**Requirement Specifications (API version)**

**Project: Cursus**

Version: 1.0

**Approval Page**

The endorsement on this document, by the authorized <<client>> representative, indicates that <<client>> and FPT have agreed on the document “Cursus- Software Requirement Specifications”.

| **Prepared by :** | Vu Duc Anh | Signature: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| --- | --- | --- | --- |
|  |  | Date: | 30 /07/ 2024 |
|  |  |  |  |
|  |  |  |  |
| **Reviewed by :** | Tuan Tran | Signature: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | Date: | \_\_\_\_ /\_\_\_\_ / \_\_\_\_ |
|  |  |  |  |
| **Supported by :** | Tuan Tran | Signature: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | Date: | \_\_\_\_ /\_\_\_\_ / \_\_\_\_ |
|  |  |  |  |
|  |  |  |  |
| **Approved by :** | Tuan Tran | Signature: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | Date: | \_\_\_\_ /\_\_\_\_ / \_\_\_\_ |

**Revision History**

| **Date** | **Version** | **Author** | **Change Description** |
| --- | --- | --- | --- |
| 30 Apr- 22 | 0.1 | Nguyen Phuong Nhat Khoa | Create SRS for Cursus |
| 5 May –24 | 0.2 | Tuan Tran | Modify SRS use case, paint point. |
| 30 July - 24 | 1.0 | Vu Duc Anh | Modify use case, project scope to API use case |

**Table Of Content**

[1. Introduction 5](#_1fob9te)

[1.1 Purpose 5](#_3znysh7)

[1.2 Overview 5](#_2et92p0)

[1.3 Intended Audience and Reading Suggestions 5](#_tyjcwt)

[Abbreviations 5](#_3dy6vkm)

[2. High Level Requirements 6](#_1t3h5sf)

[2.1 System structure 6](#_4d34og8)

[2.2 Workflow 6](#_2s8eyo1)

[2.3 Use Case Diagram 6](#_17dp8vu)

[2.3.1 General users 7](#_3rdcrjn)

[2.3.2 Administrator 9](#_lnxbz9)

[2.4 Permission matrix 11](#_35nkun2)

[3. Use Case Specifications 12](#_1ksv4uv)

[3.1 General functions 12](#_44sinio)

[3.1.1 UC 01: Sign up for Student 12](#_2jxsxqh)

[3.1.2 UC 02: Sign up for Instructor 15](#_z337ya)

[3.1.3 UC 03: Sign in 19](#_3j2qqm3)

[3.1.4 UC 04: Verify email 20](#_1y810tw)

[3.1.5 UC 05: Change password 21](#_4i7ojhp)

[3.1.6 UC 06: Forgot password 22](#_2xcytpi)

[3.2 General user 23](#_1ci93xb)

[3.2.1 UC 07: Get Course With Behavior 23](#_3whwml4)

[3.2.2 UC 08: Update profile 24](#_2bn6wsx)

[3.2.3 UC 09: View and Search course 26](#_qsh70q)

[3.2.4 UC 10: Study enrolled course 27](#_3as4poj)

[3.2.5 UC 11: View course 29](#_1pxezwc)

[3.2.6 UC 12: Enroll course 31](#_3o7alnk)

[3.2.7 UC 13: Review course 33](#_23ckvvd)

[3.2.8 UC 14: Comment course 34](#_ihv636)

[3.2.9 UC 15: View Dashboard: Top Purchased and Top Bad Courses 36](#_32hioqz)

[3.2.10 UC 16: View earnings analytics 38](#_1hmsyys)

[3.2.11 UC 17: Create course 40](#_41mghml)

[3.2.12 UC 18: Update course detail 41](#_2grqrue)

[3.2.13 UC 19: Update course content 43](#_vx1227)

[3.2.14 UC 20: View reviews 45](#_1v1yuxt)

[3.2.15 UC 21: View Bookmarks 46](#_4f1mdlm)

[3.2.16 UC 22: Add Bookmark 47](#_2u6wntf)

[3.2.17 UC 23: Delete Bookmark 48](#_mn3gooqsqu7k)

[3.2.18 UC 24: Create Payment (Purchase Course) 49](#_5vah82k44ida)

[3.3 Admin 51](#_19c6y18)

[3.3.1 UC 25: View list of students 51](#_28h4qwu)

[3.3.2 UC 26: Approve Course 52](#_lnq4jrs4gtkz)

[3.3.3 UC 27: Reject Course 53](#_37m2jsg)

[3.3.4 UC 28: View list of categories 54](#_1mrcu09)

[3.3.5 UC 29: Create a category 55](#_46r0co2)

[3.3.6 UC 30: Update a category 57](#_2lwamvv)

[3.3.7 UC 31: Soft-delete category 58](#_111kx3o)

[3.3.8 UC 32: View/Filter list of courses 60](#_206ipza)

[3.3.9 UC 33: View details of a course 61](#_4k668n3)

[3.3.10 UC 34: Activate course 62](#_2zbgiuw)

[3.3.11 UC 35: Deactivate Course 63](#_13lkttk0baju)

[3.3.12 UC 36: Approve Course 64](#_2fjb0zsc4n4h)

[3.3.13 UC 37: Reject Course 65](#_wegaekstwavn)

[3.3.14 UC 38: Approve Instructor 67](#_xf5r6qj0mx52)

[3.3.15 UC 39: Reject Instructor 68](#_nihnq7ysk3n2)

[4. Non-functional requirements 70](#_1egqt2p)

[4.1 Performance requirements 70](#_3ygebqi)

[4.2 Security Requirements 70](#_2dlolyb)

[4.3 Software quality attributes 70](#_sqyw64)

[5. Other requirements 71](#_3cqmetx)

[5.1 Localize rules 71](#_1rvwp1q)

[5.2 Image attachment 71](#_4bvk7pj)

[5.3 Date picker 71](#_1brfedor52ba)

[5.4 Internal System Error 71](#_xvir7l)

[6. Appendices](#_1x0gk37) **71**

[6.1 Message list](#_4h042r0) 71

[6.2 Email template 72](#_pkwqa1)

# Introduction

## Purpose

The Functional Requirements Specification will:

* Define the scope of business objectives, business functions, and organizational units covered,
* Identify the business processes that the solution must facilitate,
* Facilitate a common understanding of what the functional requirements are for all parties involved,
* Establish a basis for defining the acceptance tests for the solution to confirm that what is delivered meets requirements.
* The purpose of the document is to collect and analyse all assorted ideas that have come up to define the system, its requirements with respect to consumers. Also, we shall predict and sort out how we hope this product will be used in order to gain a better understanding of the project, outline concepts that may be developed later, and document ideas that are being considered, but may be discarded as the product develops.

## Overview

Overview - Litware, Inc. is a medium-sized Education technology company that has a main office in Boston.   
Litware researched about the Edtech domain and want to build an Online Course Management with name **Cursus** as [MVP](https://www.projectmanagement.com/blog-post/61937/Defining-MVP--MBI--MMF--and-MMR?__cf_chl_tk=UxPzcg2O.AGiRnz1MZ3q5CZH7iyC5KYWI6DmUjm24I0-1715072932-0.0.1.1-1407) (Minimum Value Project - simple) version to test the EU market trend.

Cursus system will allow user to be a part of Student/Studier and Instructor. Coacher will publish the course and student can purchase and study. Student could interact with Instructor as well.

Litware consider about the Client-Side architect for this Cursus to improve the user exeprience.

**Notice:** This document is only providing Use Case Specifications of **API**, no interface or UI included.

## Intended Audience and Reading Suggestions

This document is intended for:

* Development team: Responsible to develop detailed design, implement and perform unit test. Also perform test manual.
* Project Owner: Responsible to manage the Product backlog for the application.

## Abbreviations

| **Acronym** | **Reference** |
| --- | --- |
| SRS | Software requirement specification |
| UC | Use case |
| API | Application Programming Interface |

# High Level Requirements

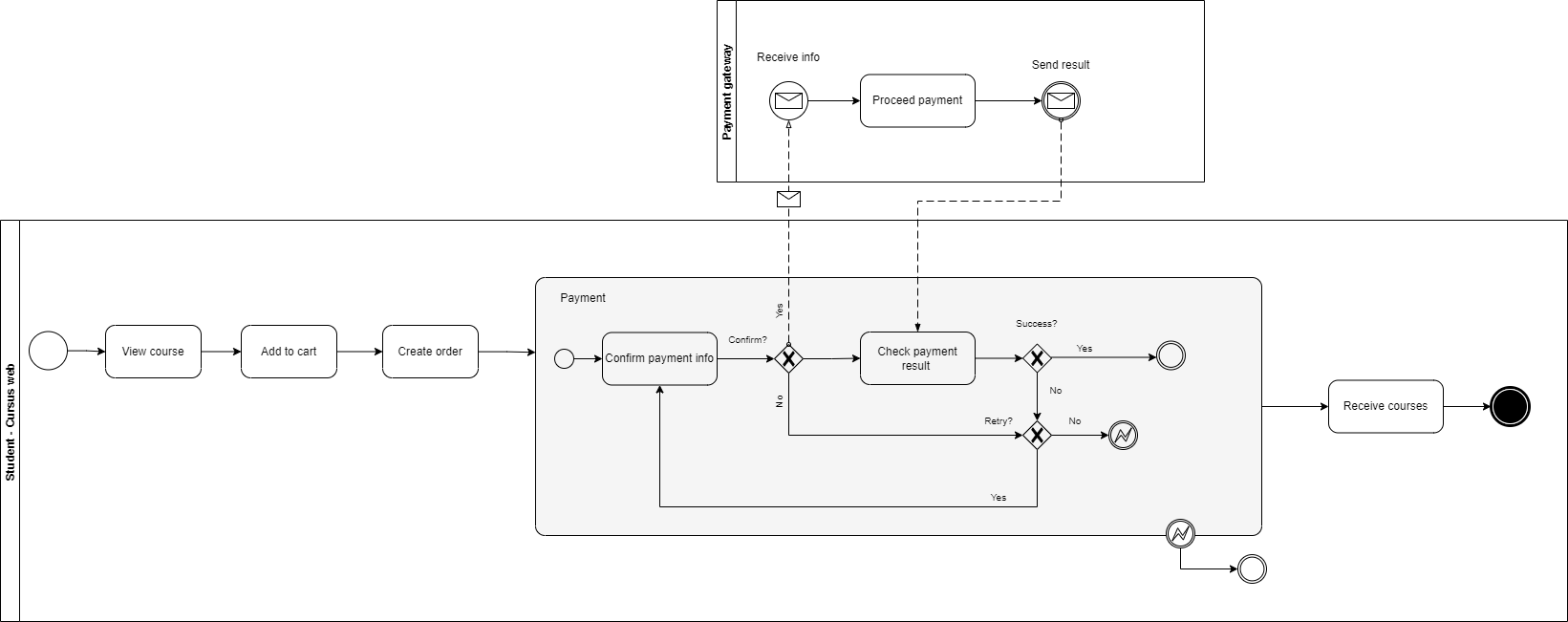
This section describes the general overview of the system functions or business processes which are depicted in different diagrams. It shows the types of users, their permissions granted to perform specific system functions and the sequence required to complete a business workflow (if any). As the section name implies, it is high-level which means it is not detailed enough. For detailed requirement specification, please see **3 Use Case Specifications** section below.

## System structure

The system architecture will be described in the software development phase.

## Workflow

This section shows the flow of tasks or steps taken by each users of the system in-order to complete a business process. The user’s actions are shown in each business process stage of the system and what happens before it can move to the next stage or revert to the previous.



## Use Case Diagram

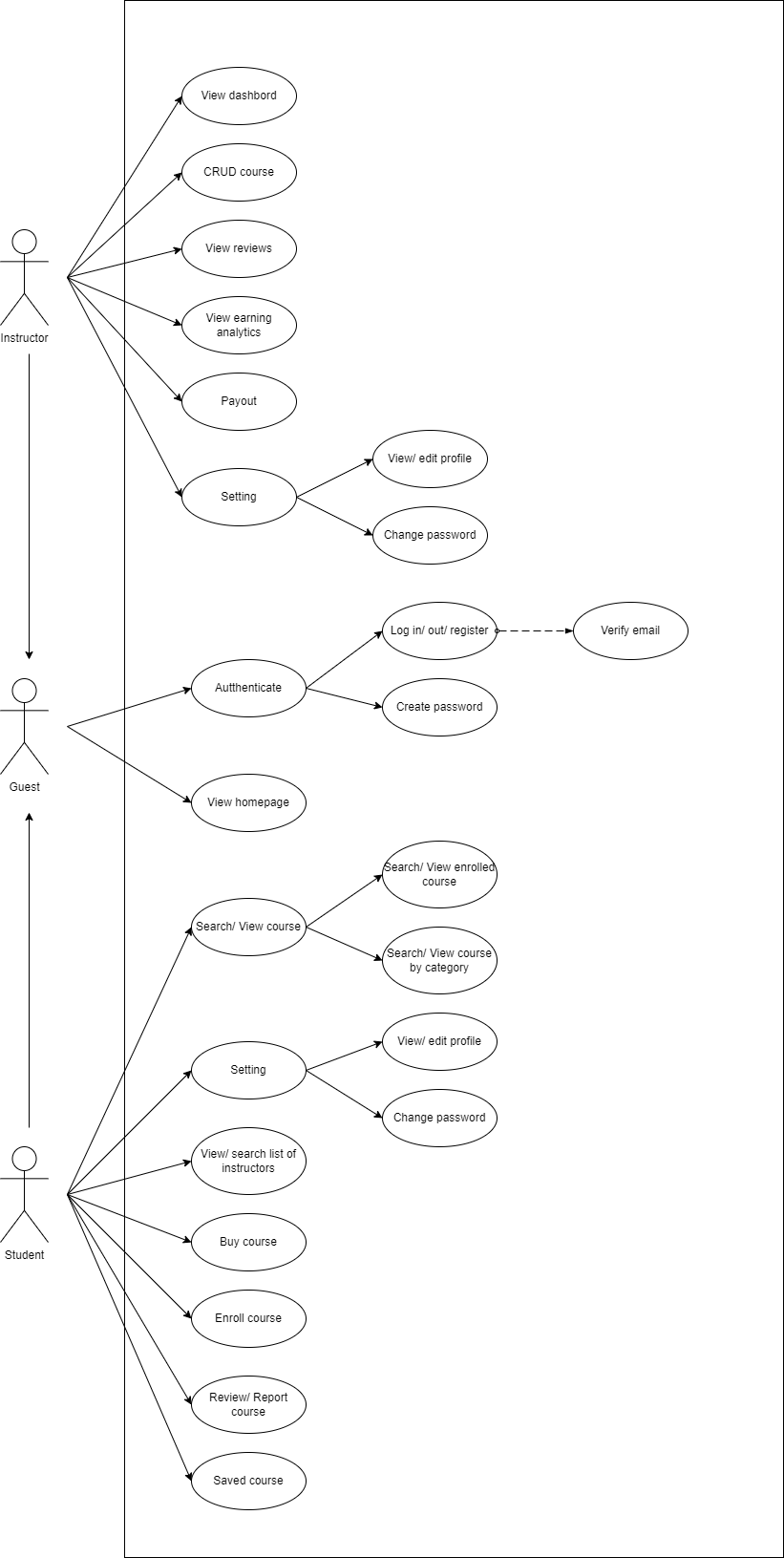
The use case diagram here shows the specific goal and objective or how the user interacts with the system. The eclipse in the system boundary represents the system use case/functions while the stickman represents the actor/user of the system. The line connecting the actor and the use case shows that the actor can perform that function in the system to achieve a goal.

### General users

**Student:** the person who is in Cursus as Student Role, can view eligible courses, purchase and enroll into the courses, give feedback, manage personal user information.

**Instructor:** the person who provides the courses like teacher, mentor, support the course. This actor can get back the course fee that Student are paid.

**Guest:** the person who visits the Cursus and looks in to be the Student or Instructor.



### Administrator

The person who managed the Cursus system. This role mainly can support to approve/reject the Instructor’s courses and manage all courses, also can take care of the health of Cursus system.



## Permission matrix

Permission Matrix mapping functions and user roles for <<App Name>> application is described as below:

Remark:

“O” means that user has permission on corresponding function. For more information about what the actor can do on that function, please refer to corresponding use case.

“X” means that user does not have permission on corresponding function.

| **Function** | **Guest** | **Student** | **Instructor** | **Admin** |
| --- | --- | --- | --- | --- |
| **Authentication** |  |  |  |  |
| Register | O | O | O | X |
| Create password | X | O | O | X |
| Log in/ log out | O | O | O | O |
| **General Function** |  |  |  |  |
| View Dashboard | O | O | O | O |
| View course enrolled. | X | O | X | X |
| View total of student/ course | X | X | O | X |
| View course analytics | X | X | O | X |
| **Setting** |  |  |  |  |
| View/ edit profile | X | O | O | X |
| Change password | X | O | O | X |
| **Purchase courses** |  |  |  |  |
| Proceed purchase course | X | O | X | X |
| **Enroll course** |  |  |  |  |
| Enroll into a course | X | O | X | X |
| View tracking of the course | X | O | X | X |
| Review course | X | O | X | X |
| Report course | X | O | X | X |
| **Dashboard for Instructor** |  |  |  |  |
| View course/ student analytics | X | X | O | X |
| **Manage course** |  |  |  |  |
| View list course | X | X | O | X |
| Create/ edit/ delete course | X | X | O | X |
| Submit course for approval | X | X | O | X |
| **Earning Affiliate** |  |  |  |  |
| View earning money | X | X | O | X |
| Payout | X | X | O | X |
| **Dashboard for admin** |  |  |  |  |
| View dashboard | X | X | X | O |
| **Manage categories** |  |  |  |  |
| Create/ view/ search/ edit/ delete categories | X | X | X | O |
| **Manage student** |  |  |  |  |
| View list of students | X | X | X | O |
| Block/ unblock student | X | X | X | O |
| **Manage instructor** |  |  |  |  |
| View list of instructors | X | X | X | O |
| Block/ unblock instructor | X | X | X | O |
| Approve/ reject the course. | X | X | X | O |
| **Manage course** |  |  |  |  |
| View list of course | X | X | X | O |
| Review/ Approve/ Reject/ Block/ Unblock course | X | X | X | O |

# Use Case Specifications

This section covers the system’s functional requirements which details what the system must do in terms of input, behavior and the expected output. It elicits the interaction between the actor(s) and the system, the system’s behavior and the results of their interactions.

## General functions

### UC 01: Sign up for Student

| **Objective:** | This use case allows a new user to sign up by providing their personal information via an API endpoint. |
| --- | --- |
| **Actor:** | 1. Guest |
| **Trigger:** | 1. The external system sends a POST request to the /api/User/sign-up-for-student endpoint with user details. |
| **Pre-condition:** | User has no account yet.  The client application must be authenticated to access the API.  The client application must provide all necessary user details in the request payload. |
| **Post-condition:** | A new student user is created in the system with a unique user ID.  A confirmation email is sent to the user. |
| **Basic flow:** | 1. Checks if the email already exists using \_userRepository.CheckEmail. If it exists, adds UserErrors.EmailAlreadyUsed. 2. Validates the phone number format using Validator.IsValidPhone and converts it to international format if valid. Checks if the phone number already exists using \_userRepository.CheckPhone. If it exists, adds UserErrors.PhoneAlreadyUsed. 3. Validates the full name, adding UserErrors.FullnameIsEmpty if it is empty or UserErrors.FullnameIsInvalid if it is invalid. 4. Validates the email, adding UserErrors.EmailIsEmpty if it is empty or UserErrors.EmailIsInvalid if it is invalid. 5. Validates the password, adding UserErrors.PasswordIsEmpty if it is empty, UserErrors.PasswordMinLength if it is too short, or UserErrors.PasswordIsInvalid if it does not meet the complexity requirements. 6. Validates the date of birth, adding UserErrors.DOBIsInvalid if it is invalid or UserErrors.DOBIsNotEnogh if the user is too young. 7. Validates the phone number, adding UserErrors.PhoneIsEmpty if it is empty, UserErrors.PhoneLength if it is not the correct length, or UserErrors.PhoneIsInvalid if it is invalid. 8. Validates the address, adding UserErrors.AddressIsEmpty if it is empty.   (9) If there are no validation errors, the system:   * Generates a new user ID using \_userRepository.AutoGenerateUserID. * Hashes the user's password using HashPassword. * Creates a new User record in the database. * Registers user details using \_userDetailService.RegisterUserDetailAsync and wallet information using \_walletService.RegisterUserWalletlAsync.   If the user details and wallet registration are successful, the system:   * Sends a confirmation email to the user's email address using \_userService.SendMailComfirm. * Returns a success response (200 OK) with a message indicating that the sign-up was successful and instructing the user to check and confirm their email.   If any validation or operation fails, the system returns a failure response with the appropriate error messages. |
| **Alternative flow** | **User Already Exists**:   1. **API** checks if the email or phone number already exists in the database. 2. If either exists, the system returns a conflict response (409 Conflict) with an error message indicating that the email or phone number is already in use. |
| **Exception flows** | **Invalid Request Payload:**   1. API validates the request payload and finds it invalid (e.g., missing required fields). 2. API returns a bad request response (400 Bad Request) with details of the validation errors.   **Unauthorized Access:**   1. External System sends a POST request to the /api/User/sign-up-for-student endpoint without proper authentication. 2. API detects the lack of authentication. 3. API returns an unauthorized response (401 Unauthorized).   **Internal Server Error:**   1. API encounters an unexpected error during processing. 2. API returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 1 | **Validate rules:**  When execute these steps, system validates data as the rules below:   * If any required fields are left blank, the system displays error message MSG 1, 6, 10,12, 15, 18 (base on what field are missing) * If username or email existed in system 🡪 show MSG 2 or 5 * Password must have 8 symbols, have at least one special characters and uppercase * Phone number must follow the format “+84” and 9 characters. |
| (2) | BR 2 | **Confirmation rule:**  After sending a confirmation email to the user's email address using \_userService.SendMailComfirm, users have to wait 60 seconds for confirmation again after sending. |

### UC 02: Sign up for Instructor

| **Objective:** | This use case allows a new instructor to sign up by providing their personal and professional information via an API endpoint. |
| --- | --- |
| **Actor:** | 1. Guest |
| **Trigger:** | 1. The external system sends a POST request to the /api/User/sign-up-for-instructor endpoint with instructor details. |
| **Pre-condition:** | User has no account yet  The client application must be authenticated to access the API.  The client application must provide all necessary instructor details in the request payload. |
| **Post-condition:** | A new instructor user is created in the system with a unique user ID.  A confirmation email is sent to the user. |
| **Basic flow:** | 1. **External System** sends a POST request to the /api/User/sign-up-for-instructor endpoint with instructor details in the request body. 2. **API** receives the request and calls the SignUpForIntrustor method with the provided RegisterInstructorDTO. 3. **SignUpForIntrustor** method performs the following validations:  * Checks if the email already exists using \_userRepository.CheckEmail. If it exists, adds UserErrors.EmailAlreadyUsed. * Validates the phone number format using Validator.IsValidPhone and converts it to international format if valid. Checks if the phone number already exists using \_userRepository.CheckPhone. If it exists, adds UserErrors.PhoneAlreadyUsed. * Validates the full name, adding UserErrors.FullnameIsEmpty if it is empty or UserErrors.FullnameIsInvalid if it is invalid. * Validates the email, adding UserErrors.EmailIsEmpty if it is empty or UserErrors.EmailIsInvalid if it is invalid. * Validates the password, adding UserErrors.PasswordIsEmpty if it is empty, UserErrors.PasswordMinLength if it is too short, or UserErrors.PasswordIsInvalid if it does not meet the complexity requirements. * Validates the date of birth, adding UserErrors.DOBIsInvalid if it is invalid or UserErrors.DOBIsNotEnogh if the user is too young. * Validates the phone number, adding UserErrors.PhoneIsEmpty if it is empty, UserErrors.PhoneLength if it is not the correct length, or UserErrors.PhoneIsInvalid if it is invalid. * Validates the address, adding UserErrors.AddressIsEmpty if it is empty. * Validates the tax number, adding InstructorErrors.TaxNumberIsEmpty if it is empty. * Validates the card number, adding InstructorErrors.CardNumberIsEmpty if it is empty. * Validates the card name, adding InstructorErrors.CardNameIsEmpty if it is empty. * Validates the card provider, adding InstructorErrors.CardProviderIsEmpty if it is empty. * Validates the certification file using ValidateCertification.  1. If there are no validation errors, the system:  * Generates a new user ID using \_userRepository.AutoGenerateUserID. * Hashes the user's password using HashPassword. * Creates a new User record in the database with the role ID for instructors. * Registers user details using \_userDetailService.RegisterUserDetailAsync. * Registers instructor-specific details using \_instructorService.RegisterInstructorAsync. * Registers wallet information using \_walletService.RegisterUserWalletlAsync.  1. If the user details and wallet registration are successful, the system:  * Sends a confirmation email to the instructor's email address using \_userService.SendMailComfirm. * Returns a success response (200 OK) with a message indicating that the sign-up was successful and instructing the instructor to check and confirm their email.  1. If any validation or operation fails, the system returns a failure response with the appropriate error messages. |
| **Alternative flow** | **User Already Exists**:   1. **API** checks if the email or phone number already exists in the database. 2. If either exists, the system returns a conflict response (409 Conflict) with an error message indicating that the email or phone number is already in use. |
| **Exception flow** | **Invalid Request Payload:**   1. API validates the request payload and finds it invalid (e.g., missing required fields). 2. API returns a bad request response (400 Bad Request) with details of the validation errors.   **Unauthorized Access:**   1. External System sends a POST request to the /api/User/sign-up-for-instructor endpoint without proper authentication. 2. API detects the lack of authentication. 3. API returns an unauthorized response (401 Unauthorized).   **Internal Server Error:**   1. API encounters an unexpected error during processing. 2. API returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 3 | **Validate rules:**  When user fills all information and click “Submit” button, system validates data as the rules below:   * If any required fields are left blank, the system displays error message MSG 2. MSG 2 is displayed below the blank field in red, italic format. * If username or email existed in system 🡪 show MSG 3 |
| (2) | BR 4 | **Create rules:**  When system validate information successfully, the system proceeds to create new account:   * Add record for new user in DB * Save user information in DB * Send confirmation email to be corresponding. |

### UC 03: Sign in

| **Objective:** | This use case allows an existing user to sign in by providing their email and password via an API endpoint. |
| --- | --- |
| **Actor:** | All types of users |
| **Trigger:** | The external system sends a POST request to the /api/User/sign-in endpoint with sign-in details. |
| **Pre-condition:** | User has already had account in system  The client application must be authenticated to access the API.  The client application must provide valid email and password credentials in the request payload. |
| **Post-condition:** | The user is authenticated, and access and refresh tokens are generated.  The user receives an authentication token for subsequent requests. |
| **Basic flow** | 1. **External System** sends a POST request to the /api/User/sign-in endpoint with sign-in details (SignInDTO). 2. **API** receives the request and calls the SignIn method with the provided SignInDTO. 3. **SignIn** method performs the following validations:  * Checks if the email is empty. If empty, returns Result.Failure(SignInErrors.InputEmpty()). * Checks if the password is empty. If empty, returns Result.Failure(SignInErrors.InputEmpty()). * Checks if the email exists in the database using \_userRepository.CheckEmail. If not, returns Result.Failure(SignInErrors.InputFieldWrong()).  1. **SignIn** method hashes the password using HashPassword. 2. **SignIn** method checks if the user credentials are correct using \_userRepository.SignIn. If incorrect, returns Result.Failure(SignInErrors.InputFieldWrong()). 3. **SignIn** method checks the user's status:  * If user.IsMailConfirmed is false, sends a confirmation email and returns Result.Failure(new Error("Email", "Your mail has not confirmed yet, please confirm to sign in")). * If user.IsDelete is true, returns Result.Failure(SignInErrors.AccountIsDelete()). * If user.IsBan is true, returns Result.Failure(SignInErrors.AccountIsBan()).  1. **SignIn** method creates a CurrentUserObject with user details. 2. **SignIn** method generates a token using \_tokenService.GenerateTokenAsync. 3. **SignIn** method generates an access token using \_tokenService.GenerateAccessTokenAsync. 4. **SignIn** method generates a refresh token using \_refreshTokenService.GenerateRefreshTokenAsync. 5. **SignIn** method saves the refresh token using \_refreshTokenService.SaveRefreshTokenAsync. 6. **SignIn** method creates a TokenDTO with the access and refresh tokens. 7. **SignIn** method returns Result.SuccessWithObject(tokenDTO). 8. **API** returns a success response (200 OK) with the authentication tokens. |
| **Alternative flow** | **User Not Found**:   1. **API** checks if the email exists in the database. 2. If the email does not exist, the system returns a conflict response (409 Conflict) with an error message indicating incorrect email or password. |

**Business Rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 5 | **Validate rules:**  After execute API, system will validate based on the existing rules. The validation logic is:   * If the value of any mandatory fields is blank, system will show an error message for the required fields as MSG * If the email does not exist in database, the system will show an error message as MSG * If the password is not correct, the system will show an error message as MSG. |
| (2) | BR 6 | **Save rules:**  System records information with last success login datetime for the user. |

### UC 04: Verify email

| **Objective:** | This use case allows a user to confirm their email address by clicking a confirmation link received via email |
| --- | --- |
| **Actor:** | Guest. |
| **Trigger:** | The user clicks on the confirmation link in their email. |
| **Pre-condition:** | The user has previously signed up (based on UC 01) and received an email with a confirmation link containing a confirmation key. |
| **Post-condition:** | The user's email is confirmed, and their status is updated in the system.  The user is redirected to the login page. |
| **Basic flow:** | 1. **User** clicks the confirmation link received via email. 2. **External System** sends a GET request to the /api/User/mail-confirm endpoint with the confirmation key as a query parameter. 3. **API** receives the request and calls the ConfirmMail method with the provided key. 4. **ConfirmMail** method performs the following steps:  * Checks if the key is empty or null. If empty, returns "Invalid value". * Decrypts the email address from the key using Encryption.Decrypt. * Checks if the email exists in the database using \_userRepository.CheckEmail. If not, returns Message.InvalidUser. * Updates the email confirmation status using \_userRepository.UpdateMailConfirm.  1. **ConfirmMail** method returns a success message with a link to the login page using \_mailService.ReturnToLoginPage. 2. **API** returns a content response (200 OK) with the success message in HTML format. |
| **Alternative flow:** | **Invalid Key**:   1. **ConfirmMail** method checks if the key is empty or null. 2. If the key is invalid, the system returns a content response (200 OK) with the message "Invalid value".   **Email Not Found**:   1. **ConfirmMail** method decrypts the email from the key and checks if it exists in the database. 2. If the email does not exist, the system returns a content response (200 OK) with the message "Invalid user". |
| **Exception flow:** | **Internal Server Error**:   1. **API** encounters an unexpected error during processing. 2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 7 | Content of email need to follow the template as Email Template provided |
| (2) | BR 8 | * Email link should be valid within 3hrs, otherwise, disable link * If successful, return to the login page of UC 02 |

### UC 05: Change password

| **Objective:** | This use case allows a user to change their account password by providing their old password and a new password. |
| --- | --- |
| **Actor:** | All types of users |
| **Trigger:** | The user requests to change their password. |
| **Pre-condition:** | User logs in successfully.  The user must be authenticated and authorized.  The user knows their current password. |
| **Post-condition:** | The user's password is successfully changed.  The user receives a confirmation email regarding the password change. |
| **Basic flow:** | 1. **User** sends a PUT request to the /api/User/change-password endpoint with the new password and the email. 2. **API** retrieves the current user information using TokenHelper.Instance.GetThisUserInfo(HttpContext). 3. **API** calls the ChangePassword method with the user's email, new password, and old password. 4. **ChangePassword** method performs the following steps:  * Validates the email format using Validator.IsValidEmail. * Validates the new password using Validator.IsValidPassword. * Retrieves the user's details from the database using \_userRepository.GetUserByEmail. * If the user does not exist, returns UserErrors.UserIsNotExist. * If the user is a Google account, returns an error indicating the password cannot be changed. * Verifies the old password by comparing it with the stored hashed password using HashPassword.. * If all validations pass, updates the user's password in the database.  1. **ChangePassword** method returns a success result. 2. **API** sends a confirmation email regarding the password change using \_userService.SendMailChangePassword.   If the email is successfully sent, **API** returns a success response (200 OK) with a confirmation message.  If the email sending fails, **API** returns a bad request response (400 Bad Request) with the error details. |

**Business Rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 9 | **Validation rule:**   * New password can’t be same with current password. Show message MSG if the same. |

### UC 06: Forgot password

| **Objective:** | This use case allows a user to reset their password by providing their email and a new password. |
| --- | --- |
| **Actor:** | General user |
| **Trigger:** | User clicks “Forgot password” at Log in screen. |
| **Pre-condition:** | User already has account on Cursus. |
| **Post-condition:** | User creates new password successfully. |
| **Basic flow:** | **1- User** sends a PUT request to the /api/User/reset-password endpoint with the email, new password, and old password in ChangePasswordDTO.  **2- API** calls the ChangePassword method with the provided email, new password, and old password.  **3- ChangePassword** method performs the following steps:   * Validates the email format using Validator.IsValidEmail. * Validates the new and old passwords using Validator.IsValidPassword. * Retrieves the user's details from the database using \_userRepository.GetUserByEmail. * If the user does not exist, returns UserErrors.UserIsNotExist. * If the user is a Google account, returns an error indicating the password cannot be changed. * Verifies the old password by comparing it with the stored hashed password using HashPassword. * If the old password does not match, returns an error indicating the passwords do not match. * If all validations pass, updates the user's password in the database.   **4- ChangePassword** method returns a success result.  **5- API** sends a confirmation email regarding the password change using \_userService.SendMailChangePassword.  If the email is successfully sent, **API** returns a success response (200 OK) with a confirmation message.  If the email sending fails, **API** returns a bad request response (400 Bad Request) with the error details. |

## General user

### UC 07: Get Course With Behavior

| **Objective:** | This use case allows a student to get course suggestions based on their behavior. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | 1. User logs in successfully 2. User accesses website successfully |
| **Pre-condition:** | User accesses website successfully  The student must be authenticated.  The student must have a role that allows access to course suggestions (Student role). |
| **Post-condition:** | The student receives a list of suggested courses based on their behavior. |
| **Basic flow:** | **1- Student** sends a GET request to the /api/Course/suggest endpoint.  **2- API** retrieves the current user's information using TokenHelper.Instance.GetThisUserInfo(HttpContext).  **3- API** calls the GetCoursesByBehavior method with the user's ID.  **4- GetCoursesByBehavior** method performs the following steps:   * Retrieves the user's behavior from \_userBehaviorRepository.GetBehaviorOfUser. * Retrieves categories related to the user's behavior from \_categoryRepository.GetCategoryByUserBehavior. * Retrieves courses related to the categories from \_courseRepository.GetCourseByUserBehavior.   **5- GetCoursesByBehavior** method returns a success result with the list of suggested courses (CourseDTO).  **6- API** returns a success response (200 OK) with the result. |
| **Exception Flows:** | **1- User Not Authenticated:**   1. API checks if the user is authenticated. 2. If the user is not authenticated, the system returns an unauthorized response (401 Unauthorized).   **2- Internal Server Error:**   1. API encounters an unexpected error during processing. 2. API returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **Description** |
| --- | --- | --- |
| (1) | BR 10 | If user is not enroll any courses, suggest them some course for “beginners” level |

### **UC 08: Update profile**

| **Objective:** | Allow user to update phone number/email to personal information above |
| --- | --- |
| **Actor:** | User (Authenticated) |
| **Trigger:** | The user requests to update their profile. |
| **Pre-conditions:** | The user must be authenticated.  The user must provide valid profile information to update. |
| **Post-conditions:** | The user's profile is updated with the provided information. |
| **Basic flows:** | 1- The user's profile is updated with the provUser sends a POST request to the /api/User/profile endpoint with the updated profile information.  2- API receives the request and calls the UpdateUserProfile service method with the provided UpdateUserProfileDTO.  3- UpdateUserProfile service method performs the following steps:   * Validates the UserID, FullName, and Address fields. * Retrieves the user and user detail records from the database using GetUserByIdAsync and GetUserDetailByIdAsync. * Updates the user's full name and the user detail's address, avatar, and date of birth. * Saves the updated user and user detail records to the database. * Returns a success result.   4- API returns a success response (200 OK) with the result.ided information. |
| **Exception Flows:** | `**Invalid Input**:   1. **API** or **UpdateUserProfile** service detects invalid input (e.g., missing fields, invalid name). 2. **API** or **UpdateUserProfile** service returns a failure response (400 Bad Request) with an appropriate error message.   **User Not Found**:   1. **UpdateUserProfile** service cannot find the user or user detail records. 2. **UpdateUserProfile** service returns a failure response with an error message indicating the user does not exist.   **Internal Server Error**:   1. **API** encounters an unexpected error during processing. 2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue.   **User Not Authenticated**:   1. **API** checks if the user is authenticated. 2. If the user is not authenticated, the system returns an unauthorized response (401 Unauthorized). |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 11 | **Rules for checking information:**   * Users can change the information, except email. * After changed, the system should be updated. |
| (2) | BR 12 | **Save information**  After the email authentication process is successful, the system adds email information to the DB. |

### **UC 09: View and Search course**

| **Objective:** | This use case allows a user to search for courses based on specific criteria. |
| --- | --- |
| **Actor:** | General users |
| **Trigger:** | The user requests to search for courses using specific search criteria. |
| **Pre-conditions:** | User accesses the website successfully.  The user must provide valid search criteria. |
| **Post-conditions:** | The system returns a list of courses matching the search criteria. |
| **Basic flows:** | 1- **User** sends a GET request to the /api/Course/search endpoint with the search criteria as query parameters.  **2- API** receives the request and calls the SearchCourses service method with the provided SearchDTO.  **3- SearchCourses** service method performs the following steps:   * Validates the search and filter fields in the SearchDTO. * Calls the \_courseRepository.SearchCourses method to search for courses based on the search criteria. * Returns a success result with the list of courses.   **4- API** returns a success response (200 OK) with the list of courses. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 13 | Only activated courses can be shown in the list. |
| (2) | BR 14 | Can filter by search keyword (course name, instructor name) |
| (3) | BR 15 | Default has 20 paging per page.  Short default as descending by Course Name, review points. |

### **UC 10:** Study **enrolled course**

| **Objective:** | This use case allows a student to study a specific course content by marking it as completed and updating their progress. |
| --- | --- |
| **Actor:** | student |
| **Trigger:** | The student initiates a request to study a specific course content. |
| **Pre-conditions:** | The student must be authenticated and authorized with the "RequireStudentRole" policy.  The course content ID must exist in the system. |
| **Post-conditions:** | The course content is marked as completed for the student.  The student's progress in the course is updated. |
| **Basic flows:** | 1- **Student** sends a PUT request to the /api/UserProcess/study/{courseContentId} endpoint with the course content ID.  **2- API** receives the request and calls the StudyCourse service method with the provided userId and courseContentId.  **3- StudyCourse** service method performs the following steps:   * Checks if the course content ID exists. * Retrieves the course version detail ID and course version ID associated with the course content ID. * Retrieves the enrolled course ID for the student and course version ID. * Checks if the student has already completed the course content. * If not completed, checks the student's current progress. * If the student is at the correct progress stage, marks the course content as completed. * Updates the student's overall progress in the course. * Returns a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| **Exception flows:** | **Course Content ID Does Not Exist**:   1. **Service** checks if the course content ID exists. 2. **Service** returns a failure result with a "Course Content ID Does Not Exist" error. 3. **API** returns a BadRequest response with the error details.   **Student Progress Not at Correct Stage**:   1. **Service** checks the student's current progress. 2. **Service** returns a failure result with a "Not Complete" error. 3. **API** returns a BadRequest response with the error details.   **Internal Server Error**:   1. **API** encounters an unexpected error during processing. 2. **API** returns a BadRequest response with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 16 | **If that content is updated and student still learn it, update the progress followed by the new one.** |

### **UC 11: View course**

| **Objective:** | This use case allows a student to view a course's details by providing the course ID. It also tracks the user's behavior for recommendation purposes. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | The student requests to view a course by its ID. |
| **Pre-conditions:** | User logs in successfully as actor above.  The student must be authenticated and authorized with the "RequireStudentRole" policy.  The course ID must be valid and exist in the system. |
| **Post-conditions:** | User view courses by category successfully  The system returns the details of the specified course.  The student's behavior is recorded for future course recommendations. |
| **Basic flows:** | 1- **Student** sends a GET request to the /api/Course/click/{courseId} endpoint with the course ID as a path parameter.  **2- API** receives the request and calls the GetCourseById service method with the provided courseId.  **3- GetCourseById** service method performs the following steps:   * Validates the courseId. * Checks if the course exists in the repository. * Retrieves the course details from the repository. * Returns a success result with the course details.   If the course retrieval is successful, **API** calls the UserBehaviorSearch service method to record the student's behavior.  **4- UserBehaviorSearch** service method performs the following steps:   * Retrieves the course and its category. * Checks if the user's behavior already exists in the repository. * Adds or updates the user's behavior in the repository.   **5- API** returns a success response (200 OK) with the course details. |
| **Exception Flows:** | **Course Not Found**:   1. **API** detects that the course does not exist. 2. **API** returns a failure response (404 Not Found) with an appropriate error message.   **Internal Server Error**:   1. **API** encounters an unexpected error during processing. 2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue.   **Invalid Course ID**:   1. **API** detects an invalid or missing course ID. 2. **API** returns a failure response (400 Bad Request) with an appropriate error message. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 17 | Only activated courses can be shown in the list. If not, show empty list |

### **UC 1**2**: Enroll course**

| **Objective:** | This use case allows a student to enroll in a specific course, initializing their progress and setting up the course content for tracking. |
| --- | --- |
| **Actor:** | student |
| **Trigger:** | The student initiates a request to enroll in a specific course. |
| **Pre-conditions:** | 1. The student must be authenticated and authorized with the "RequireStudentRole" policy. 2. The course ID must exist in the system. |
| **Post-conditions:** | The student is enrolled in the course.  The course contents are initialized for the student's progress tracking. |
| **Basic flows:** | 1- **Student** sends a POST request to the /api/Course/Enroll/{courseId} endpoint with the course ID.  **2- API** receives the request and calls the EnrollInCourseAsync service method with the provided userId and courseId.  **3- EnrollInCourseAsync** service method performs the following steps:   * Enrolls the student in the course by creating an entry in the EnrolledCourse table. * Retrieves the latest course version for the specified course. * Retrieves the course version detail ID associated with the latest course version. * Retrieves all course content IDs associated with the course version detail. * Initializes user processes for each course content, setting their completion status to false. * Returns a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| Exception Flows | * **Course ID Does Not Exist**:   1. **Service** checks if the course ID exists.   2. **Service** returns a failure result with a "Course ID Does Not Exist" error.   3. **API** returns a BadRequest response with the error details. * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns a StatusCode(500) response with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 18 | **Only paid course can be enrolled** |

### **UC 1**3**: Review course**

| **Objective:** | This use case allows an instructor to review comments and feedback for their specific course version. |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | User clicks “Review the course” at thank you screen |
| **Pre-conditions:** | 1. The instructor must be authenticated and authorized with the "RequireInstructorRole" policy. 2. The course version ID must exist and belong to the instructor. |
| **Post-conditions:** | The instructor receives the list of reviews for the specified course version. |
| **Basic flows:** | 1- **Instructor** sends a GET request to the /api/CourseComment/review/{courseVersionId} endpoint with the course version ID.  **2- API** receives the request and calls the GetCourseReviewForInstructorByCourseVersionId service method with the provided instructorId and courseVersionId.  **3- GetCourseReviewForInstructorByCourseVersionId** service method performs the following steps:   * Validates the course version ID. * Checks if the course version ID belongs to the instructor. * Retrieves the list of reviews for the specified course version. * Returns a success result with the list of reviews.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest or Unauthorized response with the error details. |
| Exception Flows | * **Course Version ID is Invalid**:   1. **Service** checks if the course version ID is a valid integer and greater than zero.   2. **Service** returns a failure result with a "Wrong Input ID" error.   3. **API** returns a BadRequest response with the error details. * **Course Version Does Not Belong to Instructor**:   1. **Service** checks if the course version ID belongs to the instructor.   2. **Service** returns a failure result with a "Wrong Course of Instructor" error.   3. **API** returns an Unauthorized response with the error details. * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns a StatusCode(500) response with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 19 | N/A |

### **UC 1**4**:** Comment **course**

| **Objective:** | This use case allows an admin or a student to comment on a specific course version. |
| --- | --- |
| **Actor:** | Admin, Student |
| **Trigger:** | The user initiates a request to comment on a specific course version. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminOrStudentRole" policy.  The course version ID must exist. |
| **Post-conditions:** | The comment is successfully added to the course version. |
| **Basic flows:** | 1- **User** sends a POST request to the /api/CourseComment/comment endpoint with the course comment details.  **2- API** receives the request and retrieves the current user's information.  **3- API** determines if the user is an admin or a student:   * If the user is an admin, it calls the AdminCommentCourse service method. * If the user is a student, it calls the CommentCourse service method.   **4- Service Method** performs the following steps:   * Validates the course version ID. * Checks if the course version ID exists. * Creates a new course comment with the provided details. * Saves the comment to the repository. * Returns a success result.   **5- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| Exception Flows | * **Course Version ID is Null**:   1. **Service** checks if the course version ID is null.   2. **Service** returns a failure result with a "Course Version ID Null" error.   3. **API** returns a BadRequest response with the error details. * **Course Version ID Does Not Exist**:   1. **Service** checks if the course version ID exists.   2. **Service** returns a failure result with a "Course Version ID Wrong" error.   3. **API** returns a BadRequest response with the error details. * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns a StatusCode(500) response with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 20 | The student must be enrolled to course |

### **UC 1**5**: View** Dashboard: Top Purchased and Top Bad Courses

| **Objective:** | This use case allows an admin to view the top purchased and top bad courses for a given year, month, or quarter. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin initiates a request to view the top purchased or top bad courses for a specified period. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  The year must be a valid year.  Only one of either month or quarter should be specified, but not both. |
| **Post-conditions:** | The system returns a list of top purchased or top bad courses based on the specified criteria. |
| **Basic flows:** | 1- **Admin** sends a GET request to the /api/CourseVersionDetail/top-purchased or /api/CourseVersionDetail/top-badcourse endpoint with the year, and optionally the month or quarter.  **2- API** receives the request and retrieves the top purchased or top bad courses for the specified period.  **3- Service Method** performs the following steps:   * Validates the year. * Ensures that only one of either month or quarter is specified. * Checks if the year exists in the repository. * Validates the month if specified. * Validates the quarter if specified. * Retrieves the top purchased or top bad courses from the repository based on the criteria. * Returns the result with the list of courses.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| **Exception flows:** | **Invalid Quarter**:   1. **Service** checks if the quarter is between 1 and 4. 2. **Service** returns a failure result with a "Quarter must be between 1 and 4" error. 3. **API** returns a BadRequest response with the error details.   **Year Does Not Exist**:   1. **Service** checks if the year exists in the repository. 2. **Service** returns a failure result with a "Year does not exist" error. 3. **API** returns a BadRequest response with the error details.   **Both Month and Quarter Specified**:   1. **Service** checks if both month and quarter are specified. 2. **Service** returns a failure result with an "Input" error indicating that only one can be specified. 3. **API** returns a BadRequest response with the error details.   **Month Does Not Exist**:   1. **Service** checks if the month exists in the repository. 2. **Service** returns a failure result with a "Month does not exist" error. 3. **API** returns a BadRequest response with the error details.   **Quarter Does Not Exist**:   1. **Service** checks if the quarter exists in the repository. 2. **Service** returns a failure result with a "There is no month in this quarter" error. 3. **API** returns a BadRequest response with the error details. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 21 | The year must be a valid year. |
| (2) | BR 22 | Only one of either month or quarter should be specified, but not both. |

### **UC 1**6**:** View earnings analytics

| **Objective:** | This use case allows an instructor to view the total available money for each of their courses. |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | The instructor initiates a request to view the available money for each of their courses. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireInstructorRole" policy.  The instructor must have at least one course. |
| **Post-conditions:** | The system returns the total available money for each course the instructor has. |
| **Basic flows:** | 1- **Instructor** sends a GET request to the /api/Wallet/received-money-each-course endpoint.  **2- API** receives the request and retrieves the current user's information.  **3- Service Method** performs the following steps:   * Retrieves the instructor ID based on the current user's ID. * Retrieves the list of course IDs associated with the instructor. * For each course, retrieves the list of course version IDs. * Calculates the total available money for each course version. * Returns the result with the total available money for each course.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| **Exception flows:** | **Exception Handling**:   1. If an exception occurs during the process, the API catches the exception. 2. The API returns a BadRequest response with the error message. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 23 | Show the total of money as USD |
| (2) | BR 24 |  |

### **UC** 17**: Create course**

| **Objective:** | This use case allows an instructor to create a new course by providing the course details. |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | The instructor submits a request to create a new course. |
| **Pre-conditions:** | The instructor must be authenticated and authorized with the "RequireInstructorRole" policy.  The category ID provided must be valid and exist in the system.  The course title must be unique. |
| **Post-conditions:** | The course is successfully created and added to the system.  The course version and details are stored, and the course image is uploaded to the Firebase storage. |
| **Basic flows:** | **Instructor** sends a POST request to the /api/Course/add-course endpoint with the course details in the request body.   1. **API** receives the request and calls the CreateCourse service method with the provided courseDTO and the current user information. 2. **CreateCourse** service method performs the following steps:    * Validates the courseDTO fields:      + Checks if CategoryId is provided and valid.      + Checks if Title is provided and unique.      + Checks if Price is provided.    * Generates a new CourseId and CourseVersionDetailId.    * Retrieves the InstructorId using the current user information.    * Uploads the course thumbnail image to Firebase storage.    * Creates a new Course object and saves it to the repository.    * Creates a new CourseVersion object and saves it to the repository.    * Retrieves the CourseVersionId for the newly created course.    * Creates a new CourseVersionDetail object and saves it to the repository.    * Creates a new Image object for the thumbnail and saves it to the repository. 3. **API** returns a success response (200 OK) with the details of the created course version. |
| **Exception Flows**: | * **Category Not Found**:   1. **API** detects that the provided category ID does not exist.   2. **API** returns a failure response (404 Not Found) with an appropriate error message. * **Title Already Exists**:   1. **API** detects that the provided course title already exists.   2. **API** returns a failure response (400 Bad Request) with an appropriate error message. * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 25 | Title must be unique |
| (2) | BR 26 | All information must be saved in the database |

### **UC** 18**:** Update **course detail**

| **Objective:** | This use case allows an instructor to update the details of a course version. |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | The instructor submits a request to update a course version detail. |
| **Pre-conditions:** | User logs in successfully as actor above.  The instructor must be authenticated and authorized with the "RequireInstructorRole" policy.  The course version ID provided must exist in the system. |
| **Post-conditions:** | The course version detail is successfully updated in the system. |
| **Basic flows:** | 1- **Instructor** sends a PUT request to the /api/CourseVersionDetail endpoint with the updated course version details in the request body.  **2- API** receives the request and calls the UpdateCoursevesiondetail service method with the provided UpdateCourseDetailDTO.  **3- UpdateCoursevesiondetail** service method performs the following steps:   * Validates the UpdateCourseDetailDTO fields:   + Checks if CourseVersionId is provided. * Retrieves the CourseId using the provided CourseVersionId. * Retrieves the current course title and details. * Compares the new title, description, and price with the existing values. * Updates the course title if it has changed. * Updates the course version detail description and price if they have changed. * Updates the UpdatedDate field with the current date and time. * Saves the changes to the repository.   **4- API** returns a success response (200 OK). |
| **Exception Flows**: | * **Course Version Not Found**:   1. **API** detects that the provided course version ID does not exist.   2. **API** returns a failure response (404 Not Found) with an appropriate error message. * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 27 | Course is activated or Pending (new) can be edited |

### **UC** 19**:** Update **course content**

| **Objective:** | This use case allows an instructor to update the content of a course version detail. |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | The instructor submits a request to update course content. |
| **Pre-conditions:** | The instructor must be authenticated and authorized with the "RequireInstructorRole" policy.  The course content ID and course version detail ID provided must exist in the system. |
| **Post-conditions:** | The course content is successfully updated in the system. |
| **Basic flows:** | **1- Instructor** sends a PUT request to the /api/CourseContent endpoint with the updated course content details in the request body.  **2- API** receives the request and calls the UpdateCourseContents service method with the provided UpdateCourseContentDTO.  **3- UpdateCourseContents** service method performs the following steps:   * Validates the UpdateCourseContentDTO fields:   + Checks if CourseContentId is provided and exists.   + Checks if CourseVersionDetailId is provided and exists.   + Validates that Title, Url, Time, and Type are provided and valid. * Retrieves the CourseVersionDetailId using the provided CourseContentId. * Creates a new CourseContent object with the updated details. * Saves the new course content to the repository.   **4- API** returns a success response (200 OK). |
| **Exception Flows**: | * **Course Content Not Found:**   1. API detects that the provided course content ID does not exist.   2. API returns a failure response (404 Not Found) with an appropriate error message. * **Internal Server Error:**   1. API encounters an unexpected error during processing.   2. API returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (2) | BR 28 | Course is activated or Pending (new) can be edited |

### 

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (2) | BR 29 | **Display rules**  The system displays a confirmation message MSG 13. MSG 13 has 2 buttons "Agree" and "Cancel" |
| (4) | BR 30 | **Rules for saving information**   * If the user selects “Cancel”, the use case stops and the system displays the previous screen. * If the user chooses "Agree", the system will: Delete community information from the system * Cannot delete the course that has student enrollment. |

### **UC 2**0**: View reviews**

| **Objective:** | Allow user to view all reviews about user's course |
| --- | --- |
| **Actor:** | Instructor |
| **Trigger:** | The instructor initiates a request to review comments on a specific course version. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireInstructorRole" policy.  The course version ID must be valid and belong to the instructor. |
| **Post-conditions:** | The system returns a list of comments for the specified course version if it belongs to the instructor. |
| **Basic flows:** | 1- **Instructor** sends a GET request to the /api/CourseComment/review/{courseVersionId} endpoint with the course version ID.  **2- API** receives the request and retrieves the current user's information.  **3- Service Method** performs the following steps:   * Validates the course version ID. * Retrieves the instructor ID associated with the course version ID. * Checks if the retrieved instructor ID matches the current user's instructor ID. * Retrieves the comments for the course version from the repository. * Returns the result with the list of comments.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |

**Business Rules** : N/A

### **UC 2**1**: View** Bookmarks

| **Objective:** | This use case allows a student to retrieve their list of bookmarked courses. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | The student sends a request to retrieve their bookmarked courses. |
| **Pre-conditions:** | The user must be authenticated with a student role. |
| **Post-conditions:** | The student receives their list of bookmarked courses or an appropriate error message. |
| **Basic flows:** | **1- Student** sends a GET request to the /api/Course/GetListBookMarked endpoint.  **2- API** verifies the student's identity and role using the token from the request.  **3- API** retrieves the user's information by calling TokenHelper.Instance.GetThisUserInfo(HttpContext).  **4- API** calls the \_bookmarkedService.GetListBookMark method with the student's UserId.  **5- Service Method** performs the following steps:   * Retrieves the list of bookmarked courses for the given UserId from the repository. * If bookmarks are found, returns them wrapped in a success result. * If no bookmarks are found, returns a failure result with an appropriate error message.   **6- API** returns the result:   * OK response with the list of bookmarked courses if successful. * BadRequest response with an error message if an exception occurs or no bookmarks are found |
| **Alternative flows:** | N/A |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 32 | Course description (max 200 chars for showing). |

### **UC 2**2**:** Add Bookmark

| **Objective:** | This use case allows a student to add a course to their list of bookmarked courses. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | The student sends a request to add a course to their bookmarks. |
| **Pre-conditions:** | The user must be authenticated with a student role. |
| **Post-conditions:** | The course is added to the student's list of bookmarked courses or an appropriate error message is returned. |
| **Basic flows:** | **1- Student** sends a POST request to the /api/Course/AddBookMark endpoint with the course ID to be bookmarked.  **2- API** verifies the student's identity and role using the token from the request.  **3- API** retrieves the user's information by calling TokenHelper.Instance.GetThisUserInfo(HttpContext).  **4- API** calls the \_bookmarkedService.AddBookmark method with the BookMarkDTO and the student's UserId.  **5- Service Method** performs the following steps:   * Checks if the course exists in the repository. * If the course does not exist, returns a failure result with an appropriate error message. * If the course exists, creates a new Bookmark object and saves it to the repository. * Returns a success result if the bookmark is added successfully.   **6- API** returns the result:   * OK response with a success message if the bookmark is added. * NotFound response if the course does not exist. * Internal Server Error response if an exception occurs. |

**Business Rules**: N/A

### UC 23: Delete Bookmark

| **Objective:** | This use case allows a student to remove a bookmarked course from their list. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | The student sends a request to remove a bookmarked course from their list. |
| **Pre-conditions:** | The user must be authenticated with a student role. |
| **Post-conditions:** | The course is removed from the student's list of bookmarked courses or an appropriate error message is returned. |
| **Basic flows:** | 1- **Student** sends a POST request to the /api/Course/RemoveBookmark endpoint with the BookMarkId to be removed.  **2- API** verifies the student's identity and role using the token from the request.  **3- API** calls the \_bookmarkedService.RemoveBookMark method with the provided BookMarkId.  **4- Service Method** performs the following steps:   * Checks if the bookmark exists in the repository. * If the bookmark does not exist, returns a failure result with an appropriate error message. * If the bookmark exists, removes it from the repository. * Returns a success result if the bookmark is removed successfully.   **5-API** returns the result:   * OK response with a success message if the bookmark is removed. * NotFound response if the bookmark does not exist. * Internal Server Error response if an exception occurs. |

### UC 24: Create Payment (Purchase Course)

| **Objective:** | This use case allows a user to generate a payment URL for depositing money into their wallet using the VnPay payment gateway. |
| --- | --- |
| **Actor:** | Student |
| **Trigger:** | The user initiates a request to create a payment URL for a specified deposit amount. |
| **Pre-conditions:** | The user must be authenticated. |
| **Post-conditions:** | A payment URL is generated and returned to the user. |
| **Basic flows:** | **1- User** sends a GET request to the /api/Payment/create-payment-url endpoint with the depositMoney parameter.  **2- API** verifies the user's identity using the token from the request.  **3- API** retrieves the user's information and calls the \_vpnPaymentService.CreatePaymentUrl method with the HttpContext, depositMoney, and userId.  **4- Service Method** performs the following steps:   * Checks if the depositMoney is greater than the minimum allowed amount (minDepositMoney). * Returns a failure result if the depositMoney is less than minDepositMoney. * Generates a unique transaction reference (vnp\_TxnRef) based on the current timestamp. * Creates a new instance of VnPayLibrary. * Configures the payment request with necessary parameters such as version, command, terminal code, amount, currency code, IP address, locale, order information, order type, return URL, and transaction reference. * Constructs the payment URL using the CreateRequestUrl method. * Returns a success result with the generated payment URL.   **5- API** returns the result:   * OK response with the payment URL if the payment URL is successfully generated. * BadRequest response if there is an error in generating the payment URL. |

**Business Rules**: N/A

## **Admi**n

### UC 25: View list of students

| **Objective:** | Allow admin to manage the user of this system by each type. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The administrator sends a request to get the list of student details. |
| **Pre-conditions:** | The administrator must be authenticated and authorized with the "RequireAdminRole" policy. |
| **Post-conditions:** | The list of students is retrieved and returned to the administrator. |
| **Basic flows:** | 1- **Administrator** sends a GET request to the /api/User/student-detail-list endpoint.  **2- API** receives the request and calls the GetListOfStudent service method.  **3- GetListOfStudent** service method performs the following steps:   * Retrieves the list of students from the repository. * Checks if the list is not empty. * Returns a success result with the list of students if the list is not empty. * Returns a failure result if the list is empty.   **4- API** returns the result to the administrator. |
| **Alternative Flows**: | * **No Students Found**:   1. **Service** detects that the list of students is empty.   2. **Service** returns a failure result with an appropriate error message.   3. **API** returns the failure result to the administrator.   **Exception Flows**:   * **Internal Server Error**:   1. **API** encounters an unexpected error during processing.   2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 33 | **Do not show password in response message** |

### 

### UC 26: Approve Course

| **Objective:** | This use case allows an administrator to approve course |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The administrator initiates the approval of the course. |
| **Pre-conditions:** | The administrator must be authenticated and authorized with the "RequireAdminRole" policy.  The course must exist in the system (not banned, not deleted) |
| **Post-conditions:** | The course approval status is updated.  An email notification is sent. |
| **Basic flows:** | **1- Administrator** sends a PUT request to the /api/Course/approve/{courseId} endpoint with the courseId.  **2- API** receives the request and calls the ApproveCourse service method.  **3- ApproveCourse** service method performs the following steps:   * Retrieves the course version by courseId. * Checks if the course version exists. * Updates the course version's IsApproved status to true and Status to Activate. * Saves the updated course version to the repository. * Sends an email notification (if required). * Returns a success result.   **4- API** returns the result to the administrator. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 34 | Approve email must be followed as ET3 |

### **UC** 27**: Reject** Course

| **Objective:** | This use case allows an administrator to reject course |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The administrator initiates the rejection of the course. |
| **Pre-conditions:** | The administrator must be authenticated and authorized with the "RequireAdminRole" policy.  The course must exist in the system (not banned, not deleted) |
| **Post-conditions:** | The course approval status is updated.  An email notification is sent. |
| **Basic flows:** | 1. **Administrator** sends a PUT request to the /api/Course/reject endpoint with the RejectCourseDTO. 2. **API** receives the request and calls the RejectCourse service method. 3. **RejectCourse** service method performs the following steps:    1. Retrieves the course version by courseId from RejectCourseDTO.    2. Checks if the course version exists.    3. Updates the course version's IsApproved status to false and Status to Deactivate.    4. Saves the updated course version to the repository.    5. Sends an email notification (if required).    6. Returns a success result. 4. **API** returns the result to the administrator. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 35 | Approve email must be followed as ET4 |

### **UC** 28**: View list of categories**

| **Objective:** | This use case allows an administrator to retrieve all categories. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The administrator initiates a request to retrieve all categories. |
| **Pre-conditions:** | The administrator must be authenticated and authorized with the "RequireAdminRole" policy. |
| **Post-conditions:** | The list of categories is returned if available.  Appropriate error messages are returned if there are issues |
| **Basic flows:** | 1- **Administrator** sends a GET request to the /api/Category endpoint.  **2- API** receives the request and calls the GetAllCategory service method.  **3- GetAllCategory** service method performs the following steps:   * Retrieves the list of categories from the repository. * Returns the list of categories wrapped in a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the list of categories. * If not successful, it returns a BadRequest response with the error details. |
| Exception Flows | **Internal Server Error**:   * 1. **API** encounters an unexpected error during processing.   2. **API** returns an internal server error response (500 Internal Server Error) with an error message indicating the issue.   **No Categories Found**:   1. **Service** returns a success result with an empty list. 2. **API** returns an OK response with the empty list. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 36 | **Only active category are appear in the list** |

### **UC** 29**: Create a category**

| **Objective:** | This use case allows an admin to add a new category to the system. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin initiates a request to add a new category. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  The category name should be unique.  The parent ID, if provided, must exist. |
| **Post-conditions:** | The new category is added to the system if all validations pass. |
| **Basic flows:** | 1- **Admin** sends a POST request to the /api/Category endpoint with the new category details.  **2-API** receives the request and calls the CreateProgramLanguage service method with the provided category details.  **3- Service Method** performs the following steps:   * Validates if the category name already exists. * Validates if the parent ID exists (if provided and not "0"). * Generates a new category ID. * Creates a new category object and populates it with the provided details and additional metadata (status, creation date, etc.). * Saves the new category to the repository. * Returns a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| Alternative Flows | * **Validation Failures**:   1. If the category name already exists, the service returns a failure result with the appropriate error message.   2. If the parent ID does not exist, the service returns a failure result with the appropriate error message.   3. If an exception occurs during the process, the API catches the exception and returns a BadRequest response with the error message. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 38 | **Unique CategoryName:**   * The system must ensure that the course name is unique and not already in use by another course. |
| (2) | BR 39 | **Field Requirements**:   * All required fields (Category name and description, except Parent Category) must be filled out before submission. |

### **UC** 30**:** Update **a category**

| **Objective:** | This use case allows an admin to update the details of an existing category (programming language) in the system. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin initiates a request to update an existing category. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  The category ID provided must exist.  The new category name should not be empty |
| **Post-conditions:** | The category details are updated in the system if all validations pass. |
| **Basic flows:** | 1- **Admin** sends a PUT request to the /api/Category endpoint with the updated category details.  **2- API** receives the request and calls the UpDateProgramLanguageInfo service method with the provided category details.  **3- Service Method** performs the following steps:   * Validates if the category ID exists. * Validates if the new category name is not empty. * Fetches the existing category details. * Updates the category details with the new information. * Saves the updated category to the repository. * Returns a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 40 | **Unique CategoryName:**   * The system must ensure that the course name is unique and not already in use by another course. |
| (2) | BR 41 | **Field Requirements**:   * All required fields (Category name and description, except Parent Category) must be filled out before submission. |

### **UC 3**1**: So**ft-d**elete category**

| **Objective:** | This use case allows an admin to soft delete a category by marking it as deleted. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin initiates a request to soft delete a category. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  The category ID provided must exist.  The category must not be associated with any courses. |
| **Post-conditions:** | The category is marked as deleted if all validations pass. |
| **Basic flows:** | 1- **Admin** sends a DELETE request to the /api/Category/{categoryId} endpoint.  **2- API** receives the request and calls the SoftDeleteProgramLanguageAsync service method with the provided category ID.  **3- Service Method** performs the following steps:   * Checks if the category has any associated courses:   + If it does, returns a failure result indicating that the category cannot be deleted because it has associated courses. * Fetches the category details:   + If the category does not exist, returns a failure result indicating that no category was found. * Marks the category as deleted (IsDelete = true). * Updates the category details in the repository. * Returns a success result.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |
| **Exception flow** | **Category Has Associated Courses**:   1. The service method finds that the category has associated courses and returns a failure result with an appropriate error message.   **Category Not Found**:   1. The service method finds that the category does not exist and returns a failure result with an appropriate error message. |

**Activity flow**

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 42 | **Rules for saving information:**  -Cannot delete the category that has course. |

### **UC 3**2**: View/Fil**ter **list of courses**

| **Objective:** | This use case allows an admin to retrieve a filtered list of courses based on the provided filtering configuration. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin initiates a request to retrieve a filtered list of courses. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  The filtering configuration must be provided correctly. |
| **Post-conditions:** | A list of courses that match the filtering criteria is returned to the admin. |
| **Basic flows:** | 1- **Admin** sends a POST request to the /api/Course/Admin/course-list-filter endpoint with a CourseListConfig object in the request body.  **2- API** receives the request and calls the GetCourseListFillterForAdmin service method with the provided filtering configuration.  **3- Service Method** performs the following steps:   * Calls the repository method to get the filtered list of courses based on the provided configuration. * Returns a success result with the filtered course list if no errors occur. * If an exception occurs, it is rethrown for handling.   **4- API** checks if the result is successful:   * If successful, it returns an OK response with the result. * If not successful, it returns a BadRequest response with the error details. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 43 | N/A |

### **UC 3**3**: View details of a course**

| **Objective:** | This use case allows an admin to retrieve detailed information about a specific course using its courseId. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin requests detailed information about a specific course. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  A valid courseId must be provided. |
| **Post-conditions:** | Detailed information about the course is returned to the admin. |
| **Basic flows:** | 1- **Admin** sends a GET request to the /api/Course/detail/{courseId} endpoint with the courseId as a path parameter.  **2- API** receives the request and calls the GetCourseDetailForAdmin service method with the provided courseId.  **3- Service Method** performs the following steps:   * Calls the repository method to get the detailed information for the specified course. * Returns a success result with the detailed course information if no errors occur. * If an exception occurs, it is rethrown for handling.   **4- API** returns the result:   * An OK response with the detailed course information if successful. * If not successful, handle and return appropriate error responses as needed. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 44 | Only courses with status "active" should be displayed in the list of available courses. |

### **UC 3**4**:** Activate **course**

| **Objective:** | This use case allows an admin to activate a specific course version using its courseVersionId. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin requests to activate a specific course version. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  A valid courseVersionId must be provided |
| **Post-conditions:** | The specified course version is activated and made available. |
| **Basic flows:** | 1- **Admin** sends a PUT request to the /api/Course/course-admin-activation/{courseVersionId} endpoint with the courseVersionId as a path parameter.  **2- API** receives the request and calls the ActiveCourseByAdmin service method with the provided courseVersionId.  **3- Service Method** performs the following steps:   * Checks if the course version exists using the repository method. * If the course version does not exist, returns a failure result with an appropriate error. * If the course version exists, activates it by calling the repository method and returns a success result.   **4- API** returns the result:   * An OK response with the result of the activation if successful. * BadRequest if the activation fails. * NotFound if the course version does not exist. * 500 Internal Server Error for unexpected errors. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 45 | The course is approved will applied the changes. |

### UC 35: Deactivate Course

| **Objective:** | This use case allows an admin to deactivate a specific course using its courseId. |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin requests to deactivate a specific course. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy.  A valid courseId must be provided. |
| **Post-conditions:** | The specified course is deactivated and made unavailable. |
| **Basic flows:** | 1- **Admin** sends a PUT request to the /api/Course/course-admin-deactivation/{courseId} endpoint with the courseId as a path parameter and the reason for deactivation as a query parameter.  **2- API** receives the request and calls the DeactiveCourseByAdmin service method with the provided courseId.  **3- Service Method** performs the following steps:   * Checks if the course exists using the repository method. * If the course does not exist, returns a failure result with an appropriate error. * If the course exists, deactivates it by calling the repository method and returns a success result.   **4- API** returns the result:   * An OK response with the result of the deactivation if successful. * BadRequest if the deactivation fails. * NotFound if the course does not exist. * 500 Internal Server Error for unexpected errors. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 46 | The course is rejected will back to the previous version. |

### UC 36: Approve Course

| **Objective:** | This use case allows an admin to approve a course |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin requests to approve a specific course. |
| **Pre-conditions:** | The user must be authenticated with an admin role. |
| **Post-conditions:** | The course is marked as approved in the system.  The course status is updated to active.  A notification email is sent to the relevant parties. |
| **Basic flows:** | **1- Admin** sends a PUT request to the /api/Course/approve/{courseId} endpoint with the courseId of the course to be approved.  **API** verifies the admin's identity and role using the token from the request.  **API** calls the \_courseService.ApproveCourse method with the provided courseId.  **2- Service Method** performs the following steps:   * Retrieves the course version by courseId from the repository. * If the course does not exist, returns a failure result with an appropriate error message. * If the course exists, updates its approval status and sets its status to active. * Saves the updated course version in the repository. * Optionally sends a notification email. * Returns a success result.   **3- API** returns the result:   * OK response with a success message if the course is approved. * NotFound response if the course does not exist. * Internal Server Error response if an exception occurs. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 47 | The course with status “Pending” can be approve |

### UC 37: Reject Course

| **Objective:** | This use case allows an admin to reject a course |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin sends a request to reject a course. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy. |
| **Post-conditions:** | The course is marked as rejected in the system.  The course status is updated to deactivated.  A notification email is sent to the relevant parties. |
| **Basic flows:** | **1- Admin** sends a PUT request to the /api/Course/reject endpoint with the RejectCourseDTO object containing the courseId and reason for rejection.  **2- API** verifies the admin's identity and role using the token from the request.  **3- API** calls the \_courseService.RejectCourse method with the provided RejectCourseDTO.  **4- Service Method** performs the following steps:   * + Retrieves the course version by courseId from the repository.   + If the course does not exist, returns a failure result with an appropriate error message.   + If the course exists, updates its approval status to false and sets its status to deactivated.   + Saves the updated course version in the repository.   + Optionally sends a notification email.   + Returns a success result.   **5- API** returns the result:   * + OK response with a success message if the course is rejected.   + NotFound response if the course does not exist.   + Internal Server Error response if an exception occurs. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 48 | The course with status “Pending” can be reject |

### UC 38: Approve Instructor

| **Objective:** | This use case allows an admin to approve an instructor's application |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin sends a request to approve an instructor. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy. |
| **Post-conditions:** | The instructor's status is updated to approved in the system.  A notification email is sent to the instructor. |
| **Basic flows:** | **1- Admin** sends a POST request to the /api/Instructor/approve/{instructorId} endpoint with the instructorId.  **2- API** verifies the admin's identity and role using the token from the request.  **3- API** calls the \_instructorService.ApproveInstructorAsync method with the provided instructorId.  **4- Service Method** performs the following steps:   * Retrieves the instructor by instructorId from the repository. * If the instructor does not exist, returns a failure result with an appropriate error message. * If the instructor exists, updates their acceptance status to true. * Saves the updated instructor in the repository. * Sends an approval notification email to the instructor. * Returns a success result.   **5- API** calls \_instructorService.SendApprovedInstructorMail to send the notification email.  **6- API** returns the result:   * OK response with a success message if the instructor is approved and the email is sent. * BadRequest response if the instructor is not approved or if an error occurs. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 49 | The instructor with status “pending” can be approve |

### UC 39: Reject Instructor

| **Objective:** | This use case allows an admin to reject an instructor's application |
| --- | --- |
| **Actor:** | Admin |
| **Trigger:** | The admin sends a request to reject an instructor. |
| **Pre-conditions:** | The user must be authenticated and authorized with the "RequireAdminRole" policy. |
| **Post-conditions:** | The instructor's status is updated to rejected in the system.  A rejection notification email is sent to the instructor. |
| **Basic flows:** | **1- Admin** sends a POST request to the /api/Instructor/reject endpoint with the RejectInstructorDTO payload.  **2- API** verifies the admin's identity and role using the token from the request.  **3- API** calls the \_instructorService.RejectInstructorAsync method with the provided RejectInstructorDTO.  **4- Service Method** performs the following steps:   * Retrieves the instructor by instructorId from the repository. * If the instructor does not exist, returns a failure result with an appropriate error message. * If the instructor exists, updates their acceptance status to false. * Saves the updated instructor in the repository. * Sends a rejection notification email to the instructor. * Returns a success result.   **5- API** calls \_instructorService.SendRejectedInstructorMail to send the notification email.  **6- API** returns the result:   * OK response with a success message if the instructor is rejected and the email is sent. * BadRequest response if the instructor is not rejected or if an error occurs. |

**Business Rules**

| **Step** | **BR Code** | **description** |
| --- | --- | --- |
| (1) | BR 50 | The instructor with status “pending” can be reject |

# **Non-functional requirements**

## **Performance requirements**

* Screen loading time should not exceed 5 seconds.

## **Security Requirements**

* The system must encrypt the user's personal information: password
* The system must comply with applicable security standards and policies
* The system should be designed to ensure that data stored in the system is protected against data loss, unauthorized access, use, modification, disclosure or misuse. Only authorized people have access to the data

## **Software quality attributes**

* The system can handle the amount of data as the number of users increases by optimizing storage and access methods.
* The application is guaranteed to run on all operating systems
  + Android: from 8.0 and up
  + iOs: from 13.1 and above
  + Internet required
* The admin portal is guaranteed to run on:
  + Windows 7 and above
  + Chrome v88.0, Firefox, Microsoft Edge

# **Other requirements**

## **Localize rules**

* Number format:
  + Decimal digits: 2
  + Date format: DD/MM/YYYY
* Application language: English

## **Image attachment**

* Allow selecting and uploading 1 image at a time.
* When selecting another image to upload, this will overwrite the old image.
* After image is uploaded, display the image as thumbnail above the button.
* Allow only image files:
  + Bitmap (.bmp)
  + JPEG (.jpg, .jpeg)
  + PNG (.png)
  + GIF (.gif)
* User can delete uploaded image by clicking on option to delete it.
* Maximum allowed size: 5MB, show message “Maximum image attachment size is 5MB.” under the field when user uploaded an image which its size over 5MB.
* If user selects not supported file type or format, show message MSG 28

## Date picker

* Validate immediately after losing focus.
* Format MM/DD/YYYY
  + Show message “The format is invalid.” under the field if user inputted date value with wrong format.
* Place holder: MM/DD/YYYY
* Default value when clicking on :<<Today>>.

## **Internal System Error**

All internal system errors will share the same error message, as was defined in 500 Error, BadMessage

# Appendices

## Message list

| **#** | **Mã số thông báo** | **Nội dung** |
| --- | --- | --- |
|  | MSG 1 | Email should not be empty! |
|  | MSG 2 | The email:'{email}' already used!! |
|  | MSG 3 | Email", $"Email {email} is invalid |
|  | MSG 4 | The phone number:'{phone}' already used!! |
|  | MSG 5 | The user with FullName '{fullname}' already exist!! |
|  | MSG 6 | Password should not be empty! |
|  | MSG 7 | Password must be at least 6 characters long |
|  | MSG 8 | Password must have at least one uppercase letter, one special character and be at least 6 characters long |
|  | MSG 9 | Old password is not matched |
|  | MSG 10 | Full name should not be empty! |
|  | MSG 11 | {name} is invalid name! |
|  | MSG 12 | Date Of Birth should not be empty! |
|  | MSG 13 | Date Of Birth is invalid! |
|  | MSG 14 | Too young to study :( |
|  | MSG 15 | Phone should not be empty! |
|  | MSG 16 | Phone must be 9 characters long! |
|  | MSG 17 | {phone} is invalid VN phone number format! |
|  | MSG 18 | Address should not be empty! |
|  | MSG 19 | User is not exist! |
|  | MSG 20 | Course content ID is not exist |
|  | MSG 21 | You must have completed the previous course first. |
| 22. | MSG 22 | Wrong email or password!! |
| 23. | MSG 23 | You have to enter your email and password!! |
| 24. | MSG 24 | Your account is deleted!! |
| 25. | MSG 25 | Your account is banned!! |
| 26. | MSG 26 | Tax number must not be empty!! |
| 27. | MSG 27 | Card number must not be empty!! |
| 28. | MSG 28 | Card name must not be empty!! |
| 29. | MSG 29 | Card provider must not be empty!! |
| 30. | MSG 30 | Certification is empty. |
| 31. | MSG 31 | Certification must be a PDF, PNG, or JPEG file. |
| 32. | MSG 32 | Certification file size must not exceed 5MB. |
| 33. | MSG 33 | Instructor Id Not Exist |
| 34. | MSG 34 | Category ID is empty, it's said that it's so lonely here |
| 35. | MSG 35 | We're currently not support this category. |
| 36. | MSG 36 | The course '{title}' already exist. Please update that course. |
| 37. | MSG 37 | Title ID is empty, it's said that it's so lonely here |
| 38. | MSG 38 | If this course is free, just type 0 for sure |
| 39. | MSG 39 | Someone was forgot to approve this course |
| 40. | MSG 40 | Course Id is not null here |
| 41. | MSG 41 | Course is not exist |
| 42. | MSG 42 | Course Id {courseId} is invalid |
| 43. | MSG 43 | We need something to identidfy where to place this content |
| 44. | MSG 44 | Not found any thing related with this Id |
| 45. | MSG 45 | Title ID is so lonely here |
| 46. | MSG 46 | We already have this title, please update the exist one |
| 47. | MSG 47 | Where is the url? |
| 48. | MSG 48 | Use another url |
| 49. | MSG 49 | Need to define learning time |
| 50. | MSG 50 | It is a video, a slide or a document? |
| 51. | MSG 51 | Where to update these thing? |
| 52. | MSG 52 | Cannot find the content, check your id |
| 53. | MSG 53 | Course version is not null here |
| 54. | MSG 54 | The course version {ToCourseVersionId} is not exist |
| 55. | MSG 55 | Cannot comment to this course |
| 56. | MSG 56 | The course commnet {courseCommentId} is not exist |
| 57. | MSG 57 | Course commnet id is not null here |
| 58. | MSG 58 | No versions of this course exist |
| 59. | MSG 59 | Course version ID {courseVersionId} is wrong |
| 60. | MSG 60 | There is a version {version} being activated |
| 61. | MSG 61 | There is currently no version activated |
| 62. | MSG 62 | The course is already deactivated |
| 63. | MSG 63 | This is not your course |
| 64. | MSG 64 | CvId not found |
| 65. | MSG 65 | CvId is empty |
| 66. | MSG 66 | This version is right now officially used and cannot change, if you want some changes, please create new version to place content for approval |
| 67. | MSG 67 | The course version is not exist |
| 68. | MSG 68 | Name must not be empty. |
| 69. | MSG 69 | Name must not exceed {maxLength} characters. |
| 70. | MSG 70 | ParentId does not exist. |
| 71. | MSG 71 | ProgramLanguage ID does not exist. |
| 72. | MSG 72 | This program language is already exists |
| 73. | MSG 73 | Amount exceeding available balance |

## Email template

ET 1: Confirmation template

| Gửi đến | <target email> |
| --- | --- |
| Subject | Email Confirmation |
| Body | Welcome to Our Learning Community!  Thank you for registering with our Learning Management System.  Get ready to embark on a journey of learning and growth with our comprehensive educational resources.  Click here to confirm your email |

ET 2: forgot password template.

| Gửi đến | <target email> |
| --- | --- |
| Subject | Forgot your password? |
| Body | Not to worry, we got you! Let's get you a new password.  RESET PASSWORD  If you didn't request to change your password, simply ignore this email.  This link will expire in 24 hours. If you continue to have problems, please feel free to contact us at net09team4@gmail.com |

ET 3: approve Instructor template.

| Gửi đến | <target email> |
| --- | --- |
| Subject | THANK YOU! |
| Body | Congratulations! We are pleased to inform you that your application has been approved. We look forward to welcoming you to our team of instructors. Please expect further communication regarding the next steps. If you have any questions or would like more information regarding the decision, please feel free to contact us at net09team4@gmail.com  Best regards,  Cursus team |

ET 4: reject Instructor template.

| Gửi đến | <target email> |
| --- | --- |
| Subject | THANK YOU! |
| Body | We regret to inform you that your application to register as an instructor has not been approved at this time. After careful consideration, the reason for the rejection is due to [Reason]  We appreciate your interest and encourage you to consider applying again in the future or exploring other opportunities with us. If you have any questions or would like more information regarding the decision, please feel free to contact us at net09team4@gmail.com.  Thank you for your understanding.  Best regards,  Cursus team |

ET 5: Change Password Notice

| Gửi đến | <target email> |
| --- | --- |
| Subject | Want to change your password? |
| Body | Not to worry, we got you! Let s get you a new password.  CHANGE PASSWORD  If you didn t request to change your password, simply ignore this email.  This link will expire in 24 hours. If you continue to have problems, please feel free to contact us at net09team4@gmail.com |

ET 6: Course Closed Notification

| Gửi đến | <target email> |
| --- | --- |
| Subject | Course's notification |
| Body | We wanted to extend our sincere gratitude for purchasing the course "[Course\_Name]". As a valued participant, we hope you've found the content enriching and valuable to your learning journey.  Please be informed that the course will be closing in 30 days. Kindly ensure that you have completed all necessary coursework and assignments by [Closing\_Date] to receive credit for this course.  Thank you once again for choosing to learn with us.  Best regards,  Cursus team |