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

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

03/04/2024

Arquitetura em Camadas

Para este projeto foi implementado código para a camada de *Business* mais específicamente, para as subcamadas de *Domain Model*, através da criação das classes do domínio, collections e objetos, sendo estas anotadas com *@Entity*, e para a camada de *Persistence*, por meio dos repositórios, ORM e queries SQL.

Modelo de Domínio

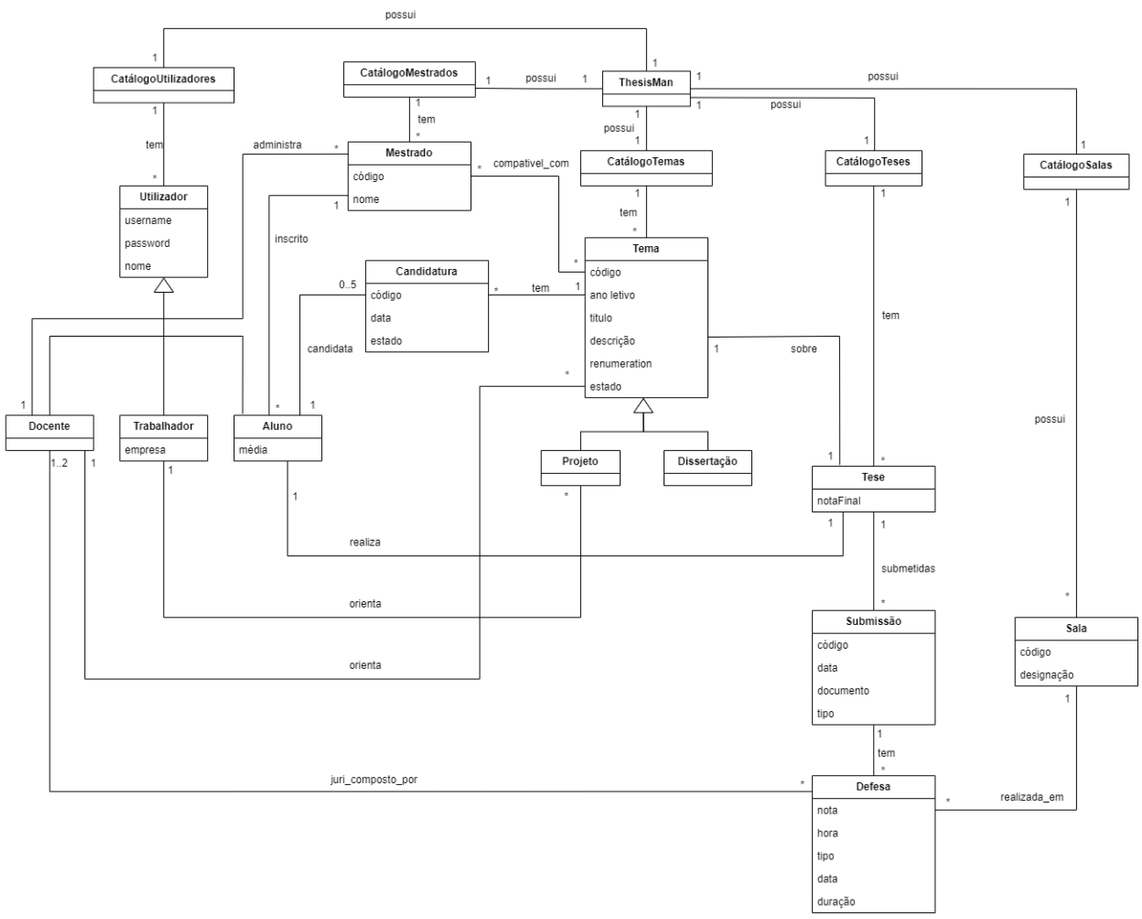
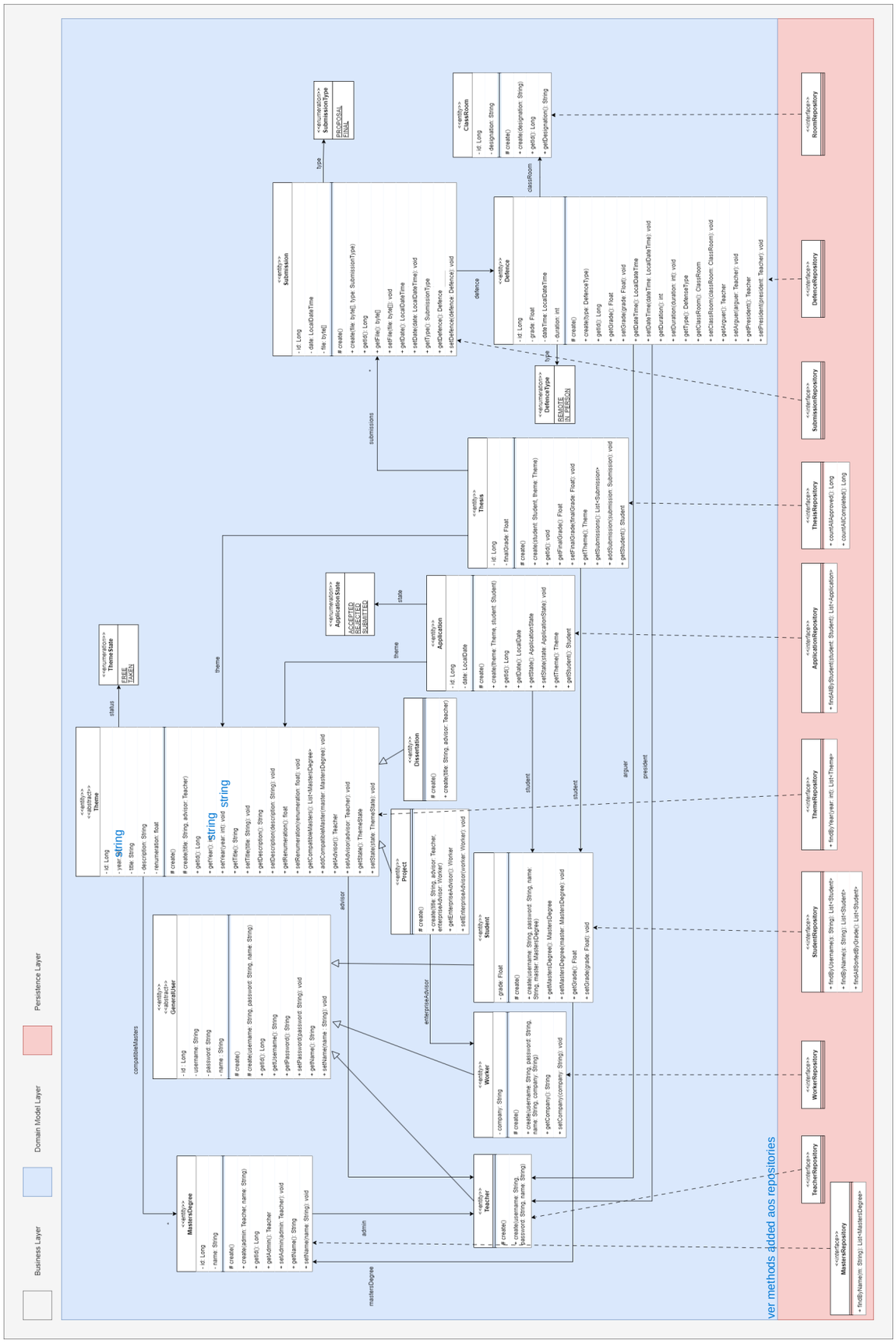
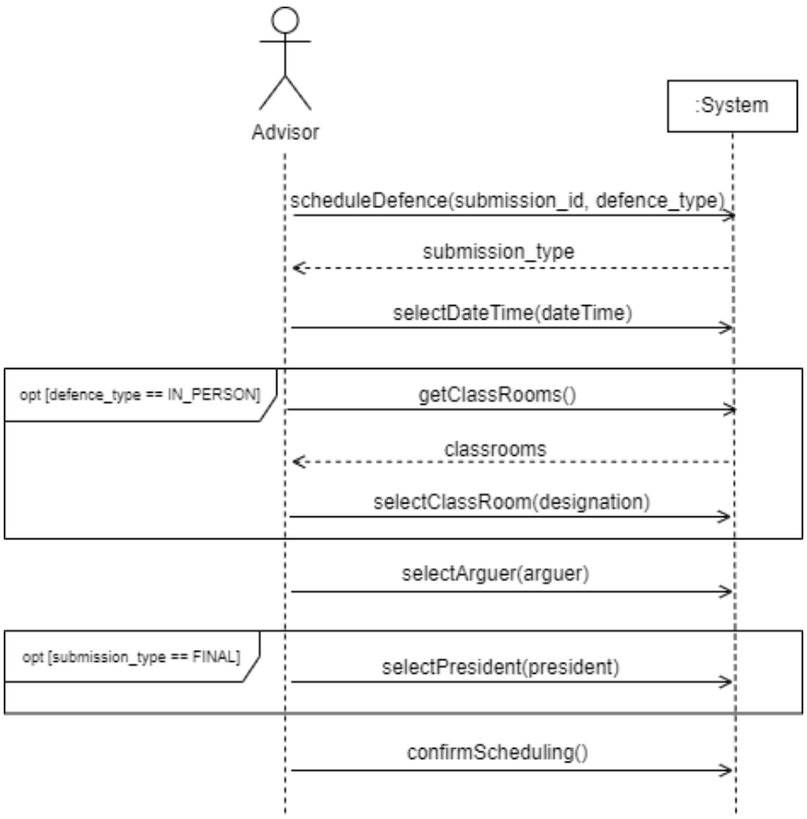


Diagrama de Classes



Nota: Visto que a implementação dos handlers e dos casos de uso não é o objetivo desta fase, os mesmos nao constam no diagrama de classes.

Diagrama de Sequência do Sistema (SSD) - UC K



Mapeamento JPA e Decisões Tomadas

Entidades

GeneralUser - Student, Teacher, Worker

A classe *GeneralUser* é uma classe abstrata usada para representar um utilizador do sistema *ThesisMan*, esta possui três classes concretas que a estendem, a classe *Student*, que representa alunos, a *Teacher*, que representa docentes, e a classe *Worker* que representa trabalhadores de empresas.

De forma a mapear a herança destas classes para a base de dados através de JPA decidimos usar a estratégia de *TABLE_PER_CLASS*, criando assim apenas três tabelas na base de dados, uma por cada classe concreta, visto que não seria necessário ter uma tabela para a classe abstrata.

Considerámos que esta foi uma boa opção pois assim, não só é possível ter uma melhor separação de cada entidade na base de dados, como também permite que cada classe concreta possua atributos próprios, e ainda evita desperdiçar espaço nas tabelas. Caso num futuro se quisessem adicionar mais atributos, uma estratégia do tipo *SINGLE_TABLE* tornar-se-ia demasiado dispendiosa em termos de espaço, dado que a tabela ficaria muito esparsa devido ao número de atributos adicionais.

Esta estratégia, *TABLE_PER_CLASS*, também apresenta vantagens comparativamente à *JOINED*, uma vez que, não guarda uma tabela para a classe abstrata, não sendo necessário realizar *JOINS* quando se pretende aceder a atributos que pertencentes à super classe.

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class GeneralUser {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  @NonNull private String username;

  @NonNull private String password;

  @NonNull private String name;
```

- Na classe abstrata *GeneralUser* utilizámos a anotação *@Entity* já que representa uma entidade da aplicação.
- Esta faz uso da anotação *@Inheritance* para escolher a estratégia *TABLE_PER_CLASS* como referido acima.
- Possui um atributo *id* do tipo *Long* com a anotação *@Id*, dado que se trata de uma primary key da tabela, fazendo uso do padrão *Identity Field*. Sendo este gerado automaticamente pela base de dados através do *@GeneratedValue* com a estratégia *AUTO*, com a qual a base de dados decide a melhor estratégia para o gerar.

```java
@Entity
public class Student extends GeneralUser {

  @NonNull @ManyToOne private MastersDegree mastersDegree;

  private Float grade;
```

- Na classe *Student*, que estende a classe *GeneralUser*, utilizámos a anotação *@Entity* já que representa um aluno e considerámos que deveria existir uma tabela de alunos.
- Esta possui um atributo *mastersDegree* que representa o mestrado onde o aluno está inscrito. Este atributo é anotado com *@NonNull* devido a estar sempre presente, este ainda está assinalado com a anotação *@ManyToOne* uma vez que muitos alunos estão inscritos no mesmo mestrado e um aluno apenas tem um mestrado (mapeado através de Foreing Key Mapping).
- Possui um atributo *grade* que representa a média atual do aluno.

```java
@Entity
public class Teacher extends GeneralUser {
```

- A classe *Teacher* que estende a classe *GeneralUser* utilizámos a anotação *@Entity* já que representa uma entidade da aplicação, sendo um Docente que irá servir de orientador de teses, daí necessitar da sua própria tabela.

```java
@Entity
public class Worker extends GeneralUser {

  @NonNull private String company;
```

- A classe *Worker*, que estende a classe *GeneralUser*, está anotada com *@Entity* já que representa uma entidade da aplicação, sendo um Trabalhador de uma empresa que irá servir de orientador externo de projetos de teses, merecendo ter a sua própria tabela.
- Esta possui ainda um atributo adicional *company* que contém o nome da empresa onde este funcionário trabalho, anotada com *@NonNull*, de modo a marcar a obrigatoriedade da sua presença.

Theme - Project, Dissertation

A classe *Theme* é uma classe abstrata usada para representar o tema de uma tese, esta possui duas classes concretas que a estendem, a classe *Project*, que representa temas de projetos submetidos por empresas, e a *Dissertation*, que representa temas de dissertação submetidos por docentes.

De forma a mapear a herança destas classes para a base de dados através de JPA decidimos usar a estratégia de *SINGLE_TABLE*, uma vez que a herança não é muito profunda e as classes concretas não apresentam muitos atributos adicionais (a *Dissertation* não possui atributos específicos próprios e a classe *Project* apenas tem um atributo *Worker*), assim a estratégia escolhida é adequada, uma vez que, como não é usado muito espaço adicional devido ao baixo número de atributos e melhora a eficiência das queries pois apenas é necessário pesquisar em uma tabela.

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Theme {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private int year = Year.now().getValue();

  @NonNull private String title;

  private String description;

  @NonNull @ManyToMany private List<MastersDegree> compatibleMasters = new ArrayList<>();

  @NonNull @ManyToOne private Teacher advisor;

  private @NonNull @Enumerated(EnumType.STRING) ThemeState state = ThemeState.FREE;

  private float remuneration;
```

- Na classe abstrata *Theme* utilizámos a anotação *@Entity* já que representa uma entidade da aplicação.
- Usámos *Long* como id, sendo este gerado na base de dados pela estratégia *AUTO*, permitindo desta forma, que o conector à base de dados decida a estratégia adequada para gerar sequências únicas de ids dependendo da base de dados específica.
- O atributo *compatibleMasters* contém a lista de mestrados compatíveis com este tema. Foi escolhida uma anotação *@ManyToMany* pelo facto de um tema poder ter vários mestrados compatíveis e um mestrado poder ser compativel com vários temas.
- Já o atributo *advisor* que representa o docente orientador deste tema foi anotado com *@ManyToOne*, havendo um mapeado através de Foreing Key Mapping, podendo assim, o mesmo docente orientar vários temas, mas um tema apenas possui um docente que o orienta.
- O atributo *state* do tipo enumerado *ThemeState* para representar o estado da tema, isto é, se este já foi ou não atribuído a algum aluno, este é anotado também com *@NonNull* e, devido a ser um enumerado com *@Enumerated* sendo escolhida a estratégia de o armazenar como *STRING* pois é mais estável do que guardar como *ORDINAL* e não havendo requisitos muito estritos de desempenho neste momento, ainda por ser mais provável alterar a ordem dos elementos do enumerado do que a designação dos mesmos.

```java
@Entity
public class Dissertation extends Theme {
```

- A classe *Dissertation* foi marcada com a anotação *@Entity* já que representa uma entidade da aplicação.
- Não possui nenhum atributo adicional relativamente à classe *Theme*, tendo apenas um *toString* e o construtor com os mesmos parâmetros.

```java
@Entity
public class Project extends Theme {

  @NonNull @ManyToOne private Worker enterpriseAdvisor;
```

- A classe *Project* foi marcada com a anotação *@Entity* já que representa uma entidade da aplicação.

- Para além do que herda da classe *Theme*, possui ainda um atributo *enterpriseAdvisor* para representar o orientador externo da empresa que submeteu o tema, este é marcado com *@NonNull* pois é necessário que esteja sempre presente, e com *@ManyToOne* (mapeado através de Foreing Key Mapping), podendo o mesmo orientador orientar vários projetos, mas cada projeto apenas possui um orientador externo.

MastersDegree

```java
@Entity
public class MastersDegree {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private @NonNull String name;
  private @NonNull @ManyToOne Teacher admin;
```

- Na classe *MastersDegree* utilizámos a anotação *@Entity* já que representa uma entidade da aplicação e considerámos que seria necessário ter uma tabela de mestrados.
- Usámos *Long* como id, sendo este gerado na base de dados pela estratégia *AUTO* com *@GeneratedValue*, permitindo desta forma, que o conector à base de dados decida a estratégia adequada para gerar sequências únicas de ids dependendo da base de dados específica.
- O atributo *name* contém o nome deste mestrado.
- Já o atributo *admin* contém o administrador, que é o coordenador do mestrado, anotado com *@ManyToOne* pois o mesmo docente pode administrar diversos mestrados, mas um mestrado só possui um administrador, existindo aqui um mapeamento através de Foreing Key Mapping.

Application

```java
@Entity
public class Application {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  @NonNull
  @Column(columnDefinition = "DATE")
  private LocalDate date = LocalDate.now();

  @NonNull
  @Enumerated(EnumType.STRING)
  private ApplicationState state = ApplicationState.SUBMITTED;

  @NonNull @ManyToOne private Theme theme;

  @NonNull @ManyToOne private Student student;
```

- A classe *Application* represeta uma candidatura por parte de um aluno a um tema de tese, daí ter sido usada a anotação *@Entity* para representar ma tabela na base de dados.
- Possui um *id* do tipo *Long* gerados, também automaticamente com *@GeneratedValue*, pela estratégia *AUTO*.
- O atributo *date* está anotado com *@NonNull* devido a este ser necessário estar sempre presente e possui também uma anotação de *@Column* com *columnDefinition="DATE"* de forma a indicar à base de dados que pretendemos apenas guardar a data, não sendo necessária a hora.
- A classe possui também um atributo *state* do tipo enumerado *ApplicationState* para representar o estado da candidatura, este está também anotado com *@NonNull*. Ainda, devido a ser um enumerado, com *@Enumerated*, tendo sido escolhida *STRING* como estratégia de armazenamento, já que é mais estável do que a *ORDINAL* e não havendo requisitos muito estritos de desempenho neste momento, ainda por ser mais provável alterar a ordem dos elementos do enumerado do que a designação dos mesmos.

- O atributo *theme* representa a referência para o tema a que esta candidatura se refere. Este será mapeado através de Foreing Key Mapping tendo como anotação *@ManyToOne*, podendo haver várias candidaturas para um mesmo tema, contudo, a candidatura apenas pode possuir um tema. Ainda foi anotado com *@NonNull* pela obrigatoriedade de estar sempre presente.
- De forma semelhante, o atributo *student* representa o aluno que realizou esta candidatura, sendo também mapeado através de Foreing Key Mapping com a anotação *@ManyToOne*, pelo motivo de um aluno poder fazer várias candidaturas mas a candidatura apenas pertencer a uma aluno. Também é anotado com *@NonNull* assinalando a necessidade de haver um aluno associado à candidatura.

ClassRoom

```java
@Entity
public class ClassRoom {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  @NonNull private String designation;
```

- A classe *ClassRoom* faz uso da anotação *@Entity* para representar uma tabela de salas de aulas para a realização das defesas de teses.
- Possui um atributo *id* do tipo *Long* anotado com *@Id* e auto-gerado pela base de dados com *@GeneratedValue* com a estratégia *AUTO*.
- Possui ainda um atributo *designation* (por exemplo: 8.2.30) anotado com *@NonNull*, uma vez que, a coluna respetiva da base de dados é de preenchimento obrigatório.

Defence

```java
@Entity
public final class Defence {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private Float grade;

  @Column(columnDefinition = "TIMESTAMP")
  private LocalDateTime dateTime;

  private int duration = 60;

  @NonNull
  @Enumerated(EnumType.STRING)
  private DefenceType type = DefenceType.IN_PERSON;

  @ManyToOne private ClassRoom classRoom;
  @ManyToOne private Teacher arguer;
  @ManyToOne private Teacher president;
```

- A classe *Defence* representa uma defesa de uma submissão, e faz uso da anotação *@Entity* para representar uma tabela de defesas de submissões de teses.
- Possui um atributo *id* do tipo *Long* anotado com *@Id* e auto-gerado pela base de dados com *@GeneratedValue* com a estratégia *AUTO*.
- Ainda, um atributo *dateTime* que representa a data e hora da defesa, assim este foi anotado com *@Column* tendo *columnDefinition = "TIMESTAMP"* sendo o *TIMESTAMP* a indicação para guardar a data e a hora na base de dados.

- A classe possui também um atributo de duração de modo a indicar a duração em minutos da respetiva defesa.
- Um atributo *grade* que representa a nota dada no final da defesa, que pode ser null até esta ser atribuída, daí o tipo *Float*.
- Um atributo *type* que indica se a defesa é presencial ou remota, e como este necessita estar sempre presente foi marcado com *@NonNull*, ainda como é um enumerado foi marcado com *@Enumerated* com a estratégia de *STRING*, em vez de *ORDINAL*, devido ao facto de possuir uma maior estabilidade e de neste momento não afetar o desempenho da aplicação, e ainda por ser mais provável alterar a ordem dos elementos do enumerado do que a designação dos mesmos.
- Um atributo *classRoom* que representa a sala onde se realiza a defesa, caso seja presencial, sendo mapeado através de Foreing Key Mapping com a anotação *@ManyToOne*, dado que podem existir várias defesas na mesma sala, mas uma defesa apenas tem uma sala atribuída.
- Um atributo *arguer* que representa o arguente da defesa, mapeado através de Foreing Key Mapping através da anotação *@ManyToOne*, sendo que um arguente pode estar presente em várias defesas, contudo, uma tese apenas pode possuir um arguente.
- Um atributo *president* que representa o presidente do juri para essa defesa, sendo mapeado através de Foreing Key Mapping com a anotação *@ManyToOne*, uma vez que, um presidente pode presidir várias defesas, no entanto, uma defesa apenas tem um presidente.

Submission

```java
@Entity
public class Submission {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  @NonNull
  @Column(columnDefinition = "TIMESTAMP")
  private LocalDateTime date = LocalDateTime.now();

  @NonNull @Lob private byte[] file;

  @NonNull
  @Enumerated(EnumType.STRING)
  private SubmissionType type = SubmissionType.PROPOSAL;

  @NonNull
  @OneToOne(cascade = CascadeType.ALL)
  private Defence defence;
```

- A classe *Submission* representa uma proposta de submissão de uma tese, daí ser marcada com a anotação *@Entity*, de modo a ser criada uma tabela de submissões na base de dados.
- Possui também um atributo *id* do tipo *Long* com *@Id* tratando-se da primary key, e anotado com *@GeneratedValue* com a estratégia *AUTO*.
- Um atributo *type* e que pode ser do tipo *PROPOSAL* ou *FINAL*, enumerado este marcado com a anotação *@Enumerated* e com o tipo *STRING*, que apesar de ser menos eficiente é mais estável, comparativamente com a estratégia de *ORDINAL*, sendo a escolhida por neste momento não afetar o desempenho da aplicação, e ainda por ser mais provável alterar a ordem dos elementos do enumerado do que a designação dos mesmos.
- Um atributo date com a anotação *@Column* com *columnDefinition="TIMESTAMP"* pois se pretende-se guardar na base de dados a data e hora da submissão do documento de proposta.
- O conteúdo do documento de proposta está no atributo *file* marcado com a anotação *@Lob* já que está destinado a guardar o conteúdo do ficheiro com a proposta na base de dados.
- Ainda existe um atributo *defense* com uma ligação *@OneToOne* pois cada submissão tem a sua respetiva defesa, e cada defesa apenas corresponde a uma submissão, este atributo ainda está assinalado com *cascade = CascadeType.ALL* de modo a propagar todas as alterações do mesmo para as entidades associadas.

Thesis

```java
@Entity
public class Thesis {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private Float finalGrade;

  @NonNull @OneToOne private Theme theme;

  @NonNull
  @OneToMany(cascade = CascadeType.ALL)
  @JoinColumn
  private List<Submission> submissions = new ArrayList<>();

  @NonNull @OneToOne private Student student;
```

- A classe *Thesis* é uma *@Entity* que é mapeada numa tabela na base de dados.
- Possui um *id* do tipo *Long* como primary key anotado com *@Id @GeneratedValue* com a estratégia *AUTO*.
- Um atributo *theme* tem uma ligação do tipo *@OneToOne*, pois cada tese apenas tem um tema associado, e um tema apenas pertence a uma tese. Este atributo ainda está assinalado com *CascadeType.ALL* de modo a propagar todas as alterações do mesmo para as entidades associadas.
- Um atributo *submissions* que representa a lista de submissões, que inclui todas as submissões propostas bem como a submissão final, relativas a uma tese, marcado com a anotação *@OneToMany*, pois uma tese pode ter mais do que uma submissão, e várias submissões podem pertencer à mesma tese, e com cascade do tipo *ALL*, de modo a propagar todas as laterações à entidade associada. Este é ainda anotado com *@JoinColumn* de forma a usar Foreign Key Mapping, colocando uma foreign key na tabela das submissões para identificar a tese a que pertence, evitando a criação de uma tabela para a associação.
- Um atributo *student* com uma ligação *@OneToOne*, pois cada estudante apenas tem uma tese associada, e uma tese só pertence a um estudante.

Repositórios

Global

As interfaces dos repositórios estendem a interface *JpaRepository* deste modo não foi necessário implementar métodos como o *findById* e *delete*. Para realizar as queries nestes, foi usado JPQL, de forma a ficar independente de dialetos diferentes de SQL.

ApplicatonRepository

```java
public interface ApplicationRepository extends JpaRepository<Application, Long> {

  /**
   * Finds all applications from a given student.
   *
   * @param student The student to search applications.
   * @return A list containing all applications from the given user.
   */
  @Query("SELECT app FROM Application app WHERE app.student = :student")
  List<Application> findAllByStudent(@Param("student") Student student);
}
```

Na interface deste repositório definimos um *findAllByStudent* de modo a conseguir executar uma query que obtenha todas as candidaturas que pertencem a um determinado estudante.

StudentRepository

```java
public interface StudentRepository extends JpaRepository<Student, Long> {

  /**
   * Finds the student with the given username.
   *
   * @param s The student username to search.
   * @return A list containing all users from the given username.
   */
  @Query("SELECT student FROM Student student WHERE student.username = :s")
  List<Student> findByUsername(@Param("s") String s);

  /**
   * Finds the student with the given name.
   *
   * @param s The student name to search.
   * @return A list containing all users from the given name.
   */
  @Query("SELECT student FROM Student student WHERE student.name LIKE %:s%")
  List<Student> findByName(@Param("s") String s);

  /**
   * Locates all students sorted in descending order by grade.
   *
   * @return A list containing all students in descending order by grade.
   */
  @Query("SELECT student FROM Student student ORDER BY student.grade DESC")
  List<Student> findAllSortedByGrade();
}
```

Na interface deste repositório definimos três métodos: *findByUsername*, este método vai procurar um estudante através do seu nome de utilizador; *findByName*, que vai procurar um estudante pela semelhança com o nome pretendido; e *findAllSortedByGrade* que retorna todos os alunos ordenados por média de forma descendente.

ThemeRepository

```java
public interface ThemeRepository extends JpaRepository<Theme, Long> {

  /**
   * Finds all themes for a given year.
   *
   * @param year The year to search theme.
   * @return A list containing all themes from the given year.
   */
  @Query("SELECT theme FROM Theme theme WHERE theme.year = :year")
  List<Theme> findByYear(@Param("year") int year);
}
```

Na interface deste repositório definimos um *findByYear* para ser possível executar uma query que obtenha todas os temas de um respetivo ano.

ThesisRepository

```java
public interface ThesisRepository extends JpaRepository<Thesis, Long> {

  /**
   * Find all thesis that have a final grade that is greater than or equal to 9.5.
   *
   * @return A list containing all approved thesis.
   */
  @Query(
      "SELECT COUNT(*) FROM Thesis thesis WHERE thesis.finalGrade IS NOT NULL AND thesis.finalGrade >= 10")
  Long countAllApproved();

  /**
   * Find all thesis that are have a final grade, that is, that were evaluated.
   *
   * @return A list containing all evaluated thesis.
   */
  @Query("SELECT COUNT(*) FROM Thesis thesis WHERE thesis.finalGrade IS NOT NULL")
  Long countAllCompleted();
}
```

Na interface deste repositório definimos um método *countAllApproved* que executa uma query que conta o número de teses em que a nota final que o estudante obteve foi maior ou igual a 10, logo os aprovados.
E ainda, um método *countAllCompleted* que conta todas as teses que possuem nota final, isto é, as teses para as quais já se realizou uma defesa final.

MastersRepository

```java
public interface MastersRepository extends JpaRepository<MastersDegree, Long> {

  /**
   * Finds the master's degree with the given name.
   *
   * @param m The name of the master to search
   * @return A list containing all masters from the given name.
   */
  @Query("SELECT master FROM MastersDegree master WHERE master.name LIKE %:m%")
  List<MastersDegree> findByName(@Param("m") String m);
}
```

Na interface deste repositório definimos um método, *findByName*, este método vai procurar um mestrado que contenha o nome pretendido.

ClassRoomRepository, DefenseRepository, SubmissionRepository, TeacherRepository, WorkerRepository

```java
public interface ClassRoomRepository extends JpaRepository<ClassRoom, Long> {}
```

```java
public interface DefenceRepository extends JpaRepository<Defence, Long> {}
```

```java
public interface SubmissionRepository extends JpaRepository<Submission, Long> {}
```

```java
public interface TeacherRepository extends JpaRepository<Teacher, Long> {}
```

```java
public interface WorkerRepository extends JpaRepository<Worker, Long> {}
```

Nestes repositórios/interfaces apenas estendemos a interface *JpaRepository* que já implementa o método *findById* necessário.

Demo

Foi criada um clase *ThesisManDemo* que contém uma *main* com a lógica para iniciar a aplicação. Esta representa uma demo que cria dados e os insere na base de dados, usando o repositórios, bem como também realiza queries através dos repositórios para os dados guardados.