Construção de Sistemas de Software 2023/2024
Projeto 1 - Fase 2 - Relatório

Rodrigo Correia - 58180
Laura Cunha - 58188
Guilherme Wind - 58640

22/05/2024

Arquitetura do Projeto

O projeto está dividido em 2 módulos:

- A aplicação que corre com base num container docker, o qual inicializa a base de dados e a aplicação *Spring*, e permite apresentar a interface web para os docentes, na qual se podem realizar os casos de uso A, B, C, D, E, I, K, L, N, O e P.
- A aplicação JavaFX que corre independentemente através da execução do seu próprio *jar*, e apresenta uma interface que permite a realização dos casos de uso realizados pelos alunos, mais concretamente o A, F, G, H, J e M.

Decisões técnicas tomadas na arquitetura da aplicação

O projeto foi implementado tendo por base a separação em camadas: Presentation e Business (Service, Domain Model, Persistence).

- Na camada de Presentation consta o código relativo à API REST, e às interfaces web e JavaFX.
- A camada de Business, inclui as camadas de Service, Domain Model e Persistence:
  - Na camada Service, estão os serviços que permitem aceder aos handlers sem quebrar a divisão em camadas, bem como os *DTO's* para realizar a passagem dos dados entre a camada de apresentação e a de Business, podendo-se alterar os dados na camada de negócio sem estes serem alterados na camada de apresentação, realizando desta forma também uma separação.
  - Na camada Domain Model encontram-se os handlers dos casos de uso (1 por UC) e as entidades que vão ser mapeadas para a base de dados, através de ORM.
  - Por sua vez, a camada de Persistence, possui os repositórios, que permitem pesquisar nos dados que estão guardados na base de dados.

A aplicação faz uso dos mecanismos de *Inversion of Control (IoC)* presentes na framework *Spring*, na medida em que esta chama o código da aplicação quando seja apropriado, através das anotações *@GetMapping, @PostMapping, @DeleteMapping* presentes nos controladores da web e da API REST.

A *Dependency Injection*, que é uma forma de *Inversion of Control*, foi realizada através do *Spring* com o uso de anotações *@Autowired* presente nos controllers para aceder aos serviços, nos serviços para aceder aos handlers, e nos handlers para aceder aos repositórios, fazendo com que o atributo anotado seja preenchido com uma instância do objeto pretendido.

Foram anotados as classes dos controladores web e da API REST, serviços, handlers e repositórios com *@Controller, @Component, @Service, @Repository*, sendo estas *Beans*, tornando-as em objetos que vão ser geridos automaticamente pelos *Spring* e que podem ser injetados quando requeridos pelas anotações *@Autowired*.

A concorrência foi gerida usando a anotação *@Transactional* em alguns métodos de alguns handlers, em casos de uso que requerem manter a consistência dos dados provenientes de vários repositórios, o que apenas é possível realizando operações atómicas. Tal como é o caso dos UC adicionar candidatura, atribuir tema, nota de defesa, marcações de sala e horas, mostrar estatísticas, e submeter documentos.

Foram usados *DTO's* como uma forma de encapsular dados e passá-los entre a camada de *Business* e *Presentation*, mantendo o isolamento das camadas.

Os repositórios presentes na camada de *Persistence* estendem a interface *CrudRepository* que contém métodos de procura, e manipulação.

Tanto na interface web como na API Rest o login realiza-se fazendo uso do *HttpSession* que vai settar cookies com um id de uma sessão, para a qual serão guardados atributos do lado do servidor para essa sessão específica.

Decisões técnicas no desenho da interface Web

Para a criação da interface web foi usado *Server Side Rendering (SSR)* através da template engine do *Thymeleaf* presente no *Spring*. Este trata de processar os templates HTML, preenchendo os campos necessários, fazendo assim o render do mesmo, enviando o HTML já processado para o cliente.

A interface web também faz uso do padrão *MVC*, sendo os **controladores** responsáveis por receber os pedidos HTTP (POST ou GET) e executar as operações necessárias, a **view** composta pelo template HTML e o **model** contendo os respetivos dados para preencher o template da view através da interface *Model* do *Spring*.

Utilizou-se um *logout* comum para os ecrãs incluídos durante o uso da aplicação, de forma a reaproveitar essa parte do código e permitir manter uma aparência consistente da mesma nos diversos ecrãs.

Os controllers foram criados utilizando o *Spring*, marcando como *@Controller* as classes e criando métodos de mapeamento (*@GetMapping* e *PostMapping*), desta forma, através de *IoC* o *Spring* irá executar o código presente nesses métodos, após receber pedidos para os *URLs* especificados na anotação.

Algumas decisões a realçar na apresentação da interface:

1. Caso o utilizador autenticado seja um trabalhador, este apenas possui acesso às abas dashboard e themes, não podendo aceder às abas de teses, defesas nem estatísticas, visto que, decidimos deixar estas informações apenas visíveis para os docentes da universidade.
2. Na página de detalhes de um tema, caso o docente autenticado seja administrador de um mestrado compatível com esse tema, será apresentado um formulário para atribuir este tema a um aluno, também pertencente a esse mestrado.
3. De forma semelhante, na página de detalhes de uma defesa, caso o utilizador autenticado seja o arguente, ou presidente, no caso da defesa ser da tese final, será apresentado um formulário para este atribuir a nota a essa defesa, podendo a mesma apenas ser atribuída uma vez.
4. Nas estatísticas de sucesso apresentadas: o número de alunos corresponde à totalidade dos estudantes; a média refere-se à média de todas as notas das defesas finais dos estudantes; por sua vez, o total de estudantes aprovados, reprovados e não avaliados, são referentes sempre à classificação atribuída à defesa final.
5. Durante a criação de um tema, caso o utilizador autenticado seja um trabalhador, será apresentado no formulário um campo adicional para escolher o orientador interno para esse tema, sendo necessário preenchê-lo.
6. A página de login é única, sendo esta utilizada por todos os tipos de utilizadores (Trabalhadores e Docentes) para se autenticaram.
7. Na aba de defesas são mostradas todas as defesas nas quais o utilizador autenticado pertence ao júri, quer seja orientador, arguente ou presidente.
8. Na aba de temas são mostrados todos os temas presentes na base de dados, independentemente do ano letivo.

Seguidamente apresentam-se prints dos vários menus da aplicação:

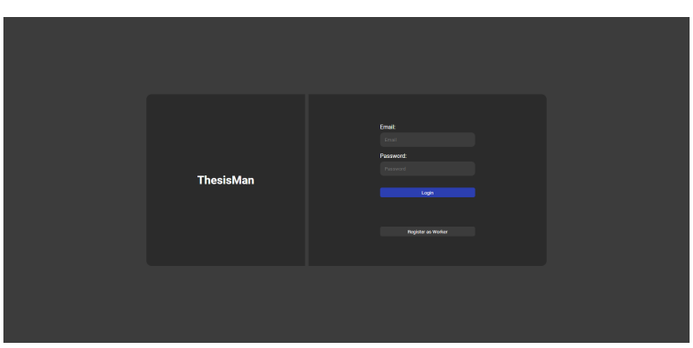

Figura 1: Login de docentes e trabalhadores

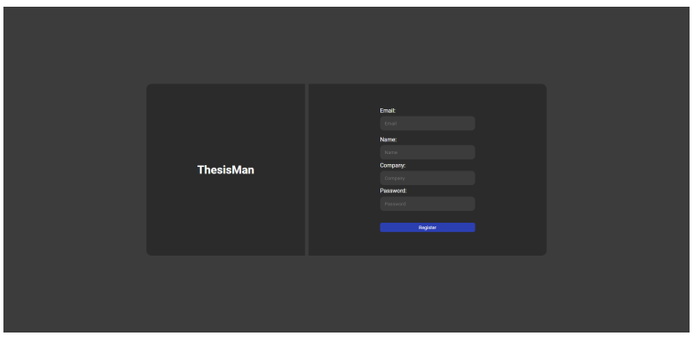

Figura 2: Registo de empresarial

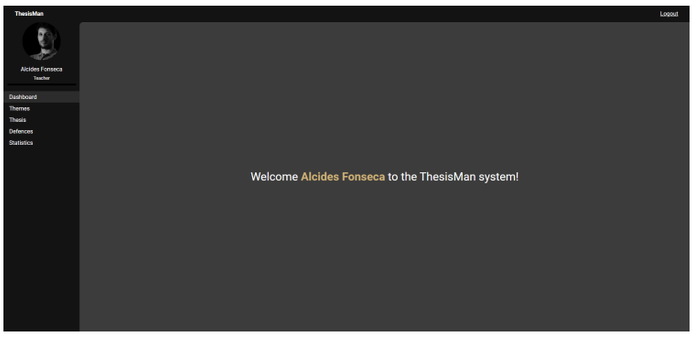

Figura 3: Dashboard

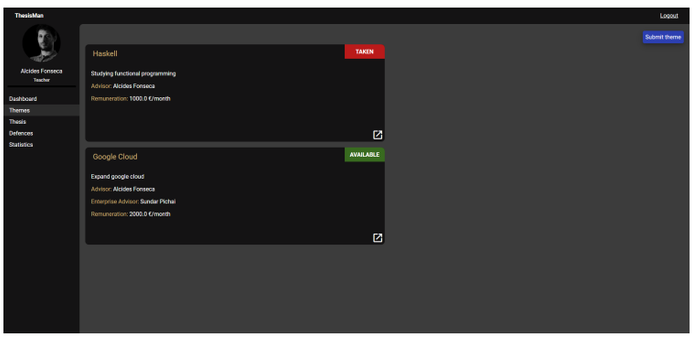

Figura 4: Temas

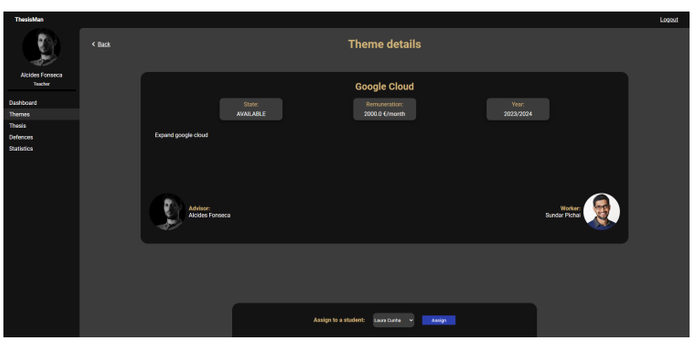

Figura 5: Temas detail

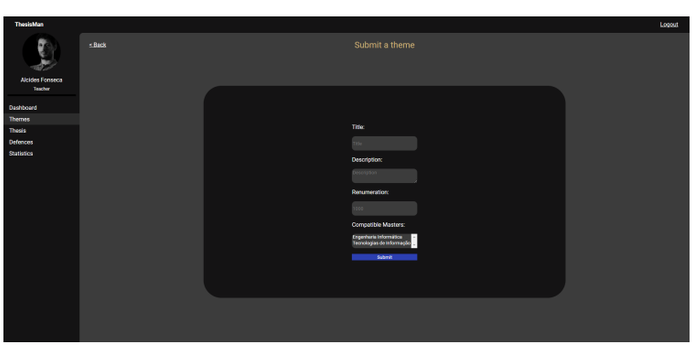

Figura 6: Submissão de tema

Figura 7: Tese

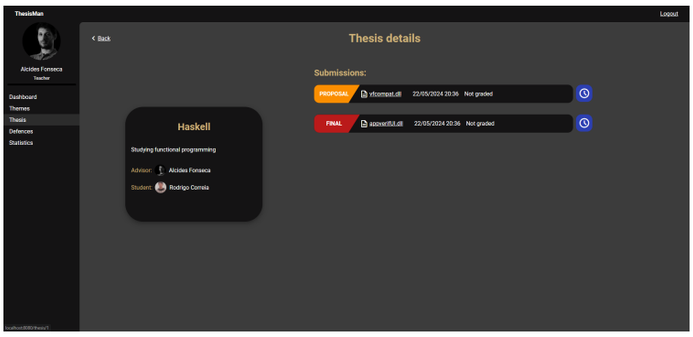

Figura 8: Tese detail

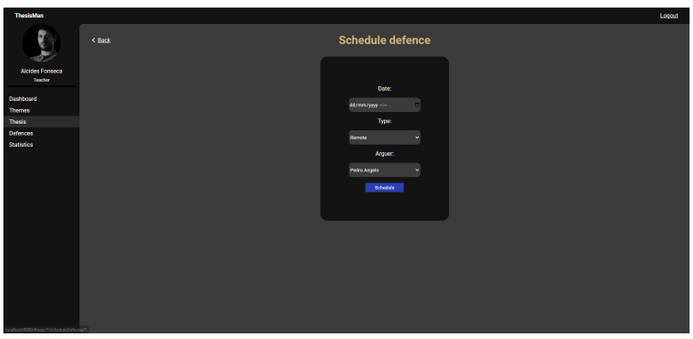

Figura 9: Agendar defesa

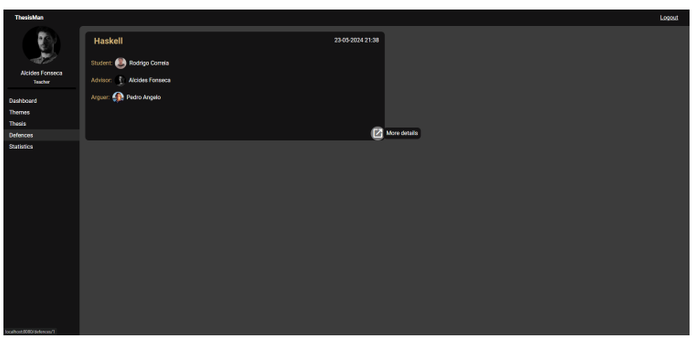

Figura 10: Defesa

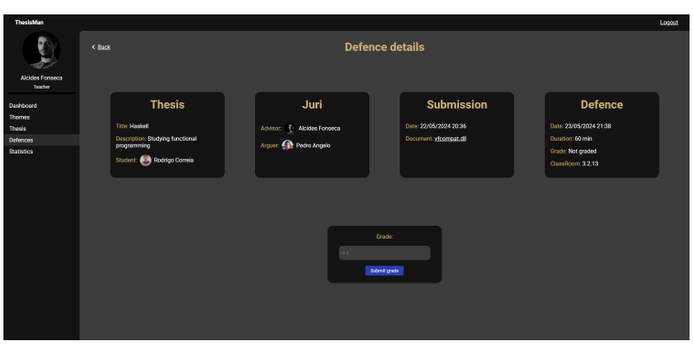

Figura 11: Defesa detail

Figura 12: Estatísticas

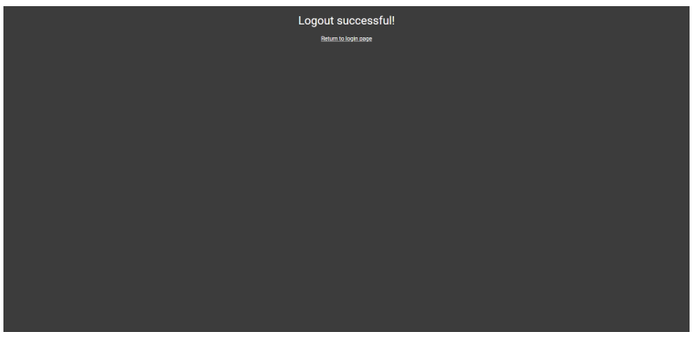

Figura 13: Logout

Decisões técnicas no desenho da API REST

Para implementar a API REST definimos controladores REST que gere os pedidos entre a aplicação e a interface *JavaFX* dos diversos casos de uso dos referentes aos alunos, através dos vários endpoints:

- GET /application: obtém todas as candidaturas
- POST /application: cria uma candidatura
- DELETE /application/applicationId: remove uma candidatura através do id da mesma
- GET /logout: realiza o logout do aluno, invalidando a sessão http
- GET /student: autentifica um aluno através do username e da password relativas à query
- POST /submission/propose: cria uma submissão do documento de proposta passado na query de uma tese especificada na mesma
- POST /submission/final: cria uma submissão do documento final passado na query de uma tese especificada na mesma
- GET /submission/final: devolve a submissão final relativa à tese especificada na query
- GET /submission/proposals: devolve a lista de propostas de submissões relativas à tese especificada na query
- GET /themes: devolve os temas relativos ao ano especificado na query
- GET /thesis: devolve a tese do estudante especificado na query

Passando a explicar algumas anotações utilizadas nos parâmetros dos métodos destes controladores:

- *@RequestParam*: realiza uma query no URL
- *@PathVariable*: permite atribuir o valor do parâmetro ao URL

Através das anotações *@GetMapping*, *@PostMapping*, *@DeleteMapping* presentes nos métodos dos controllers, tal como na interface web, fazemos uso do *IoC*.

Tal como referido anteriormente, os controladores estão anotados com *@RestController*, já que se tratam de *Beans* que vão ser usados pelo *Spring* para realizar *Dependency Injection*, e ainda possuem a anotação *@Autowired* nos atributos referentes a serviços e ao *HttpSession*.

Ainda, decidiu-se utilizar POST em vez de PUT, uma vez que queremos criar objetos e não apenas atualizá-los.

Para submeter um ficheiro no frontend e enviá-lo para o backend decidiu-se utilizar o *MultipartFile*.

Decisões técnicas no desenho da interface JavaFX

Para implementar a interface desktop foi utilizado a framework *JavaFX* fazendo uso do padrão *MVC* com o qual temos um **model** que é um objeto composto por properties que guardam dados, a **view** que é um ficheiro FXML que define os vários elementos presentes e os seus estilos, e o **controller** que contém a lógica para manipular o modelo, fazendo pedidos e tratando eventos.

Ainda, esta aplicação faz uso de pedidos à API REST de forma a obter os dados para dar display na interface.

A técnica de *IoC* está presente nos elementos da interface, sendo esta usada para inicializar e chamar eventos.

Os controllers utilizam anotações *@FXML* nos atributos e métodos presentes nos ficheiros FXML fazendo uso da técnica de *Dependency Injection*, sendo estes atribuídos pelos *JavaFX* aquando da inicialização.

Os atributos injetados possuem properties às quais foram feitos *binds* de forma a serem alteradas dinamicamente, fazendo estas uso do papel de um *observer*, de forma às mudanças realizadas no modelo serem refletidas na interface. Também fez-se uso de *ListView's* as quais possuem uma *ObservableList* que permite que quando se altere um elemento da lista esta seja atualizada na interface. Como se pode ver na figura 16, temos uma *ListView* com um *observable* que atualiza a lista quando se adicionam ou removem elementos, e um contador do número de candidaturas que possui um *bind* para ser atualizado de acordo com o número de candidaturas da lista.

Seguidamente apresentam-se prints dos vários menus da aplicação:

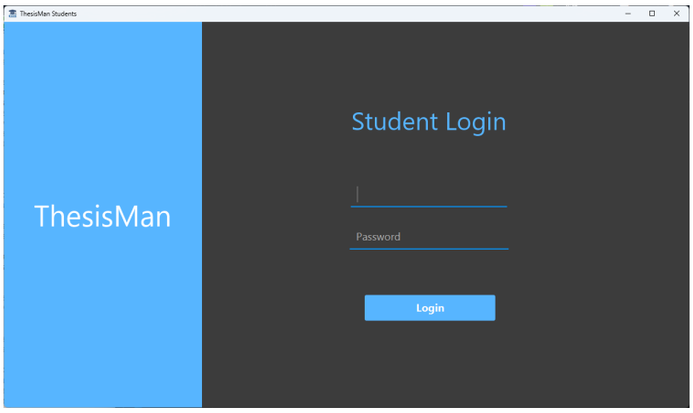

Figura 14: Login de alunos

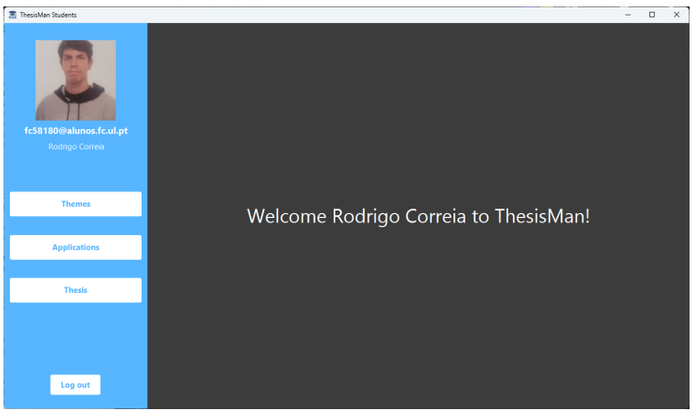

Figura 15: Dashboard

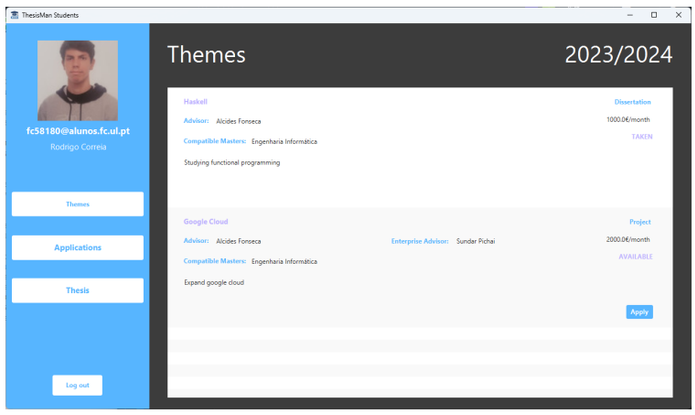

Figura 16: Temas

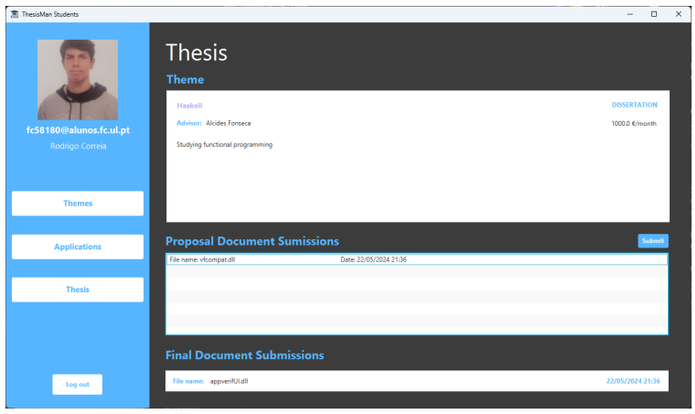

Figura 17: Teses

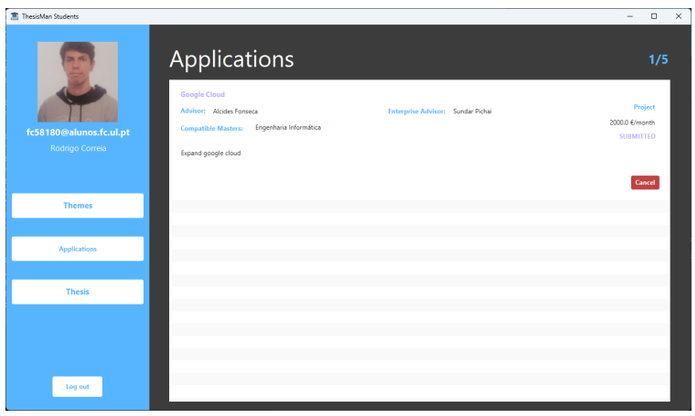

Figura 18: Candidaturas

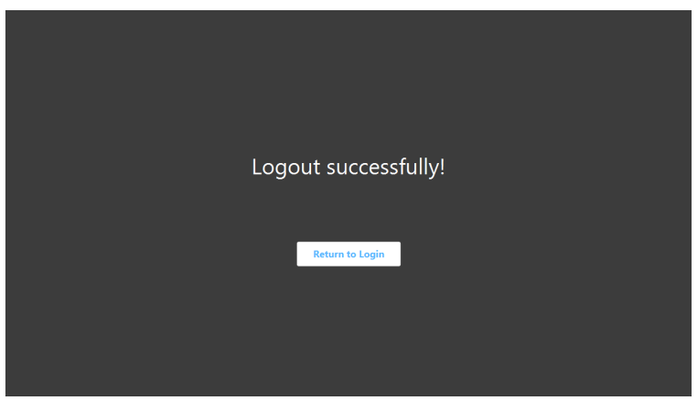

Figura 19: Logout