Relatório
CSS – Fase 2

António Estêvão – Nº 58203
Jacky Xu – Nº 58218
Maria Rocha – Nº 58208

Descrição da Arquitetura do Projeto

Para este projeto, adotou-se uma arquitetura em camadas, composta por uma camada de apresentação, uma camada de lógica de negócio e uma camada de acesso a dados. Essa escolha foi feita visando garantir uma clara separação de responsabilidades, de modo que alterações em uma camada não afetem as outras. No entanto, tanto a camada de lógica de negócio quanto a de apresentação foram subdivididas para aumentar a organização e a clareza do código

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

Utilizou-se o Maven para gerir as dependências do projeto. Ao construir o projeto utilizando o Maven, este automaticamente gere as dependências especificadas no ficheiro pom.xml. Este descarrega as dependências necessárias a partir dos repositórios Maven remotos, resolve as dependências transitivas e construi o classpath necessário para compilar e executar o projeto.

Adoção de uma arquitetura em camadas, com separação clara entre a apresentação, a lógica de negócio e o acesso a dados, visando facilitar a manutenção e a escalabilidade da aplicação.

Na camada de apresentação, utilizando JavaFX, esta foi dividida em as UI, responsável pela definição da interface gráfica e dos elementos visuais da aplicação, os controladores que lidam com os eventos realizados na UI, manipulando a lógica associada às interações do utilizador e atualizando a UI conforme necessário, e classes de serviço que são responsáveis pela comunicação entre o frontend e o backend, garantindo a transmissão correta de dados entre o cliente e o servidor

Para a parte web da aplicação, utilizando Thymeleaf, usou-se o padrão MVC (Model-View-Controller), que utiliza: modelos que representam os dados da aplicação, por exemplo as entidades JPA. A view que é responsável pela apresentação dos dados, utilizando Thymeleaf para criar as páginas HTML que são apresentadas aos utilizadores. Os controladores interceptam requisições dos utilizadores, manipula os dados através dos serviços e define que vistas devem ser retornadas.

A camada de lógica de negócio é responsável pela implementação das regras de negócio da aplicação. Esta é composta por serviços que contêm essa lógica e interagem com os repositórios para manipular os dados

A camada de acesso a dados é responsável por interagir com a base de dados. Utiliza repositórios JPA para persistir e recuperar dados. Os repositórios fornecem abstração do acesso aos dados e permitem operações CRUD de uma forma eficiente e sem necessitar de implementação

Escolha e justificação das decisões técnicas tomadas no desenho da interface Web:

Foi utilizado thymeleaf, para criar as interfaces web. O thymeleaf faz integração com o Spring boot da nossa aplicação original que só continha o JPA, permitindo então a fácil transferência de dados entre o controlador e a vista.

Foi organizada de modo a manter uma clara separação de responsabilidades, seguindo o padrão MVC (Model-View-Controller):

Controladores como o loginController lidam com os pedidos http e chamam as classes de serviço necessárias para lidar com a lógica de negócio e seleciona o template de thymeleaf a usar.

Os serviços, como o LoginService, lidam com a lógica de negócio e interagem com os repositórios para obter e manipular os dados.

Implementámos a gestão de sessões para manter o estado do utilizador autenticado. Ao efetuar login com sucesso, o ID do utilizador é armazenado na sessão, permitindo que a aplicação mantenha o contexto do utilizador entre as diferentes requisições HTTP:

Os modelos são representados pelas entidades Student, Teacher e os objetos de transferência de dados (StudentDTO, TeacherDTO). Estes modelos encapsulam os dados e a lógica de negócio da aplicação.

As vistas são representadas pelas páginas Thymeleaf, como login_teacher.html e login_uni.html. Estas vistas apresentam os dados ao utilizador e recolhem a entrada do utilizador.

Os controladores são representados pelas classes como LoginController. Eles tratam as requisições HTTP, interagem com os serviços para processar a lógica de negócio e retornam as vistas adequadas.

Escolha e justificação das decisões técnicas no desenho da API REST:

As API REST foram criadas usando o springboot, pois com as anotações específicas, como @RestController e @RequestMapping, a criação de endpoints REST é direta e intuitiva. @RestController indica que a classe é um controlador REST e que os métodos devem retornar diretamente os dados no formato JSON ou XML, tendo sido optado por utilizar JSON.

O uso de @Autowired permite a injeção automática de dependências, contribuindo para a inversão de controle (IoC). Spring automaticamente resolve e injeta as dependências necessárias quando cria uma instância de um bean.

Foram utilizadas anotações como @PostMapping para definir facilmente a resposta a diferentes tipos de requisições HTTP.

A utilização de @RequestBody String permite receber dados do frontend e trabalhar a lógica de negócio diretamente no servidor de backend.

As classes API REST apenas lidam com receber os pedidos HTTP encarregando a lógica de negócio e a resposta às classes de serviço.

Utiliza ResponseEntity que permite controlar tanto o corpo da resposta quanto o status HTTP.

A separação entre os controladores e os serviços promove uma arquitetura modular. Isto facilita a manutenção e a evolução do código.

Escolha e justificação das decisões técnicas no desenho da interface JavaFX:

A estrutura das interfaces foi definida nos ficheiros FXML. Estes ficheiros, quando ocorre um evento da interface de utilizador, chamam as classes de controlo. Estas classes são responsáveis por receber os eventos da interface e decidir como lidar com estes. Além disso, as classes de controlo chamam as classes de serviço sempre que há necessidade de comunicar com o servidor backend. As classes de serviço são responsáveis pela comunicação entre a aplicação JavaFX e o backend.

A interface foi dividida, em casos de uso específicos. Tendo assim a interface de login, responsável pela autenticação do utilizador (Esta apenas verifica se o utilizador existe na base de dados e se é um estudante, não verifica a palavra de passe), interface menu que apresenta as opções disponíveis. Casos de uso parecidos como submeter o documento para a Thesis Proposal e para Thesis Final apresentam-se na mesma interface sendo que são relacionadas.

Foi criado um screen controller, que permitiu mais facilmente guardar todas as scene existentes e ativar a que queríamos.

Cada FXML possui um controlador responsável pelas funcionalidades chamadas por este.

Foram criadas três classes de serviço para lidar com a comunicação entre a aplicação JavaFX e o backend.

Data Model foi responsável por guardar informações necessárias durante a execução da aplicação, como o ID do utilizador e a URL do servidor.

Também foram criados DTOs no servidor e classes correspondentes no frontend, para permitir o envio apenas dos dados necessários entre o servidor e a aplicação.