# Project Class Descriptions

Trif Oriana

## 1 Entities

### 1.1 UserEntity

```java
@Entity
@Table(name = "APP_USER")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "first_name", nullable = false)
    private String firstname;
    @Column(name = "last_name", nullable = false)
    private String lastname;
    @Column(name = "email", nullable = false, unique = true)
    private String email;
    @Column(name = "password", nullable = false)
    private String password;
    @OneToMany(mappedBy = "users", cascade = CascadeType.ALL)
    private List<CourseEntity> courses;
}
```

## Annotations

- `@Entity`: Indicates that this class is an entity that will be managed by the JPA (Java Persistence API).
- `@Table(name = "APP_USER")`: Specifies the name of the database table associated with this entity.
- `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Builder`: Lombok annotations for generating boilerplate code such as getters, setters, constructors, and builder methods.
- `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Defines the primary key `id` and its generation strategy.

- `@Column`: Specifies the mapping between entity attributes and database columns, including column names, nullability, and uniqueness.
- `@OneToMany(mappedBy = "users", cascade = CascadeType.ALL)`: Indicates a one-to-many relationship with the `CourseEntity` class, where each user can have multiple courses. The `mappedBy` attribute specifies the field in the `CourseEntity` class that owns the relationship.

## Fields

- `id`: Represents the unique identifier for each user.
- `firstname`, `lastname`: Store the user's first and last names.
- `email`: Stores the user's email address, which must be unique.
- `password`: Stores the user's password.
- `courses`: Represents the list of courses associated with the user. This field establishes a one-to-many relationship with the `CourseEntity` class.

Overall, the `UserEntity` class serves as a blueprint for creating and managing user records in the application's database. It encapsulates user-related data and relationships with other entities, facilitating data persistence and retrieval operations.

### 1.2 CourseEntity

```
@Entity
@Table(name = "COURSE")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CourseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "title", nullable = false)
    private String title;
    @Column(name = "description", nullable = false)
    private String description;
    @Column(name = "rating", nullable = false)
    private int rating;
    @Column(name = "assignments", nullable = false)
    private String assignments;
    @Enumerated(EnumType.STRING)
    @Column(name = "level", nullable = false)
    private Level level;
    @ManyToOne
    @JoinColumn(name = "user_id")
    private UserEntity users;
}
```

## Annotations

- `@Entity`: Indicates that this class is an entity that will be managed by the JPA (Java Persistence API).
- `@Table(name = "APP_USER")`: Specifies the name of the database table associated with this entity.
- `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Builder`: Lombok annotations for generating boilerplate code such as getters, setters, constructors, and builder methods.
- `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Defines the primary key `id` and its generation strategy.
- `@Column`: Specifies the mapping between entity attributes and database columns, including column names, nullability, and uniqueness.
- `@OneToMany(mappedBy = "users", cascade = CascadeType.ALL)`: Indicates a one-to-many relationship with the `CourseEntity` class, where each user can have multiple courses. The `mappedBy` attribute specifies the field in the `CourseEntity` class that owns the relationship.

## Fields

- `id`: Represents the unique identifier for each user.
- `firstname`, `lastname`: Store the user's first and last names.
- `email`: Stores the user's email address, which must be unique.
- `password`: Stores the user's password.
- `courses`: Represents the list of courses associated with the user. This field establishes a one-to-many relationship with the `CourseEntity` class.

# 2 DTOs

The `CourseDto` class encapsulates attributes representing a course, including its title, description, rating, assignments, and level. On the other hand, the `UserDto` class represents user-related data, containing fields for the user's first name, last name, email, and password. Both DTO classes utilize Lombok annotations (`@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Builder`) to automatically generate boilerplate code for getter/setter methods, constructors, and builder methods, reducing code verbosity and enhancing readability.

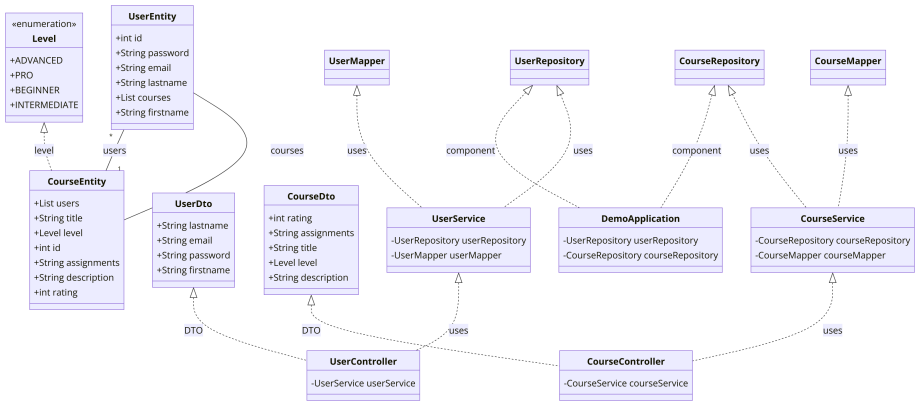


Figure 1: Night detail view

# 3 Services

The `CourseService` class is responsible for managing course-related operations in the application. This service interacts with the database through a `CourseRepository` and utilizes a `CourseMapper` for mapping between DTOs (Data Transfer Objects) and entities.

The `findAll` method retrieves all courses from the database, converts them to DTOs using the mapper, and returns a list of `CourseDto` objects.

The `addCourse` method accepts a `CourseDto` object, creates a corresponding `CourseEntity`, saves it to the database using the repository, and returns the saved course as a DTO.

The `findById` method retrieves a course by its ID from the database. If the course is found, it's converted to a DTO and returned; otherwise, an `EntityNotFoundException` is thrown.

The `updateCourse` method updates the rating of a course identified by its title. It retrieves the course from the repository, modifies its rating, saves the updated entity, and returns the updated course as a DTO.

The `deleteCourse` method deletes a course from the database based on its title. If the course exists, it's deleted; otherwise, a `RuntimeException` is thrown indicating that the course was not found.

# 4 Conclusion

This document has outlined the primary classes involved in a Spring Boot project designed for managing users and courses. Each class is equipped with annotations that define its role within the project, including entity definition, DTOs for data transfer, and controllers for handling HTTP requests.