TUNISIAN REPUBLIC

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF JENDOUBA
HIGHER INSTITUTE OF COMPUTER SCIENCE OF KEF

END OF STUDY PROJECT REPORT

Presented with the aim of obtaining the Bachelor's degree in Computer Science

Subject :

Design and development of an e-learning web application

Directed by :
First Name : Ahmed
Last Name : Hsini
first Name : Mohamed
Last name : Sandi
Supervised by: Mr.Warith Eddine JEDDI

Sustained on : ..../..../2024 Before the jury composed of:
President :
Examiner :

Academic Year 2023-2024

Dedications

With gratitude to Allah, we've crafted this modest work which we dedicate to : In memory and soul of our fathers, the dearest souls in the world, to whom we wish to express our respect, love, and connection. To our dear mothers, sources of tenderness and love, and symbols of sacrifice and kindness. To our dear ones, our siblings, who supported us at every moment of our lives. To our close friends, for their sincere moments of friendship shared and their help throughout this lengthy journey. To all our friends, in remembrance of every moment we have shared, who continue to encourage us and appreciate our efforts. Thank you enormously for being with us.

Acknowledgments

With gratitude and appreciation, we would like to thank Everyone who helped us directly or indirectly Complete this trivial job. We would like to express our sincere gratitude to Mr. Djebbi WarithEddine for his availability, his patience and his valuable advice. Their contributions are essential to our success. We sincerely thank them for their welcome, Their support and encouragement are extremely valuable Our sincere thanks and deep respect to all our teachers and staff working at the Higher Institute of Computer Science. Finally, we take this opportunity to thank the jury members who agreed to judge our work, and we hope that they find in this report the clarity and motivation qualities they expect.

Contents

Chapter 1

Introduction

In today's rapidly evolving world, traditional schooling is no longer sufficient to meet the growing demands for knowledge and skills. With the advent of globalization and the rapid advancement of technology, the landscape of education has undergone significant changes. While schools have long been the primary source of knowledge acquisition, they face challenges to keep up with the dynamic nature of modern society.

In recent years, we have witnessed a dramatic shift in the way education is delivered and consumed. The rise of the internet and digital technologies has transformed the way we learn, making information more accessible than ever before. As a result, learners are no longer confined to the four walls of a classroom. Instead, they have access to a wealth of resources and learning opportunities online.

Furthermore, the COVID-19 pandemic has accelerated this shift to online learning, forcing schools and educational institutions to adapt quickly to remote teaching and learning methods. While this transition has presented its challenges, it has also highlighted the potential of online learning to provide flexible, accessible, and personalized education to learners of all ages and backgrounds.

In this context, the need for e-learning platforms has never been greater. These platforms offer a range of benefits, including the ability to learn at your own pace, access to a wide variety of courses, and resources.

The objective is to create a Tunisian online learning application that helps students and learners to enrich their knowledge and acquire new skills in order to quickly land the job of their dreams.

This report outlines the progression of our work in detail. It is structured into 4 chapters:

- The first chapter, "General Project Overview," serves as an introductory chapter, presenting the project's overall framework, the issues at hand, and proposed solutions.
- The second chapter, "Analysis and Requirements Specification," outlines functional and nonfunctional requirements and explains the role of each actor on the site through use case diagrams.
- The third chapter, "Design," elaborates on the role of each actor through sequence diagrams and class diagrams.
- The fourth chapter focuses on presenting the site's interfaces, showcasing its appearance with illustrative screenshots and explanations, as well as the working environment.

Finally, we conclude this document with a general conclusion, evaluating the work done and proposing new perspectives to enhance our project.

Chapter 2

General Presentation

2.1 Introduction

In this chapter, we focus on presenting the hosting company where we completed our internship. In addition, we will highlight the weaknesses of our study by analyzing and critiquing the existing situation, and we will attempt to envision a solution to our problem.

2.2 Presentation of the hosting organization

This section begins with an introduction to the hosting company, followed by the presentation of its organizational chart.

2.2.1 Presentation of the company

Founded in 2006, the Higher Institute of Computer Science in Kef attracts approximately 1600 students from various geographical backgrounds in Tunisia. This institution is dedicated to the establishment of higher education and research infrastructures. It remains affiliated with the University of Jendouba as an academic institution, imbued with noble values and principles.

- General manager : Mr. Hayouni Mohamed
- general secretary : Mr. Khammassi Aissa
- address : 5 Rue Salah Ayache 7100 le Kef
- Phone/fax : 78 201 056 / 78 200 237
- Site web: `www.isikef.rnu.tn`

Figure 2.1: University Logo

2.2.2 Company organization chart

The following diagram presents a comprehensive organizational chart, highlighting the various departments of the institute in question. More specifically, this internship was carried out within the computer science department, whose management is structured as follows:

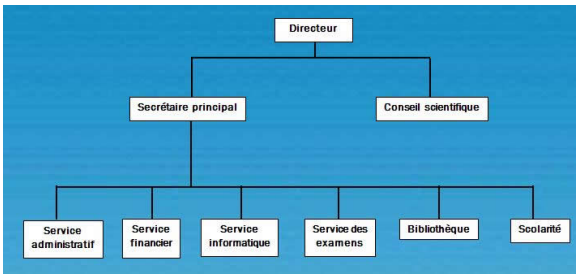

Figure 2.2: Company organization chart

2.3 Subject Presentation

Our project, entitled 'Design and Development of Modules for the E-learning Platform 3alemni.tn ' is part of the end-of-study internship for the third year of the engineering cycle at the ISI (Higher Institute of Computer Science in Ariana). Its aim is to deepen knowledge in the areas of design, programming languages, and collaborative work, preparing learners and students further for the professional world.

2.3.1 Problematic

In the era of ultra-digitalization, where most everyday objects are becoming increasingly digital, we are witnessing accelerated evolution, especially after the first 'lockdown' due to COVID-19. Indeed, in formal education, teaching and learning methods are limited, hence the need for a swift transition to digital technologies and online learning, which eliminates issues such as geographical constraints, etc. Therefore, Bee Coders aims to design an e-learning platform tailored to pedagogical challenges.

2.3.2 Study of the existing situation

Currently, continuing education is conducted in a traditional manner: with courses, learners, and trainers on-site. This type of training presents many disadvantages such as:

- Geographical constraints
- Limited number of classrooms
- Limited seating capacity
- Constraints for learners with physical disabilities or communication problems
- Costs of rent, transportation, and cleaning

To address these disadvantages, several tools have been created using new technologies. Therefore, to deepen our understanding of the subject and gain a clearer idea of the objectives of our project and its expected features, a study of these existing tools is necessary. Among the applications that address the aforementioned problems, we have chosen to study two platforms: one global and one Tunisian :

- **Coursera**
  is a for-profit U.S.-based global massive open online course provider founded in 2012[2][3] by Stanford University computer science professors Andrew Ng and Daphne Koller.[4] Coursera works with universities and other organizations to offer online courses, certifications, and degrees in a variety of subjects.
- **Studytn**
  is a Tunisian online marketplace for video based courses. At study you can learn anything you want at the comfort of your house or on the go. Instructors can sell their courses as well. Youtubers, Instagrammers and public figures can also make huge profit for sharing courses to their audience and earning a big commission for every sale they bring.

2.3.3 Critique of the existing situation

Coursera, while offering a vast array of courses and resources, does have its drawbacks. One notable inconvenience is the lack of interaction, as learners are unable to ask questions for clarification during the course. Additionally, there is no free certification available upon successful completion of a course; learners must pay for the course to obtain certification. Furthermore, access to course materials is limited, as once the course is completed, access expires, making it impossible to review the course content.
For **Study TN**, several shortcomings are apparent. Firstly, courses come with high costs, which may deter potential learners. Furthermore, there is a lack of specific targeting, meaning that the platform may not effectively cater to the needs of certain learners. Furthermore, the system's instability poses a significant issue, with anomalies occurring even at the level of accessing the platform itself. These factors collectively contribute to a less-than-ideal learning experience for users of Study TN.
Referring to the previous sections, we have identified the following limitations:

- Courses with limited access
- System instability
- Lack of interaction: learners are unable to ask questions for clarifications.

2.4 Proposed Solution and Overall Objectives

In order to address the limitations of the previously mentioned applications, we have decided to create a solution that meets the objectives and addresses the identified shortcomings.

2.4.1 Envisaged Solution

The envisaged solution involves the implementation of a web application, the 3alemni.tn e-learning platform, which provides:

- Various online courses on diverse topics(development,design,Business, etc.). The courses are divided into chapters, each containing a set of high-quality videos. All courses will be offered with pre and post-training assessments .
- A forum to facilitate interaction and ask questions for clarification for learners.

Our project is mainly divided into two parts:

- Back-End: for user management, securing personal information, course management, and forum management.
- Front-End: a set of graphical components serving as the user interface to facilitate administrator functionality and user access to courses and the forum.

2.4.2 Overall Objectives

The main objectives of our solution are as follows :

- Ensure Accessibility: Provide open access to courses without limitations, allowing learners to enroll and access course materials at any time.
- Ensure stability: Develop a robust and stable system infrastructure to minimize downtime and technical issues, ensuring uninterrupted access to course content.
- Enhance Interaction: Implement interactive features such as discussion forums, live chat support, and Q&A sessions to facilitate communication and collaboration between learners and instructors, enabling learners to ask questions and seek clarifications in real time.

2.5 Conclusion

In this chapter, we provided a comprehensive overview of our project and its context. We began by introducing the hosting organization, the Higher Institute of Computer Science in el Kef, highlighting its establishment, mission, and organizational structure. Following this, we delved into the presentation of our project, 'Design and Development of Modules for the E-learning Platform 3alemni.tn,' which aims to address the challenges of traditional education through digitalization. We discussed the problem of traditional learning methods and the shortcomings of existing e-learning platforms, such as Coursera and Study TN. By analyzing these limitations, we identified key objectives for our solution, emphasizing accessibility, stability, and interaction. In summary, this chapter sets the stage for our project's development, outlining the rationale behind our solution and its overarching goals.

Chapter 3

Analysis and Specification of Requirements

3.1 Introduction

In this chapter, we focus on understanding what our application needs to do and how it will work. We'll outline the requirements it must meet, both in terms of functionality and overall quality. Identifying the different users and their roles, we can better tailor the application to their needs. Let us dive into the specifics of what our project requires.

3.2 Identification of system actors

Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data. In our application we can identify three main actors:

- Student : These are the users who register on the platform to take courses, access educational resources, and participate in learning activities.
- Educator : These are the professionals who create and deliver courses on the platform. They may be responsible for creating educational content, assessing students.
- Administrator : LAdministrators are responsible for the overall management of the site. They can view courses, manage student registrations, monitor site activity, and resolve technical issues.

3.3 Specification of requirements

When analyzing requirements, it is important to distinguish between functional and non-functional needs. Functional requirements describe what the project's product or service should achieve, while non-functional requirements outline constraints related to its implementation and deployment

3.3.1 Functional needs

The application to be developed must offer a set of features that correspond to a set of user needs. These needs define the services users expect to be provided by this application. The main business requirements of our application are summarized in the following functionalities:

- **Students** : Students should be able to :
  - Register and authenticate.
  - Browse courses.
  - Access to educational resources such as PDF documents and course videos.

- Participation in interactive learning activities such as quizzes and discussion forums.
- Ability to communicate with instructors to ask questions or request assistance.

- **Educator**: Educators should be able to :
  - Register and Authenticate .
  - Creation and management of courses, including adding educational content, defining learning objectives.
  - Interact with students through discussion forums or emails.

- **Administrator** : The administrator should be able to :
  - Authenticate.
  - User management .
  - Monitoring courses .

3.3.2 Non-functional needs

To ensure the success of our solution, the application must verify certain properties and take into account specific constraints and requirements :

- **Confidentiality**: All information is shared only with members who have a need to know.
- **Security**: Access to information is only possible after verifying privileges and access rights. All developed web services require an authorization phase with tokens before they can be used, thanks to authentication and password encryption.
- **Ease of use**: The user interface should be user-friendly and intuitive to allow for easy navigation.
- **Compatibility**: The application must be compatible with different web browsers and devices.
- **Availability**: The application must be available 24/7 with minimal downtime for maintenance.
- **Robustness**: The system must handle exceptions in the presence of invalid data to ensure stable operation.

3.4 A use case diagram

Use case diagrams describe the general functions and scope of a system. They also highlight the interactions between the system and its actors. The use cases and actors in the use case diagrams describe what the system does and how actors use it, but they do not illustrate the internal workings of the system.

3.4.1 Global use case diagram

This is our global use case diagram, it provides an overview of the various interactions between actors and the system as a whole.

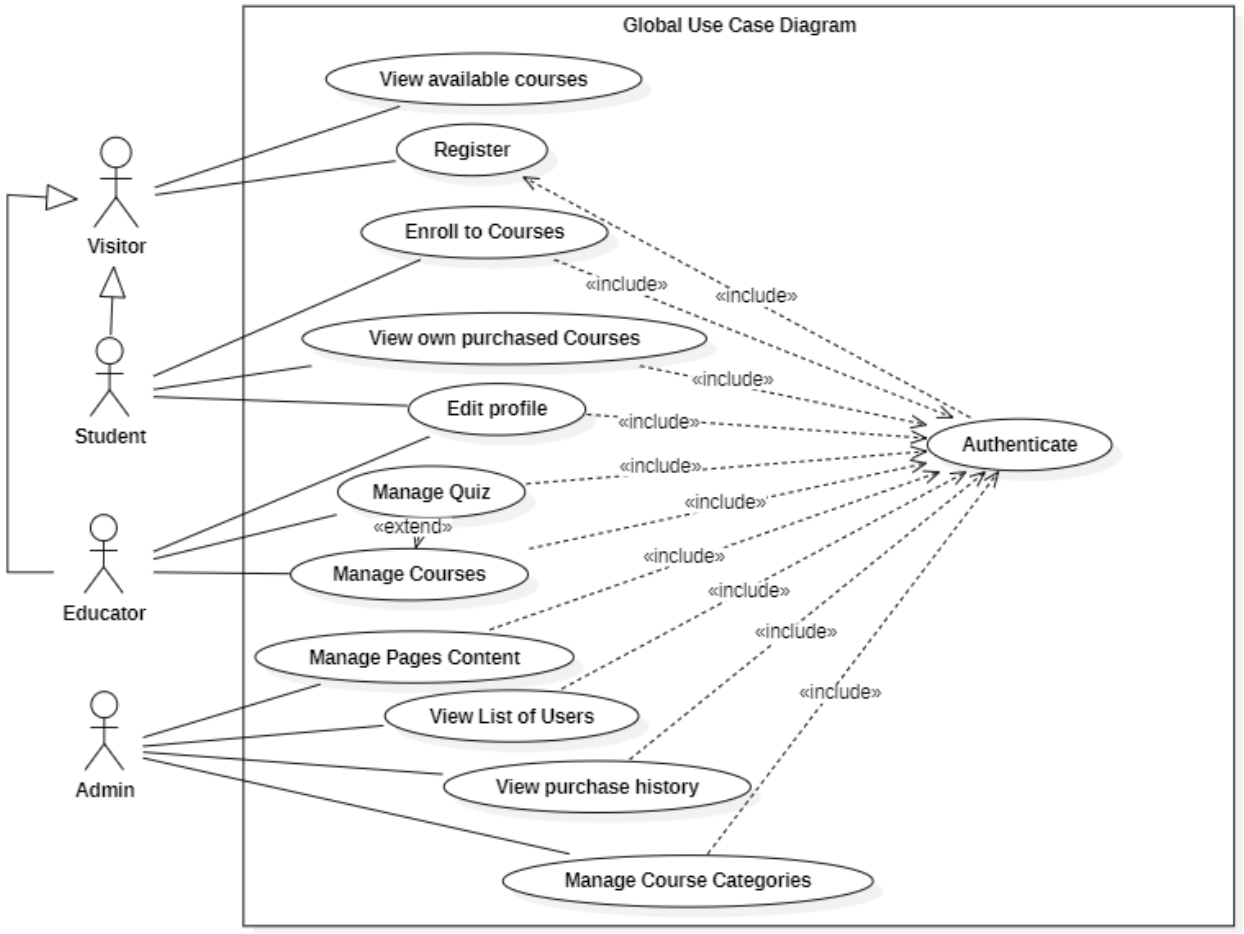

Figure 3.1: Global Use Case diagram

3.4.2 Refined use case diagram

A refined use case diagram provides a detailed view of specific interactions between actors and the system. It includes more detailed use cases and actors compared to the global use case diagram

Use case diagram « Authenticate »

The "Authenticate" use case diagram is the main case that encompasses all types of users: Administrator, Student, and Educator .

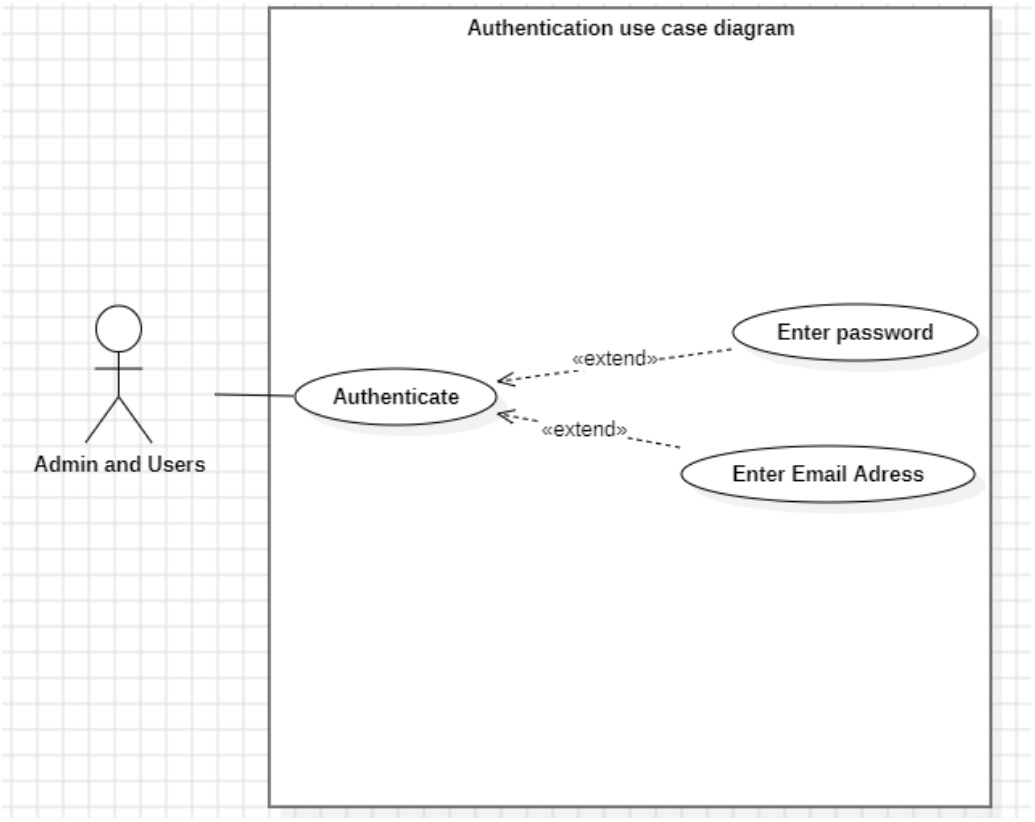

Figure 3.2: Use case diagram « Authenticate »

Table 3.1: Textual description of the « Authenticate » use case

Title of the use case	**Authenticate**
Objective	The user logs in to the system to access their account.
Actors	Administrator, Student et Educator
Preconditions	• The user must have a registered account. • The user must have their username and password.
Postconditions	The user is logged in to their account and can access the system features.
Main Success Scenario	1. The user navigates to the login page. 2. The user enters their email and password. 3. The system verifies the credentials. 4. The system offers the user access to the interfaces that are dedicated to him.
Extension A	2. If the fields are empty, the system displays an error message.
Extension B	3. **Invalid Credentials**: If the credentials provided by the user are incorrect or the account is not verified , the system displays an error message .

Use case diagram « Manage Courses »

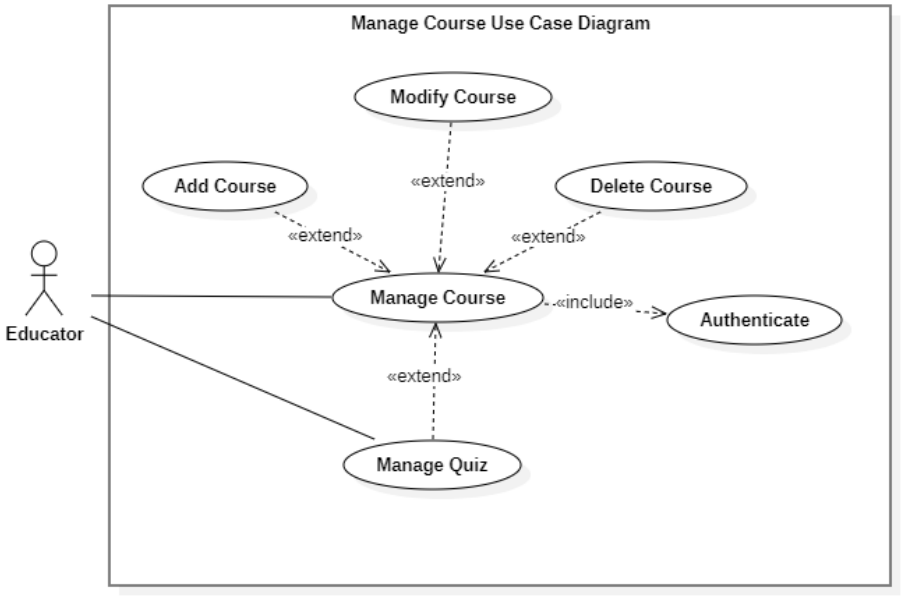

Figure 3.3: Use case diagram « Manage Courses »

Table 3.2: Textual description of the « Add Courses » use case diagram

Title of the use case	**Add Courses**
Objective	To enable educators to create and add new courses to the platform, providing students with access to new learning materials and resources.
Actors	Educator
Preconditions	• The educator or administrator must be logged into the system. • Required course information and materials must be prepared and available.
Postconditions	• A new course is successfully created and added to the platform. • Students can enroll in and access the newly created course.
Main Success Scenario	1. the Educator navigates to his dashboard . 2. The educator clicks on the "Add Course" button. 3. The system displays a form for creating a new course. 4. The educator fills in the course details, including Course title , Course description , Course category and Course image . 5. The educator submits the form. 6. The system creates the new course and shows it in the Educators dashboard .
Extensions	• **Invalid or incomplete data :** If the Educator enters invalid or incomplete data an error message is displayed .

Table 3.3: Textual description of the « Edit Courses » use case

Title of the use case	**Edit Courses**
Objective	Allow educators to modify and update the details and content of existing courses on the platform.
Actors	Educator
Preconditions	• The educator must be logged into the system. • The course to be edited must already exist in the system.
Postconditions	The selected course is successfully updated with the new details and content. Students can access the updated course information and materials.
Main Success Scenario	1. The educator navigates to the dashboard . 2. The educator selects the course they wish to edit. 3. The system displays the current course details and content. 4. The educator makes the necessary changes, including: (a) Updating the course title (b) Modifying the course description (c) Modifying the course image . (d) Uploading new or updated course materials (PDFs, videos , quizzes) 5. The educator submits the changes. 6. The system updates the course with the new information.
Extension A	4. If the educator enters invalid data: (a) The system displays an error message. (b) The educator corrects the data and resubmits.

Table 3.4: Textual description of the « Delete Course » use case

Title of the use case	**Delete Course**
Objective	Allow educators to remove existing courses from the platform.
Actors	Educator
Preconditions	• The educator must be logged into the system. • The course to be deleted must already exist in the system.
Postconditions	The selected course data and all the related materials are permanently removed from the platform.
Main Success Scenario	1. The educator navigates to their dashboard. 2. The educator selects the course they wish to delete. 3. The system prompts the educator to confirm the deletion. 4. The educator confirms the deletion. 5. The system deletes the course and all associated data. 6. The system confirms the successful deletion of the course.
Extension A	3. If the educator does not confirm the deletion: (a) The system cancels the deletion process. (b) The course remains on the platform.

Use case diagram « Manage quiz »

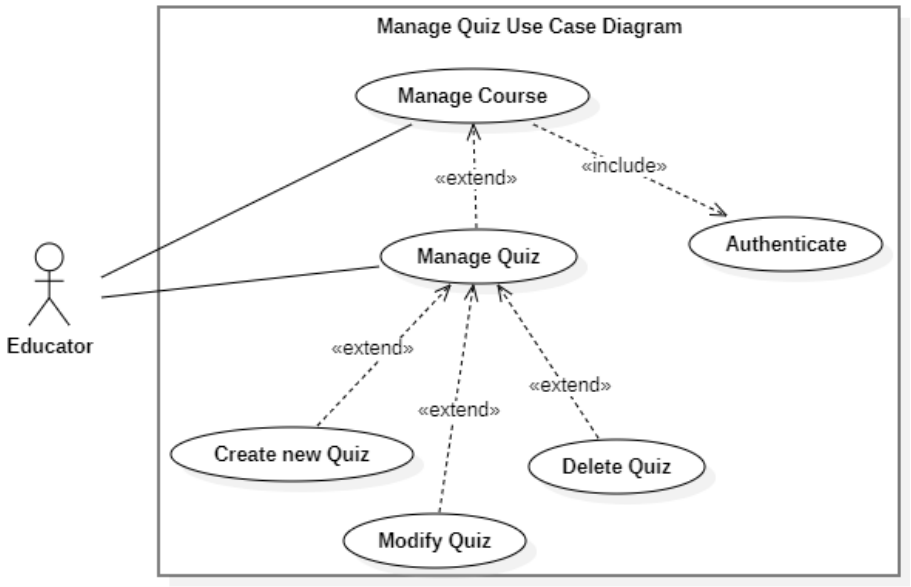

Figure 3.4: Use case diagram « Manage quiz »

Table 3.5: Textual description of the « Add New Quiz » use case

Title of the use case	**Add New Quiz**
Objective	To allow Educators to create and add a new quiz within their courses .
Actors	Educator
Preconditions	• The educator has successfully logged in to the system. • The educator must have access to the course for which they are adding a new quiz.
Postconditions	• The new quiz is successfully created and added to the course. • Students who have access to the course should be able to take the new quiz.
Main Success Scenario	1. The educator navigates to the course details page. 2. The educator clicks on the "Quiz" button to access the quiz details page. 3. The system displays an option to create a new quiz. 4. The educator clicks on the "Create New Quiz" button. 5. The educator enters the quiz details and questions. 6. The educator submits the new quiz. 7. The system confirms the creation and adds the quiz to the course.
Extensions	• **Invalid Data:** If the educator enters invalid data, an error message is displayed.

Table 3.6: Textual description of the « Modify Quiz » use case

Title of the use case	**Modify Quiz**
Objective	To allow Educators to modify an existing quiz within their courses to update or improve the quiz content.
Actors	Educator
Preconditions	• The educator has successfully logged in to the system. • The educator must have access to the course and the quiz they wish to modify.
Postconditions	• The quiz is successfully modified within the course. • Students who have access to the course should see the updated quiz content.
Main Success Scenario	1. The educator navigates to the course details page. 2. The educator clicks on the "Quiz" button to access the quiz details page. 3. The system displays a list of existing quizzes. 4. The educator presses " edit " on the quiz they wish to modify. 5. The educator makes the necessary changes to the quiz details and questions. 6. The educator submits the modifications. 7. The system confirms the modifications and updates the quiz in the course.
Extensions	• **Invalid Data:** If the educator enters invalid data, an error message is displayed.

Table 3.7: Textual description of the « Delete Quiz » use case

Title of the use case	**Delete Quiz**
Objective	To allow Educators to delete an existing quiz from their courses.
Actors	Educator
Preconditions	• The educator has successfully logged in to the system. • The educator must have access to the course and the quiz they wish to delete.
Postconditions	• The quiz is successfully deleted from the course. • Students who have access to the course should no longer see the deleted quiz.
Main Success Scenario	1. The educator navigates to the course details page. 2. The educator clicks on the Quiz to access the quiz details page. 3. The system displays a list of existing quizzes. 4. The educator selects the quiz they wish to delete. 5. The educator confirms the deletion. 6. The system deletes the quiz from the course and confirms the deletion.
Extensions	• **Cancellation:** If the educator cancels the deletion, no changes are made.

Use case diagram « Edit Profile »

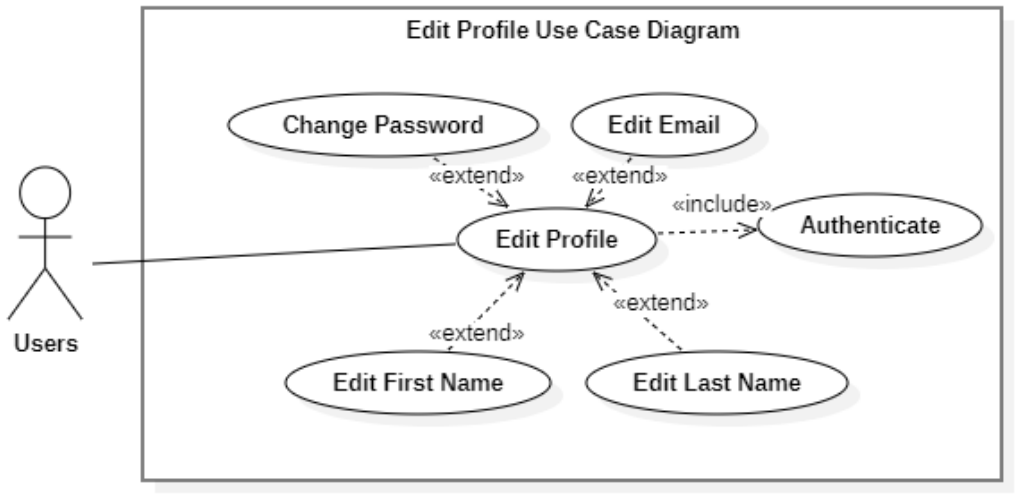

Figure 3.5: Use case diagram « Edit Profile »

Table 3.8: Textual description of the « Edit Profile » use case

Title of the use case	**Edit Profile**
Objective	To allow users to update their personal information .
Actors	Student, Educator
Preconditions	• The user has successfully logged in to the system. • The user navigates to their profile page.
Postconditions	• The user's profile information is successfully updated. • The updated profile information is saved in the system.
Main Success Scenario	1. The user press on " Edit Profile " in the navigation bar . 2. The system displays the current profile information. 3. The user updates the desired information (e.g., name, email, password, profile picture). 4. The user submits the changes. 5. The system validates the input data. 6. The system saves the updated profile information. 7. The system confirms the successful update to the user.
Extensions	• **Invalid Data:** If the user enters invalid data, the system displays an error message and prompts the user to correct the information. • **Cancellation:** If the user cancels the edit, no changes are made to the profile.

Use case diagram « Enroll to Course »

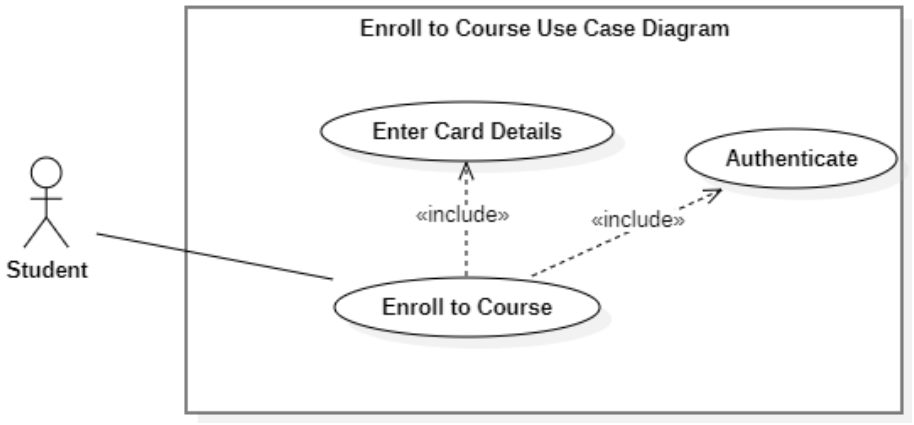

Figure 3.6: Use case diagram « Enroll to Course »

Table 3.9: Textual description of the « Enroll to Course » use case

Title of the use case	**Enroll to Course**
Objective	To allow students to enroll in course and access it's related materials .
Actors	Student
Preconditions	• The student has successfully logged in to the system. • The student has a valid payment method if the course requires a fee.
Postconditions	• The student is successfully enrolled in the selected course. • The course is added to the student's list of " My courses ".
Main Success Scenario	1. The student navigates to the course catalog. 2. The student selects a course to enroll in. 3. The student press on "Buy Now " . 4. The student provides card details if required and confirms the enrollment. 5. The system processes the enrollment and updates the student's course list.
Extensions	• **Invalid Authentication:** If the student is not authenticated, the system prompts the student to log in.

Use Case Diagram « Manage Categories»

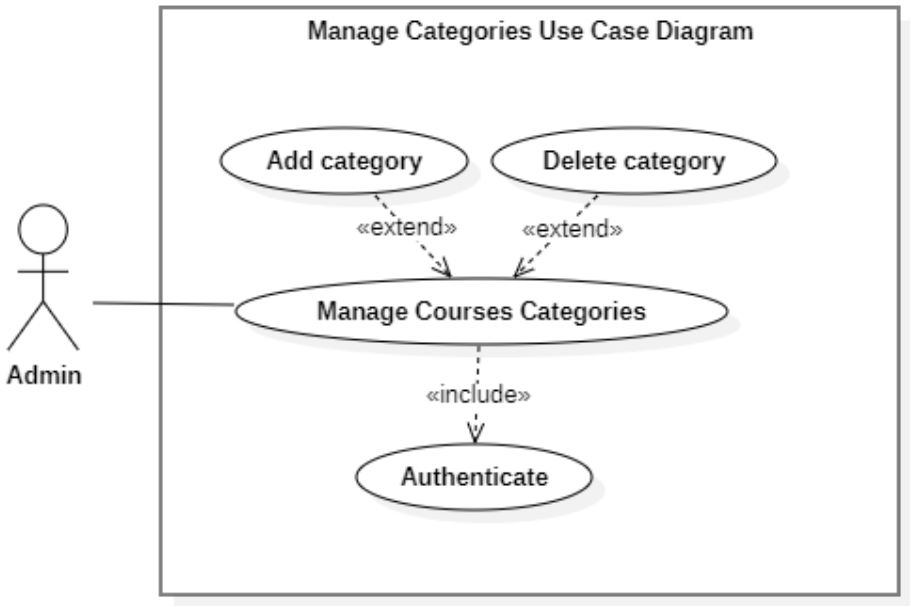

Figure 3.7: Use Case Diagram « Manage Categories»

Table 3.10: Textual description of the « Add Category » use case

Title of the use case	**Add Category**
Objective	To allow the admin to add a new category to the system.
Actors	Admin
Preconditions	The admin is authenticated .
Postconditions	The new category is successfully added to the system.
Main Success Scenario	1. The admin navigates to the category management section . 2. The admin selects the option to add a new category . 3. The admin fills in the details for the new category, such as name, description, etc. 4. The admin submits the form to add the category. 5. The system validates the information and adds the new category to the system.
Extensions	• **Invalid Information:** If the admin submits incomplete or invalid information, the system displays error messages prompting the admin to correct the errors and resubmit the form.

Table 3.11: Textual description of the « Delete Category » use case

Title of the use case	**Delete Category**
Objective	To allow the admin to delete an existing category from the system.
Actors	Admin
Preconditions	The admin is authenticated .
Postconditions	The specified category is successfully deleted from the system.
Main Success Scenario	1. The admin navigates to the category management section of the system. 2. The admin selects the category to be deleted from the list of existing categories. 3. The admin confirms the deletion action. 4. The system removes the selected category from the system.
Extensions	• **Category In Use:** If the category to be deleted is currently associated with any items or content in the system, the system displays a warning message informing the admin about the dependencies and prompts for confirmation.

Table 3.12: Textual description of the « Modify Category » use case

Title of the use case	**Modify Category**
Objective	To allow the admin to modify the details of an existing category in the system.
Actors	Admin
Preconditions	The admin is authenticated .
Postconditions	The specified category is successfully updated with the new details in the system.
Main Success Scenario	1. The admin navigates to the category management section of the system. 2. The admin selects the category to be modified from the list of existing categories. 3. The admin edits the details of the selected category, such as name, description, etc. 4. The admin submits the form to update the category. 5. The system validates the information and updates the selected category with the new details.
Extensions	• **Invalid Information:** If the admin submits incomplete or invalid information, the system displays error messages prompting the admin to correct the errors and resubmit the form.

Chapter 4

Design

4.1 Introduction

In this chapter, we delve into the detailed design of the system, focusing on both the dynamic and static aspects. The design process bridges the gap between high-level requirements and implementation, providing a clear blueprint for developers. Key elements in this chapter include sequence diagrams and class diagrams.

4.2 Sequence Diagrams

Sequence diagrams offer a dynamic view of the system, illustrating how objects interact through a particular sequence of messages. They highlight the temporal aspect of processes, showcasing the order of interactions and the flow of information among different components. This helps in understanding the behavior of the system under various scenarios.
In the next section we will be presenting some of our main application functionalities sequence diagrams to help easily understanding the work flow

4.2.1 Sequence Diagram « Register »

In this section, we will present the sequence of steps for registration using this sequence diagram :

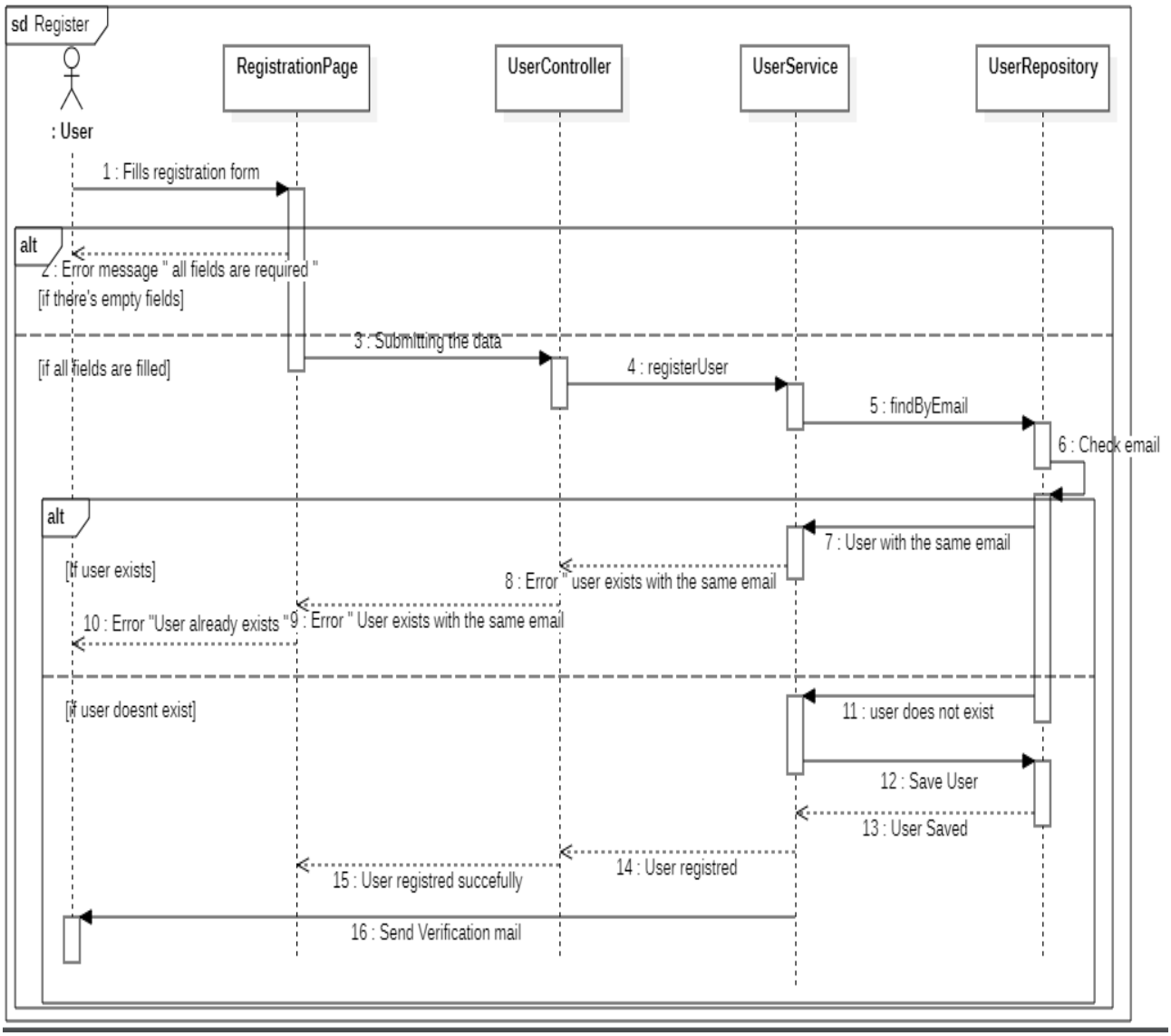

Figure 4.1: Sequence Diagram « Register »

Textual Description of the Registration Sequence Diagram

1. **User:** Fills out the registration form on the RegistrationPage.
2. **Validation Check:**
   - If there are empty fields, RegistrationPage returns an error message: "All fields are required".
3. **Submit Form:** If all fields are filled, RegistrationPage submits the data.
4. **UserController:** Calls `registerUser` in UserService.
5. **UserService:** Calls `findByEmail` in UserRepository.

6. **UserRepository:** Checks if the email exists.
7. **Email Exists:**
    - If email is found:
        - UserService returns an error to UserController.
        - UserController returns the error "User already exists" to RegistrationPage.
        - RegistrationPage displays the error message.
8. **New User:**
    - If email is not found:
        - UserService calls `saveUser` in UserRepository.
        - UserRepository saves the user data.
9. **Success:**
    - UserRepository confirms the user is saved.
    - UserService returns success to UserController.
    - UserController returns success to RegistrationPage.
    - RegistrationPage displays a success message and sends a verification email.

4.2.2 Sequence Diagram « Login »

In this section we will present the login process that will allow our application's users to access to their specific pages .

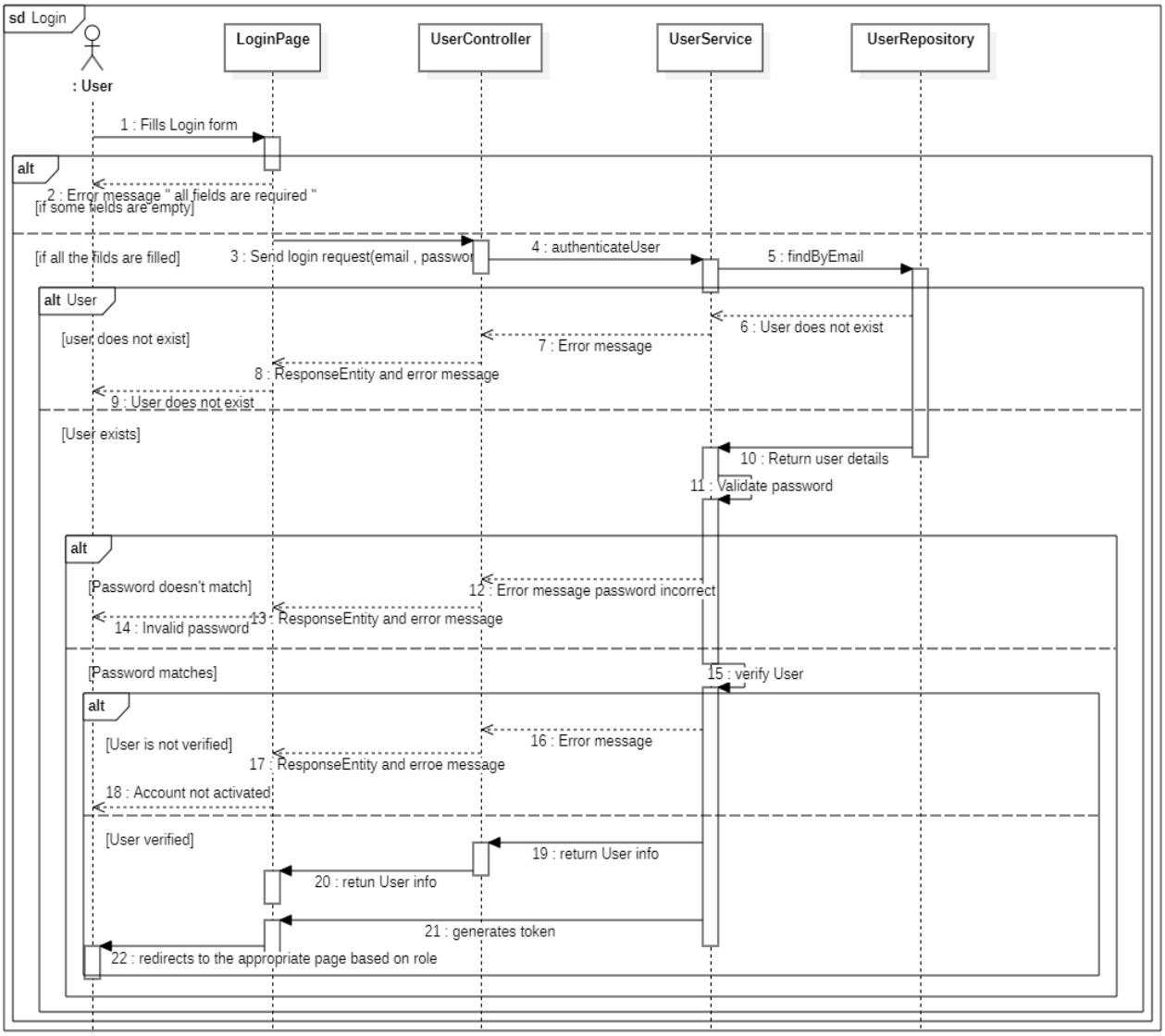

Figure 4.2: Sequence Diagram « Login »

Textual Description of the Login Sequence Diagram

1. **User:** Fills out the login form on the LoginPage.
2. **Validation Check:**
   - If some fields are empty, LoginPage returns an error message: "All fields are required".
3. **Submit Login Request:** If all fields are filled, LoginPage sends the login request (email, password).
4. **UserController:** Calls `authenticateUser` in UserService.
5. **UserService:** Calls `findByEmail` in UserRepository.

6. **UserRepository:** Checks if the user exists.
7. **User Exists Check:**
    - If user does not exist:
        - UserService returns an error to UserController.
        - UserController returns the error "User does not exist" to LoginPage.
        - LoginPage displays the error message.
8. **Password Validation:**
    - If user exists:
        - UserService validates the password.
        - If password doesn't match:
            * UserService returns an error to UserController.
            * UserController returns the error "Password incorrect" to LoginPage.
            * LoginPage displays the error message.
        - If password matches:
            * UserService verifies the user.
            * If user is not verified:
                · UserService returns an error to UserController.
                · UserController returns the error "Account not activated" to LoginPage.
                · LoginPage displays the error message.
            * If user is verified:
                · UserService returns user info to UserController.
                · UserController generates a token.
                · UserController redirects to the appropriate page based on user role.

4.2.3 Sequence Diagram « Reset Password »

In this section we will present the reset password process that allows our users to reset their password after loosing it and gets access to their account .

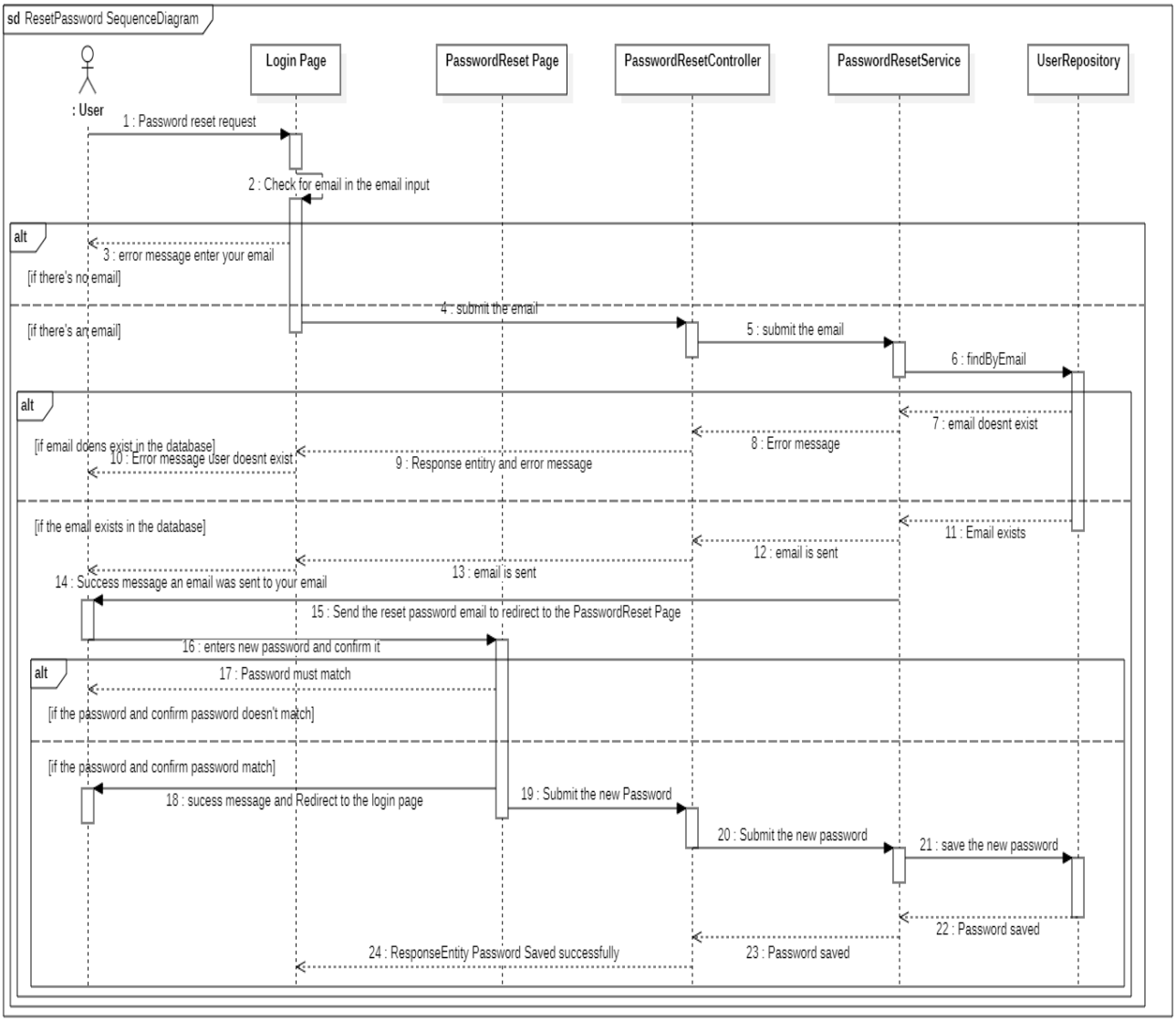

Figure 4.3: Sequence Diagram « Reset Password »

Textual Description of the Reset Password Sequence Diagram

1. **User:** Sends a password reset request on the LoginPage.
2. **Validation Check:**
   - If no email is provided, LoginPage returns an error message: "Enter your email".
3. **Submit Email:** If email is provided, LoginPage submits the email.
4. **PasswordResetController:** Calls `findByEmail` in PasswordResetService.
5. **PasswordResetService:** Calls `findByEmail` in UserRepository.

6. **UserRepository:** Checks if the email exists.
7. **Email Exists Check:**
    - If email does not exist:
        - PasswordResetService returns an error to PasswordResetController.
        - PasswordResetController returns the error "User does not exist" to LoginPage.
        - LoginPage displays the error message.
    - If email exists:
        - PasswordResetService sends a password reset email.
        - PasswordResetController confirms the email is sent.
        - LoginPage displays a success message.
8. **Password Reset:** User enters a new password and confirms it.
9. **Password Match Check:**
    - If passwords do not match:
        - LoginPage returns an error message: "Passwords must match".
    - If passwords match:
        - LoginPage submits the new password.
        - PasswordResetController calls `saveNewPassword` in PasswordResetService.
        - PasswordResetService saves the new password via UserRepository.
        - UserRepository confirms the password is saved.
        - PasswordResetService returns success to PasswordResetController.
        - PasswordResetController returns success to LoginPage.
        - LoginPage displays a success message and redirects to the login page.

4.2.4 Sequence Diagram « Edit Profile »

In this section we will present the steps to Edit the users profiles and update it , the steps are presented in this sequence diagram :

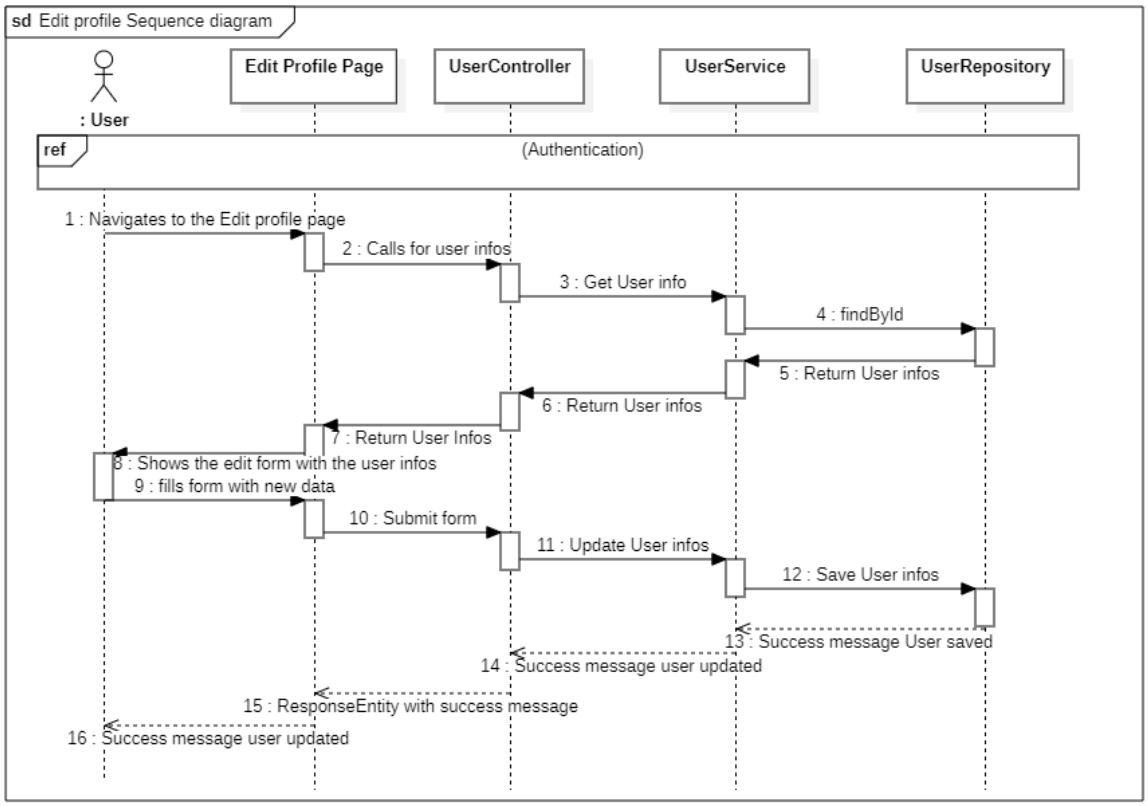

Figure 4.4: Sequence Diagram « Edit Profile »

Textual Description of the Edit Profile Sequence Diagram

1. **User:** Navigates to the Edit Profile page.
2. **UserController:** Calls for user info.
3. **UserService:** Calls `findById` in UserRepository.
4. **UserRepository:** Retrieves and returns user info.
5. **UserController:** Returns user info to Edit Profile Page.
6. **Edit Profile Page:** Shows the edit form with user info.
7. **User:** Fills the form with new data and submits it.
8. **UserController:** Calls `updateUser` in UserService.
9. **UserService:** Calls `save` in UserRepository to update user info.
10. **UserRepository:** Saves the updated user info and confirms.
11. **UserService:** Returns success message to UserController.
12. **UserController:** Returns success message to Edit Profile Page.
13. **Edit Profile Page:** Displays success message to User.

4.2.5 Sequence Diagram « Create Courses »

For this section we will deep into the Educators most important activity is creating his own courses , this Sequence describes the process of creating a course :

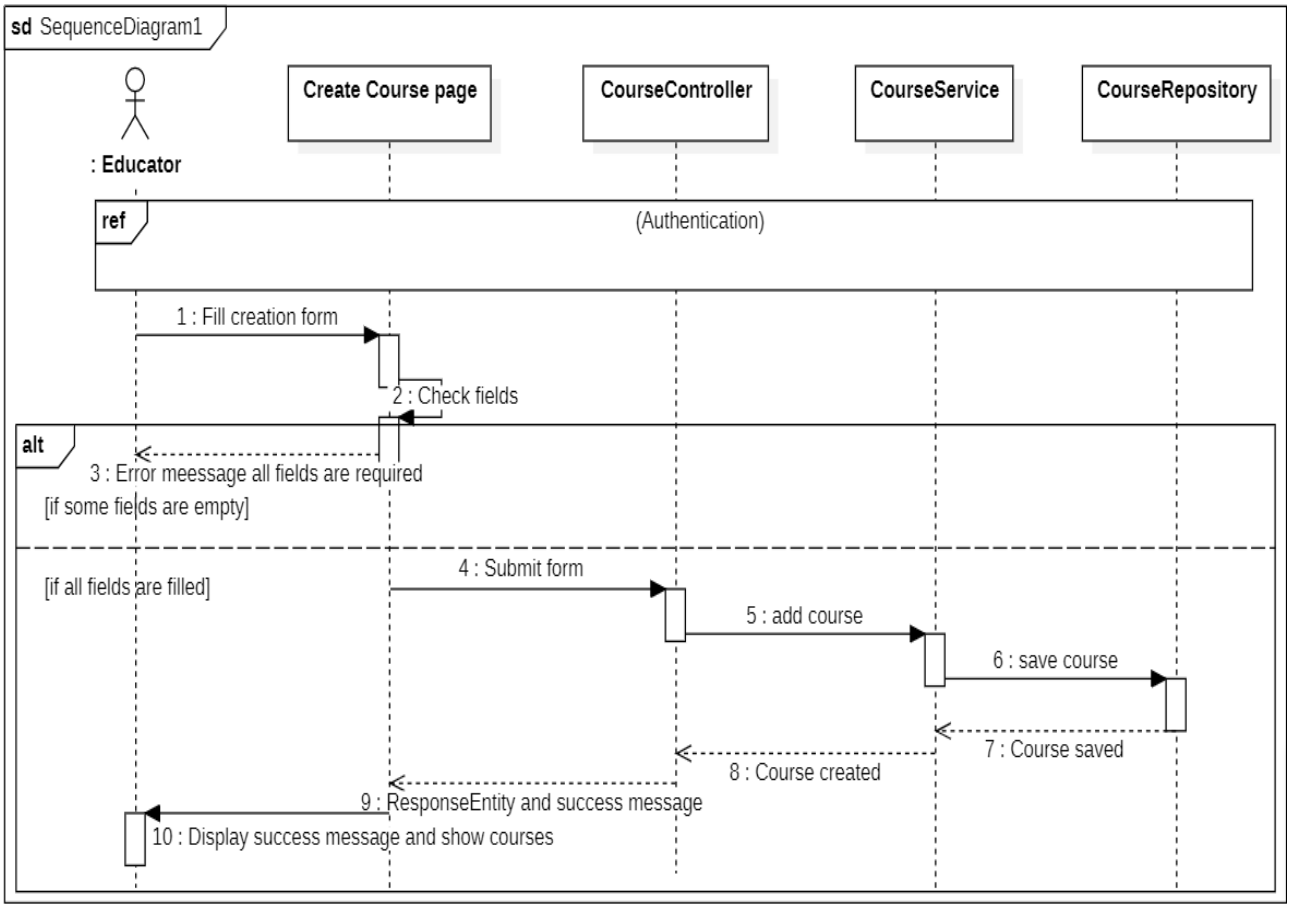

Figure 4.5: Sequence Diagram « Create Course »

Textual Description of the Create Course Sequence Diagram

1. **Educator:** Fills out the course creation form on the Create Course page.
2. **Create Course Page:** Checks if all fields are filled.
   - If some fields are empty, returns an error message: "All fields are required".
3. **Submit Form:** If all fields are filled, Create Course Page submits the form.
4. **CourseController:** Calls `addCourse` in CourseService.
5. **CourseService:** Calls `saveCourse` in CourseRepository.
6. **CourseRepository:** Saves the course and confirms.
7. **CourseService:** Returns success message to CourseController.
8. **CourseController:** Returns success message to Create Course Page.
9. **Create Course Page:** Displays success message and shows the courses.

4.2.6 Sequence Diagram « Edit Course »

Now for Edit Course sequence diagram it's represented as the following diagram :

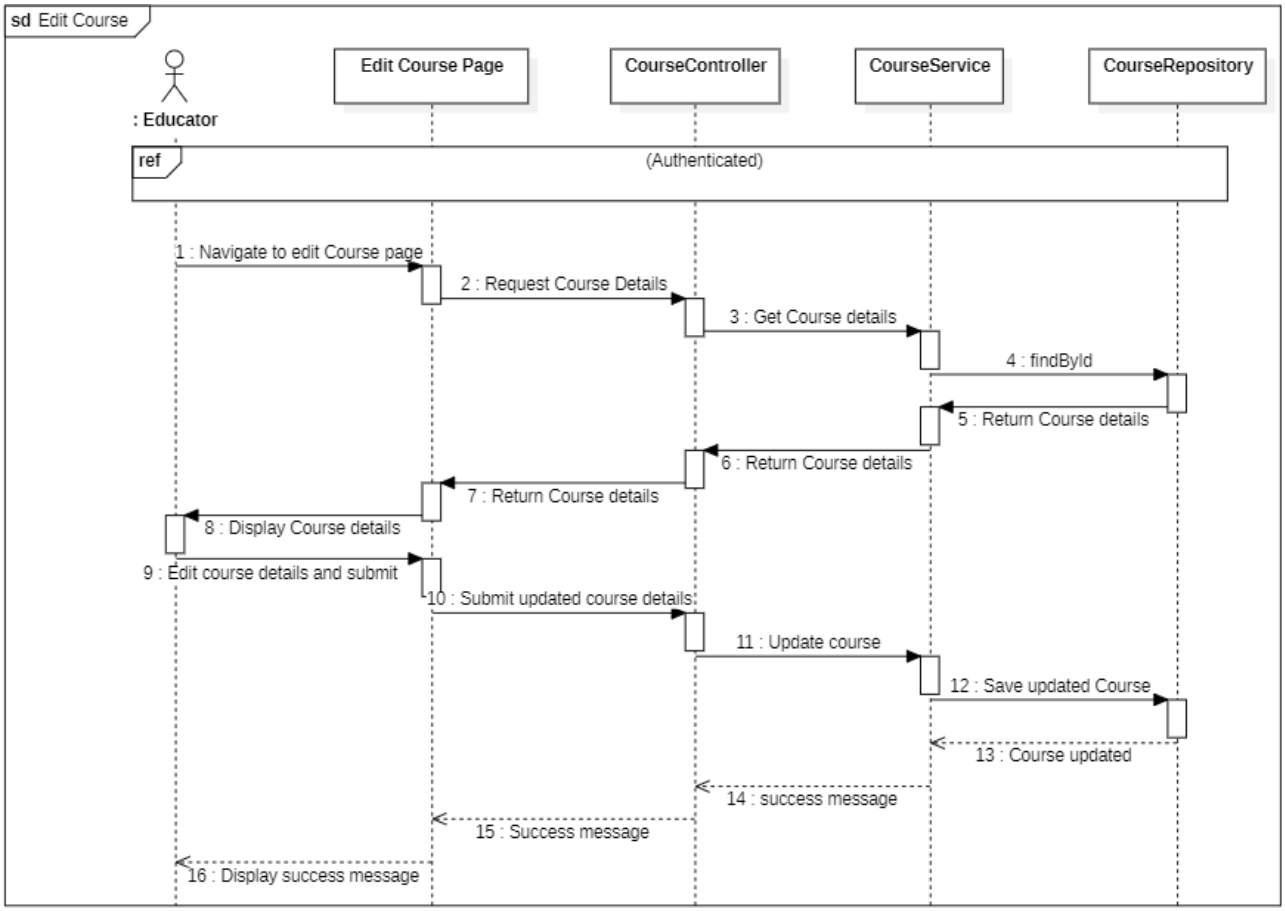

Figure 4.6: Sequence Diagram « Edit Courses »

Textual Description of the Edit Course Sequence Diagram

1. **Educator:** Navigates to the Edit Course page.
2. **Edit Course Page:** Requests course details from the CourseController.
3. **CourseController:** Retrieves course details from CourseService.
4. **CourseService:** Fetches course by ID from CourseRepository.
5. **CourseRepository:** Returns course details to CourseService.
6. **CourseService:** Sends course details to Edit Course Page.
7. **Edit Course Page:** Displays course details.
8. **Educator:** Edits and submits course details.
9. **Edit Course Page:** Submits updated details to CourseController.
10. **CourseController:** Updates and saves course via CourseService.
11. **CourseService:** Confirms update and returns success message.
12. **Edit Course Page:** Displays success message.

4.2.7 Sequence Diagram « Delete Course »

For the next Diagram it represents the Course deletion process :

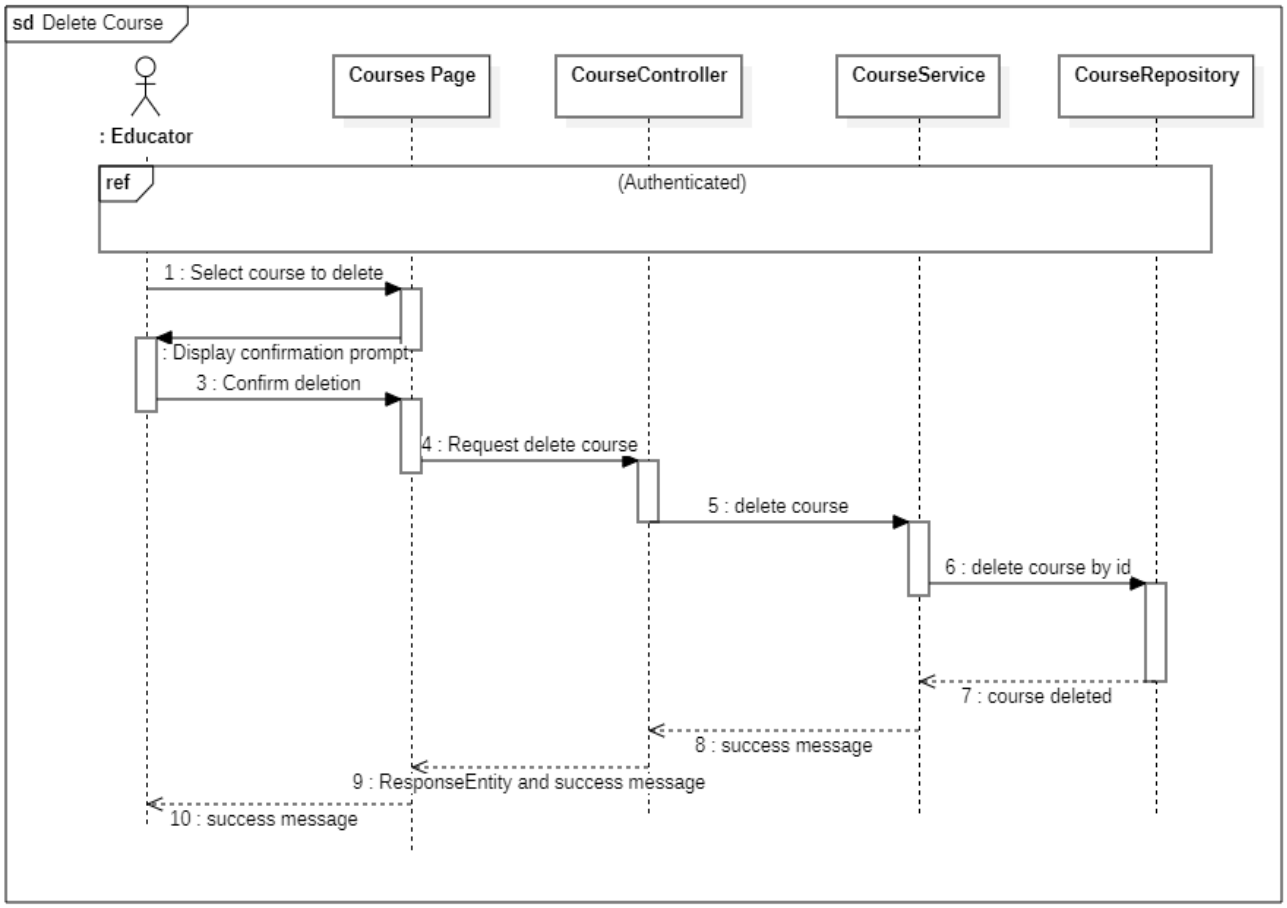

Figure 4.7: Sequence Diagram « Delete Course »

Textual Description of the Delete Course Sequence Diagram

1. **Educator:** Selects course to delete on Courses Page.
2. **Courses Page:** Displays confirmation prompt.
3. **Educator:** Confirms deletion.
4. **Courses Page:** Requests course deletion from CourseController.
5. **CourseController:** Calls `deleteCourse` in CourseService.
6. **CourseService:** Deletes course by ID in CourseRepository.
7. **CourseRepository:** Confirms course deletion.
8. **CourseService:** Sends success message to CourseController.
9. **CourseController:** Sends success message to Courses Page.
10. **Courses Page:** Displays success message.

4.2.8 Sequence Diagram « Create Quiz »

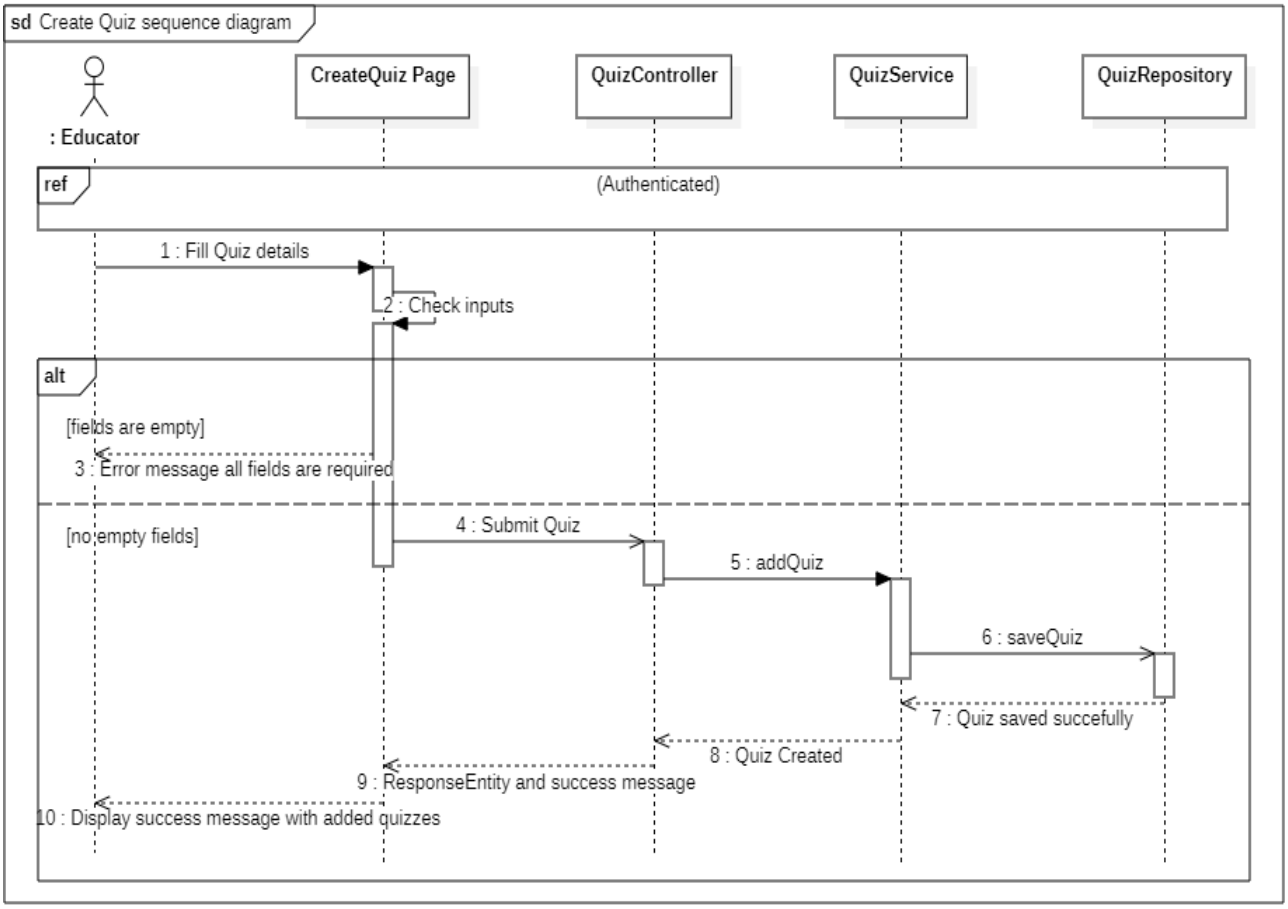

Figure 4.8: Sequence Diagram « Create Quiz »

Textual Description of the Create Quiz Sequence Diagram

1. **Educator:** Fills out quiz details on CreateQuiz Page.
2. **CreateQuiz Page:** Checks if all fields are filled.
3. **Error Handling:** If fields are empty, display an error message.
4. **Submit Quiz:** If no empty fields, CreateQuiz Page submits the quiz details.
5. **QuizController:** Calls `addQuiz` in QuizService.
6. **QuizService:** Calls `saveQuiz` in QuizRepository.
7. **QuizRepository:** Saves the quiz and confirms success.
8. **QuizService:** Sends success message to QuizController.
9. **QuizController:** Sends success message to CreateQuiz Page.
10. **CreateQuiz Page:** Displays success message.

4.2.9 Sequence Diagram « Edit Quiz »

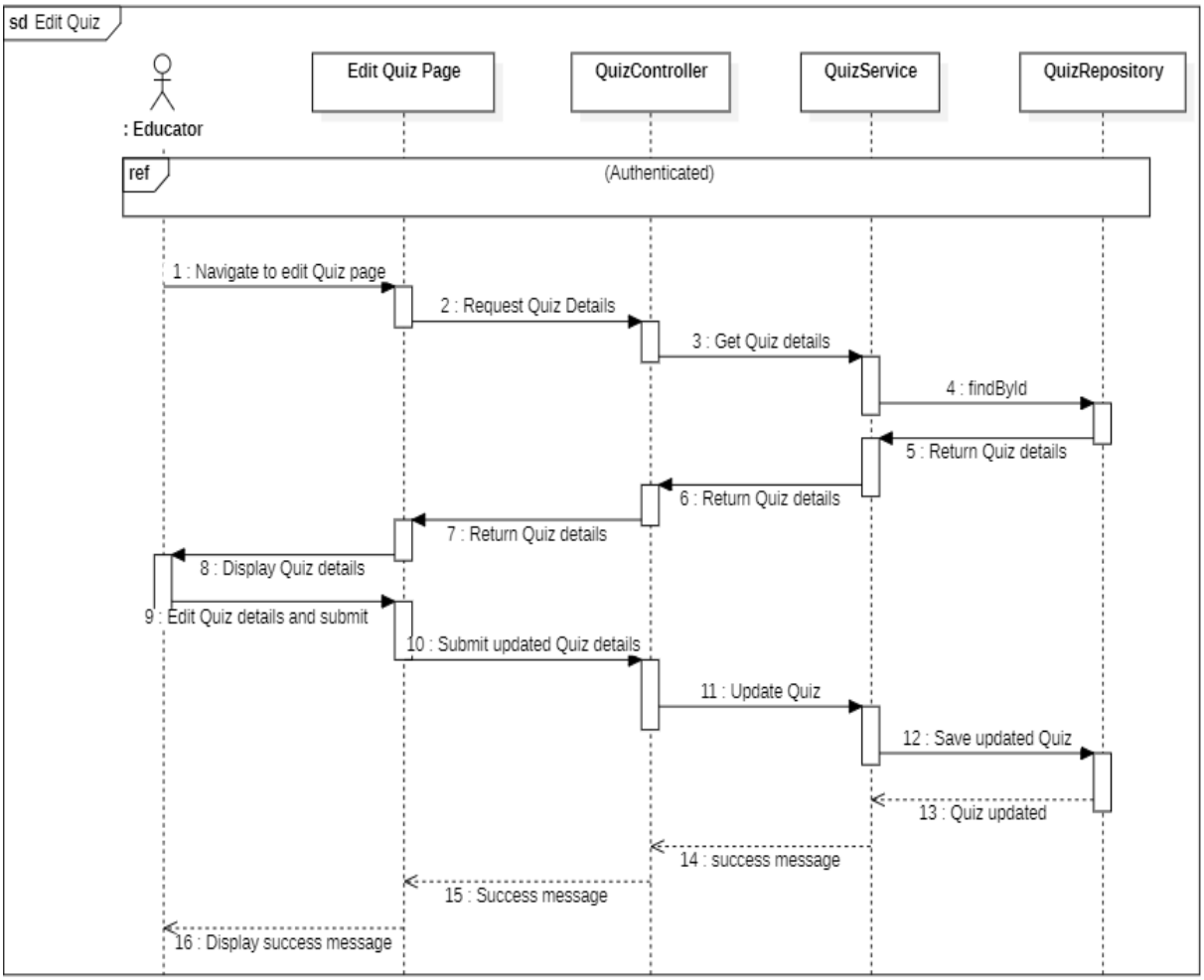

Figure 4.9: Sequence Diagram « Edit Quiz »

Textual Description of the Edit Quiz Sequence Diagram

1. **Educator:** Navigates to the Edit Quiz page.
2. **Edit Quiz Page:** Requests quiz details from QuizController.
3. **QuizController:** Retrieves quiz details from QuizService.
4. **QuizService:** Fetches quiz by ID from QuizRepository.
5. **QuizRepository:** Returns quiz details to QuizService.
6. **QuizService:** Sends quiz details to QuizController.
7. **QuizController:** Sends quiz details to Edit Quiz Page.
8. **Edit Quiz Page:** Displays quiz details to Educator.
9. **Educator:** Edits quiz details and submits the form.
10. **Edit Quiz Page:** Submits updated quiz details to QuizController.

11. **QuizController:** Updates quiz in QuizService.
12. **QuizService:** Saves updated quiz in QuizRepository.
13. **QuizRepository:** Confirms quiz update.
14. **QuizService:** Sends success message to QuizController.
15. **QuizController:** Sends success message to Edit Quiz Page.
16. **Edit Quiz Page:** Displays success message.

4.2.10 Sequence Diagram « Delete Quiz »

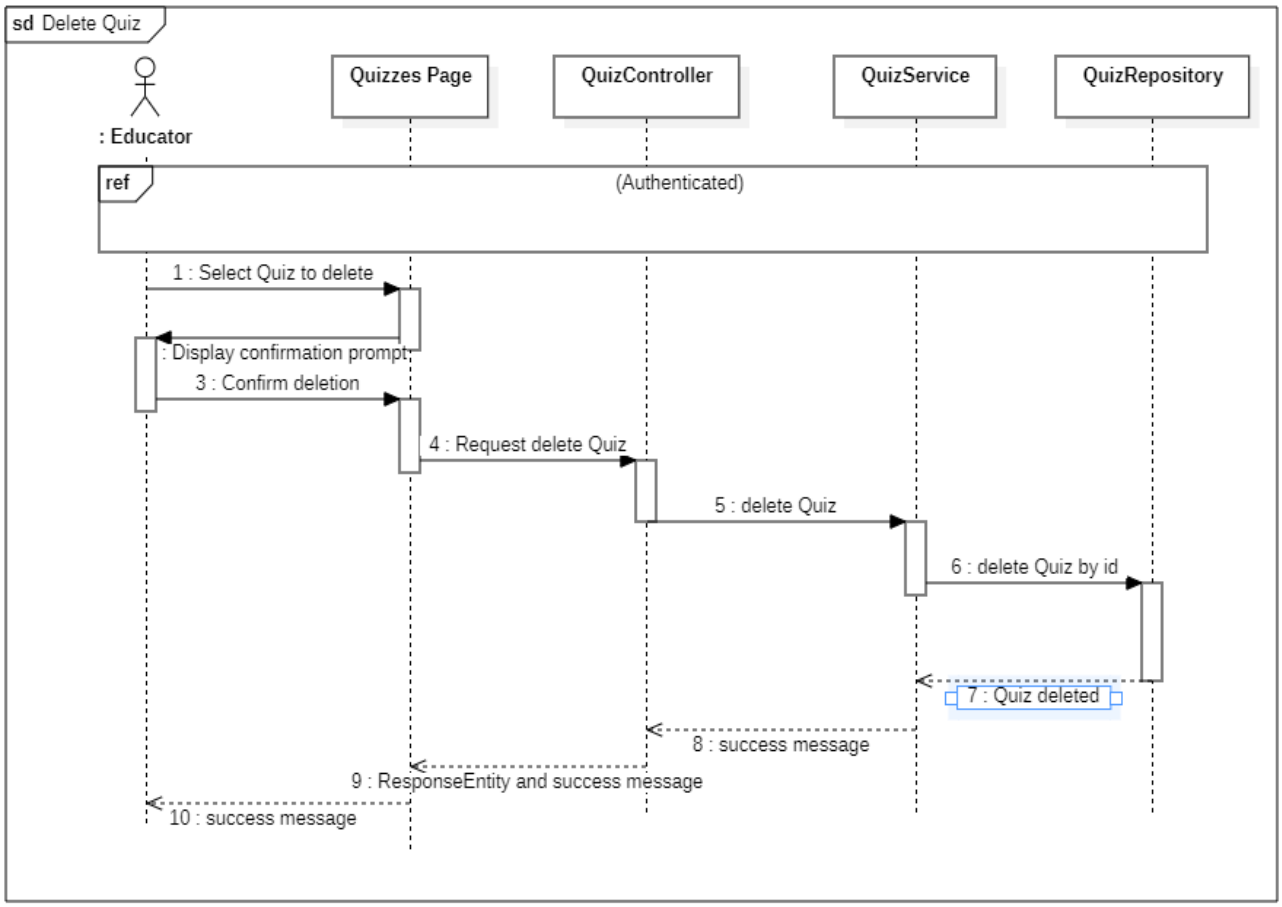

Figure 4.10: Sequence Diagram « Delete Quiz »

Textual Description of the Delete Quiz Sequence Diagram

1. **Educator:** Selects quiz to delete on Quizzes Page.
2. **Quizzes Page:** Displays confirmation prompt.
3. **Educator:** Confirms deletion.
4. **Quizzes Page:** Requests quiz deletion from QuizController.
5. **QuizController:** Calls `deleteQuiz` in QuizService.
6. **QuizService:** Deletes quiz by ID in QuizRepository.
7. **QuizRepository:** Confirms quiz deletion.
8. **QuizService:** Sends success message to QuizController.
9. **QuizController:** Sends success message to Quizzes Page.
10. **Quizzes Page:** Displays success message.

4.2.11 Sequence Diagram « upload file »

The next sequence diagram represents the upload files for a course :

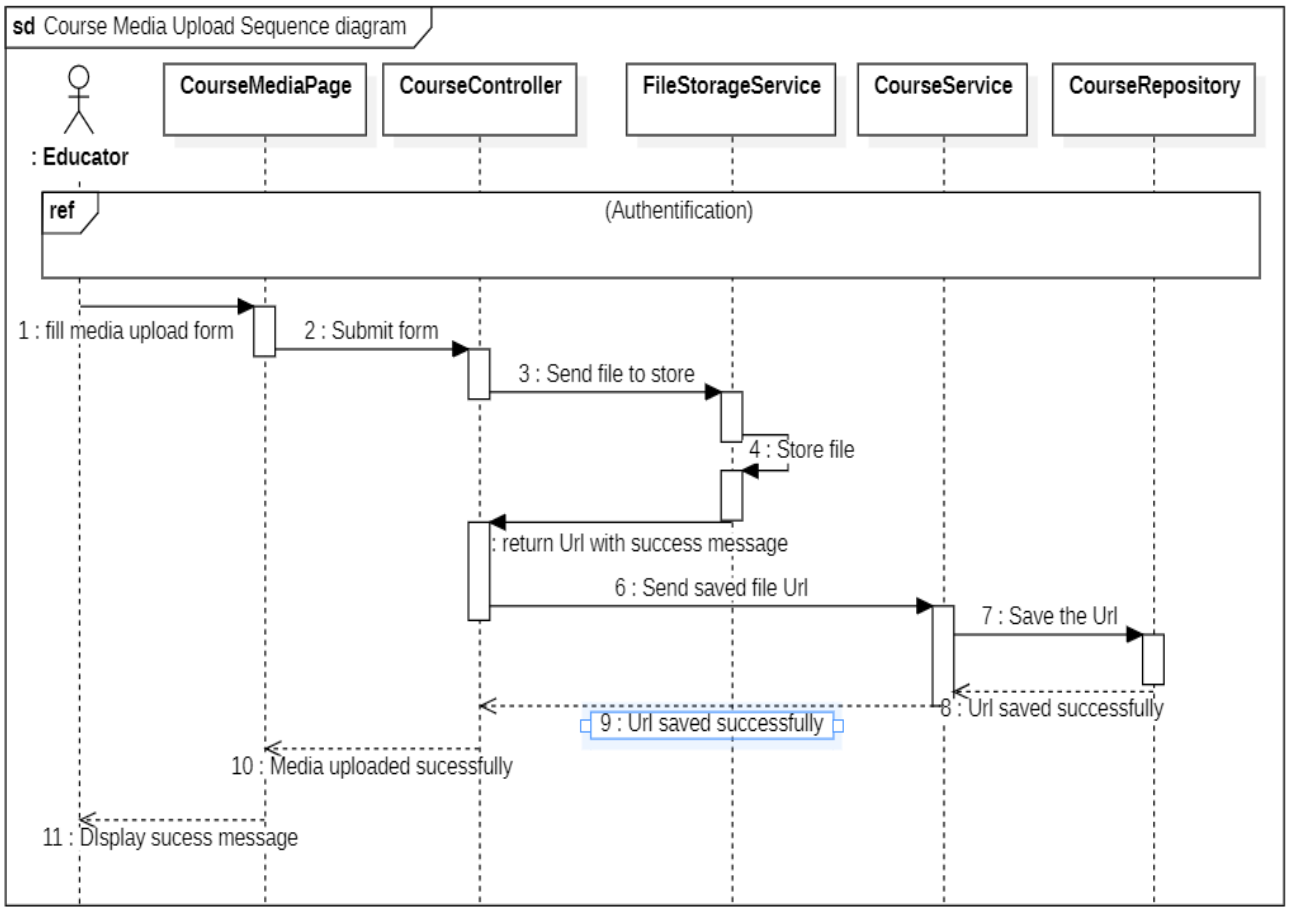

Figure 4.11: Sequence Diagram « upload file »

Textual Description of the Upload File Sequence Diagram

1. **Educator:** Fills media upload form on CourseMediaPage.
2. **CourseMediaPage:** Submits the form.
3. **CourseController:** Sends file to FileStorageService.
4. **FileStorageService:** Stores the file.
5. **FileStorageService:** Returns URL with success message.
6. **CourseController:** Sends saved file URL to CourseService.
7. **CourseService:** Saves the URL in CourseRepository.
8. **CourseRepository:** Confirms URL saved successfully.
9. **CourseService:** Sends success message to CourseController.
10. **CourseController:** Sends success message to CourseMediaPage.
11. **CourseMediaPage:** Displays success message.

4.2.12 Sequence Diagram « Purchase Course »

The next Sequence diagram represents the purchasing functionality :

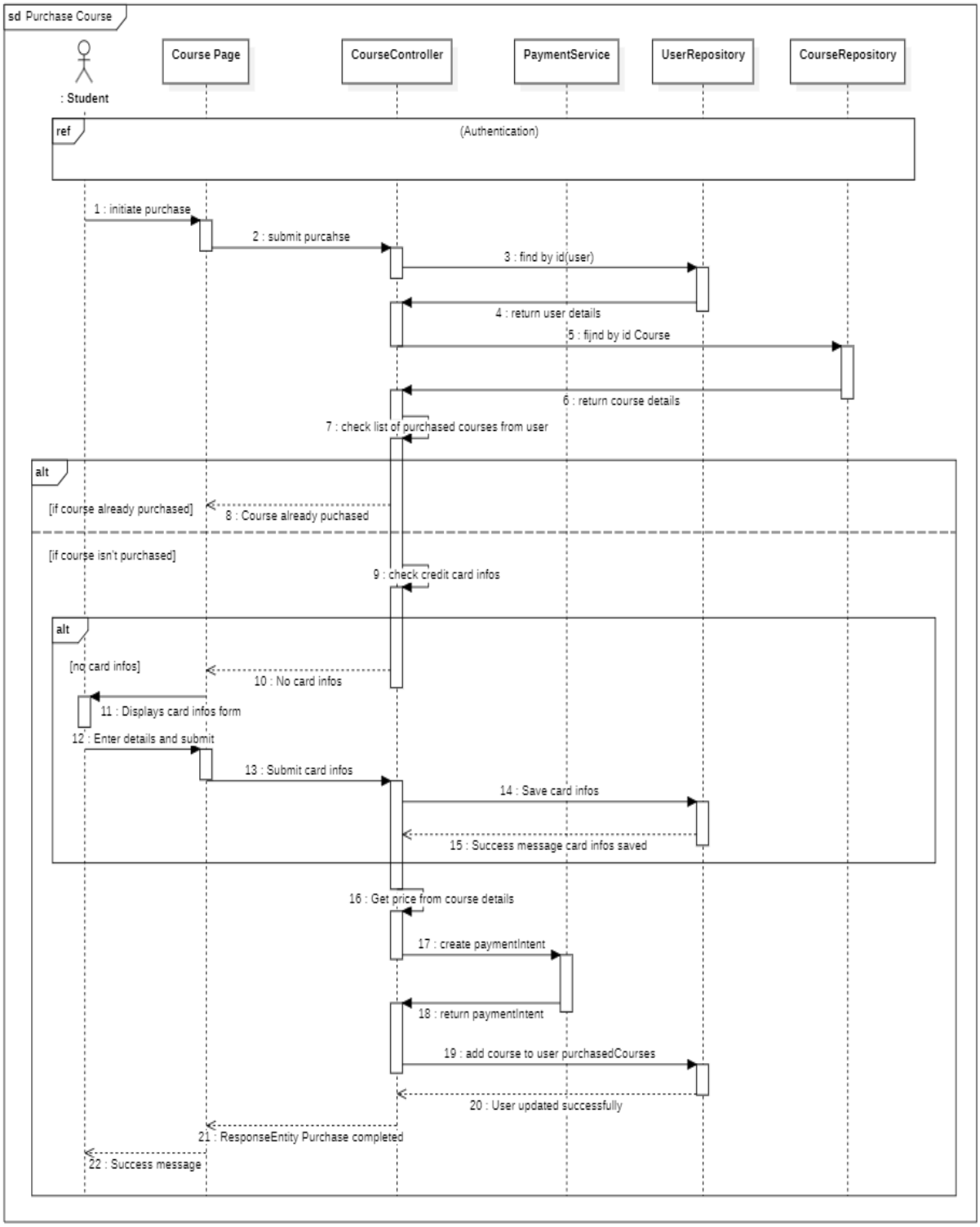

Figure 4.12: Sequence Diagram « Purchase Course »

Textual Description of the purchase Course Sequence Diagram

1. **Student:** Initiates purchase on Course Page.
2. **Course Page:** Submits purchase request to CourseController.
3. **CourseController:** Finds user by ID from UserRepository.
4. **UserRepository:** Returns user details.
5. **CourseController:** Finds course by ID from CourseRepository.
6. **CourseRepository:** Returns course details.
7. **CourseController:** Checks if the course is already purchased by the user.
    - **If already purchased:** Sends message that course is already purchased.
8. **If not purchased:** Checks credit card information.
    - **If no card info:** Displays card info form.
    - **Student:** Enters card details and submits.
    - **CourseController:** Submits card info to PaymentService.
    - **PaymentService:** Saves card information and continue the purchasing process normally .
9. **PaymentService:** Gets price from course details.
10. **PaymentService:** Creates payment intent.
11. **PaymentService:** Returns payment intent.
12. **CourseController:** Adds course to user's purchased courses.
13. **UserRepository:** Updates user information.
14. **CourseController:** Sends purchase completed response.
15. **Course Page:** Displays success message.

4.2.13 Sequence Diagram « Quiz Attempt »

This section represents the quiz Attempt Sequence diagram :

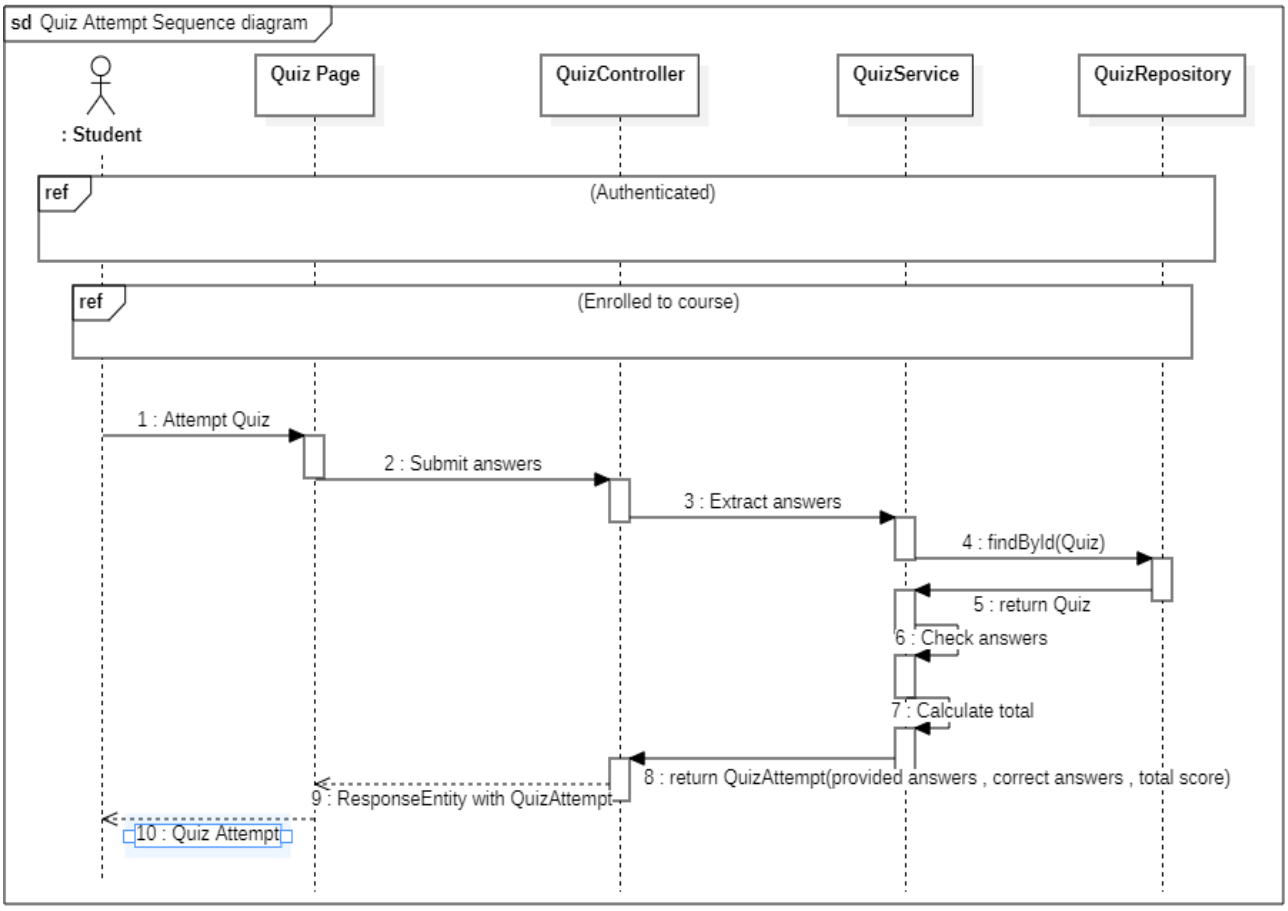

Figure 4.13: Sequence Diagram « Quiz Attempt »

Textual Description of the Quiz Attempt Sequence Diagram

1. **Student:** Attempts quiz on Quiz Page.
2. **Quiz Page:** Submits answers to QuizController.
3. **QuizController:** Extracts answers and sends request to QuizService.
4. **QuizService:** Fetches quiz by ID from QuizRepository.
5. **QuizRepository:** Returns quiz details.
6. **QuizService:** Checks answers and calculates total score.
7. **QuizService:** Returns quiz attempt results (provided answers, correct answers, total score) to QuizController.
8. **QuizController:** Sends quiz attempt results to Quiz Page.
9. **Quiz Page:** Displays quiz attempt results.

4.2.14 Sequence Diagram « Comment »

This section represents the Commenting functionality Sequence diagram :

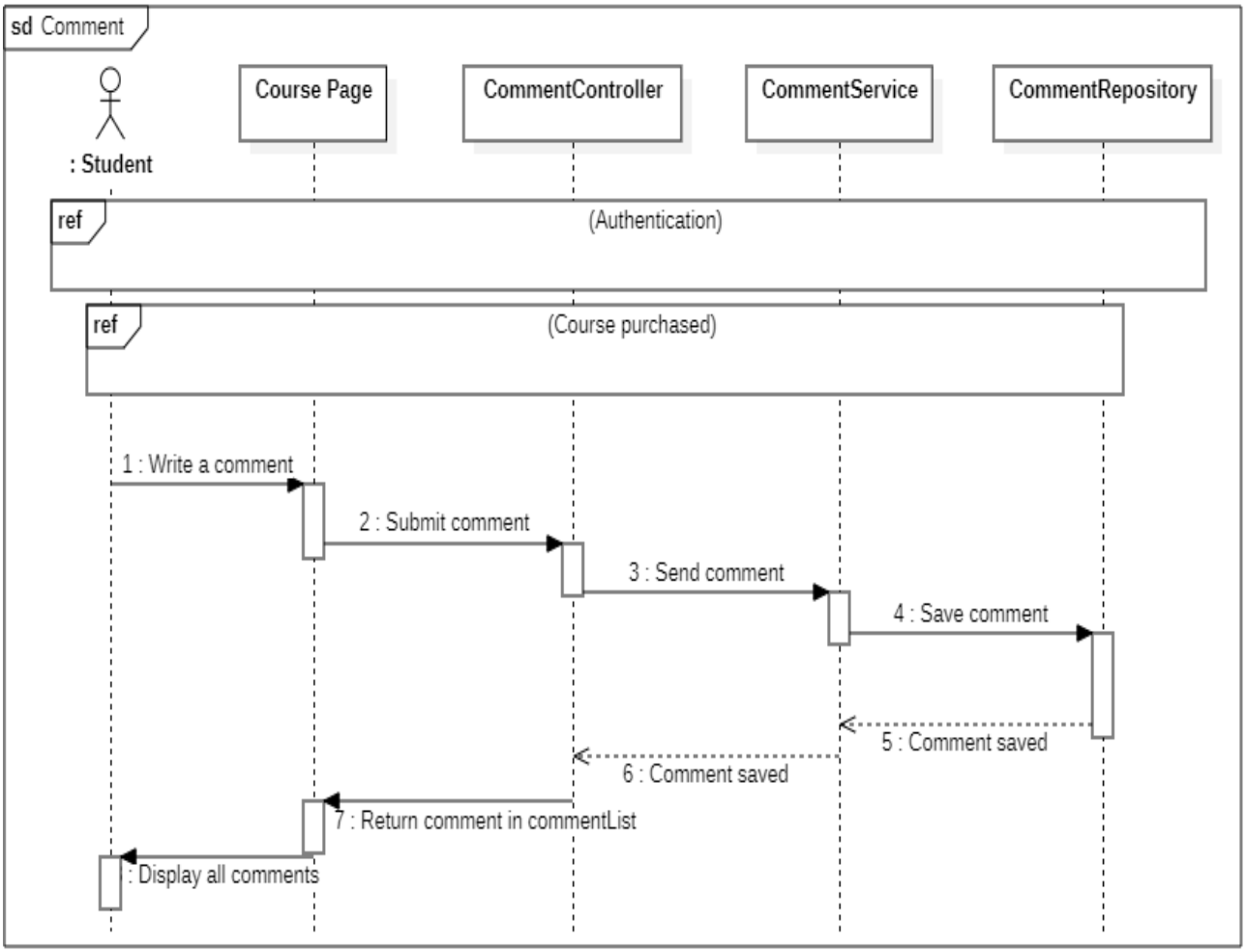

Figure 4.14: Sequence Diagram « Comment »

Textual Description of the Comment Sequence Diagram

1. **Student:** Writes a comment on Course Page.
2. **Course Page:** Submits the comment to CommentController.
3. **CommentController:** Sends the comment to CommentService.
4. **CommentService:** Saves the comment in CommentRepository.
5. **CommentRepository:** Confirms comment saved
6. **Course Page:** Displays all the old and new comments .

4.2.15 Sequence Diagram « Add Category »

The next diagram represents the add category sequence diagram :

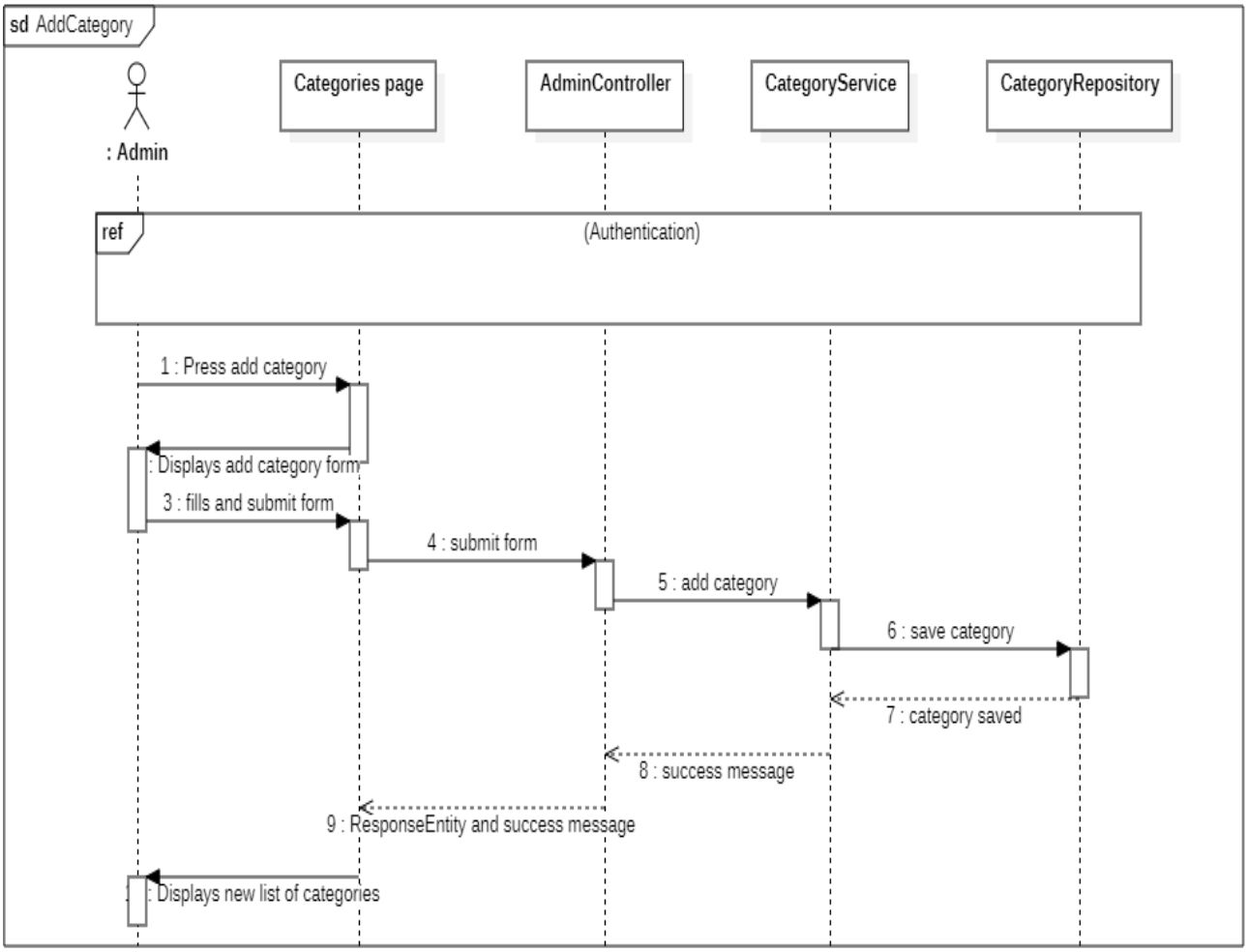

Figure 4.15: Sequence Diagram « Add Category »

Textual Description of the Add category Sequence Diagram

1. **Admin** presses the "Add category" button on the **Categories page**.
2. The **Categories page** displays the "Add category" form.
3. **Admin** fills in the form and submits it.
4. The form submission is sent to the **AdminController**.
5. The **AdminController** calls the **CategoryService** to add the category.
6. The **CategoryService** saves the category using the **CategoryRepository**.
7. The **CategoryRepository** confirms that the category is saved.
8. A success message is sent back to the **AdminController**.
9. The **AdminController** returns a ResponseEntity with a success message to the **Categories page**.
10. The **Categories page** displays the updated list of categories.

4.2.16 Sequence Diagram « Delete Category »

This section represents the Delete category functionality Sequence diagram :

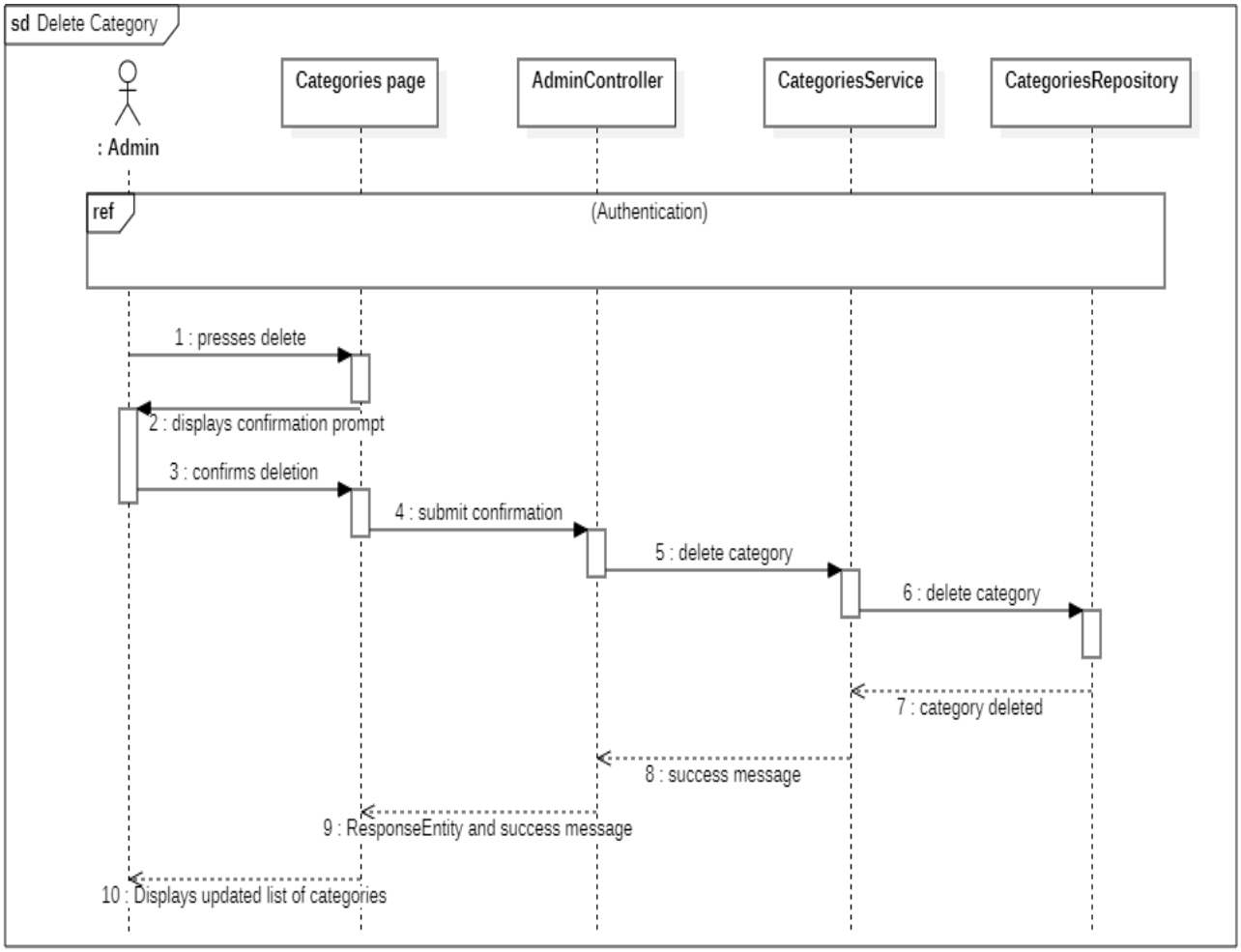

Figure 4.16: Sequence Diagram « Delete Category »

Textual Description of the Delete Category Sequence Diagram

1. **Admin** presses the "Delete category" button on the **Categories page**.
2. The **Categories page** prompts for confirmation to delete the category.
3. **Admin** confirms the deletion.
4. The confirmation is sent to the **AdminController**.
5. The **AdminController** calls the **CategoryService** to delete the category.
6. The **CategoryService** removes the category using the **CategoryRepository**.
7. The **CategoryRepository** confirms that the category is removed.
8. A success message is sent back to the **AdminController**.
9. The **AdminController** returns a ResponseEntity with a success message to the **Categories page**.
10. The **Categories page** displays the updated list of categories.

4.3 Classes Diagram

4.3.1 Introduction

A class diagram is a type of diagram in the Unified Modeling Language (UML) that shows the structure of a system by displaying its classes, their attributes, methods, and the relationships among them. It helps in visualizing and designing the static aspects of a software system, providing a blueprint for system construction.

4.3.2 Global Class Diagram

Chapter 5

Implementation