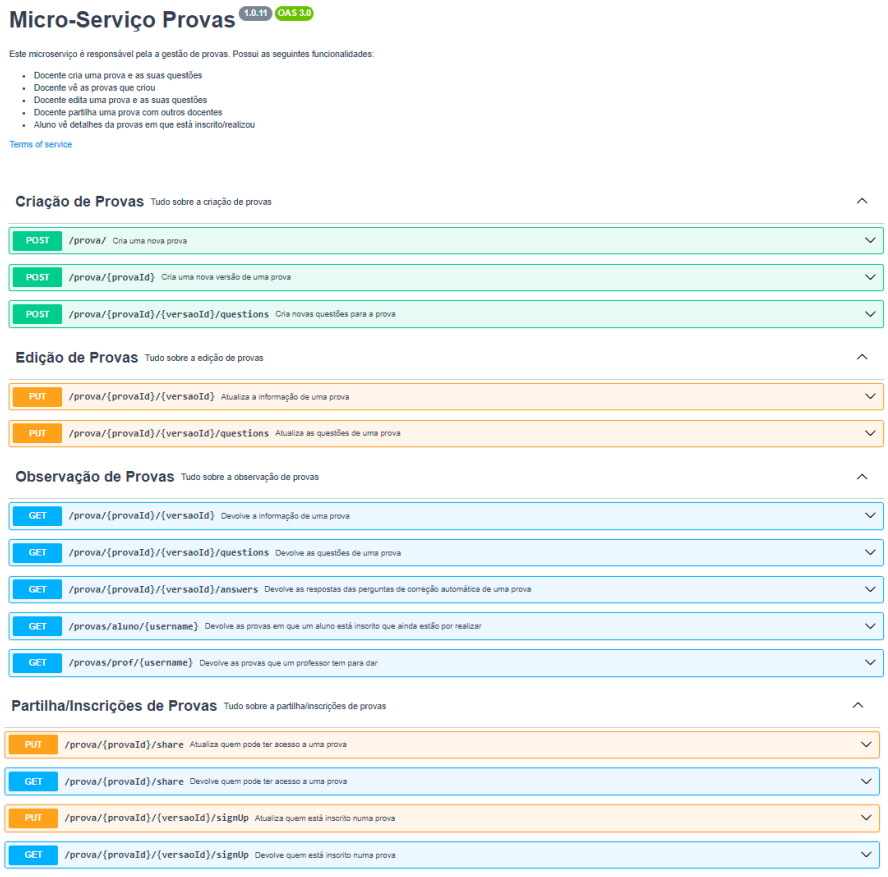


Figura 16: API - Provas

### 3.7.2 Arquitetura

O diagrama da figura 17 representa a arquitetura final do micro-serviço das provas. Seguem-se então algumas notas sobre ela:

- Todas as entidades são classes utilizadas para auxiliar nos acessos à base de dados.
- Todas as classes são inicializadas retirando a informação *json* que é recebida pelo micro-serviço.
- Adicionou-se uma classe **Espaco** que representa um espaço de uma pergunta de espaços, que contem um identificador, o número do espaço na pergunta, o seu valor/cotação, o texto correto e o *id* da questão a que está associado. Esta alteração permitiu facilitar o desenvolvimento.

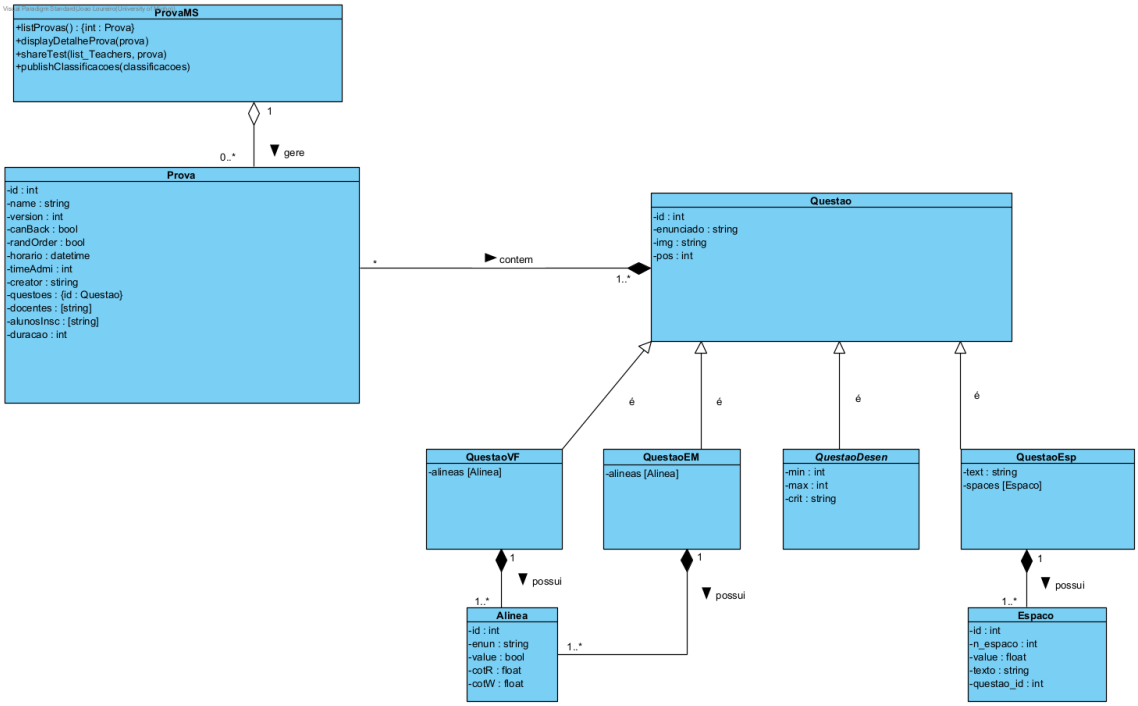


Figura 17: Arquitetura - Provas

Dividimos a estrutura deste projeto em vários módulos *python* de forma a facilitar o desenvolvimento:

- o módulo *app* é essencialmente o *facade* do micro-serviço (controla as rotas e os respetivos métodos).

- o módulo *ContactDB* generaliza os acessos à base de dados deste micro-serviço, semelhante a um *DBO*.
- o módulo *provas* utilizam o *ContactDB* para aceder à base de dados.
- o módulo *Classes* possui as classes mencionadas anteriormente.
- os ficheiros restantes referem-se a testes à base de dados.

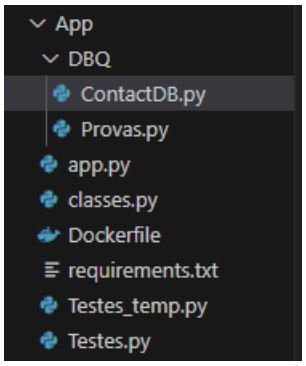


Figura 18: Estrutura do Projeto - Provas

### 3.7.3 Base de Dados

A base de dados deste micro-serviços sofreu algumas alterações, de forma a eliminar duas tabelas que podiam ser juntadas numa só, de forma a facilitar o desenvolvimento. As tabelas *texto_questao_espaco* e *criterio_questao_espaco* foram "fundidas"numa só, a tabela ***espacos***, que representa a informação dos espaços a preencher numa pergunta de espaços. Possui os seguintes atributos:

- *id*: id do espaço
- *n_espaco*: número do espaço na pergunta
- *correto*: cotação caso aluno acerte no espaço
- *texto*: texto correto a inserir
- *questao_id*: id da questão em que o espaço está inserido

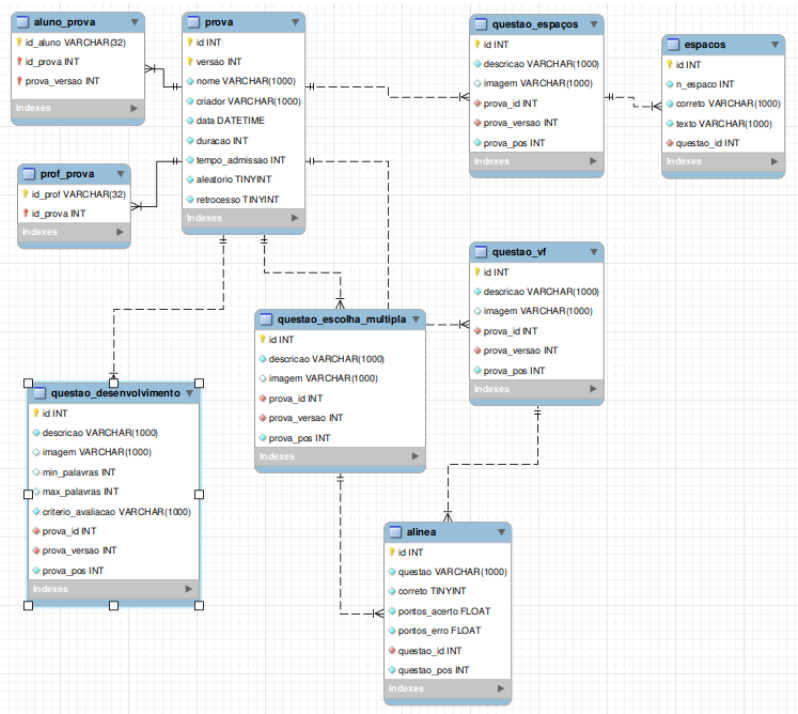


Figura 19: Modelo Lógico - Prova

### 3.7.4 Problemas

No geral, o desenvolvimento deste micro-serviço correu bem, pelo o que a maior dificuldade revelou-se em entender ao certo como o *sqlalchemy* funcionava, já que foi a primeira vez os dois desenvolvedores usaram esta biblioteca. Para além disto, como este micro-serviço contém muitas rotas que têm de devolver muita informação no formato *.json*, foi um verdadeiro desafio testar todas e corrigir os erros nelas presentes, algo em que a aplicação *Postman* ajudou bastante, estando portanto confiantes que todas as rotas estão a funcionar devidamente.

Quanto ao *Docker*, não sentimos muitas dificuldades, apesar da falta de experiência dos responsáveis por este micro-serviço, já que utilizamos como base os ficheiros usados para o *docker* de outros micro-serviços, e adaptamos ao nosso.

### 3.7.5 Análise Crítica

Pela a nossa estimativa e pelos os testes realizados, acreditamos que este micro-serviço esteja 100% funcional.

Em termos de funcionalidades, estimamos que conseguimos implementar 90% das funcionalidades referentes a este micro-serviço, estando em falta as funcionalidades referentes à eliminação de provas e questões já existentes, que não conseguimos implementar a tempo.

## 3.8 Micro-Serviço Realização de Provas

Em relação ao **Micro-Serviço Realizar Prova**, foi necessário realizar algumas alterações tanto a nível de arquitetura como da própria API.

### 3.8.1 API

Em relação à API, a versão final deste micro-serviço conta com a API presente na seguinte imagem:

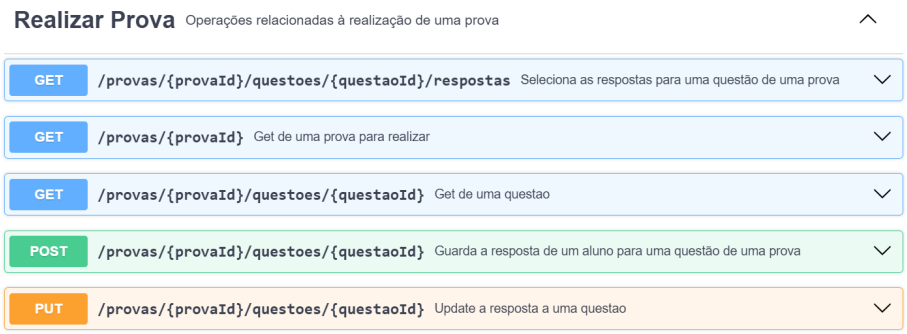


Figura 20: API - Realizar Prova

Esta API sofreu algumas alterações em relação à anterior. Mantivemos os métodos *GET* e *POST* da rota **/prova/{idProva}/questoes/{idQuestao}** e acrescentamos um método *PUT* que funciona como um update a uma resposta a uma questão. Adicionamos um método *GET* da rota **/provas/{idProva}** que devolve a prova que pretendemos realizar.

### 3.8.2 Arquitetura

Em relação à arquitetura, o diagrama ilustrado na figura seguinte representa a arquitetura final completa deste Micro-Serviço. Para simplificar aqui estão algumas notas da arquitetura:

- ContactDB, RespostaAluno, e app são módulos e não classes.
- RespostaQuestao e CriaInstancias são classes e não módulos.
- Todas as classes são inicializadas retirando a informação *json* que é recebida pelo microserviço.
- O módulo app pode ser visto como o *facade* do micro-serviço.
- O módulo ContactDB é um módulo que generaliza os acessos à Base de Dados deste micro-serviço.

- O módulo RespostaAluno utiliza o módulo ContactDB para aceder à base de dados.

A nível de alterações, adicionamos a classe CriaInstancias e detalhamos melhor quais as funções e atributos que cada módulo/classe tem.

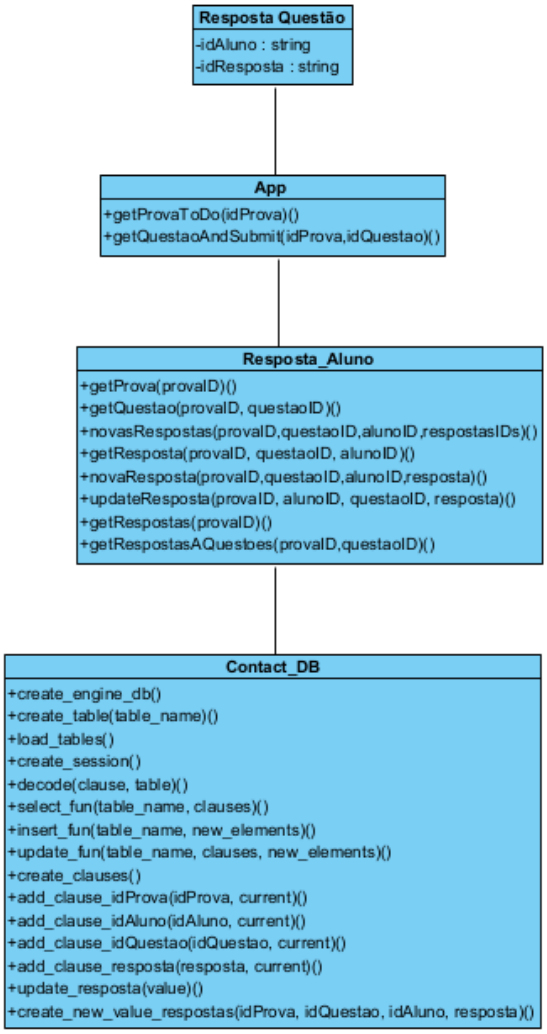


Figura 21: API - Realizar Prova

### 3.8.3 Base de Dados

Em relação à base de dados, a única tabela permaneceu inalterada. Esta tabela, denominada **respostas_alunos**, mantém a mesma informação, incluindo o ID da prova, o da questão, o do aluno e a resposta correspondente.

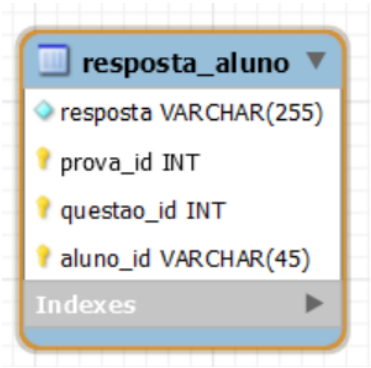


Figura 22: Estrutura do Micro Serviço

### 3.8.4 Problemas

No geral, o desenvolvimento deste micro-serviço correu sem grandes dificuldades associadas. Isto deveu-se à constante comunicação entre os dois desenvolvedores e à facilidade com que encontramos a resolução de erros que enfrentamos, bem com a facilidade de encontrar bibliotecas *Python* que podíamos usar para o projeto e que cuja utilização é bastante simples, sendo *sqlalchemy* um exemplo disso.

As dificuldades foram mais acentuadas em tarefas relacionadas com **Docker** devido à falta de experiência de cada um dos elementos responsáveis por este micro serviço.

### 3.8.5 Análise Crítica

Embora não tivéssemos apresentado grandes dificuldades na execução do micro-serviço, chegamos à fase final do trabalho ainda com todas as rotas funcionais porém não implementadas no projeto devido à falta de sincronização com o grupo inteiro pelo número de elementos ser bastante extenso, mas apresentamos um micro-serviço funcional e eficiente, tal como, proposto na fase anterior.

Em termos dos requisitos que este micro-serviço teria de implementar, conseguimos cobrir todos os requisitos propostos inicialmente:

- O Aluno tem uma opção para responder a Provas;
- O Aluno seleciona a questão que pretende responder;
- O Aluno submete uma resposta;

- O Aluno termina a Prova a qualquer momento.

# 4 Padrão *Observer*

O padrão *Observer* não chegou a ser implementando, no entanto vamos referir como o mesmo iria ser feito.

1. O Docente decide publicar as notas.
2. O pedido chega à API-Gateway.
3. A API-Gateway manda um pedido ao micro-serviço correção prova para publicas as notas.
4. A API-Gateway manda um pedido ao micro-serviço prova para notificar os alunos que as notas foram publicadas.
5. O micro-serviço prova recebe o pedido e coleta os respetivos alunos que realizaram a prova.
6. O micro-serviço prova contacta o micro-serviço utilizadores a informar quais os alunos a notificar.
7. O micro-serviço utilizadores recebe o pedido e os alunos, associa aos respetivos emails.
8. O micro-serviço utilizadores manda o pedido ao micro-serviço notificações a com a associação identificador dos utilizadores e respetivos email.
9. Por último, o micro-serviço notificações cria as novas notificações na sua base de dados.

# 5 Análise Crítica

Embora o projeto esteja inacabado, achamos que desenvolvemos um bom sistema, do ponto de vista arquitetural. Uma boa parte das funcionalidades que não estejam totalmente implementadas. Encontramos sérios problemas aquando a ligação do serviço como um todo, sobretudo problemas de incompatibilidade nos formatos das mensagens na receção e resposta de pedidos de cada micro-serviço, mas também há certos micro-serviços que estavam a ter problemas nas rotas. No entanto, achamos vivamente que se tivéssemos mais algum tempo para concluir o projeto, teríamos grande parte se não toda a aplicação montada, visto que certas componentes estão praticamente completas e outras estejam completas.

# 6 Conclusão

Com este trabalho prático, conseguimos aplicar todos os conhecimentos adquiridos na Unidade Curricular de Requisitos e Arquiteturas de Software, mais concretamente, todos os passos envolvidos na engenharia de requisitos, desde o seu levantamento ao desenvolvimento das funcionalidades deles derivadas.

Um dos maiores desafios foi, sem duvidas, coordenar uma equipa tão grande de forma a garantir a coesão entre micro-serviços, como referido anteriormente. Mesmo assim, apesar de não termos todas as funcionalidades implementadas e as que estão não estarem 100% corretamente funcionais, estamos orgulhosos e satisfeitos com o produto final, pelo o que achamos que cumprimos os objetivos do trabalho prático, sem ter dúvidas de que, com mais um bocado de tempo de desenvolvimento, teriamos conseguido implementar todas as funcionalidades propostas.

# 7 Anexos

## 7.1 Imagens Views

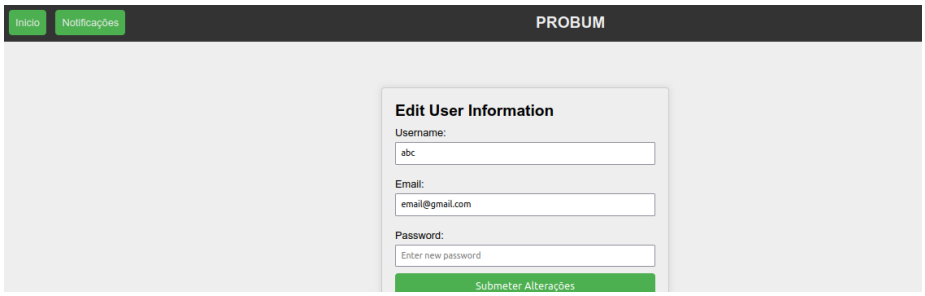


Figura 23: View 1

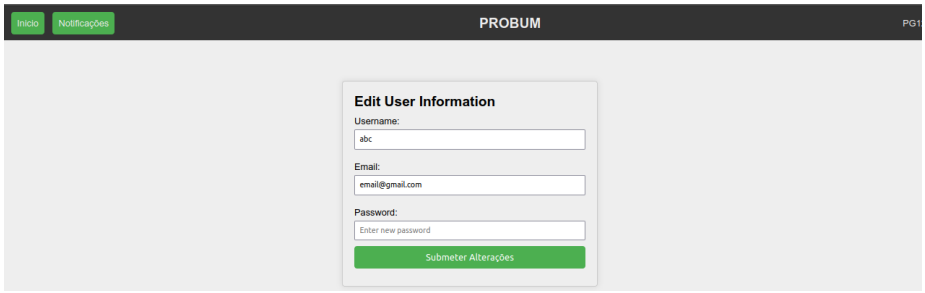


Figura 24: View 2


Figura 25: View 3

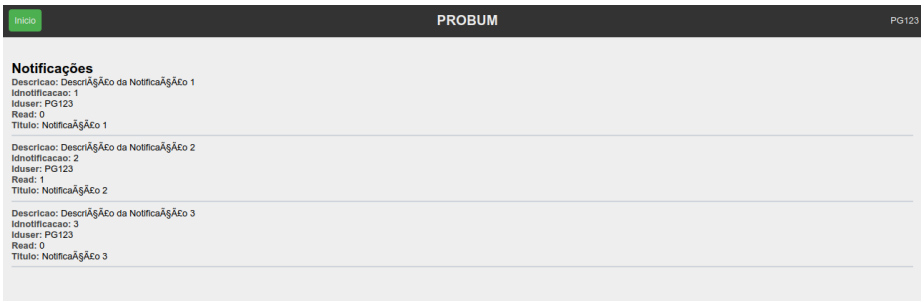


Figura 26: View 4


Figura 27: View 6

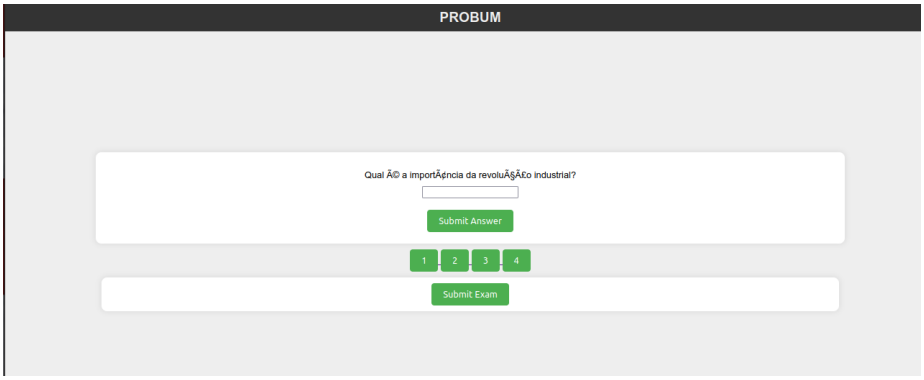


Figura 28: View 6

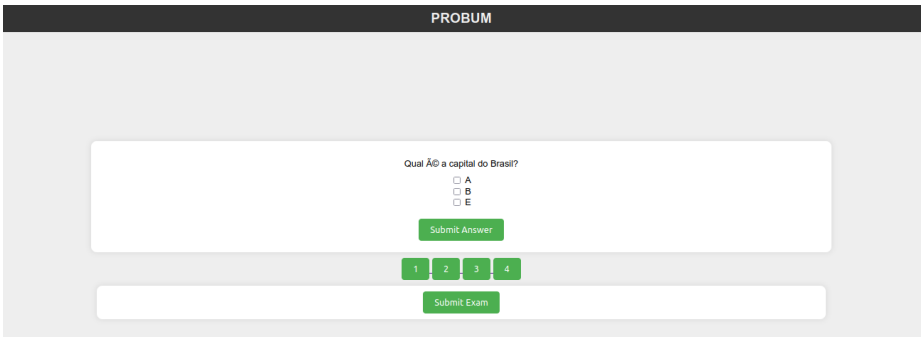


Figura 29: View 7

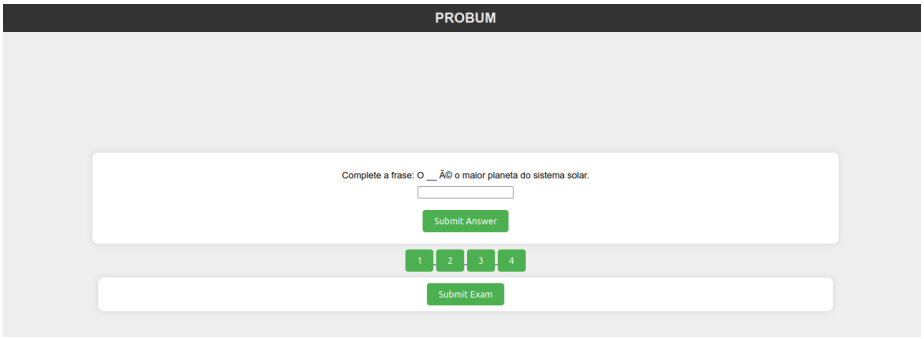


Figura 30: View 8

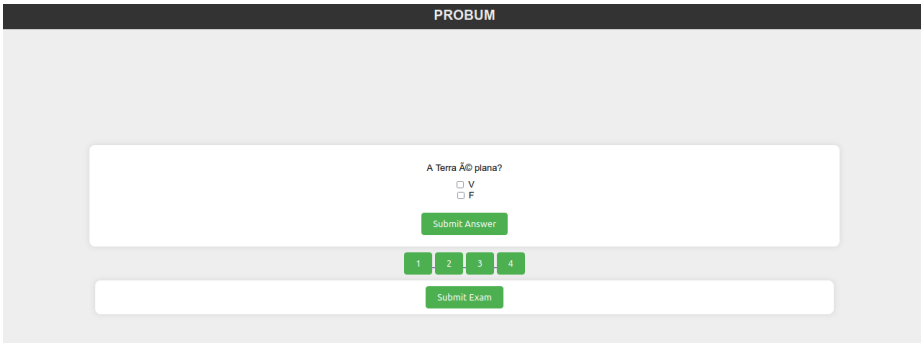


Figura 31: View 9

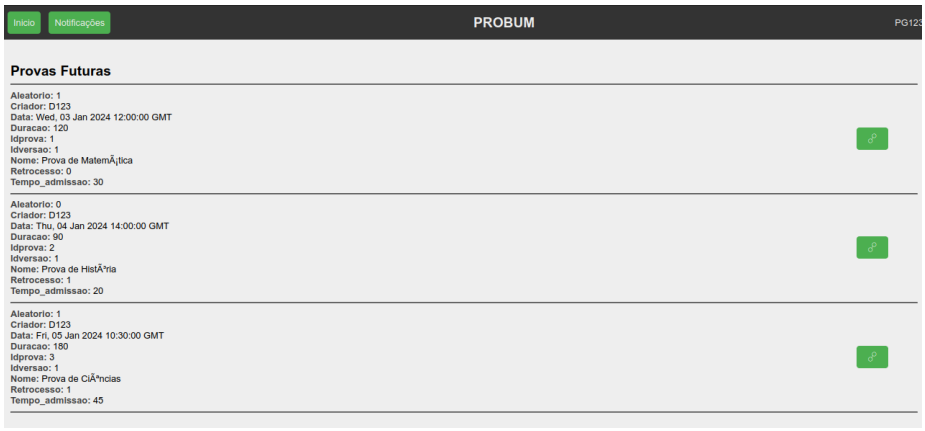


Figura 32: View 10


Figura 33: View 11