Relatório do projeto Thesisman

Disciplina de Construção de Sistemas de Software:

- Martim Pereira fc58223
- Daniel Nunes fc58257
- João Pereira fc58189

Modelo domínio atualizado

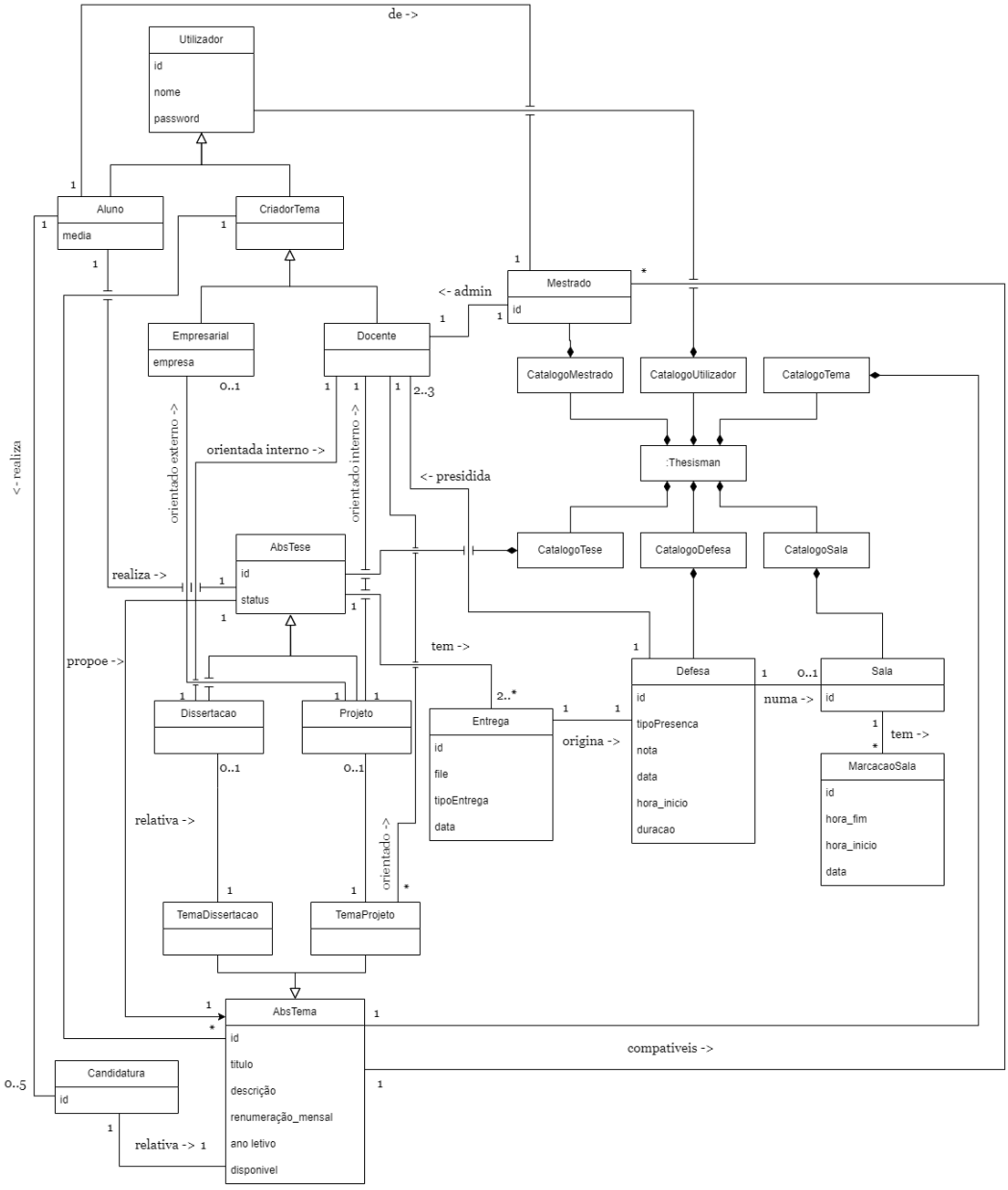
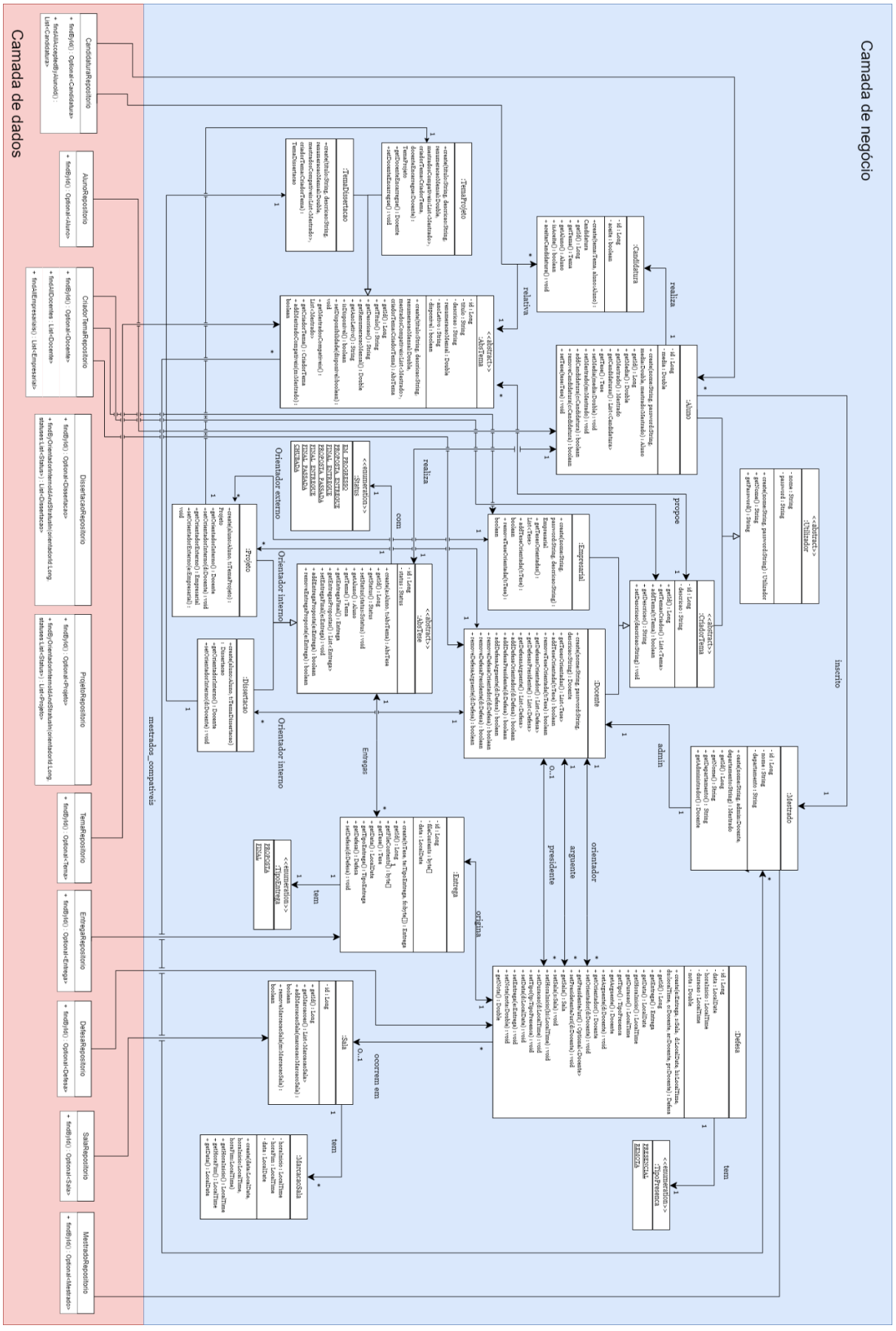


Diagrama de classes atualizado



Mudanças face á fase 1

Tema:

Em vez de termos uma Entidade Tema passámos a ter uma AbsTema que é extendida por TemaProjeto e TemaDissertacao tendo sido escolhida uma @Inheritance(strategy = InheritanceType.SINGLE_TABLE) de modo a partilharem a mesma tabela. Decidimos de esta forma pois apesar de o TemaProjeto ter um atributo a mais que TemaDissertacao

Tese:

Em vez de termos uma Entidade Tese passámos a ter uma AbsTese que é extendida pro um Projeto e Dissertacao tendo sido escolhida uma @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)

Arquitetura do Projeto

Componentes principais:

- **Controllers**: responsáveis por receber e tratar requisições http
- **Serviços:** executam a lógica de negócios e coordenam a comunicação entre controllers e handlers
- **Handlers:** implementam a lógica de negócios e realizam operações com dados
- **Mappers:** tem a responsabilidade única de converter DTOs em entidades e entidades em DTOs
- **DTOs:** utilizados para transferência de dados entre camadas da aplicação
- **Entidades:** representam os dados persistidos na base de dados sendo acedidos através de repositórios
- **Repositorios:** repositórios JPA que nos permitem obter e guardar entidades da base de dados

Decisões técnicas e arquiteturais:

- **Padrões arquiteturais:** uso de MVC para ter uma separação clara entre a camada de apresentação, lógica de negócios e acesso aos dados
- **DTO Pattern:** facilita a transferência de dados entre as diferentes camadas assegurando também a separação entre as mesmas
- **Dependency Injection:** utilizada para injetar os componentes necessários noutros que necessitem deles (um handler anotado com @Autowired num serviço que o usa), promovendo a inversão de controlo

- **Responsabilidade única:** cada classe tem responsabilidade única e bem definida, tal como o uso de handlers por caso de uso.

Gestão de concorrência:

- **@Transactional:** os métodos dos handlers estão todos anotados com @Transactional de modo a fornecer suporte para transações da base de dados, sendo usada par definir o scope de uma única transação, de modo a podermos garantir que os métodos anotados ocorreram todos na mesma transação e que distinguir se o método foi terminado com sucesso (se não foi lançada nenhuma exceção) e por isso a transação será confirmada (commit) e as alterações serão salvas permanentemente na base de dados ou se o método lançou uma exceção e é necessário fazer um rollback, descratando efetivamente todas as alterações feitas na base de dados. Isto ajuda efetivamente a manter a consistência dos dados, evitando estados inconsistentes em caso de falha.
- **Optimistic Locking (@Version):** todas as entidades foram anotadas com @Version para que sempre que uma entidade é lida do banco de dados é recuperado o campo @Version. Isto vai permitir que ao confirmar uma transação o JPA verifique se o campo ainda corresponde ao valor na base de dados, lançando exceção quando não corresponde, ou seja, verifica se não foi já efetuada uma outra alteração á entidade por outra transação que foi commited. Isto permite lidar com concorrência utilizando Optimistic Locking, detetando conflitos quando várias transações tentam modificar a mesma entidade ao mesmo tempo.
- **Pessimistic Locking:** nas transações identificadas como mais críticas foram utilizados EntityManagers de modo a pudermos dar lock nas entidades a serem alteradas. Isto pode trazer problemas como deadlocks e diminuição de desempenho, mas overall o grupo avaliou que estas transações iriam beneficiar do uso do lock.

Fluxo dos Casos de Uso

- Popular a base de dados

De modo a puderem ser testados os casos de usos tanto na aplicação JavaFX como na web app foram criadas entidades através do PopulateService que é chamado ao usar o endpoint “/” (na web app quando é chamado o localhost:8080) e no javafx quando a aplicação é ligada automaticamente.

NOTA: como não é feito drop dos repositórios na função de populate irá gerar um erro se se popular a base de dados 2 vezes.

- Desktop-app

- Login com autenticação da universidade (aluno)

Ao iniciar a aplicação esta abre uma página de login que permite que o aluno introduza as suas credenciais (email + password) na view login.fxml que através do LoginController comunica com o RestAPIAlunoService que por usa vez irá fazer um pedido POST ao endpoint “/login”.

Este enpoint vai comunicar com o LoginService que reencaminha o pedido para o LoginHandler, onde está a lógica por detrás do login de um aluno. Este handler, caso o login seja sucedido (aluno com o email passado existe), retorna o Aluno e cria um DTO chamado RespostaLoginAluno com as informações necessárias e retorna-o até á view. Caso o aluno não tenha sido encontrado é lançada uma exceção que é propagada até ao controller e traduzida para o lado do cliente com HttpStatus específicos. Do lado do servidor dependendo do status retornado ou sai da página de login para a próxima vista correspondente (vista para aluno sem tese e para aluno com tese) ou fica na mesma e aparece uma mensagem de erro descritiva.

- Listar temas disponíveis neste ano letivo

Após ter sido efetuado o login de um aluno caso este não tenha ainda uma tese associada é carregada a view list_temas.fxml, na qual o aluno tem a capacidade de visualizar todos os temas do ano letivo. Esta view antes de ser renderizada é carregada com os temas propostos no ano letivo atual recorrendo ao ListarTemasController que comunica com a RestAPIAlunoService que realiza um pedido GET ao endpoint “/temas” presente na RestAPIController sendo o ano recebido como @RequestParam. Este por usa vez comunica com o ListarTemasService que obtêm a List<Tema> do ListarTemasHandler, sendo estes convertidos pelo serviço em List<TemaDTO> recorrendo a um TemaMapper. Esta lista de DTOs é devolvida até ao ListarTemasController do lado do cliente de modo a puder expô-los na view.

- Candidatar a um tema (limite de 5), por parte dos alunos

Este caso de uso é realizado no mesmo menu do caso de uso apresentado no UC anterior, onde juntamente á lista mencionada anteriormente um botão “Candidatar” que ao ser pressionado, após ter sido selecionado um tema, criar uma candidatura por parte do aluno ao tema selecionado, através da RestAPIAlunoService que efetua um pedido POST ao endpoint “/candidatura” presente no RestAPIController, passando o id do aluno e o id do tema. Este por sua vez contacta o CriarCandidaturaService que pede ao CriarCandidaturaHandler que tenta adicionar a nova Candidatura ao Tema á lista de candidaturas do Aluno, verificando se o mestrado do aluno é compatível com os listados no tema e se este não realizou já 5 candidaturas (limite máximo). Para lidar com a concorrência a lista de candidaturas de um aluno, apesar de esta ser rara/improvável, passou por dar um PESSIMISTIC_WRITE lock em cada um cada uma das candidaturas de maneira a

assegurar não ocorrem lost updates na atualização da lista. Finalmente o CriarCandidaturaHandler devolve um boolean que representa se a candidatura foi criada com sucesso. Caso a adição não tenha sucedidos pela falha de um dos requisitos aparece um pop-up do lado do cliente com uma mensagem descritiva da razão da não adição á lista.

- Cancelamento de uma candidatura a um tema

Este caso de uso permite ao aluno remover uma candidatura que realizou previamente através da view que é acedida ao clicar no botão “Minhas Candidaturas”. Para concretizar isto é usado o ListCandidaturasController que ao ser pressionado o botão irá ser obtida a List<CandidaturaDTOs> relativas ao aluno para ser apresentada na view “list-candidaturas.fxml” que irá ser obtida (a lista) recorrendo a um pedido GET ao endpoint /candidaturas/{alunoId}. Nesta também se encontra um botão “Remover” que ao selecionar uma candidatura da lista, sendo passado o id desta ao pedido DELETE ao endpoint /candidaturas/{candidaturaId} que comunica com o CancelarCandidaturaService que passa ao CancelarCandidaturaHandler que está encarregue de tentar a deleção da lista de candidaturas do aluno. Para gerir a concorrência no acesso á lista neste caso optámos por Optimistic Locking lançado as exceções adequadas relacionadas com a alteração da tabela por outra transação (como consideramos estes pontos raros por apenas acontecerem a nosso ver no caso de um Aluno estar logado duas vezes em dispositivos diferentes). O handler devolve a List<Candidatura> atualizada caso a remoção tenha sido um sucesso que é convertida numa List<CandidturaDTO> que é propagada até ao lado do cliente de modo a atualizar a lista de candidaturas na view. Caso a remoção falhe por alguma razão são lançadas exceções que são propagadas até ao cliente após serem traduzidas em HttpStatus e aparece um pop-up com uma mensagem descritiva relativa á razão de falha da remoção. Isto permite que a vista seja atualizada dinamicamente imediatamente após o processamento da remoção da candidatura.

- Submissão de documento de proposta de tese & documento final de tese por parte dos alunos

Estes dois casos de uso são bastante semelhantes em termos de lógica e por isso forma agrupados. Quando um aluno tem tese apos o login é carregado a view “teseDetails.fxml” que apresenta as informações relativas á tese do aluno, como o seu estado e entregas realizadas. Também nela se encontram os botões “Enviar Final”, “Enviar Proposta” e “Logout”. Os dois primeiros botões permitem a realização destes casos de uso. Após pressionar um deles irã ser dada a opção de o aluno escolher um ficheiro que deseja submeter do sistema de ficheiros e após esta seleção será feito ou um pedido POST ao endpoint “/tese/{id}/propostas” ou “/tese/{id}/final” (dependendo do botão pressionado). Este pedido vai ser recebido no RestAPIController do lado do servidor que irá chamar a função respetiva no

SubmeterDocumentoService que vai pedir ao handler que tente criar a Entrega relativa á tese dada pelo id do aluno passado no endpoint ({id}). No caso de ser uma entrega de proposta é devolvida a List<Entrega> com as entregas propostas que o aluno já fez. Esta antes de ser devolvida ao cliente é convertida recorrendo a um EntregaMapper numa List<EntregaDTO> caso o handler tenha conseguido criar a entrega com sucesso. A lista devolvida vai servir para atualizar a view automaticamente após a criação da nova entrega.Caso contrário são lançadas exceções que são propagadas até ao cliente como HttpStatus específicos.

- Web App

- Fragmento topBarMenu

Para aceder às diferentes views que permitem a concretização dos casos de uso mencionados abaixo foi criado um fragmento que é introduzido no topo de cada página quando esta é carregada e que contêm botões que permitem carregar outra página da aplicação web. Estes botões são dinâmicos pois aparecem de acordo com a permissão do utilizador, como por exemplo o empresarial apenas ter um botão de logout e para submeter temas.

- Registo de utilizadores empresariais

Para concretizar este caso de uso foi criado o “registrar_empresarial.html” que é carregado pressionando o botão de “Registar Empresarial” no ecrã de login. Este template permite ao utilizador preencher um formulário com os campos necessários para a criação de um empresarial. Ao submeter este formulário, caso esteja todo preenchido, é mandado um pedido POST para o endpoint /criar-empresarial presente no WebLoginController passando os diversos campos do formulário. Este por sua vez chama o método correspondente do RegistrarEmpresarialService que propaga o pedido para o handler que trata da lógica específica. Caso o registo falhe por algum motivo é lançada uma exceção que é propagada até ao LoginController que altera o login.html de modo a explicitar ao utilizador o motivo da falha.

- Login de utilizadores empresariais & docentes

Este caso de uso é concretizado no “login.html” carregado pelo endpoint “/login” no WebLoginController que permite preencher um formulário com o email e password de um utilizador e ao submeter o formulário estes são passados no pedido POST para o endpoint “/login”, que irá comunicar com o LoginService de maneira a obter as informações do user que está a tentar dar login. Este fala com o LoginHandler que irá tentar encontrar um CriadorTema que tenha o email passado no pedido e caso este exista obtêm informações relativas ao seu tipo e se este é admin de algum mestrado.

Estas informações juntamente com o id do CriadorTema são passadas no DTO RespostaLogin pelo LoginService para o controller de modo a puderem ser usadas durante a sessão atual. Caso esta estrutura chegue ao controller é feito “redirect:/menu” para ser carregada a próxima view. Caso o email do utilizador não exista na base de dados aparece uma mensagem de erro em “login.html”.

- Submissão de temas por parte dos docentes & utilizadores empresariais

Para concretizar estes casos de uso foi criado um ficheiro “submeter_tema.html” que ao ser carregado obtém não só a lista de mestrados como a lista de docentes (apenas no caso de o utilizador ser um empresarial e pretender definir um Docente como orientador interno) de modo a poder apresentá-los no formulário da página onde o utilizador poderá preencher os diferentes campos necessários para criar um tema. Para obter estas duas listas são realizados pedidos ao SubmeterTemaHandler que comunicará com o SubmeterTemaHandler para obter a lista com as entidades convertidas em DTOs recorrendo ao MestradoMapper e CriadorTemaMapper.

Este após ser preenchido e submetido realiza um pedido POST ao endpoint “/submeter-tema” que por sua vez irá tentar criar um tema com os parâmetros passados e dependendo do tipo do utilizador este será um TemaProjeto ou um TemaDissertacao. Para fazer isto ele comunica com o SubmeterTemaService que irá reencaminhar o pedido de criação ao SubmeterTemaHandler. Este irá realizar a logica de negócio por detrás da criação de um novo tema e lançará exceções caso não seja possível a criação do mesmo. Caso levante alguma exceção esta será propagada até ao controller e este irá atualizar a vista com uma mensagem de erro em “submeter_tema.html” que descreva a razão de falha de criação.

- Atribuição dos temas aos alunos (da parte do administrado)

Para concretizar este caso de uso foi criado o “atribuir_tema.html” que ao ser carregado relaiza um pedido GET ao endpoint “”/atribuir-tema” no WebAtribuicaoTemaController obtém a informação necessária relativa a alunos do seu mestrado que ainda não têm um tema atribuído (não estão a realizar tese), os temas que são compatíveis com o mestrado que o utilizador administra e a lista de docentes (pois ao atribuir um TemaProjeto pode ser necessário escolher um docente como orientador caso este ainda não esteja definido). Esta informação é obtida comunicando com o AtribuicaoTemaService que irá em primeira instância obter o mestrado do administrador e posteriormente a List<Tema> e List<Aluno> do AtribuicaoTemaHandler e a List<Docente> do ListarDocentesHandler e converte-os em listas dos respetivos DTOs recorrendo ao CriadorTemaMapper, TemaMapper e AlunoMapper. Estas listas são incluídas no AtribuicaoTemaInfo pelo servoço de modo a serem todas passadas ao controller de uma vez.

Após esta informação ser carreada na vista o utilizador consegue dar “match” entre um aluno e um tema selecionando-os em cada uma das listas. Ao clicar no botão presente vai permitir a submissão do formulário que despoleta um pedido POST ao endpoint “atribuir-tema” passando os ids do aluno e do tema juntamente com a info obtida previamente. O controller após detetar este pedido vai obter a informação doserviço e handler de se o tema selecionado tem orientador e caso não tenha é feito “redirect:/escolher_orientador” que apresenta a previamente obtida lista de docentes para seleção e após a mesma é feito o pedido de criação de um Tese ao serviço passando o id do aluno, tema e docente (caso tenha sido selecionado). Este chega ao AtribuirTemaHandler que irá realizar a lógica de criação de uma Tese por parte do Aluno relativamente ao Tema e retorna á view “atribuir_tema”. Caso algo corra mal é lançada uma exceção pelo handler que propaga até ao controller e carrega a vista com uma mensagem de erro descritiva do que correu mal.

- Marcação da defesa da proposta de tese por parte do orientador da tese

Para a realização deste caso de uso foi criada uma vista “marcar_defesa_proposta.html” que é inicialmente carregada fazendo um GET ao endpoint “/marcar-defesa-proposta” no WebMarcacaoDefesaController pedindo a informação das salas na base de dados e dos docentes, sendo estas obtidas através de pedidos ao MarcacaoDefesaService que comunica com os handlers MarcacaoDefesaHandler e ListarDocentesHandler para obter as mesmas, estando o serviço encarregue de fazer a conversão em DTOs recorrendo a SalaMapper e CriadorTemaMapper.

Após isto a view é carregada como um formulário que permite: escolher se a defesa é remota ou presencial, e caso seja esta última efetuar a escolha da sala, escolher a data da defesa, escolher a hora da defesa e escolher quem será o arguente (que pode diferir de defesa para defesa). Após o preenchimento e submissão do formulário é feito um pedido POST ao endpoint “/marcar-defesa-proposta” que irá tentar criar uma Defesa associada a um salaId numa certa data e hora com presença do orientador da tese e o arguente selecionado. Para isto é feito um pedido ao MaracrDefesaService que reencaminha para o MarcarDefesaHandler que irá verificar todos os requisitos para a defea ser marcada nas condições passadas, desde a sala estar disponível ao arguente e orientadores estarem disponíveis nessa data. Para este efeito é usado Pessimistic Locking e são feitos PESSIMISTIC_WRITE locks nos docentes e sala de modo que estes não sejam modificados enquanto verificamos a disponibilidade dos mesmos. Caso estes e outros requisitos sejam cumpridos a Defesa é criada e caso contrário são lançadas exceções que serão propagadas até ao controller que as tratará e colocará como uma mensagem de erro na view.

- Marcação da defesa final de tese por parte do orientador da tese e nomeação do júri

Para a realização deste caso de uso foi criada uma vista “marcar_defesa_final.html” que é inicialmente carregada fazendo um GET ao endpoint “/marcar-defesa-final” e segue um raciocínio igual ao mencionado no caso de uso anterior.

Após isto a view é carregada da mesma maneira que foi mencionada no caso de uso anterior e é feita a mesma lógica, sendo a única diferença a necessidade de selecionar também um docente que atue como presidente de júri, sendo também a sua disponibilidade verificada no MarcarDefesaHandler.

- Registo da nota de defesa da proposta da tese, por parte do orientador da tese & da nota da defesa final da tese por parte do presidente de júri

Para a realização deste caso de uso foi criada uma vista “registar_nota_defesa.html” que é ao ser carregada despoleta um pedido GET ao endpoint “registar-nota-defesa” no WebRegistarNotaDefesaController que irá obter uma List<DefesaDTO> que contém não só as defesas que o docente orienta como aquelas em que participou como presidente de júri e que estejam em condições de serem avaliadas. Para obter esta lista o controller comunica com o RegistarNotaDefesaService que obtém a List<Defesa> com as defesas nas condições descritas anteriormente, sendo convertidas em DefesaDTOs pelo RegistarNotaDefesaService recorrendo ao DefesaMapper.

Após a view ser carregada aparece a lista que expões ao utilizador a defesas que podem ser avaliadas e permite que uma seja selecionada e escrito o inteiro de 0 a 20 que corresponde á nota a registar. Com a submissão do formulário é feito um pedido POST ao endpoint “/registar-nota-defesa” no WebRegistarNotaDefesaController que recebe o id da defesa e a nota a atribuir a esta. O controller comunica então com o RegistarNotaDefesaService que pedirá ao RegistarNotaDefesaHandler que tente registar a nota, sendo realizado as verificações necessárias para verificar se este registo é válido (se quem está a registar é o orientador/presidente). Para evitar lost updates aplicamos um Pessimistic Locking usando um PESSIMISTIC_WRITE lock na defesa de modo a ter controlo total sobre a Defesa antes de atribuir a nota á defesa.

Caso o registo não seja um sucesso são lançadas exceções que propagam até ao controller e que são colocadas na view descrevendo a razão que impossibilitou a avaliação da defesa.

- Recolha de estatísticas sobre a taxa de sucesso dos alunos

De modo a puderem ser apresentadas estatísticas relativas á taxa de sucesso dos alunos aos docentes foi criada a view “estatísticas.html” que apresenta as seguintes estatísticas: numero de alunos a realizar tese no ano atual, número de alunos chumbados na defesa final no ano atual, número de alunos que passaram na defesa final este ano e a média das notas das defesas finais no ano atual e as mesmas

estatísticas mas na perspetiva global (todos os alunos que alguma vez realizaram tese que estejam na base de dados). Para isto é feito um pedido GET ao endpoint "/estatísticas" presente no WebEstatisticasController que comunica com o EstatisticasService para obter as métricas mencionadas acima passando essa responsabilidade para o EstatisticasHandler que as guarda no EstatisticasInfo DTO de modo a ser propagado primeiro para o serciço e depois para o controller para que este possa carregá-lo na view.