

# ThesisMan – Relatório Técnico

*Construção de Sistemas de Software*

*Projeto Prático Fase #2*



**Ciências  
ULisboa**

Daniela Camarinha fc58199

Gonçalo Pinto fc58178

José Brás fc55449

## Descrição da arquitetura de camadas:

A arquitetura em camadas na Engenharia de Software serve para estruturar sistemas complexos em camadas distintas, de forma a gerir essa mesma complexidade dividindo responsabilidades bem definidas pelas camadas. A comunicação entre camadas faz-se de forma que os pedidos fluam para baixo e as respostas a esses pedidos para cima.

Neste 2º Trabalho, vamos falar um pouco sobre todas as camadas.

A camada de Apresentação é uma camada que interage diretamente com os clientes. Neste projeto teremos uma aplicação Web que utiliza ... e uma aplicação desktop que beneficia da plataforma de software JavaFX com um ficheiro fxml associado a caso de uso.

A camada de Aplicação serve como intermediária entre a Apresentação e o Domain Model. As aplicações Desktop e Web incluem controllers e serviços. Controllers lidam com as requisições HTTP, e a camada troca instâncias de classes DTO (Data Transfer Objects) entre o cliente e o servidor.

Os nossos handlers contém a grande parte da Lógica de Negócios e executam os casos de uso da nossa aplicação

A camada de Negócio utilizará o padrão Domain Model que organiza o código baseado em conceitos do domínio, suas características e associações.

A Camada de Persistência é uma camada intermédia entre o Acesso aos Dados e Lógica de Negócio que serve para abstrair os conceitos OO e os tipos de dados representados na base de dados. Aqui são utilizados conceitos ORM e Java Persistence Annotation para automaticamente criar as relações entre as entidades na Camada de Dados.

Por sua vez, a Camada de Dados é uma camada baseada numa base de dados em PostgreSQL.

## Escolha e justificação das decisões técnicas tomadas na arquitetura da aplicação, suportadas pelos padrões desenvolvidos na aula.

Neste projeto foram usados vários padrões, estudados em aula, relativos á arquitetura de uma aplicação, como é o caso de *APIs REST*. De modo a desenvolver uma aplicação destinada a alunos que fosse capaz de comunicar com um servidor utilizou-se uma arquitetura *REST* realizada através de duas classes: **RestAPIClientService**, **RestThesismanController**. Ambas as classes permitem o utilizador da aplicação falar com o servidor através de troca de mensagens sobre o protocolo http.

De modo a representar a informação necessária de objetos nas vistas da aplicação criamos uma classe *Controller* para cada página da app. Estes vão refletir todas as mudanças de dados de modo ao utilizador se aperceber das mesmas. Uma vez que utilizamos um padrão *MVC*, Model-View-Controller, estes tipos de classes são essenciais para a nossa aplicação. Para além disso, para desenvolver a aplicação utilizamos a plataforma de software *JavaFX*, que nos deu ferramentas para implementar a aplicação seguindo o padrão *MVC*.

Adicionalmente, recorreremos a objetos *DTO* para encapsular os dados necessários para o funcionamento da aplicação de modo a possibilitar a transferência destes do servidor para o utilizador. Para além disso, o uso de *DTOs* permite separar os objetos da maneira como estes são representados, possibilitando assim a mudança de um sem afetar o outro.

Para realizar a lógica de negócio criamos várias classes de *Handlers* e de modo a abstrair nas camadas superiores a forma como as operações são realizadas, optamos por usar uma classe *Service*. Para além de promover a abstração do código, esta permite mudar a forma como os *Handlers* atuam sem mudar a classe que utiliza o *Service*.

## Escolha e justificação das decisões técnicas no desenho da interface Web.

Executar a aplicação web:

- Executar o Docker numa máquina
- Abrir o browser e ir para <ip> :8080 sendo *ip* o ip da máquina com o Docker ligado.

A aplicação web é a interface utilizada pelos docentes e utilizadores empresariais. Os utilizadores podem fazer o login (UC A e B). Se as credenciais estiverem erradas, o utilizador terá de se registar. Pressupõe-se que seja um utilizador empresarial. (UC C).

Os docentes podem:

- Submeter temas (UC D)
- Atribuir temas aos alunos (UC I)
- Marcar e registar a nota da defesa da proposta de tese (UCs K e L)
- Marcar e registar a nota da defesa final de tese (UCs N e O)

- Ver estatísticas (UC P)
- Atualizar o estado da Candidatura

Os utilizadores empresariais podem:

- Submeter temas (UC E)
- Marcar e registar a nota da defesa da proposta de tese (UCs K e L)
- Marcar e registar a nota da defesa final de tese (UCs N e O)

As páginas são renderizadas utilizando marcação HTML, enquanto a estilização é aplicada através de CSS.

Os controllers gerem as requisições HTTP e mapeiam os casos de uso para os endpoints:

- @GetMapping: Métodos que processam requisições GET para carregar páginas ou dados do servidor.
- @PostMapping: Métodos que processam requisições POST para enviar novas informações ao servidor.

## Escolha e justificação das decisões técnicas no desenho da API REST.

Para possibilitar a comunicação entre a camada de Apresentação e de Negócio, utilizamos duas classes, **RestAPIClientService** e **RestThesismanController**, que possibilitam um serviço de rest api. Este serviço permite a troca de mensagens sobre o protocolo Http entre o utilizador da aplicação e o server.

A classe **RestAPIClientService** está situada do lado cliente e apresenta uma função para cada caso de uso específico do aluno. A esta estão associados dois atributos, *alunoId* e *instance*, que representam o id do aluno da sessão atual e a instância da classe a ser usada pelo utilizador para comunicar com o servidor. Cada função da classe envia um pedido ou “POST” ou “GET” para o servidor com as informações necessárias no corpo da mensagem através de um ficheiro Json ou através de parâmetros no “url” do pedido. Caso o retorno do servidor não seja o esperado, é lançada uma caixa de alerta a avisar do imprevisto.

A classe **RestThesismanController** está situada do lado do servidor e apresenta uma função de resposta para cada pedido do mesmo. De modo a realizar os requerimentos do aluno, é chamada uma função, específica de cada caso de uso, na classe **ThesismanServiceImp**, correspondente aos serviços do servidor, que por sua vez faz “forwarding” do pedido para o handler correto e assim efetuar o pedido do user. Foi decidido implementar a classe **ThesismanServiceImp** de modo a promover a abstração do código e flexibilidade do mesmo. Após realizar o pedido do aluno, é enviada uma resposta ao mesmo com um status “BAD\_REQUEST” ou “OK”. O status é definido por erros ou estados incompatíveis que resultam num funcionamento errado do servidor.

## Escolha e justificação das decisões técnicas no desenho da interface JavaFX.

Executar a aplicação desktop:

- `cd desktop-app`
- `mvn clean javafx:run`

A aplicação desktop, que funciona como a interface para os alunos, beneficia da plataforma de software JavaFX. Cada página é gerada através da linguagem de marcação FXML. Existe um controller e um ficheiro FXML para cada um dos casos de uso pedidos.

### LoginController

É a classe que contém todos os elementos FXML que vão interagir com a página de login e vão interagir com o caso de uso A.

Neste controller, o aluno pode fazer o Login na aplicação para entrar na sua conta da FCUL. Terá de inserir o seu e-mail e a sua palavra-passe.

### MenuController

É a classe que contém todos os elementos FXML que vão interagir com a página de menu.

A partir do menu, o aluno pode escolher entre vários casos de uso: listar temas, criar ou cancelar a candidatura, submeter o documento para a proposta da tese e submeter o documento que corresponde à tese final.

### ListTemasController

É a classe que contém todos os elementos FXML que vão ser usados para executar o caso de uso F para listar os temas disponíveis neste ano letivo, por parte dos alunos.

A página contém uma tabela de temas. Cada tema tem um id, um título, uma descrição, a sua remuneração, o id do seu submissor a lista de mestrados compatíveis.

### CriarCandidaturaController

É a classe que contém todos os elementos FXML que vão ser usados para executar o caso de uso G para criar uma candidatura a um tema.

O aluno pode escolher entre vários temas compatíveis com o seu mestrado na tabela, selecionar um tema e clicar no botão para criar a candidatura.

### CancelarCandidaturaController

É a classe que contém todos os elementos FXML que vão ser usados para executar o caso de uso H para cancelar uma candidatura.

O aluno pode escolher entre vários temas que tinha previamente escolhido, selecionar um tema e clicar no botão para cancelar a candidatura.



```
Mestrado mestrado1 = new Mestrado(nome:"Engenharia Informática");
Mestrado mestrado2 = new Mestrado(nome:"Engenharia Biomédica");
mestradoRepository.save(mestrado1);
mestradoRepository.save(mestrado2);
```

Dois docentes:

```
Docente docente1 = new Docente(departamento:"Informática", isAdmin:true, name:"Alcides", contact:"Alcides@mockdocente.pt");
Docente docente2 = new Docente(departamento:"Informática", isAdmin:true, name:"Pedro", contact:"Pedro@mockdocente.pt");
docenteRepository.save(docente1);
docenteRepository.save(docente2);
```

Dois temas:

```
Tema tema1 = new Tema(titulo:"Tema1", descricao:"Descrição1", remuneracaoMensal:1000, docente1);
Tema tema2 = new Tema(titulo:"Tema2", descricao:"Descrição2", remuneracaoMensal:2000, docente2);
```

Dois alunos:

```
Aluno aluno1 = new Aluno(average:18.1, name:"Maria", contact:"fc123@alunos.pt", mestrado1);
Aluno aluno2 = new Aluno(average:12.5, name:"João", contact:"fc345@alunos.pt", mestrado2);
```

Três Candidaturas:

```
Candidatura candidatura1 =
    new Candidatura(new Date(), EstadoCandidatura.APROVADO, aluno1, tema1);
Candidatura candidatura2 =
    new Candidatura(new Date(), EstadoCandidatura.EMPROCESSAMENTO, aluno2, tema2);
Candidatura candidatura3 =
    new Candidatura(new Date(), EstadoCandidatura.APROVADO, aluno1, tema2);
```

Duas Teses:

```
Tese tese1 = new Dissertacao(candidatura1);
Tese tese2 = new Dissertacao(candidatura3);
```

Duas defesas:

```
Defesa defesa1 = new Defesa(isFinal:false, isPresencial:false);
Defesa defesa2 = new Defesa(isFinal:true, isPresencial:false);
```

## Formato dos e-mails

Para controlar os emails dos docentes e utilizadores empresariais definimos regras que estes têm de seguir de modo a serem válidos. Para o email de um docente, este tem de acabar com @mockdocente.pt. Já no caso de ser um email de um utilizador empresarial, o mesmo tem de acabar em @utilempresarial.pt de modo a ser válido.