

**<<PROJECT ADEMY - ONLINE COURSE DRAW LEARNING PLATFORM >>**

**Software Design Specification**

– HCM, November 2023 –

**Record of changeS**

| **Date** | **A\* M, D** | **In charge** | **Change Description** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

\*A - Added M - Modified D - Deleted

**Table of Contents**

[I. Overview 5](#_heading=h.gjdgxs)

[1. Code Packages 5](#_heading=h.30j0zll)

[2. Database Design 6](#_heading=h.1fob9te)

[a. Database Schema 6](#_heading=h.3znysh7)

[b. Table Description 6](#_heading=h.2et92p0)

[II. Code Designs 9](#_heading=h.tyjcwt)

[1. Authentication/Customer Login 9](#_heading=h.3dy6vkm)

[a. Class Diagram 9](#_heading=h.1t3h5sf)

[b. Class Specifications 10](#_heading=h.4d34og8)

[c. Sequence Diagram 12](#_heading=h.1gdu0474sut4)

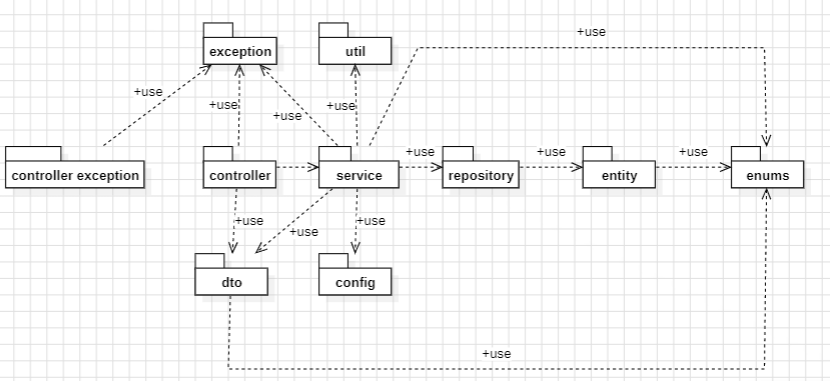
[2. Add to cart/Buy course 13](#_heading=h.1wf3o716ev2q)

[a. Class Diagram 13](#_heading=h.c1bwpiqcxdz7)

# I. Overview

## 1. Code Packages

*[Provide the package diagram for each sub-system. The content of this section including the overall package diagram, the explanation, package and class naming conventions in each package. Please see the sample & description table format below (****please note: package names don’t follow Java package naming convention yet****)]*



***Package descriptions***

| ***No*** | ***Package*** | ***Description*** |
| --- | --- | --- |
| ***01*** | ***config*** | ***contains configuration files or classes responsible for handling application settings and configurations.*** |
| ***02*** | ***controller*** | ***responsible for handling incoming requests from clients (e.g., web browsers) and controlling the flow of the application.*** |
| ***03*** | ***controller.exception*** | ***contain classes related to exception handling in the application's controllers.*** |
| ***04*** | ***exception*** | ***contains custom exception classes and error-handling logic used throughout the application.*** |
| ***05*** | ***dto*** | ***contains classes that define data structures for transferring data between different parts of the application.*** |
| ***06*** | ***entity*** | ***contains classes that represent data models or entities.*** |
| ***07*** | ***enums*** | ***contain enumerated types (enums) that define a fixed set of constant values.*** |
| ***08*** | ***repository*** | ***contains classes responsible for interacting with a database or data storage.*** |
| ***09*** | ***service*** | ***contains classes that encapsulate the application's business logic. These classes are responsible for processing data, performing operations, and interfacing with the data storage.*** |
| ***10*** | ***util*** | ***contains utility classes or helper functions that provide common functionalities used across the application.*** |

## 2. Database Design

### a. Database Schema

*[Provide the tables relationship like example below – following MySQL database naming convention]*

### b. Table Description

| **No** | **Table** | **Description** |
| --- | --- | --- |
| *01* | *account* | *The 'account' table stores user account information with essential fields, including a unique identifier ('id'), creation date ('create\_date'), email address ('email'), password ('password'), active status ('active'), user role ('role'), and an optional status flag ('status'). This table is used for managing user accounts within the system.* |
| *02* | *cart* | *The 'cart' table keeps track of shopping cart data. It has three key components:*  *'id' - A unique identifier for each cart.*  *'cart\_cookie' - A place to store cart information, handy for unregistered users (can be empty).*  *'customer\_id' - Links the cart to a specific customer (can be empty). This table helps manage shopping carts in an online store.* |
| *03* | *certification* | *This table is designed to manage certification records, including lecture names, dates, and their associations with courses and customers* |
| *04* | *comment* |  |
| *05* | *course* | *This table is used to manage courses, including details like descriptions, durations, names, prices, instructor associations, and status information.* |
| *06* | *course\_content* | *This table is designed to manage course content, including descriptions, titles, video links, and their associations with course sections.* |
| *07* | *course\_content\_completion* | *This table is designed to keep track of when users finish course content and whether they have completed it, connecting it to specific customers.* |
| *08* | *course\_exam\_completion* |  |
| *09* | *customer* | *This table is designed to manage customer details, including birthdates, names, genders, avatars, and their associations with user accounts.* |
| *10* | *exam* |  |
| *11* | *feedback* |  |
| *12* | *file\_meta* | *This table is designed to manage metadata for files, including their names, storage paths, and version details. img\_id: A unique identifier for each file's metadata.*  *file\_name: The name of the file (optional).*  *file\_path: The file's storage location or path (optional).*  *version: The version or revision information of the file (optional).* |
| *13* | *instructor* | *This table is designed to manage instructor details, including names, phone numbers, specializations, and their associations with user accounts, along with avatar information. id: A unique identifier for each instructor.*  *fullname: The full name of the instructor.*  *phone\_number: The phone number of the instructor (optional).*  *specialization: The area of specialization or expertise of the instructor.*  *account\_id: An identifier linking the instructor to their account (optional).*  *avatar\_url: A link to the instructor's avatar or profile picture (optional).* |
| *14* | *my\_learning* | *This table is designed to link customers to the courses they are actively engaged in, helping manage the learning progress within the system. customer\_id: An identifier for the customer who is learning a course.*  *course\_id: An identifier for the course that a customer is currently learning.* |
| *15* | *review* | *This table is designed to manage customer reviews and feedback for courses, including comments, ratings, and associations with both courses and customers. id: A unique identifier for each review.*  *comment: The text comment or feedback provided by the customer, always required.*  *feedback\_date: The date when the feedback was given (optional).*  *rating: A numerical rating provided by the customer, indicating their evaluation of the course.*  *update\_date: The date when the review was last updated (optional).*  *course\_id: An identifier for the course being reviewed.*  *customer\_id: An identifier for the customer who provided the review.* |
| *16* | *salary* | *This table is designed to manage financial transactions, including the transaction amount, currency, email information, recipient wallet, and the association with instructors.* |
| *17* | *section* | *This table is designed to structure course content by creating sections, defining their order, and associating them with specific courses.*  *id: A unique identifier for each section.*  *title: The title or name of the section (optional).*  *course\_id: An identifier linking the section to the course it belongs to.* |
| *18* | *staff* | *This table is designed to store details about staff members, including birthdates, names, genders, hiring dates, and their associations with user accounts.*  *id: A unique identifier for each staff member.*  *birthday: The staff member's date of birth (optional).*  *full\_name: The full name of the staff member (optional).*  *gender: The gender of the staff member (optional).*  *injob\_date: The date when the staff member joined the job (optional).*  *account\_id: An identifier linking the staff member to their user account (optional).* |
| *19* | *token* | *This table is designed to handle user authentication tokens, including their expiration and revocation status, token values, types, and their connections to user accounts.*  *id: A unique identifier for each token.*  *expired: A binary indicator (1 or 0) showing if the token has expired (optional).*  *revoked: A binary indicator (1 or 0) showing if the token has been revoked (optional).*  *token: The actual token value used for authentication, always provided.*  *token\_type: The type of token, such as access or refresh (optional).*  *account\_id: An identifier linking the token to a user account (optional).* |
| *20* | *transaction* | *This table is designed to manage financial transactions, including payment methods, total amounts, and their associations with customers.*  *id: A unique identifier for each transaction.*  *payment\_method: The method used for the payment (optional).*  *total\_amount: The total amount involved in the transaction.*  *customer\_id: An identifier connecting the transaction to a specific customer (optional).* |

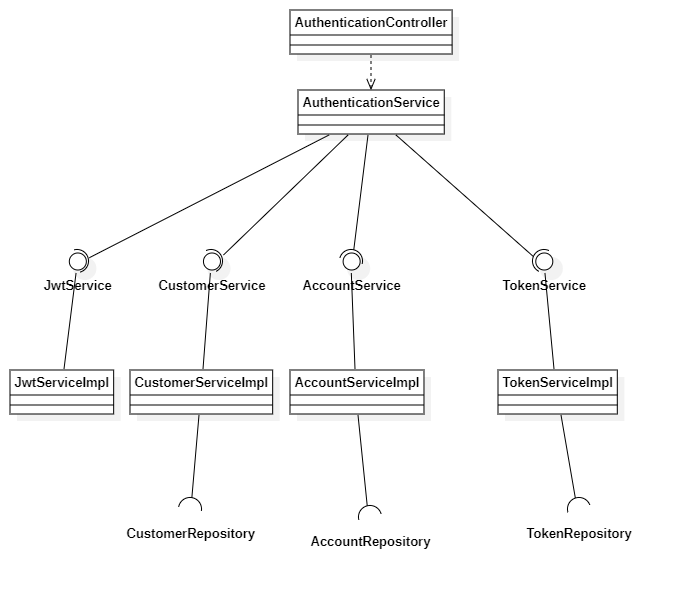
# II. Code Designs

## 1. Authentication/Customer Login

*[Provide the detailed design for the function <Feature/Function Name1>. It include Class Diagram, Class Specifications, and Sequence Diagram(s)]*

### a. Class Diagram

*[This part presents the class diagram for the relevant feature]*

**

### b. Class Specifications

*[Provide the description for each class and the methods in each class, following the table format as below]*

#### Authentication Controller Class

*[Provide the detailed description for the class methods]*

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *Authenticate* | *This method is responsible for handling authentication requests via HTTP POST at the "/authentication" endpoint. It takes an `AuthenticationRequest` object as input, which is expected to be included in the request body. The method processes this input, authenticates the user based on the provided credentials, and generates an authentication token.*  *Inputs: - `request` (Type: AuthenticationRequest) - The `AuthenticationRequest` object is used to deserialize the JSON request body and contains user credentials, such as an email and password.*  *Outputs: - Returns a `ResponseEntity` with an HTTP status of 200 (OK) and a `Cookie` object. The `Cookie` contains the authentication token as its value and is set with a 24-hour expiration time. It's accessible from the root path ("/") of the website and is added to the HTTP response headers.* |
| *02* | *Customer Register* | *This method facilitates user registration by taking a `RegisterRequest` object from the request body, processing the registration request, and generating an authentication token.*  *Inputs:*  *- `request` (Type: RegisterRequest) - The `RegisterRequest` object is deserialized from the JSON request body and contains user registration information, typically including details like username, password, and other user-related data.*  *Outputs:*  *- Returns a `ResponseEntity` with an HTTP status of 200 (OK) and a `Cookie` object. The `Cookie` is created with the authentication token obtained from the `authenticationService.register(request)`. It's set with a relatively long expiration time, accessible from the root path ("/") of the website, and included in the HTTP response headers.* |
| *03* | *Instructor Register* | *This method facilitates the registration of instructors within the system. It accepts an `InstructorRegisterDto` object from the request body, processes the registration request, and generates an authentication token for the newly registered instructor.*  *Inputs:*  *- `instructorDto` (Type: InstructorRegisterDto) - The `InstructorRegisterDto` object is deserialized from the JSON request body and contains information required for registering an instructor, such as their personal details and credentials.*  *Outputs:*  *- Returns a `ResponseEntity` with an HTTP status of 200 (OK). It also includes a `Cookie` object in the HTTP response. The `Cookie` is named "USER" and contains the authentication token obtained from the `authenticationService.registerInstructor(instructorDto)`. The `Cookie` is configured with a long expiration time, is accessible from the root path ("/") of the website, and is added to the HTTP response headers.* |
| *04* | *Change Password* | *This method allows a user to change their password by validating an OTP (One-Time Password). The user's identity is extracted from the security context, and the OTP provided in the request is compared to the server-stored OTP for validation.*  *Inputs:*  *- `changePasswordRequest` (Type: ChangePasswordDto) - The `ChangePasswordDto` object is deserialized from the JSON request body and contains information required for changing the user's password, including the OTP provided by the user.*  *Outputs:*  *- Returns a `ResponseEntity` with a generic type of `String`. If the OTP validation is successful, it invokes the `authenticationService.changePassword(changePasswordRequest)` method to change the user's password and returns the result as a response with an HTTP status of 200 (OK). If the OTP validation fails, it returns an HTTP response with a status of 401 (UNAUTHORIZED).* |

#### Authentication Service Class

***Class Methods***

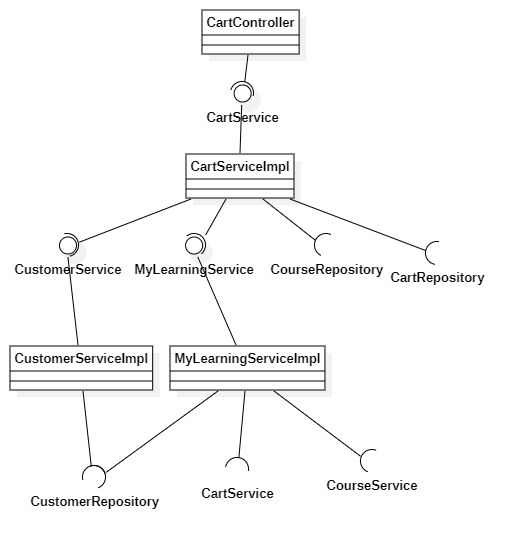
*[Provide the detailed description for the class methods]*

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *Customer Register* |  |
|  |  |  |

### c. Sequence Diagram

## 2. Add to cart/Buy course

### Class Diagram



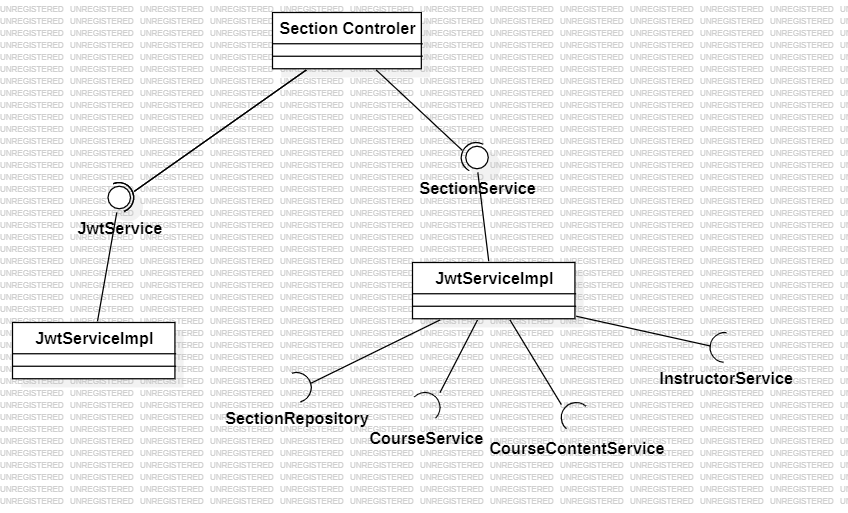
## 3. Create course/Create lesson

### Class Diagram

### 

## 4. Edit Section

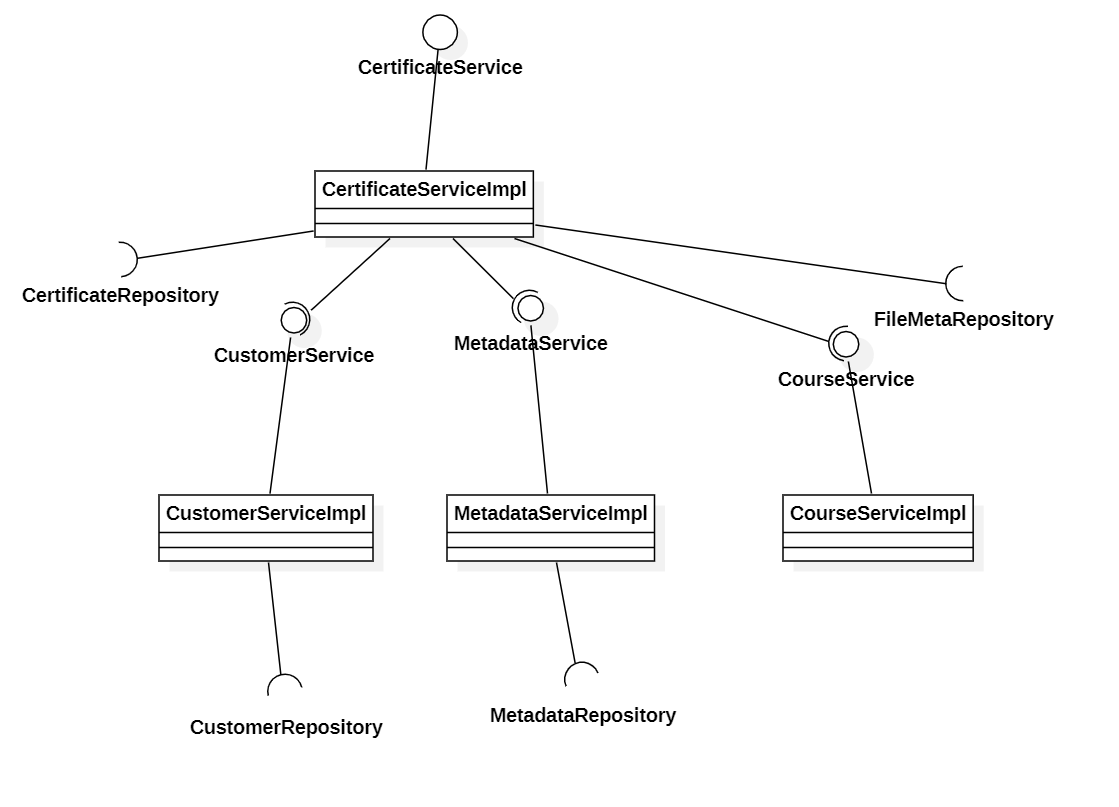
### Class Diagram



| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *SectionController* | *Controls Section CRUD API endpoints* |
| *02* | *createSection* | *Creates a new Section. Takes SectionCreateDto and token in header. Calls SectionService to create Section, returns SectionDto.* |
| *03* | *getSection* | *Gets a Section by id. Takes id parameter. Calls SectionService, returns SectionDto.* |
| *04* | *getSections* | *Get all Sections for a Course. Takes a token and course id. Calls SectionService, returns list of SectionDto.* |
| *05* | *deleteSection* | *Deletes a Section. Takes id and token. Calls SectionService to delete, returns SectionDto.* |
| *06* | *editSection* | *Updates a Section. Takes SectionDto and token. Calls SectionService to update, returns SectionDto.* |
| *07* | *CourseController* | *Controls Course CRUD API endpoints.* |
| *08* | *getAllCourse* | *Gets all Courses with pagination. Takes name, page, maxPage, isRandom params. Calls CourseService, returns list of CourseDto.* |
| *09* | *getInstructorCourseList* | *Get Courses for an Instructor. Take a token in the header. Gets email from token, calls CourseService, returns list of CourseDto.* |
| *10* | *getCourseByID* | *Get a Course by id. Takes id param, calls CourseService, returns CourseDto.* |
| *11* | *getAllInfoCourse* | *Gets all info for a Course. Takes id and token. Calls CourseService, returns CourseAllInfoDto.* |
| *12* | *getCourseDefaultInfo* | *Gets default info for a Course. Takes id, calls CourseService, returns CourseDefaultInfo.* |
| *13* | *createCourse* | *Creates a new Course. Take CourseCreateDto and token. Gets email from token, calls CourseService to create Course, returns CourseDto.* |
| *14* | *updateCourse* | *Updates a Course. Take CourseEditDto and a token. Calls CourseService to update, returns CourseDto.* |
| *15* | *deleteCourse* | *Deletes a Course. Takes id and token. Gets email from token, calls CourseService to delete, returns CourseDto.* |
| *16* | *JwtService* | *Provides JWT token generation and validation functions.* |
| *17* | *extractUserEmail* | *Extracts and returns user email from JWT token string.* |
| *18* | *generateToken* | *Generates JWT token from user details and optional extra claims. Returns a token string.* |
| *19* | *isTokenValid* | *Validates JWT token against user details. Returns boolean validity.* |
| *20* | *SectionService* | *Provides Section business logic and data access.* |
| *21* | *createSection* | *Creates a new Section. Take SectionDto, courseId, email. Calls repo, returns SectionDto.* |
| *22* | *getAllSectionOfCourse* | *Get all Sections for a Course. Take courseId, email. Calls repo, returns SectionDtos.* |
| *23* | *getSectionById* | *Gets Section by id. Calls repo, returns SectionDto.* |
| *24* | *getAllSectionByCourseID* | *Get Sections by courseId. Calls repo, returns SectionDtos.* |
| *25* | *getSectionEntityBySectionID* | *Gets Section entity by id. Calls repo, returns Section.* |
| *26* | *getListSectionDetailByCourseID* | *Gets Section details for Course. Loops Sections, gets lessons, returns SectionDetailDtos.* |
| *27* | *updateSection* | *Updates a Section. Take SectionDto, email. Calls repo, returns SectionDto.* |
| *28* | *removeSection* | *Deletes a Section. Takes id, email. Calls repo, returns SectionDto.* |
| *29* | *getSectionDefaultInfoByCourseID* | *Gets default Section info for Course. Loops Sections, gets names, returns SectionDefaultInfos.* |
| *30* | *CourseService* | *Provides Course business logic and data access.* |
| *31* | *getAllCourseByPaging* | *Gets paged Courses. Calls repo, returns CourseDtos.* |
| *32* | *searchCourseById* | *Get Course by id. Calls repo, returns CourseDto.* |
| *33* | *addCourse* | *Adds a new Course. Take CourseDto. Calls repo, returns CourseDto.* |
| *34* | *searchCourseByNameAndFilter* | *Searches Courses by name. Calls repo with filters, returns CourseDtos.* |
| *35* | *createCourseUsingJwt* | *Creates a Course from JWT. Takes createDto, email. Calls repo, returns CourseDto.* |
| *36* | *isCourseCreateByRightEmail* | *Checks Course created by email. Returns boolean.* |
| *37* | *searchCourseEntityById* | *Gets Course entity by id. Calls repo, returns Course.* |
| *38* | *getCoursesByInstructorEmail* | *Get Courses for email. Calls repo, returns CourseDtos.* |
| *39* | *updateCourse* | *Updates a Course. Take CourseDto, email. Calls repo, returns CourseDto.* |
| *40* | *getAllInfoOfCourse* | *Get all Course info. Take an id. Calls services, returns CourseAllInfoDto.* |
| *41* | *getAllCourseDefaultInfo* | *Gets default Course info. Take an id. Calls services, returns CourseDefaultInfo.* |
| *42* | *getCourseByCourseContentID* | *Gets Course by content id. Calls repo, returns Course.* |
| *43* | *removeCourseUsingCourseID* | *Deletes Course by id, email. Calls repo, returns CourseDto.* |
| *44* | *getPercentOfCourseCompleted* | *Gets completion percent for Course. Loops contents, counts completed, calculates percent.* |

## 5. Create Certificate Automation

### Class Diagram



## 6. Generate OTP for changing password

### Class Diagram