MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Online platform for learning

FastLearn

DIPLOMA THESIS

Student: **Paul-Ioan Cloţan**

Project Supervisor: **Teodora Sanislav**

2023

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

VISED,

DEAN,
Prof. Eng. Liviu MICLEA, PhD

HEAD OF DEPARTMENT,
Prof. Eng. Honoriu VĂLEAN, PhD

Student: **Paul-Ioan Cloţan**

Online platform for learning

FastLean

1. **The Problem:** *The design, development, and implementation of an online learning web application. The platform aims to provide reliable information and encourages collaboration between fellow users.*

2. **Project Content:** *Presentation page, Introduction, Theoretical approach, Analysis and design, Implementation, Testing and validation, Conclusions and References*

3. **Place of documentation:** *Technical University of Cluj-Napoca, Automation Department*

4. **Consultants:** -

5. **Thesis emission date:** *October 21, 2022*

6. **Thesis delivery date:** *July 5, 2023*

Student signature 

Project's supervisor signature _____

MINISTRY OF EDUCATION
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Declaraţie pe proprie răspundere privind autenticitatea proiectului de diplomă

Subsemnatul **Paul-Ioan Cloţan,** legitimat cu CI seria SB nr**.** 835057, CNP 5000204324791**, autorul lucrării: Online platform for learning, FastLearn** elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la **Facultatea de Automatică şi Calculatoare**, specializarea **Automatică si Informatică Aplicată(in limba engleză),** din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea iulie a anului universitar 2022-2023, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, anularea examenului de licenţă.

Data:
05.07.2023

Paul-Ioan Cloţan
_____
(semnătura)

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

SYNTHESIS

of the Diploma Thesis:

Online platform for learning

FastLearn

Student: **Paul-Ioan CLOȚAN**

Project Supervisor: **Lecturer Eng. Teodora SANISLAV, PhD**

1. **Problem definition**: The creation of a web application that addresses the complex educational needs of its end-users.

2. **Application domain**: The application was designed using a combination of modern techniques and technologies to ensure the longevity of the application. The application was developed using the Model-View-Controller (MVC) in Visual Studio 2022 with the C# programming language. The technologies used include but are not limited to .NET Framework, Entity Framework, and SQLite for the back-end and Razor and CSS for the front-end.

3. **Obtained Results:** A fully operational online learning platform capable of hosting reliable information. The application supports five user roles: *Unregistered User*, *Student*, *HelpingStudent*, *Professor*, and *Admin*, each having dedicated and easy-to-use user interfaces. Among the implemented features, it includes progress tracking, versatile content editing, and *HelpingStudent* to *Student* tutoring opportunities.

4. **Testing and Verification:** The quality of the application was ensured through manual testing of all the features, both during and at the end of the development phase.

5. **Personal Contributions:** The design and implementation of an online platform for learning. The introduction of a new user role, the *HelpingStudent*, rewarded to help the *Students* understand particularly difficult subjects.

6. **Documentation Sources:** Online articles and documents.

Student signature 

Project's supervisor signature _____

Contents

1 Introduction

1.1 Overview

Starting with the earliest societies, learning from the experience of our peers enabled humanity to evolve and prosper over countless centuries. One of the biggest challenges we continue to face is selecting and sharing reliable, helpful information. This task becomes more challenging as the human population and the quantity of information available increase with our evolution.

Over time, special techniques and systems were developed to ensure the sustainable growth of our species. These efforts resulted in the educational system we know today. While its effectiveness is hard to contest, recent years, particularly the pandemic ones, have shown us the advantages of incorporating an online learning environment.

Compared to traditional, paper-based learning, online information is more manageable and easier to disseminate. For instance, rectifying an error in a printed and distributed coursebook requires recalling all the faulty samples. This process is both time and energy-consuming and harmful to the environment as it consumes high volumes of paper. On the other hand, an online course is far easier to manage, as the error can be corrected in the span of a few days from the moment it was found.

Despite the numerous benefits of online learning, verifying the authenticity of available information is harder than ever before. The theme of this thesis, *FastLearn*, aims to solve this problem by providing an online environment where the quality of the information can be trusted.

The presentation of this diploma thesis is organized into four chapters that describe how *FastLearn* was designed and implemented. The first chapter is meant to present the context, the motivation, and the objectives of this application. The second chapter performs an analysis of similar platforms. The third chapter presents how *FastLearn* was designed and the steps required to implement the application. Additionally, relevant test cases and how the identified errors were resolved are showcased. The fourth chapter contains a review of the development process and a list of potential improvements for future application iterations.

1.2 Objectives

The main objective of *FastLearn* is to enhance the learning experience of its end-users by providing reliable information. Given the complex nature of the task, user roles with clear responsibilities must be designed. The application aims to provide user-role-specific features that will help achieve this goal.

The proposed objectives in the elaboration of the *FastLearn* web application are elaborated below:

- Provide an overview of the available information to unregistered users to help them decide if creating an account is appropriate.
- Offer a high degree of customization regarding the format in which the information is presented.
- Create well-defined user types with specific responsibilities.
- Design a robust and easy-to-use user interface.
- Implement progress tracking and knowledge assessment tools.
- Motivate the end-users of the application to help each other in their learning journey.
- Create a dedicated user role tasked with helping other users in one-to-one sessions.
- Guarantee straightforward integration of future updates by using modern technologies with long-term support.

1.3 Specifications

The theme of this thesis, *FastLearn*, aims to assist the user in expanding his knowledge on a desired topic. To make this possible, the application provides the learning enthusiast with reliable features presented as a welcoming user interface.

The functionalities of the application are accessible based on the role of the user. To this end, five user roles were defined. The first three, *Unregistered User*, *Student*, and *HelpingStudent*, represent the application's target audience. The access to information and features is gradually increased as the user advances through the roles mentioned above. For instance, the *Student* has full access to the educational content of the platform, being able to enhance and test his knowledge as well as request help with particularly challenging topics. The fourth type of user, the *Professor*, represents the domain expert that provides reliable information. This user can design custom courses using a wide range of content formatting tools to ensure the quality of the material. The last user role, the *Admin*, is tasked with monitoring the well-being of the platform. They have the authority to adjust user roles and impose penalties on users who violate platform rules.

The application was developed following modularization and code reusability guidelines. The logic application's logic is implemented using straightforward solutions, and strict naming conventions are used to improve the readability of the code.

2 State of the art

Over the last few decades, the information available to us increased exponentially, thus making it impossible to check its authenticity. For this reason, platforms like *FastLearn* have become necessary in our day-to-day life. This project aims to ease the learning experience by developing an interactive web application with accurate information supplied by experts in their respective fields. Given the scarcity of verified information on the internet, there are multiple e-learning platforms that aim to help anyone who wants to expand their knowledge. A few examples are: *SQLZOO* [1], *GeeksForGeeks* [2], *Udemy* [3], *Coursera* [4] and many others.

A diligent analysis was made regarding three of the web platforms mentioned in the above paragraph.

SQLZOO [1] is a free-to-use learning environment destined for both beginner and experienced programmers looking to improve their SQL (Structured Querry Language) skills. Its modest but intuitive design offers an easy-to-follow learning path for beginners. The design of the platform can be observed in Figure 2.1 below.

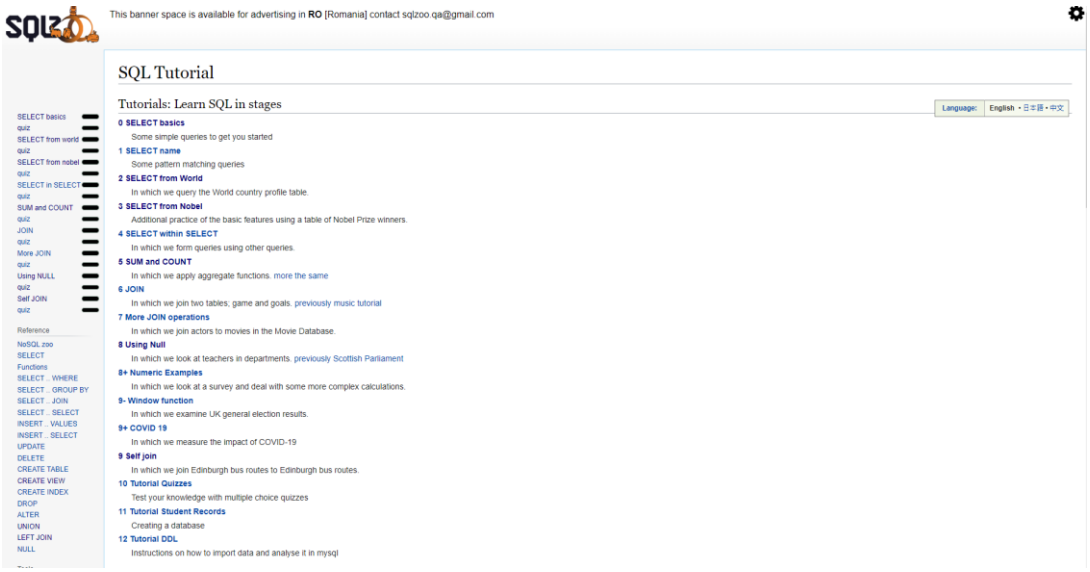

Figure 2.1 - SQLZOO landing page (source [1])

The learning material is structured in lessons, each using and building upon the previously studied concepts. The classes are composed of a series of practical assignments gradually increasing in difficulty. The exercises require the user to formulate a valid query to interrogate the provided database and return the desired output. Upon writing a potential solution, the user can run it directly on the platform and receive instant feedback regarding his solution. In Figure 2.2, the functionalities presented above are illustrated.

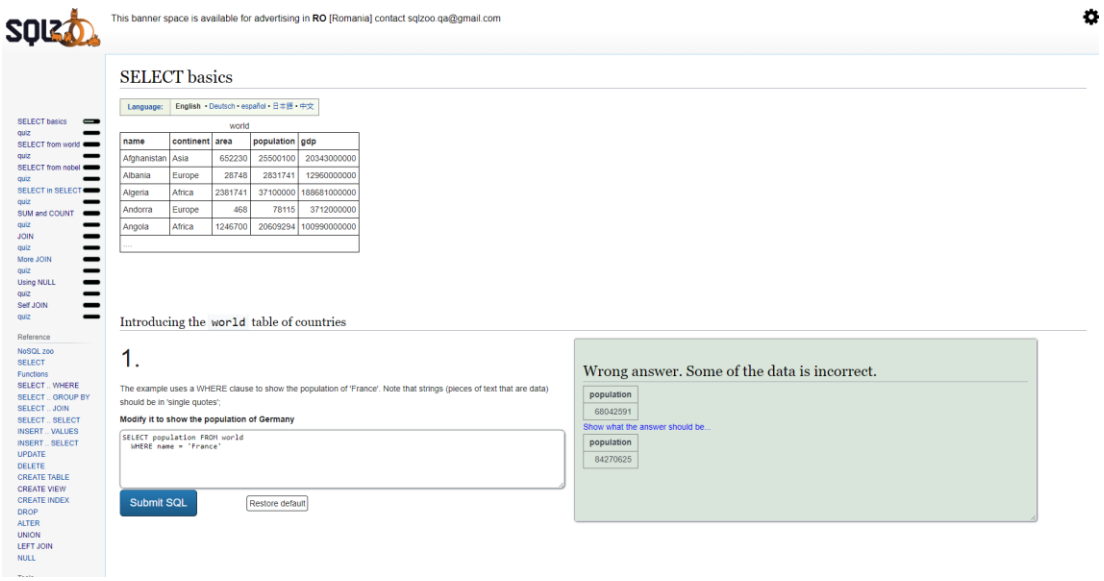

Figure 2.2 - SQLZOO lesson overview (source [1])

At the beginning of the lesson, a quick view of the database, in our instance, the „world“ table, is given. Quickly after this, the first assignment is presented. From now on, it is up to the user to write a potential solution in the designated space and test it. In Figure 2.2, the user receives feedback in case of an incorrect solution. Initially, only the first table is provided, and if the user so desires, the correct result of the query can be viewed by clicking on the button „Show what the answer should be...“.

The previously presented features are available without creating an account, which is ideal for quickly accessing and dusting up one‘s knowledge on the desired topic. While using *SQLZOO* like this is viable for experienced programmers, for a beginner, it is recommended to have an account. One of the advantages of a logged-in user is the possibility to track their progression and examine solutions to past exercises. This feature allows a novice to revise the already mastered knowledge effectively.

SQLZOO is, without a shadow of a doubt, a great learning resource, as it offers a comprehensive learning path to learn SQL that will enhance one's individual learning experience. When comparing it with the topic of this thesis, *FastLearn*, the latter stands out due to its emphasis on collaboration. It encourages its users to participate in one-on-one dedicated sessions to help the user in need understand the topic better. This method builds upon the individual study approach and offers insights the user wouldn't have encountered otherwise.

The second application, *GeeksForGeeks* [2], is an online platform designed to help any programming passionate enhance their skill. With an intuitive user interface, it encapsulates a wide range of articles, courses, tutorials, and practice exercises in various programming languages. A significant portion of the content is available free of charge. The content is gradually made available to users, ranging from unregistered to paying users. Without an account, a user can access tutorials on many topics, as seen in Figure 2.3.

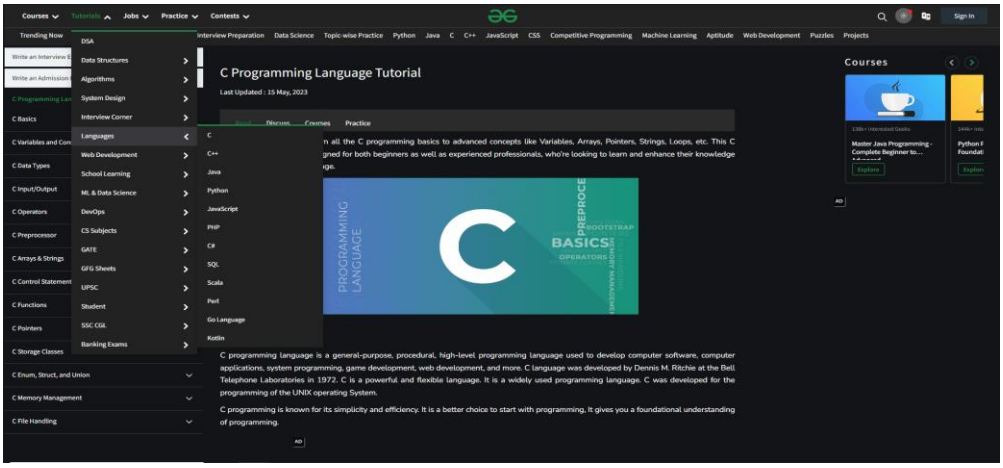

Figure 2.3 - GeeksForGeeks tutorial categories (source [2])

Using the website as an unlogged user may fulfill the needs of some, but the advantages of registering on *GeeksForGeeks* are undoubtedly plenty. One of the advantages of creating an account is having access to the large collection of practice problems and solving those problems directly on the site, in their IDE(Integrated Development Environment), which can be observed in Figure 2.4. Another benefit of being registered is interacting and testing your knowledge by competing with other users. Lastly, one can sign up for a course by paying an enrollment fee. Enrolling in such a course will grant access to detailed video content and other course-specific features.

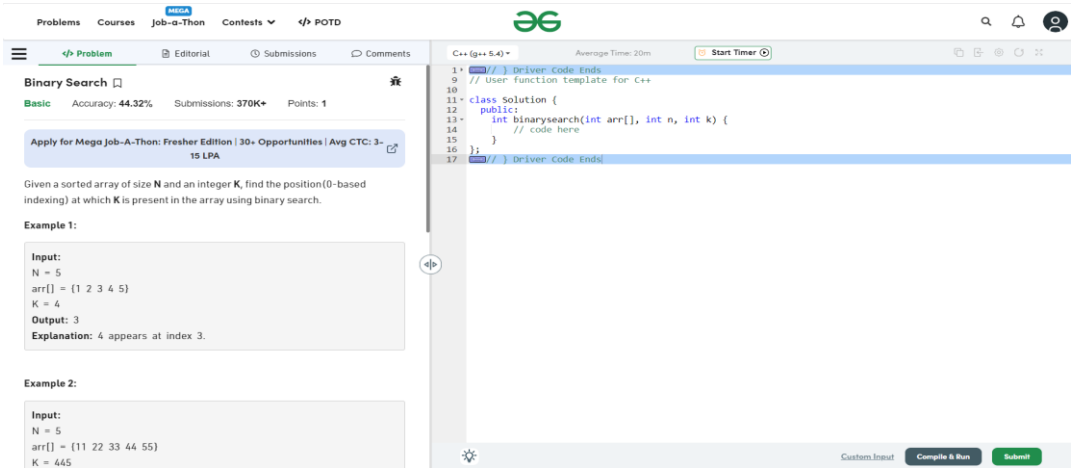

Figure 2.4 - GeeksForGeeks exercise page (source [2])

GeeksForGeeks is a reliable platform well-suited for learning, but there is no free assistance for users that need a little bit more help to understand a particular concept. The topic of this thesis, *FastLearn*, solves this issue by introducing a new user role. This new user is incentivized to help students in need in one-to-one chat sessions by earning platform currency.

While *GeeksForGeeks* is a user-friendly and reliable resource for anyone who wants to enhance their programming skills, the last learning website analyzed, *Udemy* [3], is more suitable for someone who looks for more general topics. The broader

selection of course subjects hosted on *Udemy* makes it a better choice for the general user. Some of the categories can be observed in Figure 2.5.

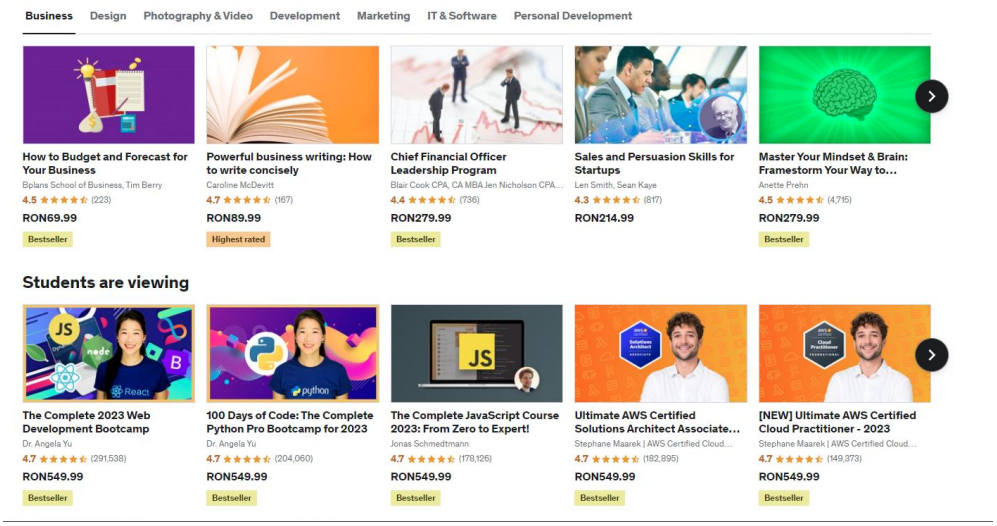

Figure 2.5 - Udemy landing page (source [3])

Udemy is an online learning platform aiming to foster a global learning environment. The website offers its services to both personal users and businesses. The general user can create a personal account, while businesses can opt for special company accounts. Creating a personal account gives you access to free content, a scheduler for learning activities, and filters that will help you find the best resource. Most courses on the website require an enrollment fee. Before paying, the user can browse the curriculum to understand better the topics covered in the video materials that will be made available once joining the course. Based on the specific offer, some advantages may be life-long access, special practical assignments, and gaining a certificate upon completion. The landing page for a logged in user can be observed below in Figure 2.6.

While *Udemy* is well-suited for individual use, it has plans designed to meet the needs of small to large companies. *Udemy Business* is a subscription-based service for companies that grant access to top courses selected from Udemy.com, available on any device. Based on the selected plan, the company can create custom learning paths and get advanced analytics and insights through a special account. The employees of the company gain access to a carefully selected collection of courses. The landing page for Udemy Business can be seen below in Figure 2.7.

Udemy offers insightful video materials that will help refine a particular skill through self-study. However, the topic of this thesis, *FastLearn*, advocates for a collaborative approach. It encourages its users to participate in one-on-one dedicated sessions to help the user in need understand the topic better.

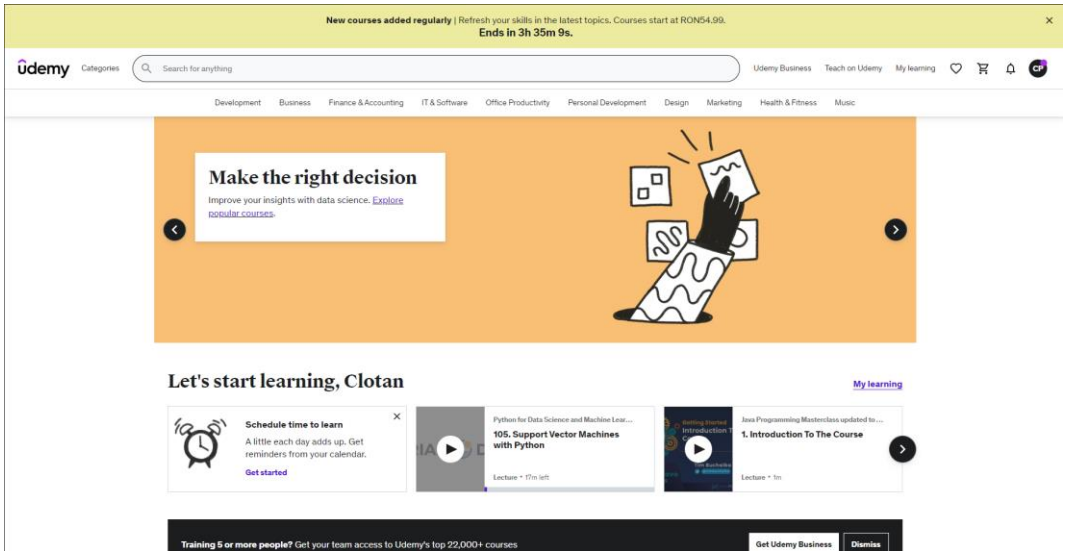

Figure 2.6 - Udemy logged-in user landing page (source [3])

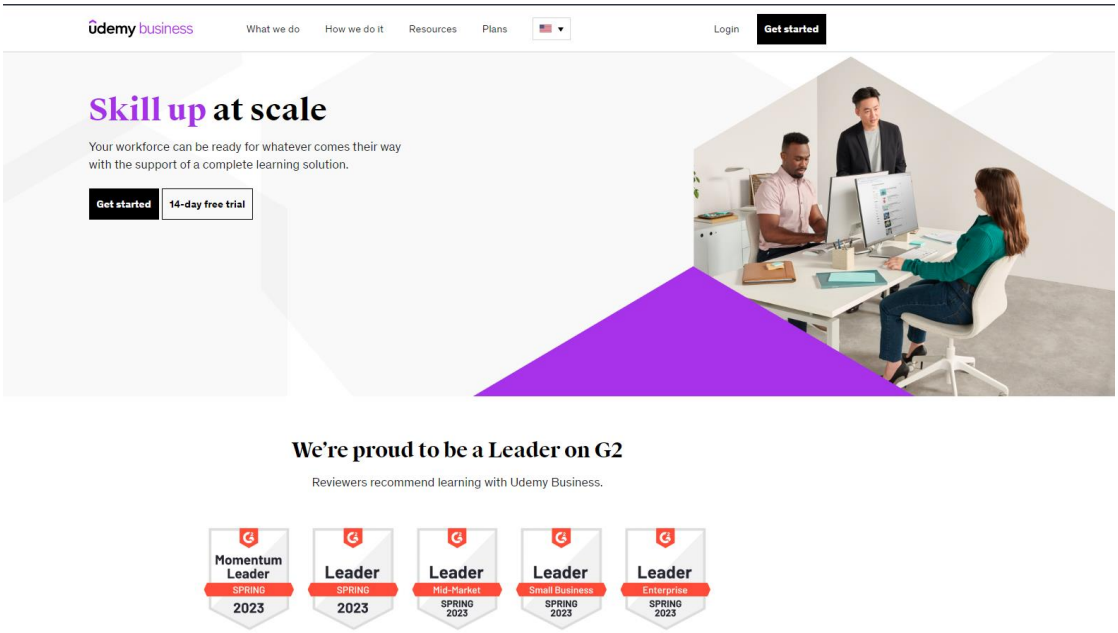

Figure 2.7 - Udemy Business (source [3])

In conclusion, all three applications that were discussed provide significant insights. The first platform, *SQLZOO* offers an excellent example of what an intuitive learning path for programmers at all levels should look like. The second platform, *GeeksForGeeks* perfectly illustrates how one can integrate the lessons taught by *SQLZOO* and delivers high-quality content in a modern and easy-to-use user interface. The third and last analyzed platform, *Udemy* is a well-known website portraying a good business model and a testament to the importance of resources similar to the theme of this thesis, *FastLearn*.

3 Analysis, design and implementation

The goal of *FastLearn* is to create a web application dedicated to providing reliable information to its end-users, the students. The motivation for creating a tool like *FastLearn* stems from the exponentially increasing volume of available information caused by the rapidly expanding digitalization of our world. By eliminating false information and only presenting authentic one, and making it available at a click's distance, *FastLearn* can enhance the academic performance of any student worldwide.

In the following sections of this chapter, a comprehensive presentation of the structure, logic, and technologies used in the design and implementation phase of this diploma thesis, *FastLearn*, will be provided.

3.1 Analysis of the FastLearn software application

3.1.1 Analysis of the user roles of the FastLearn web application

One of the objectives of this diploma thesis, *FastLearn*, is to enhance the learning journey of its users. Crucial to attaining this goal are carefully designed user roles to ensure information reliability and the platform's smooth functioning. With this in mind, five user roles were defined. The first one is the *Unauthenticated User*, and the remaining four role types: *Student*, *HelpingStudent*, *Professor,* and *Admin*, require to be logged in.

The analysis begins with *Unauthenticated User*, as it represents the default role of a newcomer to the platform. As suggested by the classification name, the user is not logged in and registered in the platform. This role allows viewing the list of available courses and their respective syllabus, analyzing the course reviews, and ultimately registering and logging into the platform. The goal of this role is to enable the new users to determine if the content of the platform suits their educational requirements.

The next analyzed user type, *Student*, represents the default role a newly created account is assigned. From the start, the user has everything needed to start studying. The next step will be enrolling in the desired course. Once enrolled, the *Student* gets access to the content of the lesson, structured in chapters and subchapters. Each subchapter contains information on a particular topic and possibly special materials provided by the *Professor* to facilitate understanding the subject. The *Student* has access to the materials afferent to the currently studied subchapter. To advance to the following subchapter, the *Student* must pass a test designed by the *Professor*. The progress through a specific course is tracked, enabling the *Student* to quickly resume his learning. In the case in which *Student* needs help with a particular topic, he/she can open a one-to-one chat request for the specific subchapter and get help from a more experienced *Student*. At any moment, the user can request to advance to the next role of the platform and become a *HelpingStudent*, or leave a review for a course.

The purpose of the *HelpingStudent* is to support the end-user of the platform in understanding subjects the *Student* needs help with. A *HelpingStudent* has all the facilities of a *Student*. In addition, this user can examine the chat requests opened by the *Students*, join the chat and solve the issue. At the end of a chat session, the *HelpingStudent* will be recompensed with platform currency, called Points. The Points can be spent to speed up the learning process by lowering the passing requirements of specific subchapter tests.

The next analyzed role is the *Professor*. The main responsibility of this user is to provide comprehensive courses with reliable information that the end-users of *FastLearn* will be able to study. To facilitate this, a *Professor* can edit every aspect of his lessons, starting from the content of the subchapters to the image the course will be associated with and also read the reviews of the *Students*. Besides managing his educational offers, this user plays a crucial role in the election process of the *HelpingStudents*. Once a *Student* registers a request to become a *HelpingStudent*, a *Professor* must review the *Student*'s preparation and recommend him for promotion to the *Admin* if it is the case.

The *Admin* manages the platform and ensures everyone fulfills their role. Through the contact information provided on the *About* page of the application, the platform users can report concerning events that require the attention of an *Admin*. To solve the signaled issues, this user can attribute warnings and suspensions or demote the reported users. Another responsibility of this role is to ensure the reliability of the courses. To this extent, this user type can examine the entire content of a class and remove or reinstate it from the official course list of the website, depending on the case. Lastly, an *Admin* is the final reviewer of the *HelpingStudent* requests, being able to accept or reject them.

3.1.2 Analysis of design methods, programming languages and development environments used

3.1.2.1 Model-View-Controller

In the early design stages of the *FastLearn* web application, the need for a clear and intuitive separation in the program logic was identified. The *Model-View-Controller* (MVC) [5] design pattern fits this requirement as it was developed by the computer scientist Trygve Mikkjel Heyerdahl Reenskaug with the goal of breaking up a complex user „application into smaller manageable components“ [6].

The MVC is a widely used design pattern as it provides an excellent method to decouple the user interface, data, and application logic, thus improving the development experience of complex applications [5]. The separation of concerns is made via its three components, the View, the Model, and the Controller.

The Model component is responsible with the data layer of the application. It encapsulates the structure of the data and the rules by which the data can be altered. In the context of this application, the Model accurately describes how the persistent data can be altered and stored.

The View corresponds to the Front-End of the application. This component represents the UI (User Interface) through which the user can interact with the application's data. The View renders the information available in the Model, with the help of the Controller.

The Controller is responsible with handling the application logic. Sometimes also reffered as the „Brain“ of the application (Figure 3.1), it controlls all the data flows of the application. It serves as the connection point between the View and the Model. In the Controller, the application logic is applied on the data received from the View, and the Model information is updated accordingly. Afterwards, the updated information is pulled from the Model and sent to the View to be displayed to the user.

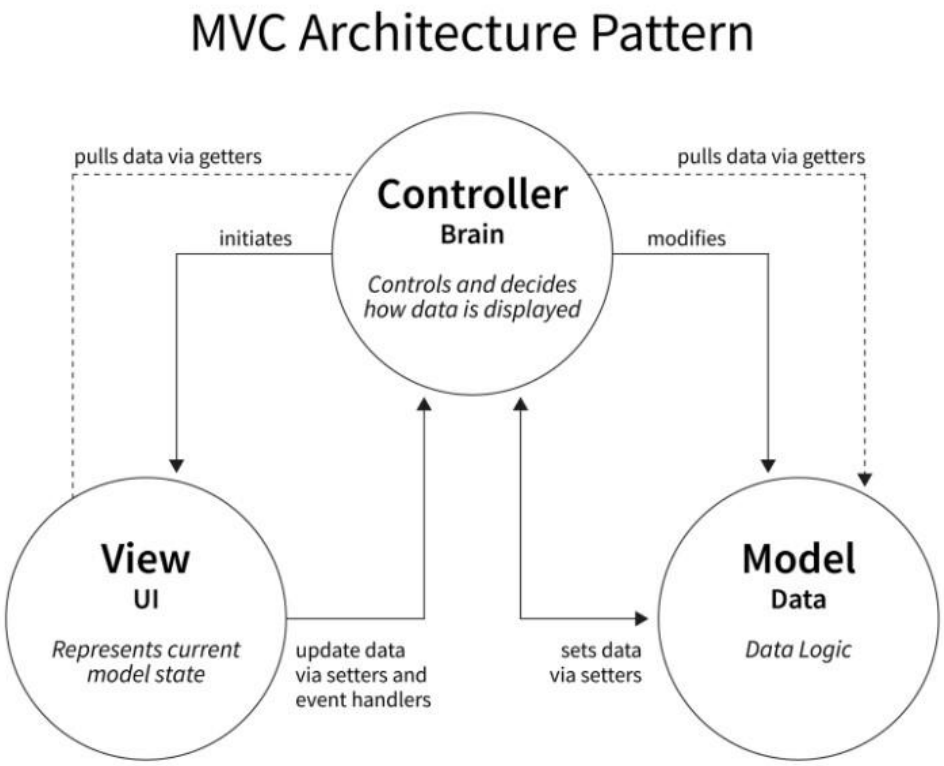

Figure 3.1 - MVC Architecture Pattern (source [7])

3.1.2.2 ASP.NET and C#

ASP.NET is a free open-source and cross-platform development platform dedicated to building various web, mobile, and even IoT applications [8]. A great feature that offers versatility to this platform is its included package manager, *NuGet*. *NuGet* contains over 100.000 tools designed to aid the development cycle of complex applications. Among the supported programming languages by *ASP.NET*, the theme of this thesis, *FastLearn*, was developed using *C#* programming language.

At its core, *C#* is an object-oriented programming language developed to create objects and define their behavior [9], making the development process of an application more intuitive. Throughout the years, the designing process of reliable applications was

highly improved. Features like automatic garbage collection and robust exception handling allow the programmer to focus on the application's logic. Another remarkable feature of *C#* is the Language Integrated Query (LINQ), which enables more efficient data handling.

3.1.2.3 Entity Framework

Entity Framework (EF) [10] is an *Object-Relational Mapping* (ORM) framework that provides an easy way for the developers to interact with the database directly through the code. As presented in Figure 3.2, the data layer of the application, in our case, the Models, have direct correspondents in the database under the form of tables.

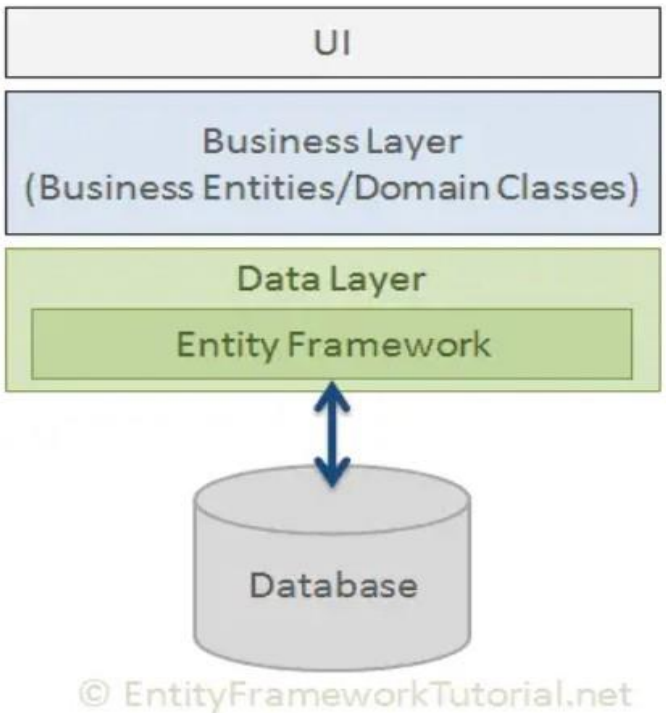

Figure 3.2 - Entity Framework (source [10])

The interaction with the database is highly simplified, as the developer can perform database operations at a higher level of abstraction. *EF* provides numerous advantages to its users. One of the most important features this framework comes with is the possibility to use migrations. The migrations provide a mechanism to incrementally develop the database schema as the data layer of the application grows more and more complex. As presented in Figure 3.3, this enables the use of the Code-First approach.

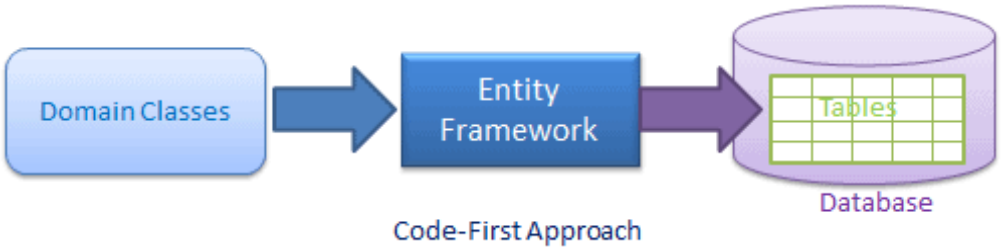

Figure 3.3 - Code-First approach (source [11])

The Code-First, introduced in *Entity Framework* 4.1 [11], is an approach that focuses on creating the domain of the application first. The development cycle using the Code-First method defines the domain classes, configures the classes, and, lastly, updates the database schema through migrations [11]. Furthermore, *EF* enforces built-in conventions which improve code readability and project understanding.

3.1.2.4 Dependency injection and Autofac

Autofac [12], one of the most used IoC (Inversion of Control) containers in *C#*, is used to manage the dependencies within complex applications. An IoC container also facilitates the use of *Dependency Injection*.

Dependency Injection (DI) is a programming technique in which the dependencies of an object are provided to it from an external source instead of being defined locally. This approach enhances code readability with a modular code structure.

In Figure 3.4, the working principle of an IoC container can be observed. The module of the application, in our case, HomeController, that needs a dependency, ProductService, makes a request to the DI container, which will provide the dependency. Usually, the dependency is provided to the module by injecting it into the constructor of the class. In the case of Figure 3.4, this means that the *DI* container instantiates the ProductService dependency in the Controller of the HomeController class.

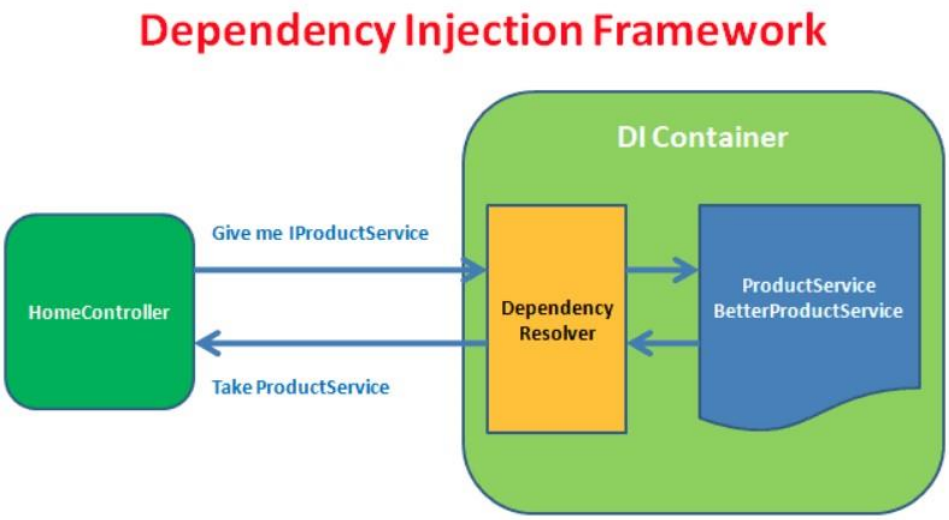

Figure 3.4 - Dependency injection framework (source [13])

3.1.2.5 HTML, CSS, JavaScript and CKEditor

HTML (Hyper Text Markup Language) [14] is used to give structure to a website. It makes use of headings, paragraphs, and tags. The *HTML* code of a webpage is rendered with the help of a browser, resulting in a plain structure in which the information is shown.

CSS (Cascading Style Sheets) [14] allows the programmer to customize the aspect of the page and make it more appealing. Highly versatile, *CSS* empowers the developer to change the user interface's appearance to suit the application's needs.

Javascript (JS) [15], firstly known as LiveScript [15], is a lightweight programming language usually used to allow client-side interactions with the user. *JS* enables the developer to define context-specific functionalities directly in the client. For example, at the click of a specific button, a *JS* script will run and validate certain fields of a form present on the *HTML* page.

Figure 3.5 perfectly summarizes the roles of the previously described technologies. Combining the capabilities of *HTML*, *CSS*, and *JavaScript*, developers can create an interactive interface to enhance the end-user experience.

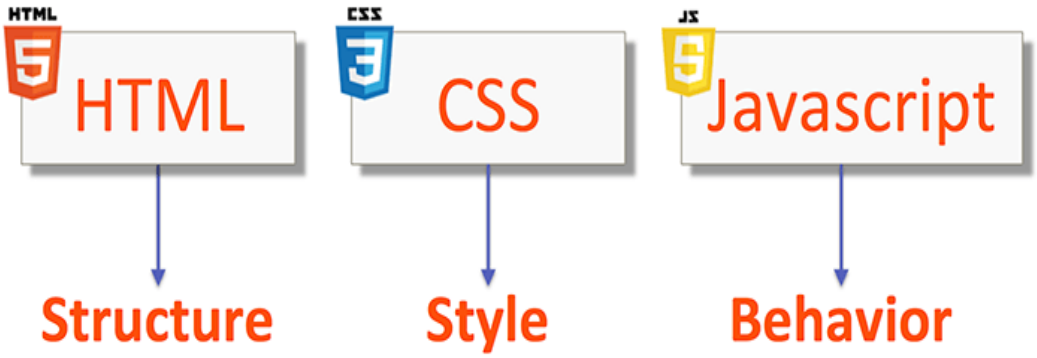

Figure 3.5 - HTML, CSS and JavaScript interaction (source [14])

CKEditor [16] is a modern, reliable, highly customizable text editor developed using TypeScript and JavaScript. These qualities make it a perfect match for the theme of this thesis, *FastLearn*, as the *Professors* need a high degree of flexibility when designing the structure and aspects of their course.

3.2 Design of the FastLearn software application

3.2.1 Database design

One of the goals of this diploma thesis, *FastLearn*, is to design a reliable application to enhance the learning experience of its end-users by providing accurate information on the topic they want to study. As one can imagine, this objective requires efficient data storage and management. In the design phase of *FastLearn*, it was determined that the solution was the use of a database. The database provides a scalable and efficient method to store and organize the high load of information to be handled by the application.

Employing a Code-First approach, at the end of the development phase of *FastLearn*, the need for the following entities was identified: *AspNetRoles*, *AspNetUsers*, *UsersDatas*, *Warnings*, *UserAnswers*, *Tests*, *SubChapters*, *SubChapterFiles*, *Questions*, *Messages*, *HelpingStudentApplications*, *EnrolledStudentInCourse*, *Courses*, *CourseReviews*, *Chats*, *Chapters* and *Answers*. A comprehensive analysis of the aforementioned entities is provided in the following paragraphs.

The analysis starts with the entities responsible for user authentication and user role management. The entities in question are *AspNetRoles* and *AspNetUsers*. In the design phase of the application, it was decided to opt for the „individual user" authentication method provided by the ASP.NET MVC 5 project template. This decision is based on the need for a high level of security for the user data. The first entity, *AspNetRoles*, stores all the application roles in the Name field. The second entity, *AspNetUsers,* contains critical user account information. From the vast collection of provided fields, the fields used in developing this application are the Email and the PasswordHash (the password is hashed for security reasons).

The rest of the entities were developed in an auxiliary project module handles all the data transactions with the auxiliary database. This approach was employed to provide a clear separation between the authentication security of the application and the data handling procedures.

The first analyzed entity from the second project is *UserDatas*. This entity contains information about the user, vital for the good functioning of the application. The Email field is meant to help the identification of the user and provide contact information if it should be needed. The IsSuspended and SuspendedUntil fields store if the user's access to the platform is denied, and if yes, until when. The Points field keeps track of the platform currency the user has accumulated through activities. The Points can be spent to unlock certain features of the application. The UserRole field is meant to store the user's role to optimize the number of requests required, as the user's role is often required when handling the data.

In what follows, the base entities that make storing the course data possible are analyzed. The entities in question are *Courses*, *Chapter*, and *SubChapter*. More entities are required to develop a fully-defined course, which are ulteriorly discussed. The *Courses* entity contains basic information about a class, such as the CourseName and CourseDescription. Apart from these, fields like Active, DeactivationReason, and IssueResolved are designed to enable the review process of a course. Once a class is created, the Active field is false by default, the course needing to be reviewed and approved by an *Admin* user type. Additionally, these fields provide a way to take down the course from the platform if issues with its content are found. The last field of this entity is called ImagePath, in which the location on the server of the main course image is stored. The next fundamental entity in defining a course is the *Chapter*. The before-mentioned entity contains minimal but essential information about the chapter, such as ChapterTitle, ChapterDescription, and ChapterNumber. The first two fields define the title of the chapter and offer a short description of the chapter, while the last field keeps

the order of the chapter in the course. The last fundamental entity needed to define a course is the *SubChapter* entity. The fields of this entity are SubChapterTitle, SubChapterDescription, and SubchaterNumber. The first one defines the title of the chapter, while the last one establishes the order of the subchapter among all the subchapters of the chapter it belongs to. The SubChapterDescription is a special field where the course information is stored. The *Professor* user type can add information in this field through the *CKEditor* text editor, and the formatted information are stored in the database.

The next described domain entities are used to enable the courses to have dedicated test quizzes: *Tests*, *Questions*, *Answers* and *UserAnswers*. The *Tests* entity contains information defining the minimum score to pass the test, called MinimumScore, and a short test description stored in the field TestDescription. The *Questions* entity contains information about the text of the question, called QuestionString. The *Answers* domain class contains the text of the answer, stored in the AnswerText field, and keeps track if the answer is correct or not, via the IsCorrect field. The *UserAnwers* entity stores information about the responses of a *Student* for a certain test. The fields of this domain class stores all the answers of the *Student* in the field ChosenAnswersIdListSerialized, which is a string field containing the id's of the answers chosen, the score obtained by the test taker, and if the test was passed or not.

The *SubChapterFiles* entity allows the *Professor* user type to add supplementary files for a certain subchapter in order to help the *Student* user type to understand the topic presented in that subchapter. This entity contains information about the name of the file, the date it was uploaded, and the path where the resource can be found on the local server.

The next set of entities enables the chat functionality of the platform. The *Chats* and *Messages* entities are responsible for storing the data related to the chatting session between two users, the *Student* that needs help, a regular *Student,* and the *HelpingStudent*. The *Chats* entity contains information regarding the chat session topic and, if the chat is still active, through the IssueResolved field. Once the *Student* considers the issue was solved, he can close the conversation, and subsequently, the IssueResolved field will be marked as confirmed. The second entity, *Messages*, stores the messages sent in the context of a chat session. The MessageContent field stores the text of the message, while the TimeStamp field stores the date and time when the message was sent.

The *CourseReviews* entity, as its name suggests, enables a *Student* to leave a review for a particular course. Through the ReviewText and Stars fields, a short description of the experience the *Student* had while enrolled in this course and a star rating between one and five stars can be provided. The IsActive and DeactivationReason fields ensure that the reviews remain informative and constructive and do not derail from this purpose. Once a *Student* leaves a review, it must be approved by an *Admin*. An *Admin* can deactivate a review and provide a reason if a specific feedback message is inappropriate.

The *EnrolledStudentsInCourses* entity keeps track of the progress of the *Students* in the courses they are enrolled in. Besides the external relations this entity has to enable the progress tracking feature, it has the CompletedCourse field, which monitors if the student finished the course or not.

The *HelpingStudentApplications* entity makes the ascension of a regular *Student* to the rank of *HelpingStudent* possible. The FinishedCoursesIds field is of type string, and keeps a serialized list of the Ids of the courses the *Student* has completed until the moment he requested the promotion. The status of the application is also tracked, as once approved, the status of the application will change.

The last entity described is called *Warnings*. Through the WarningReason field, this entity enables the administrators to warn certain users if their behavior breaks the rules of the platform. At three warnings, the user is suspended for three whole days.

The conceptual model of the database, togheter with the entities described above, can be obserbed in the Bachman diagram below (Figure 3.6).

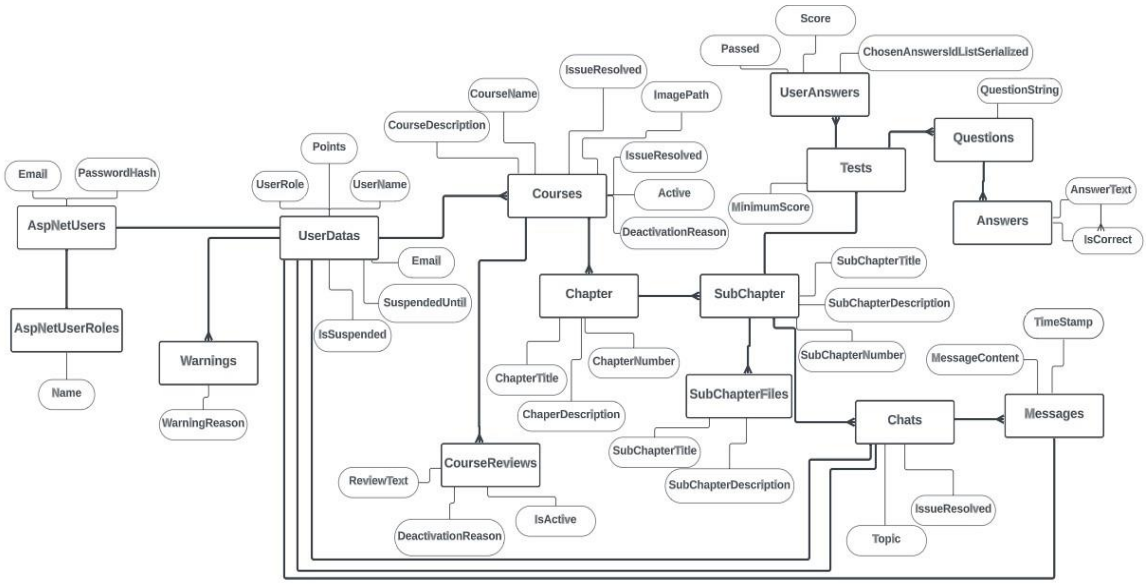

Figure 3.6 - Bachman diagram of the database

Figure 3.7 presents the entity relationship legend used in Figure 3.6.

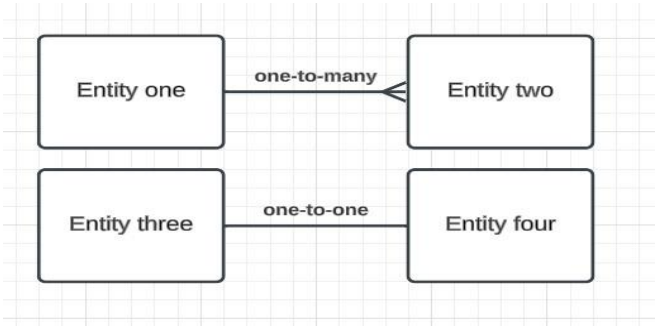

Figure 3.7 - Entity relation legend for the Bachman Diagram

The entities described above form the conceptual model of the database. The next step is to define the logical model of the database. This can be achieved by defining the relational schemes. The relational schema of an entity contains its atributes, the constraints imposed on the atributes and the primary key (a unique numerical identifier) for each instance of the entity.

Next, the relational schemes of the database are presented. In the definition process of this schemes, the following acronyms are used: Auto Increment (AI), Primary Key (PI), Foreign Key (FK), Not Null (NN) and Relational Schema (RS).

RS *UserDatas* =(UserDataId(PK, int, NN, AI), UserId(FK, string, NN), UserName(string, default:"Not Chosen."), Points(int, default: 0), EnrolledCourseId(FK, int, NN), Email(string, NN), SuspendedUntil(DateTime, default: DateTime.Now), IsSuspended(boolean, default: False), UserRole(string, default: student), HelpingStudentApplicationId(FK, int, NN))

RS *Warnings* = (WarningId(PK, int, NN, AI), WarningReason(string, NN), UserDataId(FK, int, NN))

RS *EnrolledStudentInCourses* = (EnrolledStudentInCourseId(PK, int, NN, AI), CourseId(FK, int, NN), LastCompletedChapterId(int, NN), LastCompletedSubChapterId(FK, int, NN))

RS *Courses* = (CourseId(PK, int, NN, AI), CourseName(string, NN), CourseDescription(string, NN), Active(boolean, default: false), OwnerId(string, NN), DeactivationReason(string, NN), IssueResolved(boolean, NN), ImagePath(string, NN))

RS *CourseReviews* = (CourseReviewId(PK, int, NN, AI), UserId(FK, int, NN), ReviewText(string, NN), Stars(int, NN, MinVal: 1, MaxVal:5), CourseId(FK, int, NN), DeactivationReason(string, NN), IsActive(boolean, default: false))

RS *Chapters* = (ChapterId(PK, int, NN, AI), ChapterTitle(string, NN), ChapterDescription(string, NN), CourseId(FK, int, NN), ChapterNumber(int, NN, AI))

RS *SubChapters* = (SubChapterId(PK, int, NN, AI), SubChapterTitle(string, NN), SubChapterDescription(string, NN), ChapterId(FK, int, NN), SubChapterNumber(int, NN, AI))

RS *SubChapterFiles* = (SubChapterFilesId(PK, int, NN, AI), Title(string, NN), Description(string, NN), FilePath(string, NN), UploadedDate(value: DateTime.Now), SubChapterId(FK, int, NN))

RS *Tests* = (TestId(PK, CK, int, NN, AI), MinimumScore(int, NN, MinVal: 50, MaxVal: 100), TestDescription(string, NN)) (TestId - SubChapterId one-to-zero-to-one relation with the SubChapter)

RS *UserAnswers* = (UserAnswerId(PK, int, NN, AI), Passed(boolean, NN), Score(int, NN , MinVal:0, MaxVal: 100), UserId(FK, int, NN), TestId(FK, int, NN), ChosenAnswersIdListSerialized(string, NN))

RS *Questions* = (QuestionId(PK, int, NN, AI), QuestionString(string, NN), QuestionPointPercentage(int, default: 1), TestId(FK, int, NN))

RS *Answers* = (AnswerId(PK, int, NN, AI), AnswerText(string, NN), IsCorrect(boolean, NN), QuestionId(FK, id, NN))

RS *HelpingStudentApplications* = (HelpingStudentApplicationId(PK,int, NN, AI), FinishedCoursesIds(string, NN), StudentId(FK, int, NN), ProfessorId(FK, int, NN), IsApproved(boolean, NN, default: false))

RS *Messages* = (MessageId(PK, int, NN, AI), MessageContent(string, NN), UserId(FK, int, NN), TimeStamp(DateTime, default: DateTime.Now), ChatId(FK, int, NN))

RS *Chats* = (ChatId(PK, int, NN, AI), Topic(string, NN), StudentId(FK, int, NN), HelpingStudentId(FK, int, NN), SubChapterId(FK, int, NN), IssueResolved(boolean, NN))

3.2.2 Web application design

In the early stages of the application development, two main design methods were considered: Use-Case Driven design and domain-driven design.

Domain-Driven Design(DDD) is a software development method that focuses on the domain logic and a solid understanding of what the behavior of the application should be. One of the biggest advantages of using DDD is easier communication between the developers of the application and the domain experts. On the other hand, Use-case Driven Design focuses on the interaction between the application and the user. It revolves around identifying the needs of the end-user, and desing the application to best fulfill them.

As specified before hand, one of the core objectives of *FastLearn* is to provide an intuitive and user-friendly interface, centered around the educational requirements of the student. With this goal in mind, the design of the application was done using a Use-case Driven Design. At the end of the design phase, five user roles were defined: *Unregistered User*, *Student*, *HelpingStudent*, *Professor* and *Admin*. In the following paragraphs, the specific use-cases of each role will be analyzed.

3.2.2.1 Design of the Unregistered User

This type of role is destined to allow the users without an account to assess if the resources of the platform have the potential to satisfy their needs. The identified use-cases can be observed in Figure 3.8.

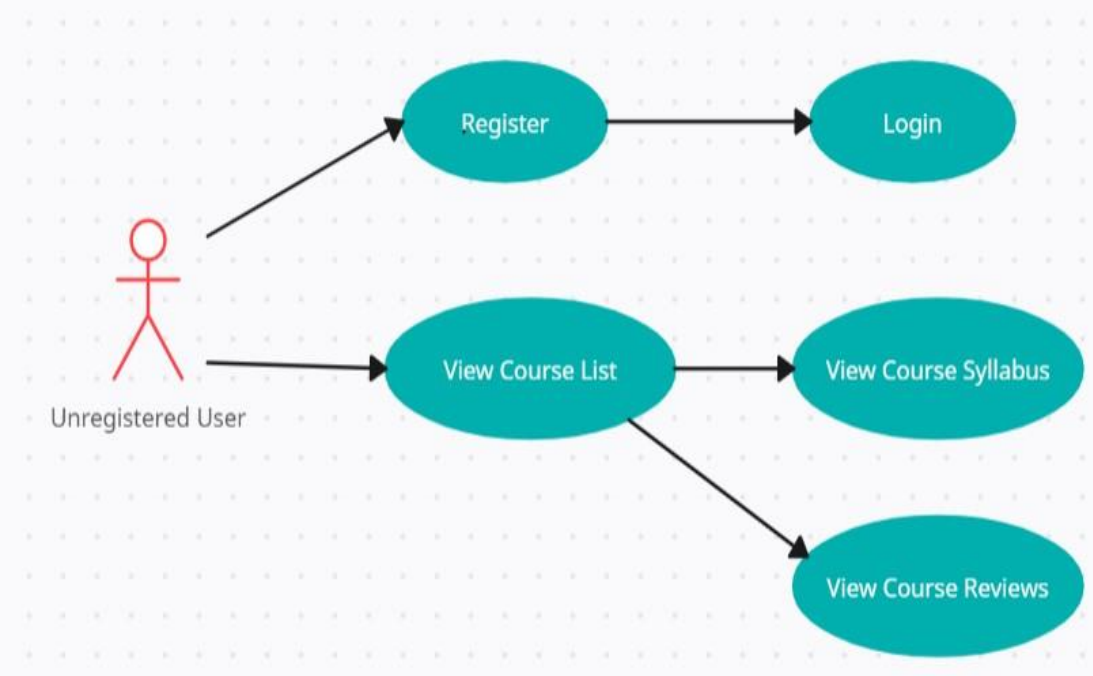

Figure 3.8 - Unregistered User use-case diagram

As shown in the above figure, an Unregistered User has two main use cases. The first one is examining the courses available on the platform through the provided syllabus and reviews. The second option this user has is to join the community of *FastLearn* and start studying the course he finds appropriate for his needs.

3.2.2.2 Common use-cases for the registered user-types

This section presents the features accessible to any registered user type. The functionalities elaborated here are available to all four remaining user roles: *Student*, *HelpingStudent*, *Professor* and *Admin*. The functionalities in question can be observed in Figure 3.9.

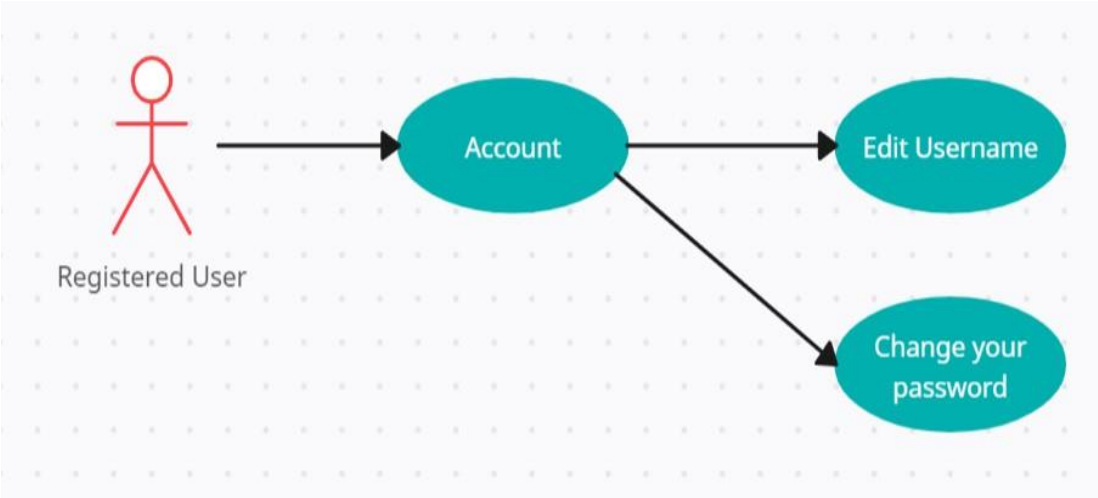

Figure 3.9 - Registered User use-case diagram

As shown in the above figure, every logged-in user can access the account page. On this page, the user can view account-specific information, such as his username and the number of Points he owns. Furthermore, from this page, the username and password can be changed according to the wishes of the owner.

3.2.2.3 Design of the Student user-role

The *Student* represents the target user of *FastLearn* and represents the default user-role a new account receives upon registration. The specific use-cases that resulted at the end of the design phase can be observed in Figure 3.10.

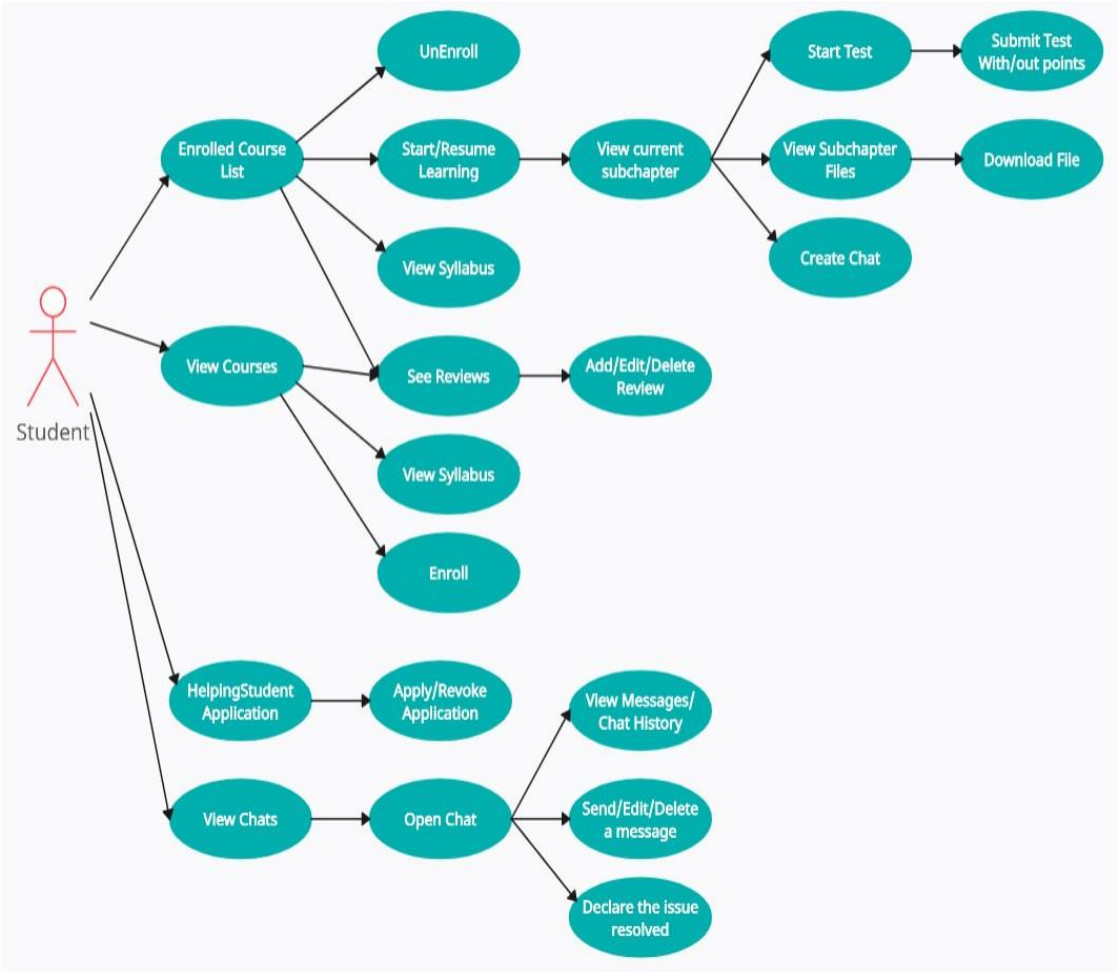

Figure 3.10 - Use-case diagram for specific Student functionalities

The *Student* has the possibility to look at the courses available on the platform. Additionally, this user can add his own review to a specific course. Once a review is created, the user can edit or delete it any time he/she wants.

The most important functionality of this user role is the possibility to enroll in a course and the features unlocked by this procedure. Once enrolled in a course, the *Student* can start studying the materials prepared by the *Professor* by accessing the *EnrolledCourseList* and clicking on the name of the course (Start/Resume Learning option). Next, the user is redirected to the last subchapter he completed by taking the associated test. In the View current subchapter page, several options are available.

The first one is to take the subchapter-specific test and progress to the next subchapter of the course by accessing the *Start Test* option. Upon accessing this option, the test begins. At the end of the test, the *Student* has the option to submit the test in two ways: *Submit Test*, which normally evaluates the test, and *Submit Test with Points*, which will unlock a less-strict grading system for the test. Both *Submit Test* and *Submit Test with Points* redirect the user to the test overview page. If the test was passed, the next subchapter can be opened, if not, the user will be guided back to the current subchapter. The second option the user has in the *View current subchapter* page is to view the files the *Professor* provided for that specific subchapter through the *View subchapter files* option and download them if the he/she so desires. The last important option the *Student* has in the *View current subchapter* page is request the help of a *HelpingStudent* through the option *Create chat*, thus opening a one-to one chat request.

Once located in the *EnrolledCourseList* submenu, the *Student* has three remaining options, besides starting or resuming his learning. Through the *UnEnroll* option, the user can stop his learning activities regarding this course. The last two options are to view the reviews of the course, and manage their reviews or to view the syllabus of the course.

The next presented feature is accessible through the *View Chats* submenu, and allows the user to manage the chats they are a part of. Upon opening the previously mentioned submenu, a list of all the chats will be displayed. The chats are separated into two categories: Active Chats and Archived Chats. For the active chats, the chat can be accessed through the *Open Chat* option. In the chat window, the *Student* can explain their problem and communicate with the *HelpingStudent* that will join their chat. Besides viewing and sending messages, the user can also manage their messages. The last option available in this window is declaring the scope of the chat achieved and closing the chat through the *IssueResolved* button. For the second category of chats, Archived Chats, the user can view the chat history of a specific conversation through the „Chat History“ option.

To conclude the features available for a *Student*, the *HelpingStudentApplication* submenu will be described. In this window, the user can request a promotion to the rank of *HelpingStudent* by pressing the *Apply* button. Once the application is submitted, the live progress of their progress can be tracked.

3.2.2.4 Design of the HelpingStudent user role

This section will present the *HelpingStudent* role and describe its features. This user can access all the previously mentioned functionalities in the section named „Design of the Student user-role“. The facilities that differentiate the two user roles can be observed in Figure 3.11.

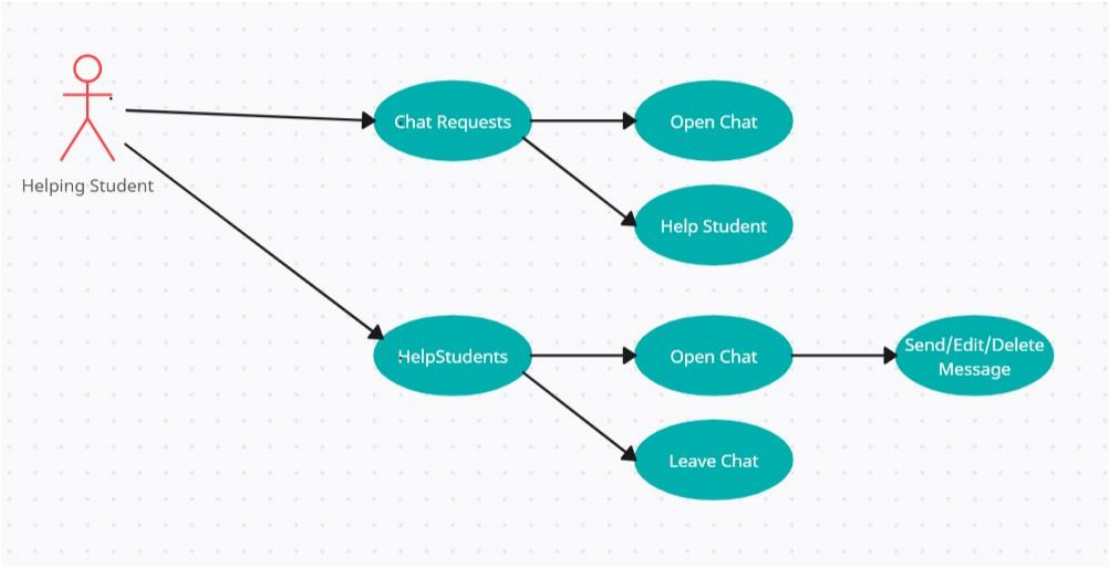

Figure 3.11 - HelpingStudent role use-cases

In the *Chat Requests* submenu, a list of all the active chat requests will be presented. Each active request will contain a short description of the request, including the course name, the chapter, and the subchapter the *Student* needs help with. A particular request has two associated options. The first one lets the *HelpingStudent* to open the chat and see the description the *Student* provided for his problem, while the second one, *Help Student*, allows the *HelpingStudent* to join the chat and offer his expertise if they consider they are able to do so.

The *HelpStudents* submenu will display the Active Chats and the chat history of the user. For a chat listed under Active Chats, the *HelpingStudent* can open the chat through the Open Chat option and start helping the *Student*. The second option, namely Leave Chat, allows the user to leave a chat if they consider it the best course of action. If the *HelpingStudent* leaves the chat, the chat history is preserved. For the courses listed under the Arhived Chats, the chat history mentioned earlier, the user can view the messages sent in that conversation through the Chat History option.

3.2.2.5 Design of the Professor user role

The scope of the *Professor* is to provide authentic information to be consumed by the end-users of this application. To achieve this goal this user has a wide selection of tools at their disposal, which can be observed in Figure 3.12.

In the following paragraphs, the below-mentioned features are presented. To start, it must be mentioned that the *Professor* can share their expertise by creating a course on a specific subject. The information available is structured in chapters and subchapters, each containing a lesson.

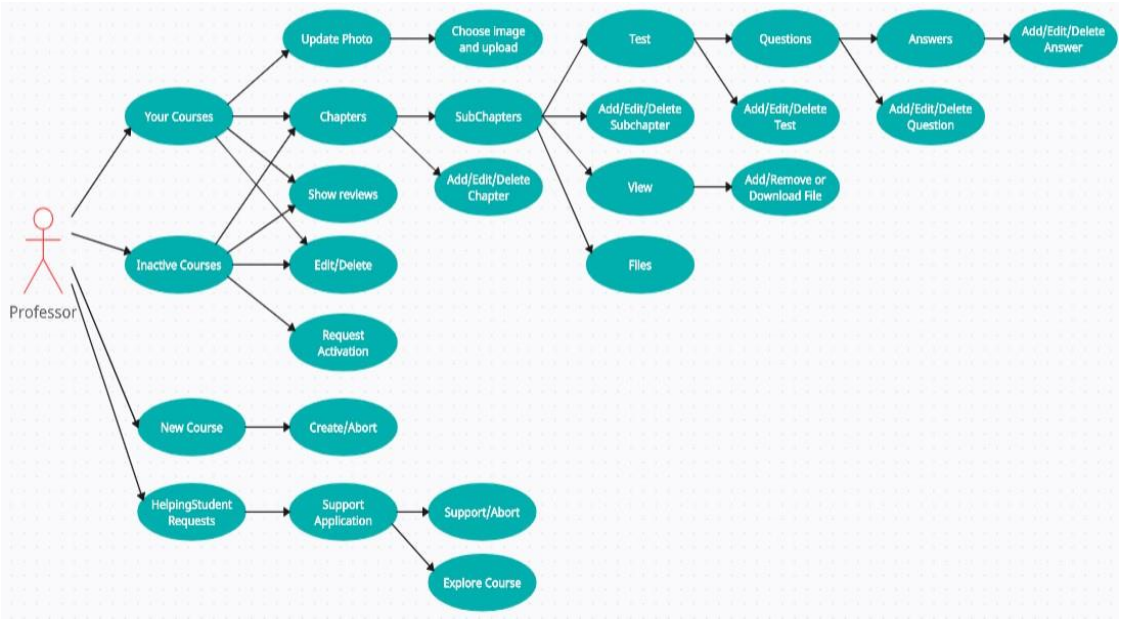

Figure 3.12 - Professor use case diagram

The first feature presented is the *New Course* submenu. In this menu, the *Professor* can start creating a new course by selecting the topic of the course and a short description to help the *Student* understand if the course suits their educational needs.

In the *Your Courses* submenu, several options for each course listed are available. The *Professor* has the option to manage the front page of the course through the Edit and Update Photo options. The course can be entirely removed from the platform by using the option Delete. The Show Reviews option allows the *Professor* to view the feedback of the course-taker and improve the course. The main option available in this submenu is the Chapters option. This option enables the *Professor* to update the content of the course progressively. First in line are the chapters of the course. In the Chapters window, a list of the chapters will be displayed with the option to Edit the chapter title and description or delete the chapter altogether. Alongside those two options, the user can access the subchapter management panel specific to a particular chapter. In the subchapter panel, accessible through the SubChapters option, several features are available. First and foremost, the *Professor* can add, edit or delete a specific subchapter. When a subchapter is created or edited, the user can provide the information specific to that lesson and format it in a dedicated, flexible text editor. Another important option is the Files option. In the dedicated window, the *Professor* can view, add or remove files meant to ease the *Student's* learning journey. The *Professor* can preview how the subchapter will look for the end-user at any given moment. The Test option enables the *Professor* to create a unique test for the current subchapter, with afferent questions and answers. The questions can have multiple answers, but only one can be correct.

Next, the *Inactive Courses* submenu will be discussed. This window is dedicated for the courses that need the attention of the *Professor*. A course is displayed in this submenu, if an issue about the course is signaled, and the *Admin* decides major revisions

are needed. The Chapters option allows the *Professor* to solve the issue. When the user considers the course can be activated, the Request Activation allows them to request a new review of the course.

The *HelpingStudentRequests* submenu contains a list of all the *Student* applications that desire to help their fellow learning companions by becoming a *HelpingStudent*. Once an application is submitted, the next step is to be reviewed by the *Professor*, which can decide if the *Student* is qualified for the promotion. For each particular application, the *Professor* can evaluate the *Student's* proficiency. This can be achieved by examining the curriculum of all the courses the *Student* has completed. At the end of the assessment, the *Professor* can support the application and recommend the *Student* for promotion through the Support option.

3.2.2.6 Design of the Admin user role

The *Admin*, short for administrator, is meant to oversee the good functioning of FastLean. To achieve this goal, the features presented in Figure 3.13 are put at his disposal.

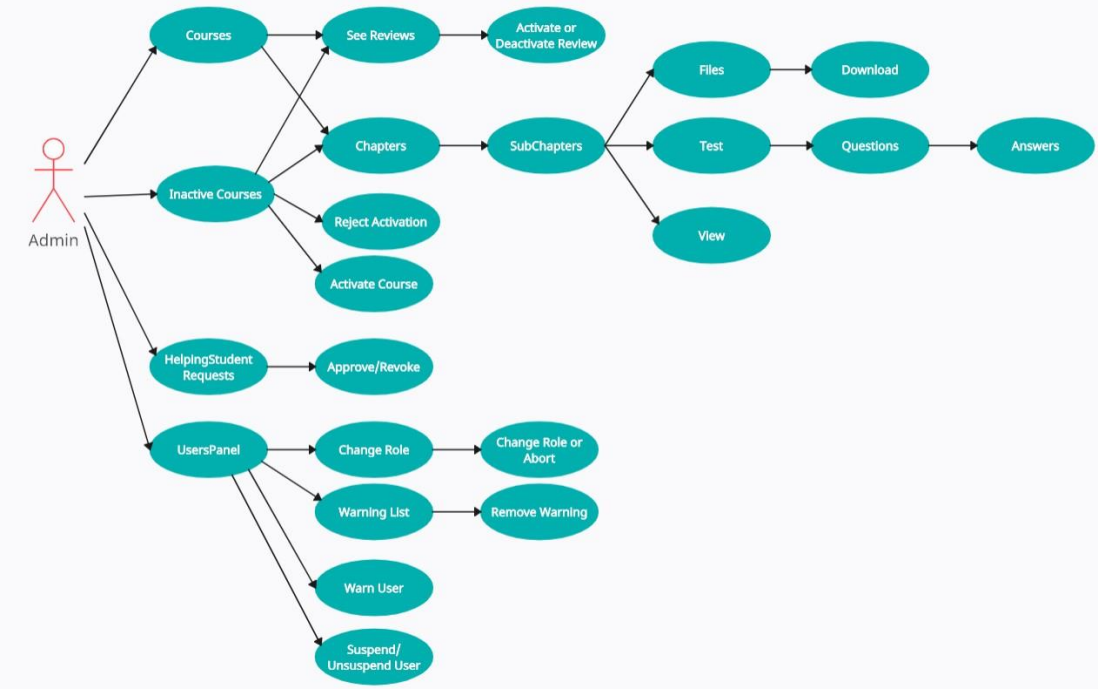

Figure 3.13 - Admin use case diagram

The presentation of the features begins with the features ensuring the high quality of the courses available on the platform. The first two submenus, *Courses*, and *Inactive Courses*, serve this scope. The first one, *Courses*, contains a list of all the available courses to the users. The second one, *Inactive Courses*, displays a list of all the courses that have issues. In both submenus, the *Admin* can check the course content through the Chapters option and view the course reviews through the View Reviews button. Besides the standard features, an *Admin* can put a course under review through the Deactivate Course option available in the Courses submenu or re-

instantiate a course under review through the Activate *Course* option available in the *Inactive Courses* submenu.

In the *HelpingStudentRequests* submenu, a list of all *HelpingStudentRequests* approved by a *Professor* are displayed. The *Admin* has the role to check the *Student's* warnings and past behavior, and assess if they are suited for the promotion. If the result of the assessment is positive, the *Admin* can approve the request through the Approve option.

The *UserPanel* submenu displays a list of all the users registered on the platform. The role of this menu is to enable the *Admin* to reward or punish the users that have exemplary behavior or, respectively, disturb the users of the platform in one way or another. The ChangeRole option allows the *Admin* to promote or demote a user to one of the roles available on the platform: *Student*, *HelpingStudent*, *Professor*, and *Admin*. The Warning List option opens a window where the history of warnings for a particular user can be viewed. Any warning can be revoked from this menu if this is deemed appropriate. The complementary option, to warn a user, is found in the principal submenu under the Warn User button. When a particular user reaches a number of three warnings, this results in an automatic suspension from the platform for three days. The last two options, SuspendUser and UnsuspendUser are to be used if the user's behavior gravely affects the platform.

3.2.3 Web application architecture design

In this section, the application is showcased using the MVC template. The partition can be observed in Figure 3.14.

The *AccountController* and *ManageController* are responsible for the user sign-in, login, and password recovery.

The *AnswerController*, *QuestionController*, and *TestControlle* enable the creation of the tests the *Students* use to test their knowledge and advance through a particular course.

The *CourseController*, *ChapterController* and *SubChapterController* allow the management of a course.

The *ChatController* and *MessageController* enable the *Students* and *HelpingStudents* to have one-to-one conversations.

The *CourseReviewController* is used to manage the reviews left by end-users for a particular course.

The *UsersController* enables the management of user roles, warnings, and suspensions.

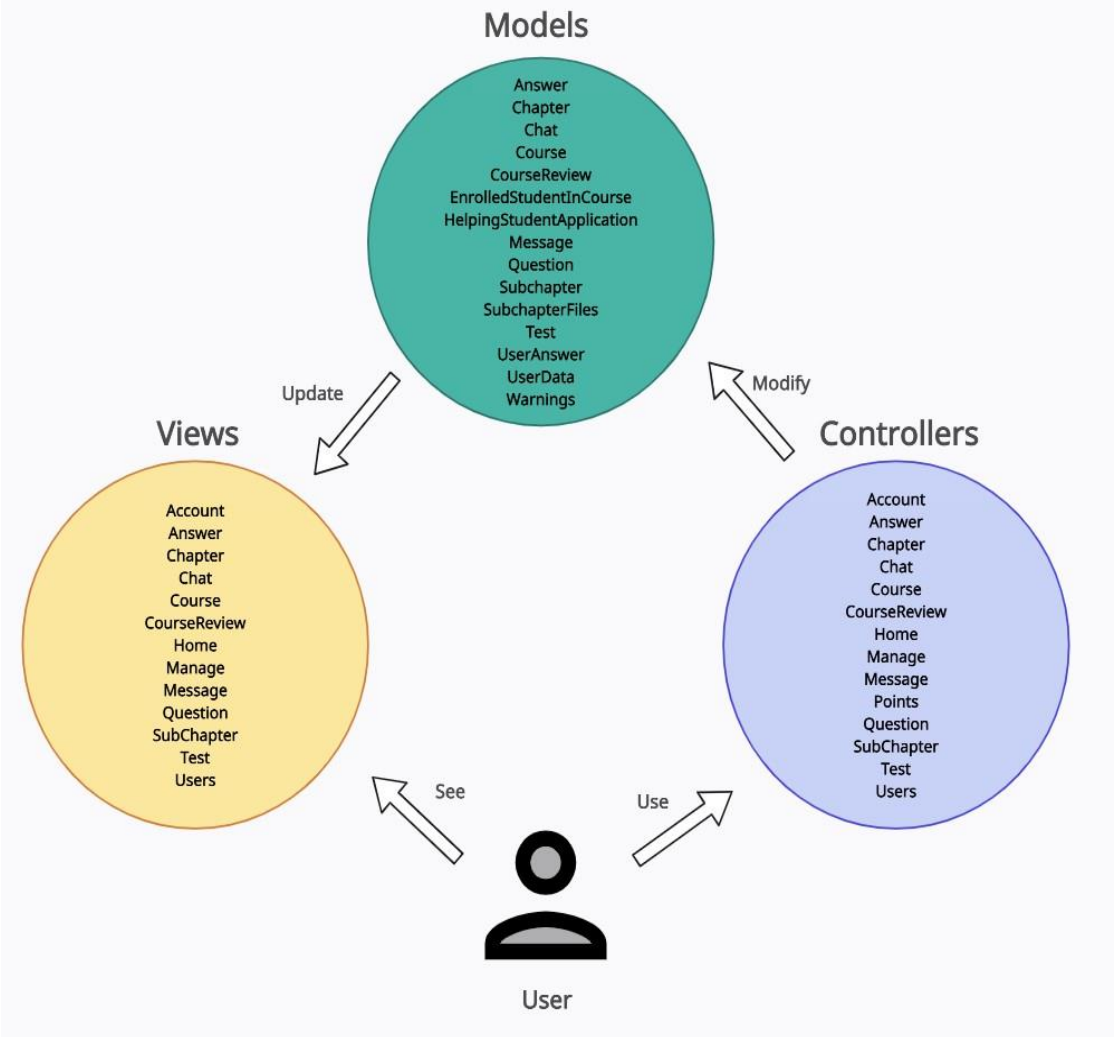

Figure 3.14 - MVC diagram for FastLearn web application

3.3 Implementation of the FastLearn software application

3.3.1 Database implementation

As mentioned in the previous sections, two databases were used to ensure the functioning of *FastLearn*. The first database is responsible for registering and logging in the users, while the second one stores the data needed to run the functionalities of the platform, such as course-related information.

The first mentioned database can be recreated by creating a new project using the ASP.NET Web Application(.NET Framework) project template. In the newly opened window, after selecting the name and the location of our new project, the Create button must be pressed. In the „Create new ASP.NET WebApplication“ window, the settings are indicated in Figure 3.15.

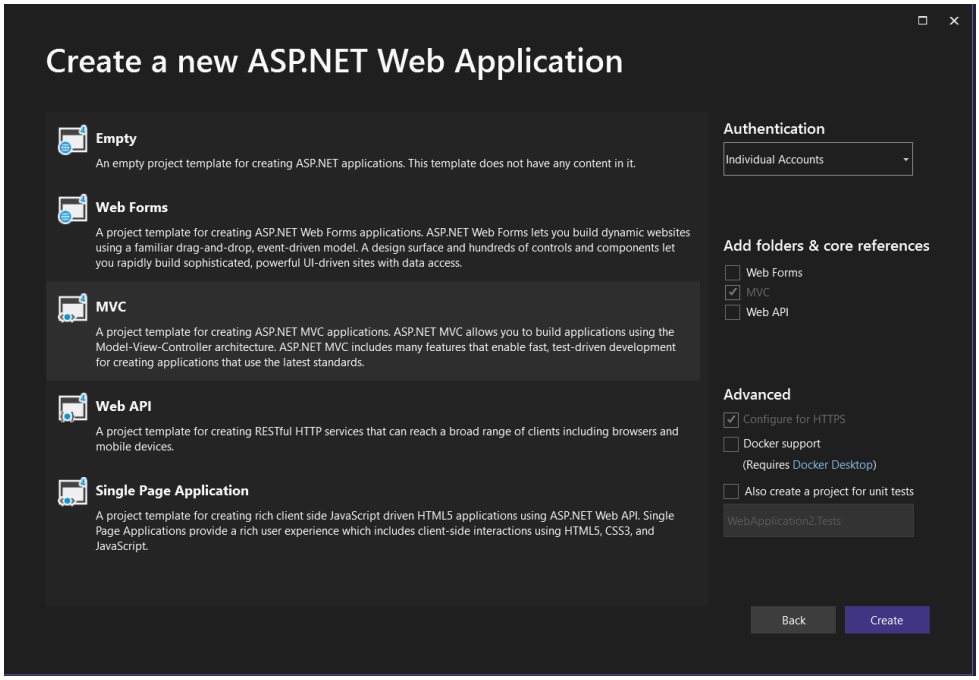

Figure 3.15 - Project configuration window

For the second database of the application, the Code-First approach was used. As the features of *FastLearn* were being implemented, the models of the application were updated to support the functionality. Once one of the models was changed, through the use of migrations, the database was brought up-to-date. In the following paragraphs, an example of this procedure will be showcased.

The presentation illustrates how the chapters of a course are implemented. The first step in the implementation of this feature was creating the *Chapter.cs* model. The final product can be observed in Figure 3.16.

```
{
    16 references
    public class Chapter
    {
        [Required]
        13 references
        public int ChapterId { get; set; }

        31 references
        public int ChapterNumber { get; set; }

        [Required]
        4 references
        public string ChapterTitle { get; set; }

        [Required]
        4 references
        public string ChapterDescription { get; set; }

        17 references
        public int CourseId { get; set; }

        13 references
        public virtual Course Course { get; set; }

        17 references
        public virtual ICollection<SubChapter> Subchapters { get; set; }
    }
}
```

Figure 3.16 - Chapter.cs model class

In the definition process of the *Chapter.cs* model class, several Entity Framework conventions must be respected. The model class must have a unique identifier defined, which has the attribute *[Required]* and is named: „ClassName“+“Id“, so in our case, the field is called *ChapterId*. The second convention allows the definition of a one-to-many relationship between a course and its chapters. In Figure 3.16 half of the convention can be observed. The *Chapter.cs* class must have an instance of the course it belongs to defined, and also have a copy of the *CourseId*. In this case, lines 24 and 26 satisfy this convention. The second half of the one-to-many convention is defined in the *Course.cs* model class and requires the instruction presented in Figure 3.17.

```
13 references
public virtual ICollection<Chapter> Chapters { get;set; }
```

Figure 3.17 - Course model class one to many convention instruction

Once the model classes and relationships between them are defined, the database must be updated. Entity Framework facilitates this process. To reflect the new changes onto the database, the user must do the following steps.

- Open the Package Manager Console
- Create a new migration, using the command „add-migration [*MigrationName*]“
- Update the database using the command „update-database“.

Between the second and third steps mentioned, the generated resulting migration should be similar to the in Figure 3.18.

```
2   {
3       using System;
4       using System.Data.Entity.Migrations;
5
        2 references
6       public partial class AddChapters : DbMigration
7       {
            0 references
8           public override void Up()
9           {
10              CreateTable(
11                  "dbo.Chapters",
12                  c => new
13                      {
14                          ChapterId = c.Int(nullable: false, identity: true),
15                          ChapterTitle = c.String(nullable: false),
16                          ChapterDescription = c.String(nullable: false),
17                          CourseId = c.Int(nullable: false),
18                      })
19                  .PrimaryKey(t => t.ChapterId)
20                  .ForeignKey("dbo.Courses", t => t.CourseId, cascadeDelete: true)
21                  .Index(t => t.CourseId);
22
23          }
24
            0 references
25          public override void Down()
26          {
27              DropForeignKey("dbo.Chapters", "CourseId", "dbo.Courses");
28              DropIndex("dbo.Chapters", new[] { "CourseId" });
29              DropTable("dbo.Chapters");
30          }
31      }
32  }
33
```

Figure 3.18 - Chapter.cs migration file

The migration in Figure 3.18 has two main parts. The Up() and Down() methods. The first one contains the instructions that describe how the database will be changed using the update-database method, while the Down() method allows the modifications to be reverted if needed.

At the end of the implementation phase, the second database contains complex interactions. To help the reader better understand those interactions, Figure 3.19 showcases the database diagram, a detailed and comprehensive visual description of the database. In case the reader wishes to obtain the provided diagram, this can be obtained using *Microsoft SQL Server Management Studio 18*(SSMS), find the *Database Diagrams* folder, and select the *New Database Diagram* option.

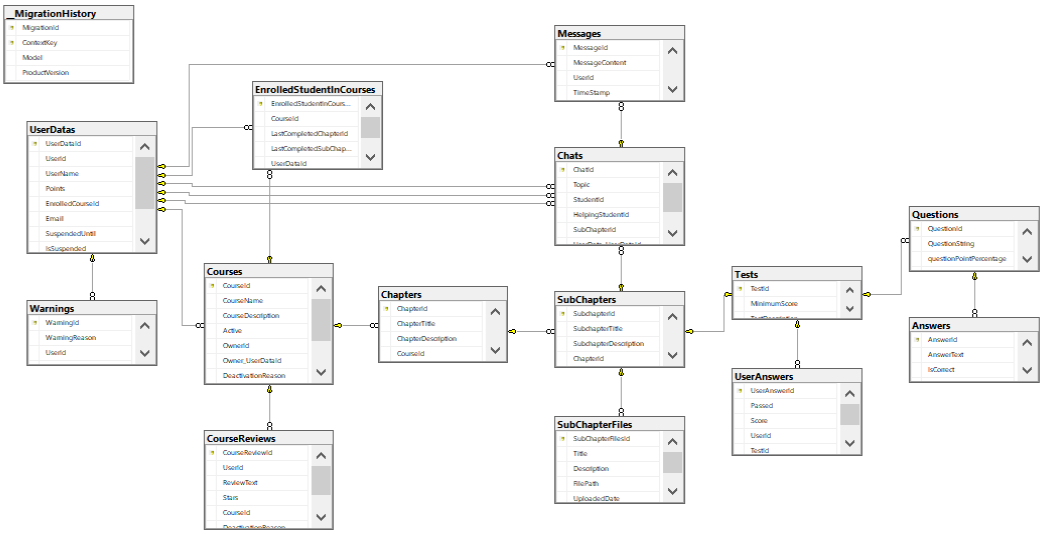

Figure 3.19 - Database diagram

3.3.2 Web application structure

In this section, a clear overview of the structure and design of *FastLearn* is presented. The project is structured into two modules, namely the *.Web* model, and the *.Data* module. An overview of the mentioned modules can be observed in Figure 3.20.

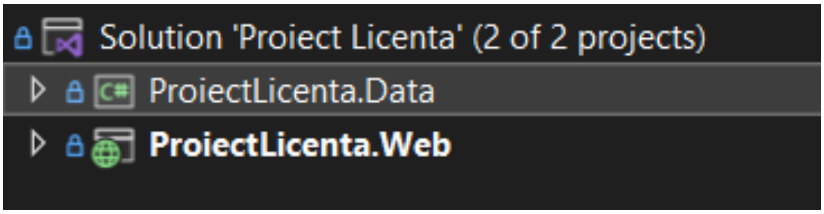

Figure 3.20 - Solution module overview

The *.Web* module is the main module of the application. In this part of the project, the authentication and role management storage and management are handled. Besides the before-mentioned functionalities, this module contains the View and the Controllers that implement the rest of the functionalities of *FastLearn*. In *Figure **3.21*** an overview of this module is provided.

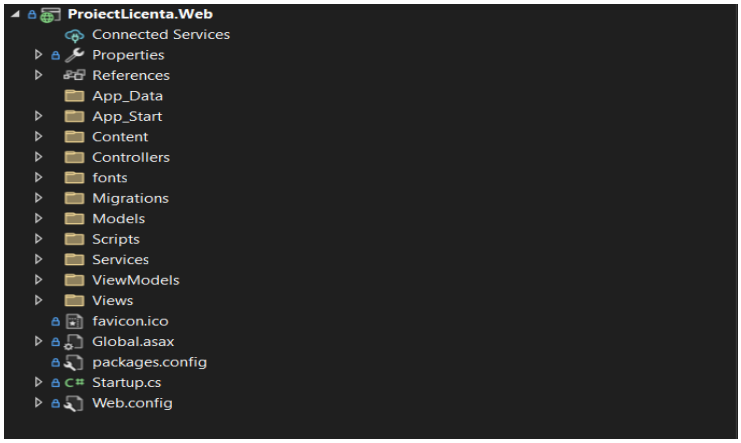

Figure 3.21 - .Web module overview

In the Controllers folder, the Controller component of the MVC architecture is implemented. The View component is implemented in the *Views* folder, observable in Figure 3.22. Each subsequent folder contains the .cshtml classes used to deliver the information to the end-user.

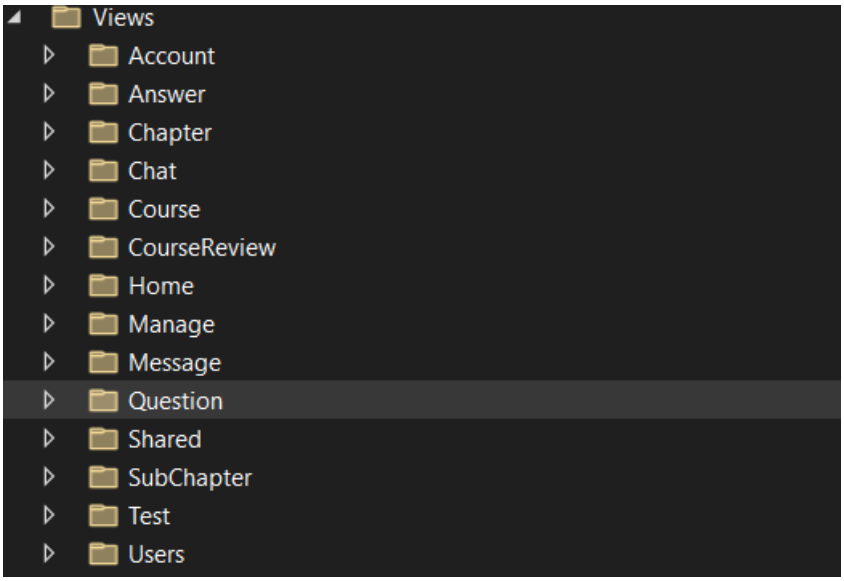

Figure 3.22 - View component of the MVC architecture

The remaining module, *.Data* module, encapsulates the Model component of the MVC architecture and all the subsequent database interactions. An overview of this module is available in Figure 3.23.

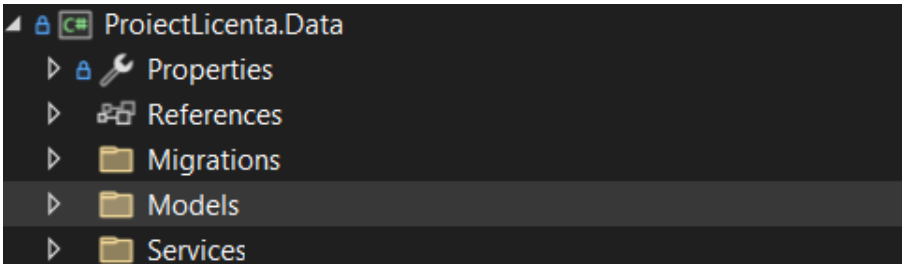

Figure 3.23 - .Data module overview

The three main components needed in the implementation phase are the *Migrations*, *Models*, and *Services* folders. The *Migrations* folder is managed entirely by Entity Framework, a new migration file being generated at each update of the database. The *Models* directory contains the classes describing the structure and rules of the Model component of MVC. The Services folder contains the services responsible for data manipulation within the database corresponding to this module. The database in question is the second database described in section 3.3.1. An overview of the Services directory is provided in Figure 3.24.

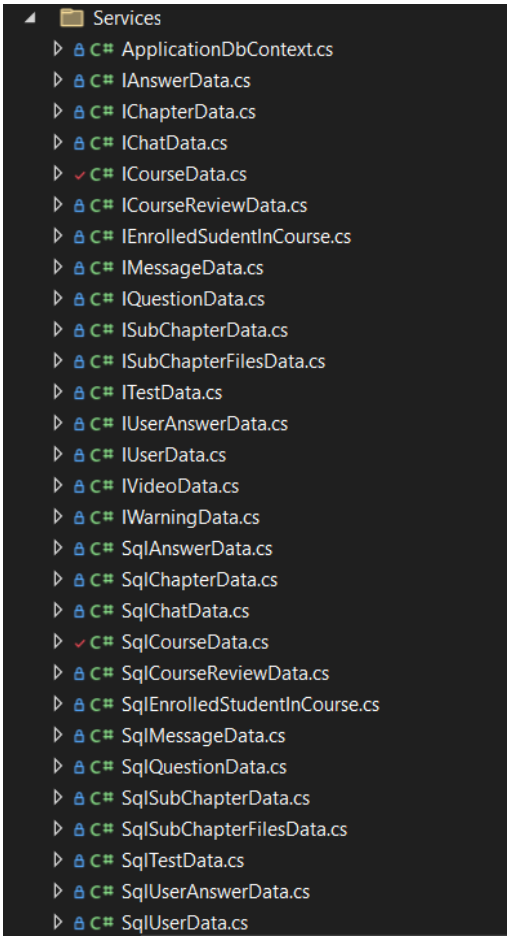

Figure 3.24 - Service directory overview

Figure 3.24 perfectly illustrates how the services are structured. The sub-services are composed of an interface that imposes the methods that must be implemented to ensure the well functioning of the platform, and the implementation of those methods. The interface and corresponding implementation follow a strict naming convention. The interface is named following the pattern "I[ModelName]Data", while the implementation follows the pattern "Sql[ModelName]Data". For instance, the service classes handling the course review logic are named *ICourseReviewData* and *SqlCourseReviewData*.

The Autofac IoC container, explained in section 3.1.2 is used to allow communication between the two modules. In the .Web project, a dedicated dependency

container in the *App_Start* directory. To implement this container, the Autofac IoC package must be installed. Next, a few steps must be followed:

- Instantiate the container.
- Register the controllers.
- Register the interfaces and their implementation.
- Register the DbContexts of the two databases used.
- Build the container and resolve the dependencies.

In Figure 3.25 the above-mentioned steps are illustrated in a simplified manner. It must be mentioned that in order to have a valid implementation of the container, all the services must be registered. To complete the implementation of the container, the developer must use the syntax used in lines 28-29 and register the remaining services.

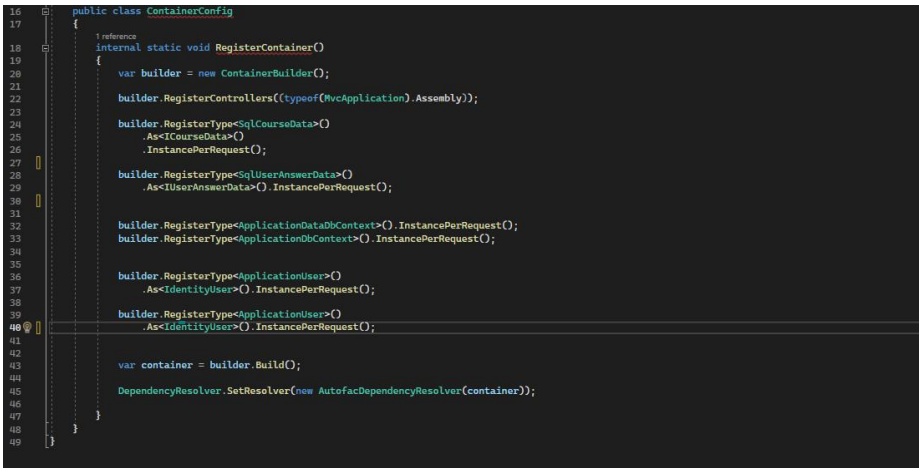

Figure 3.25 - Container class simplified example.

Once the container is set up, the connection between the two modules is successfully realized. One more step is necessary in order to use the models and services from the *.Data* module in the Controllers of the *.Web* module. The developer must inject the needed dependency in the respective controller. An example of this procedure can be observed in *Figure **3.26***. The *MessageController* requires functionalities regarding message management. To this end, a private read-only instance of the interface must be declared, and initialized in the controller's constructor, as illustrated below. *Autofac* will manage the rest, and the developer will be able to use the methods present in *SqlMessageData*(the implementation of *IMessageData*).

Figure 3.26 - Dependency injection in controller example

3.3.3 Module implementation

In the following section, the implementation of the user-roles described in Chapter 3.1.1 is explained with the help of code snippets and user interface screenshots.

3.3.3.1 Unregistered User module

This module represents the first medium of interaction between a new user and the theme of this thesis, *FastLearn*. The first page opened can be observed in Figure 3.27.

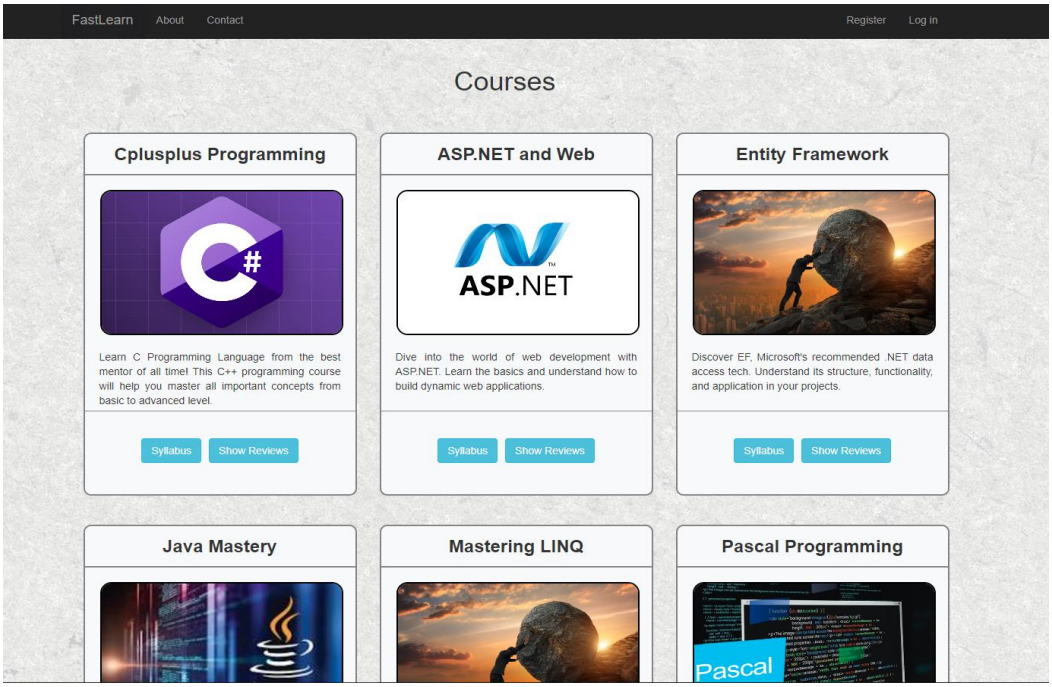

Figure 3.27 - Landing page for Unregistered User

A user without an account has access to several features to help him decide if the content of the platform is suited for his educational needs and, consequently, if he/she needs to create an account.

The first feature in question is browsing the courses of the platform. Each course listed on this page has a dedicated card allocated in which the course title, a custom or default image, and a short description of the course are provided. Based on this information, the new user can decide if a particular course needs to be further analyzed or not.

The second feature available on this page allows the user to study the syllabus of a course he/she is interested in through the option Syllabus. If this option is chosen, the table of contents of the respective course is opened.

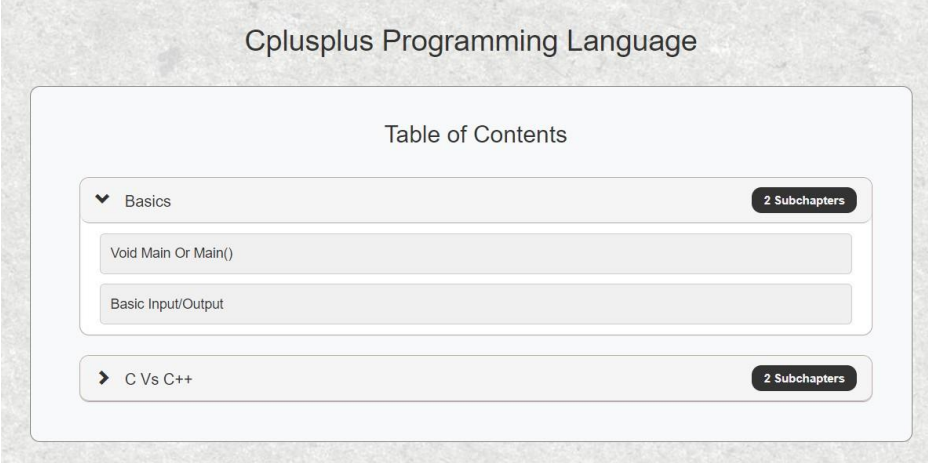

Figure 3.28 - Syllabus page

In Figure 3.28, a list of all the chapters of the course is displayed. For each chapter, the number of subchapters of that chapter is shown, and if the user desires, they can see a list of each chapter's subchapter title. This can be achieved by clicking the arrow button present in front of the name of each chapter.

The code that gives the core structure of the *Syllabus* page can be observed in Figure 3.29. Each chapter will have a dedicated space, hosting the subchapter count and drop-down.

```
<div id="info-box">
    <center>
        <h3 style="margin-top:15px;margin-bottom: 30px;" id="h3">Table of Contents</h3>
    </center>

    <div class="panel-group" id="accordion">
        @foreach (var chapter in Model.Chapters)
        {
            int subChapterCount = chapter.Subchapters.Count();

            <div class="panel panel-default">
                <div class="panel-heading">
                    <div class="title-row">
                        <h4 class="panel-title">
                            <span class="arrow" data-toggle="collapse" data-target="#collapseChapter@(chapter.ChapterId)" aria-expanded="false"></span>
                            @Html.DisplayFor(model => chapter.ChapterTitle)
                        </h4>
                        <span class="badge badge-secondary">@subChapterCount Subchapters</span>
                    </div>
                </div>



                <div id="collapseChapter@(chapter.ChapterId)" class="panel-collapse collapse">
                    @foreach (var subchapter in chapter.Subchapters)
                    {
                        <div class="subchapter-box">
                            <span class="subchapter-title">@Html.DisplayFor(model => subchapter.SubchapterTitle)</span>
                            @if (User.Identity.IsAuthenticated && !User.IsInRole("professor") && (string)ViewData["isEnrolled"] == "true")
                            {
                                @Html.ActionLink(">> More", "ViewSubchapter", "SubChapter", new { subchapterId = subchapter.SubchapterId }, new { @class = "btn btn-info btn-sm" })
                            }
                        </div>
                    }
                </div>

            </div>
        }
    </div>
```

Figure 3.29 - Syllabus core structure code

The interactive behavior of the page is achieved with the use of JavaScript. In Figure 3.29 the JS code used to enable this feature is presented. By combining the implementation presented in Figure 3.29 and Figure 3.30 the feature can be implemented.

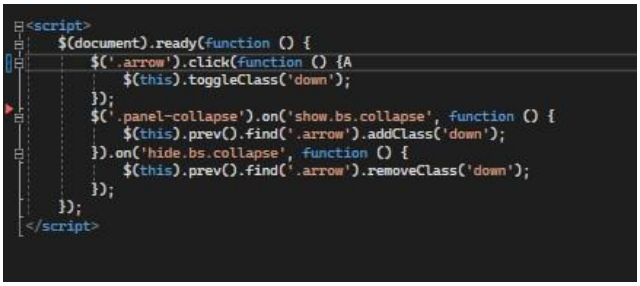

Figure 3.30 - Syllabus interactive code component

The second feature available to the *Unregistered User* is the possibility to view the reviews of a course. The list of reviews can be accessed through the Show Reviews option.

Figure 3.31 - Review page

Figure 3.31 shows how the reviews will be displayed. For each review, a short text description of the user's experience and a corresponding star rating is presented.

In the case in which a suitable course is found, the user has the option to create an account.

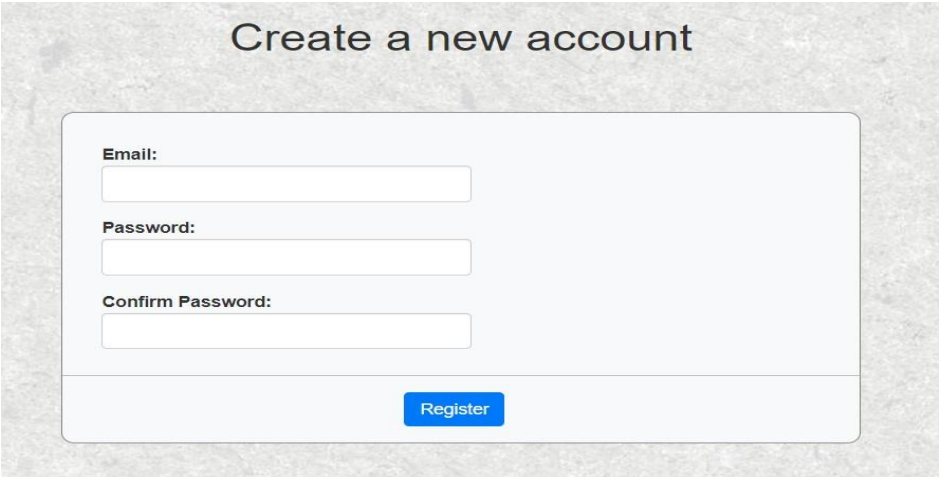

Figure 3.32 - Register page

Figure 3.32 presents the registration form. To create an account, the user must provide a valid email address and a password that follows standard security rules.

Once an account is created, the *Login* option presented in Figure 3.33 can be used.

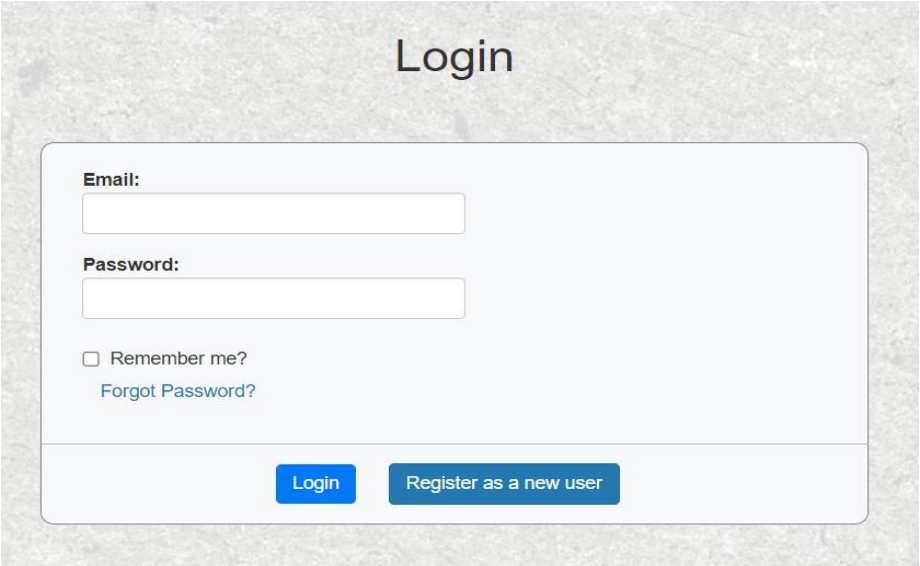

Figure 3.33 - Login page

At any point, if the user forgets his password, it can be recovered by opting for the option Forgot Password? On click, the page presented in Figure 3.34 opens.

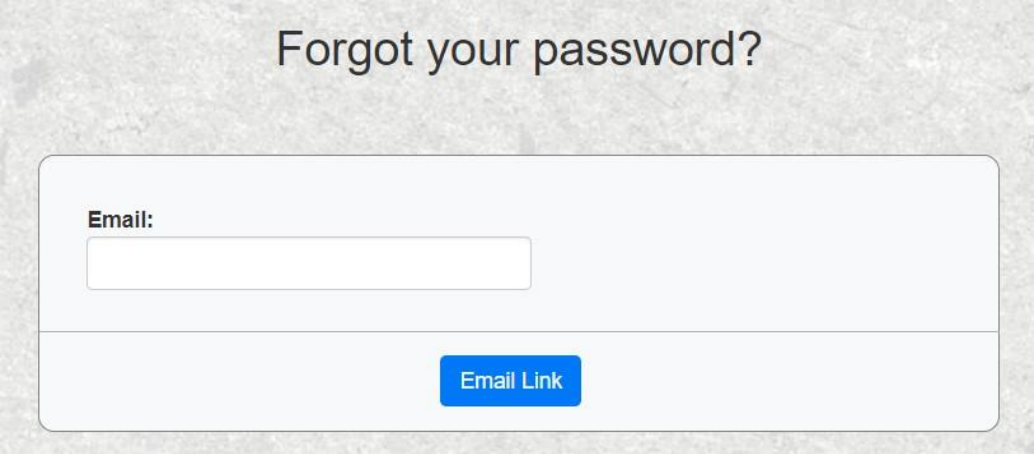

Figure 3.34 - Forgot password page

By providing a valid email in the required field, a recovery email will be sent. By using the unique link sent, the user can change the password for his/her account.

The final feature the *Unregistered User* has is viewing the *About* and *Contact* page of the platform. The first page, the *About* page, contains general information about the platform. The second page, the *Contact* page, contains contact information useful if the users need to contact the administrators of the platform.

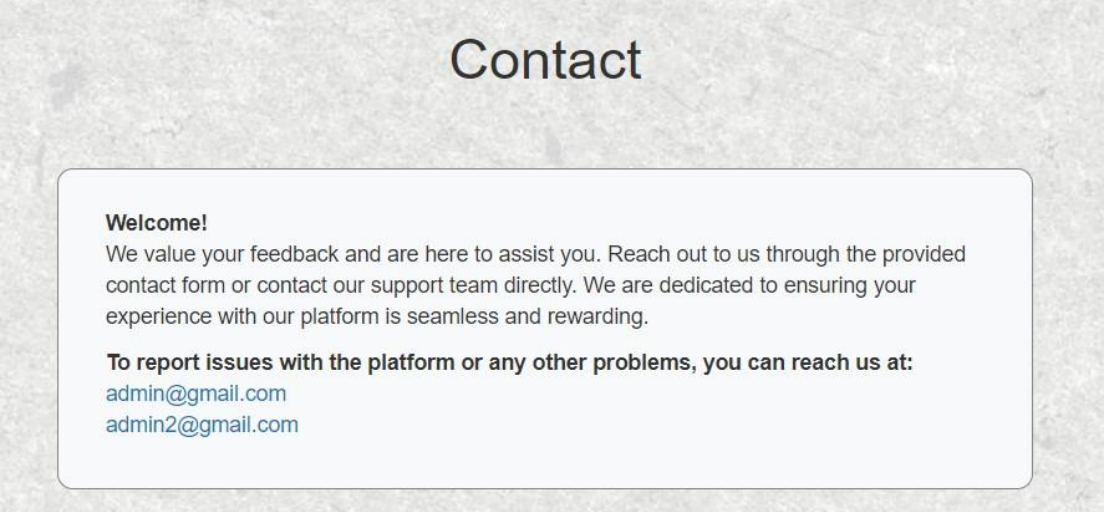

Figure 3.35 - Contact page

Figure 3.35 presents the *Contact* page of the platform. In this window, the contact information of the management is provided. If the user has any issues with the platform, a request can be sent to the listed emails.

3.3.3.2 Student module

The *Student* role is the default role a newly registered user receives. This module is designed for the end-users of the platform, and gives access to multiple features meant to enhance his/her learning journey.

The default page of this module lists all the courses available on the platform, in which the user is not enrolled yet. The landing page of this module is presented in Figure 3.36.

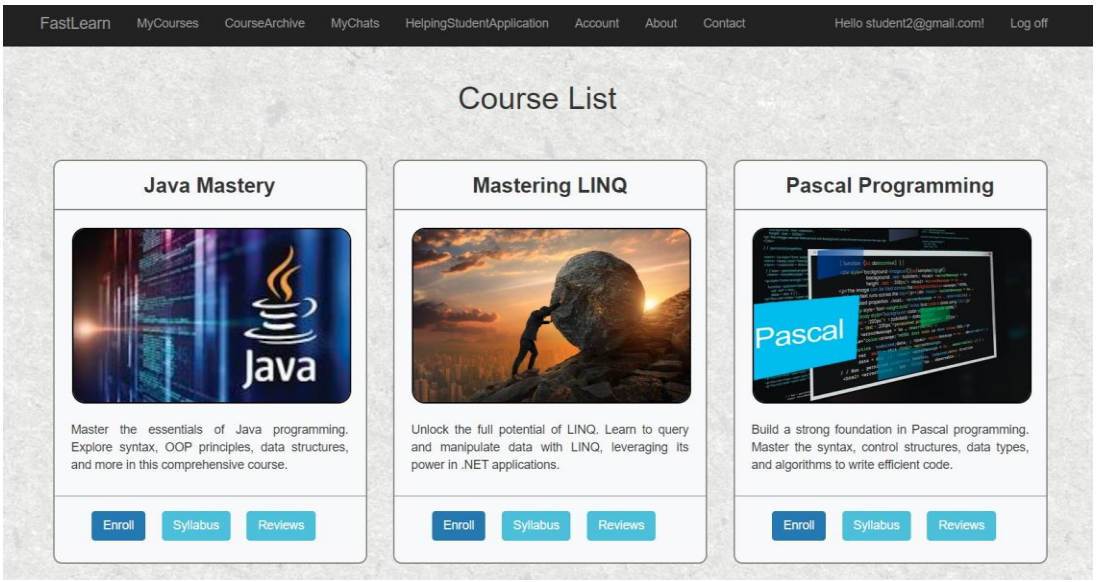

Figure 3.36 - User module landing page

As in the previous module, the user can view the syllabus and reviews of a particular course. Additionally, a *Student* can leave a review and edit or delete an already existing one. Figure 3.37 illustrates the mentioned features.

Figure 3.37 - Course Review page for Student

To add a review for a particular course, the user can select the *Add a new Review* option. Figure 3.38 shows the form that allows this functionality. The user will be able to insert a text review, and a star rating between 1 and 5. This will allow other users to better assess the course quality and decide if enrolling is a good option.

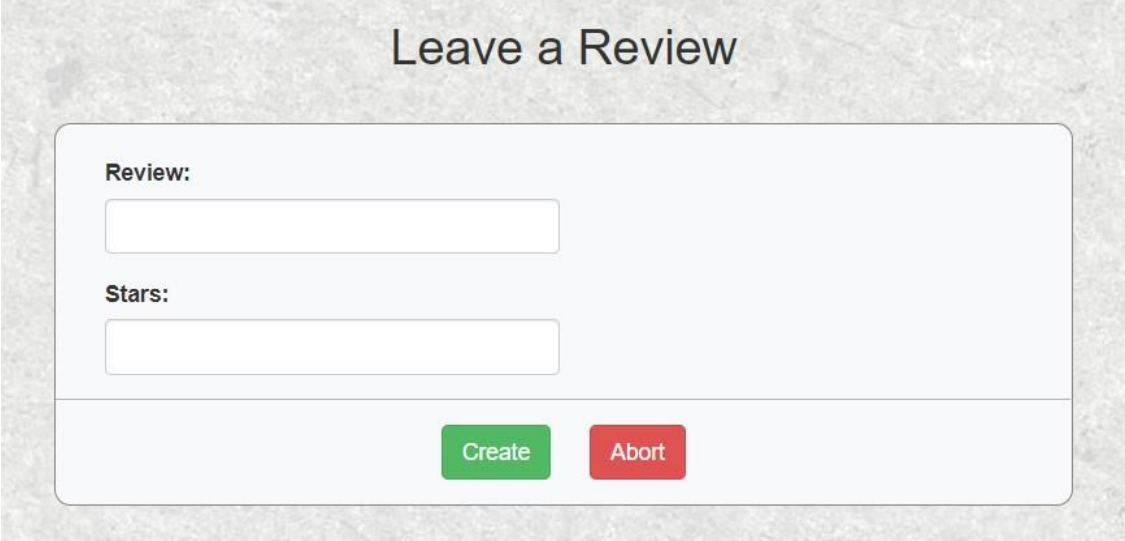

Figure 3.38 - Add a review page

Once a review is submitted, it must be approved by an *Admin*. Once the review is approved, the user can edit or remove the review. The Edit option will enable the user to edit the text and the star rating of the review. It must be mentioned that the restrictions mentioned for the Leave a Review form must be respected in the Edit form as well.

The Delete option, presented in Figure 3.39, allows the removal of the review. In this form, the user can proceed with the deletion of the review through the option Delete or keep the review through the option Abort.

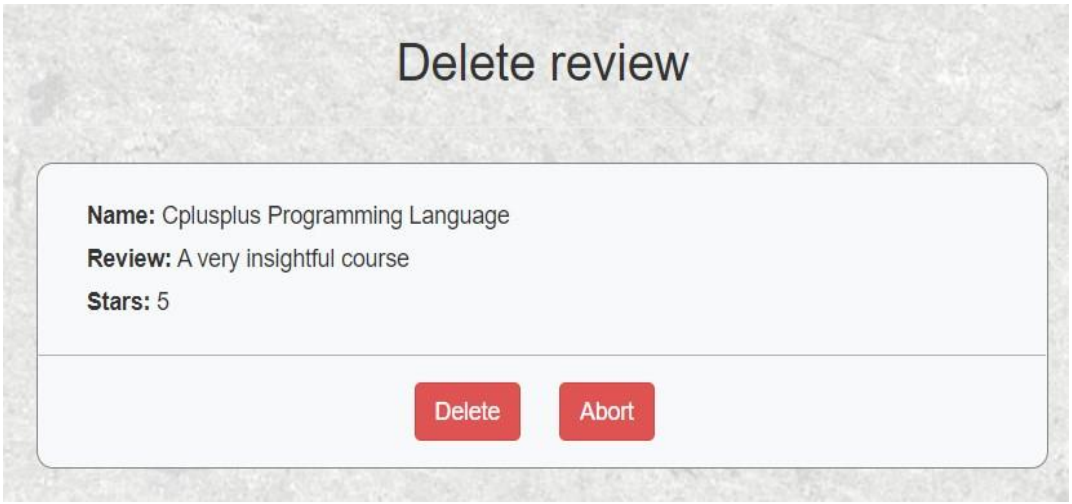

Figure 3.39 - Delete review form

The next important feature available on the landing page is the Enroll option. By choosing this option, the window presented in Figure 3.40 will open. On this page, the user can enroll in a course by choosing the option Enroll, or change his mind through the Abort option.

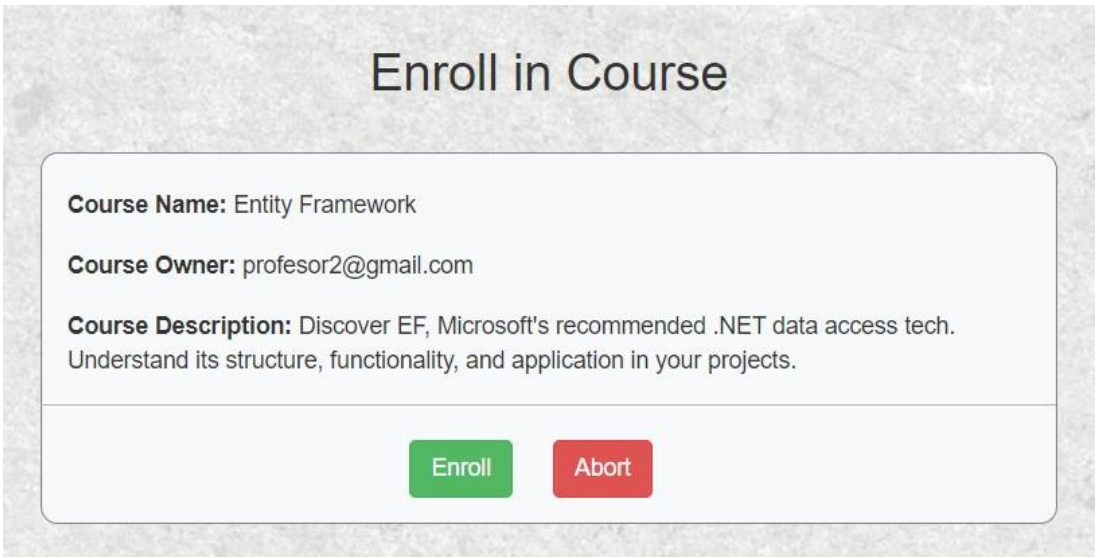

Figure 3.40 - Enroll in course window

Once the user is enrolled in a course, he/she can access the course from the next submenu, *MyCourses*. In this submenu, all the courses in which the user is enrolled and are not completed will be displayed. As shown in Figure 3.41, the user has access to several features from this submenu. For a particular course, the user can choose to cancel his enrollment, if he/she determines the information is not helpful. This can be

achieved through the option UnEnroll. Opting to unenroll from a course, will automatically result in the loss of progress the user has made in that particular tutorial.

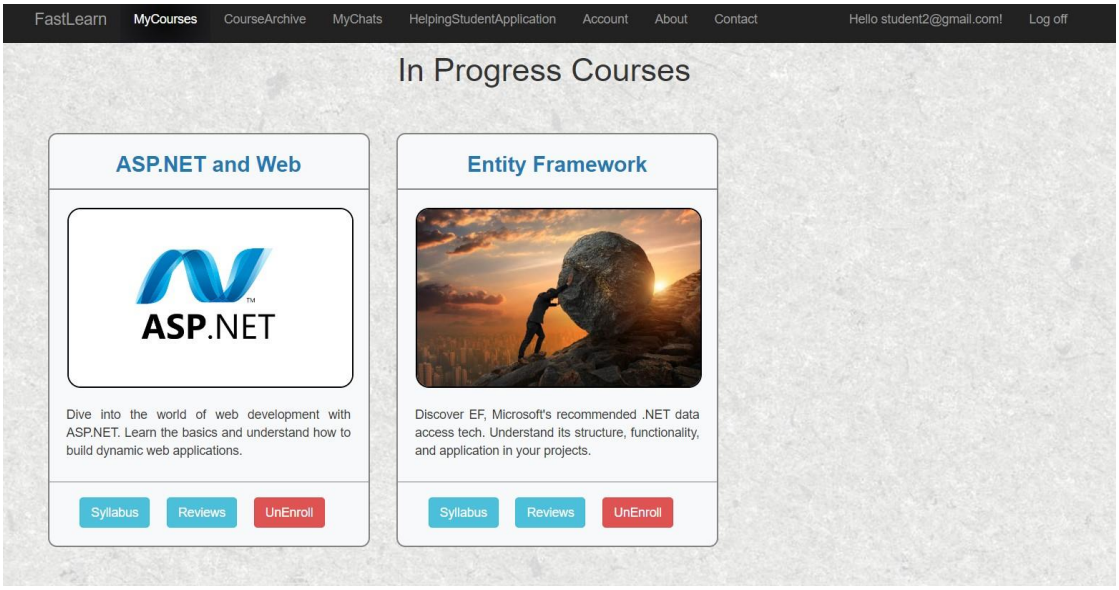

Figure 3.41 - Enrolled and not completed course page

As in the previous submenu, the user can view the syllabus and the reviews of the course. Additionally, on the *Syllabus* page, the user can go view the content of the subchapters through the >>More button.

Figure 3.42 - Syllabus page additional option

Figure 3.42 shows the previously mentioned option. By expanding the chapter list, for each subchapter listed, the user can view the content of the subchapter by clicking on the >>More button.

Figure 3.43 shows the newly opened page. In this window, the content for a specific subchapter is displayed. Several options are available to the user that explores the course. The first one is to view the additional materials provided by the *Professor* to facilitate the understanding of the lesson by selecting the See Files option. The second option allows the user to view the next subchapter by selecting the >>SubChapter button. The third option is to go back to the list of courses, through the option Courses.

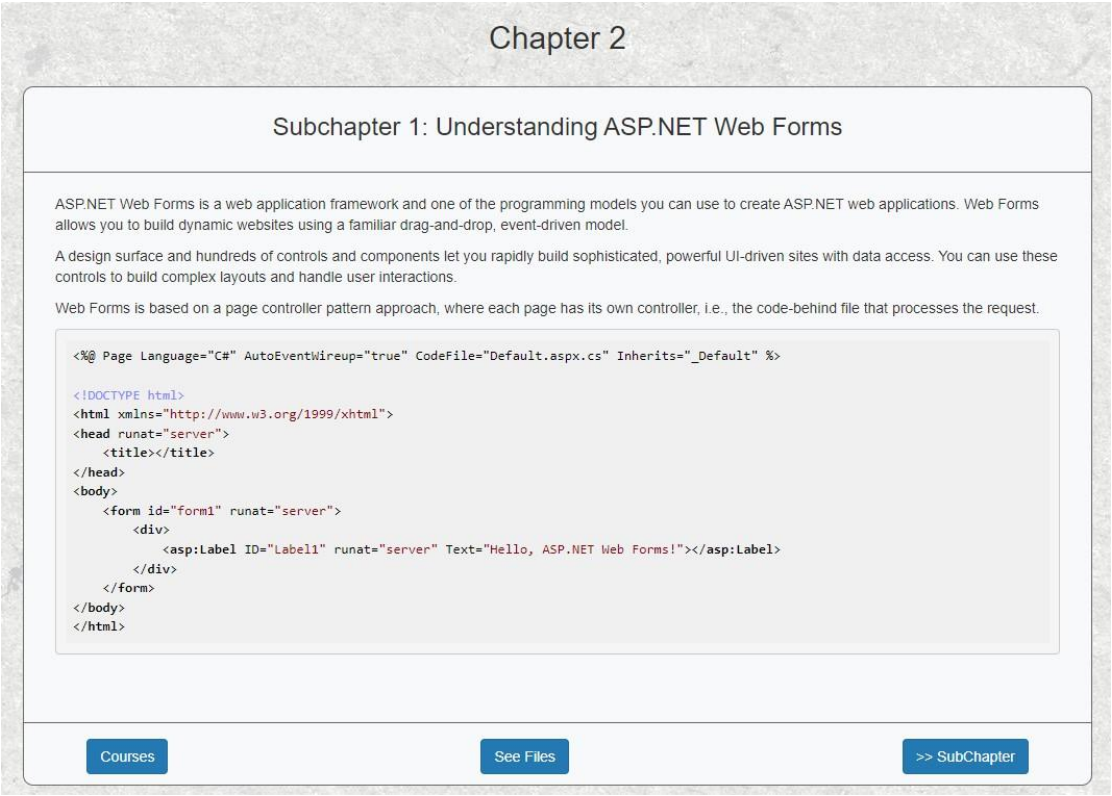

Figure 3.43 - Subchapter window

The See Files option shows a list of all the materials the *Professor* has provided for the *Student*. The end-user has the option to any particular file and study it.

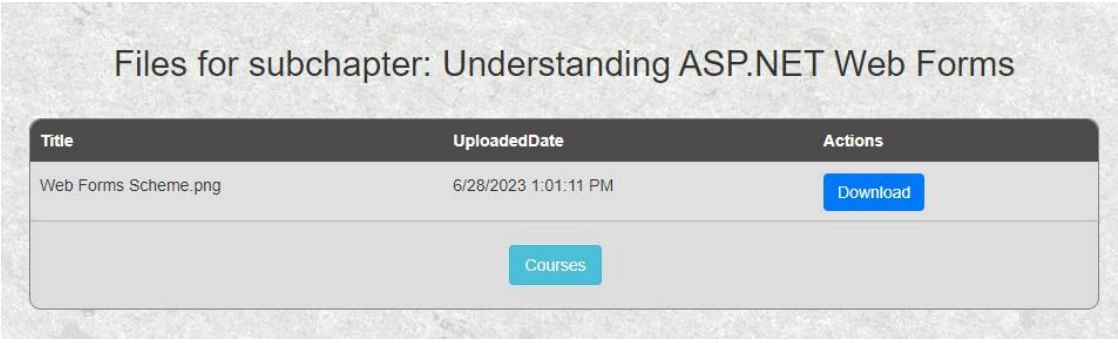

Figure 3.44 - Subchapter file list

The last and final option available in the *MyCourses* submenu is starting or resuming the learning process for a particular course. This option can be achieved by selecting the title of the course(highlighted in blue), as can be observed in Figure 3.41. This feature will redirect the user to the subchapter he is currently studying, opening a window similar to the one in Figure 3.43, with a few more features.

Figure 3.45 - Options of the subchapter content page, for the currently studied subchapter

The Courses, See Files, and >>SubChapter buttons have the same functionality as described above. The new features are Create a new Chat, and Start Test and can be observed in Figure 3.45. The Start Test feature opens a special test prepared by the Professor in which the user will test his knowledge. A sample test can be observed in Figure 3.46. Each question has a single correct answer.

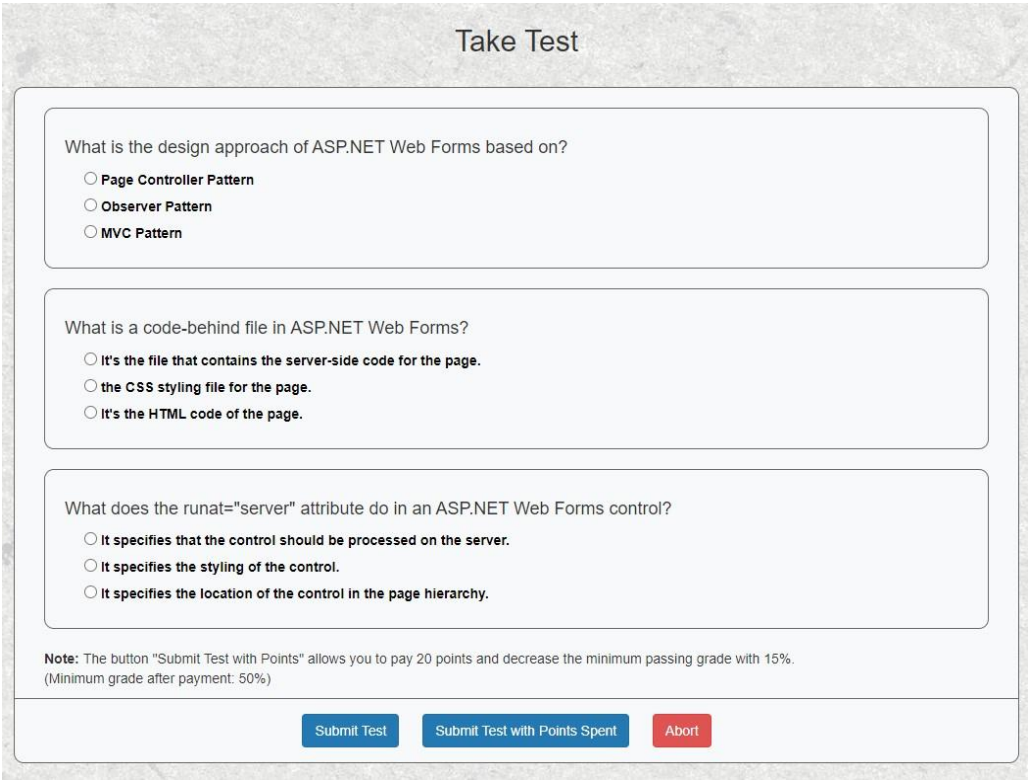

Figure 3.46 - Test page

The test can be submitted only when all the questions are answered. When the user considers he/she has finished the test, he/she has two methods to submit the test. The first option, Submit Test automatically grades the test and shows the results.

Figure 3.47 shows how the results of the test are displayed. In the upper part of the page, the score of the user, the minimum score for admission and the result of the test are displayed. Following this, for each question, the answer of the user and the correct answer are displayed. If the answer is correct, a green highlight is applied, else the question and answer will be highlighted in red.

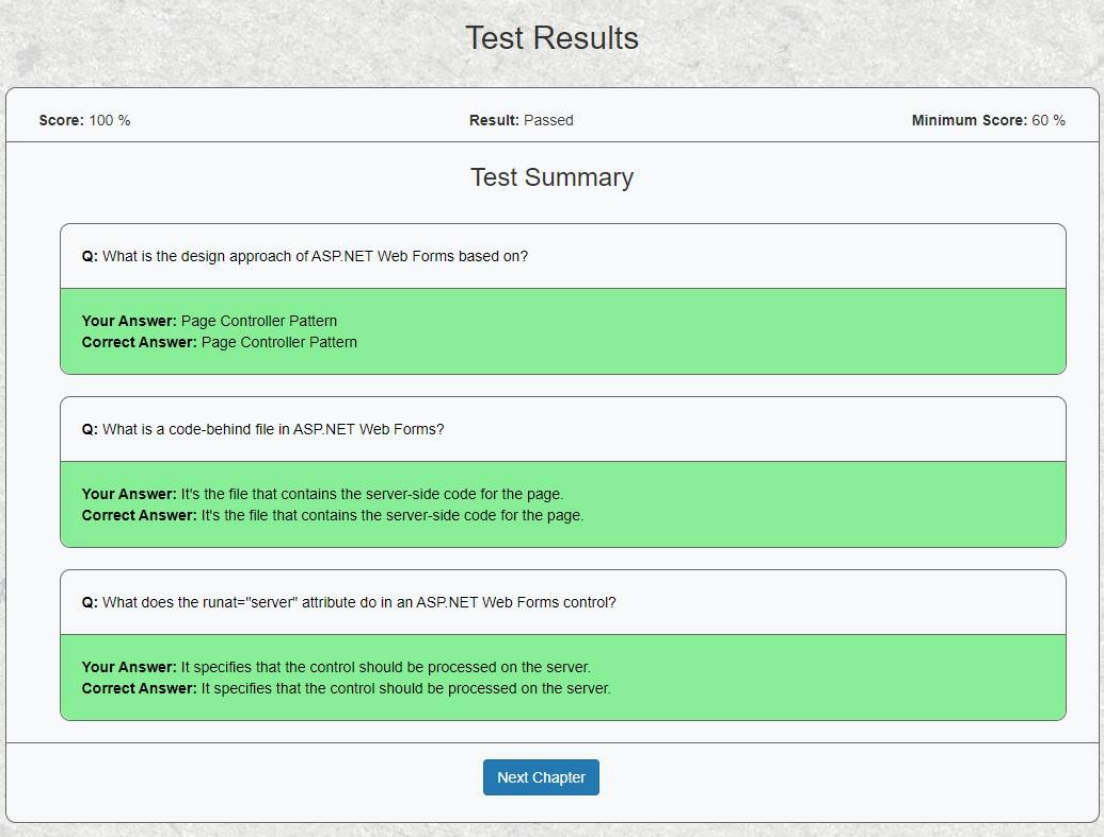

Figure 3.47 - Test result page

The second option to submit a test, Submit Test with Points Spent, allow the user to use the platform currency in order to unlock an easier version of the test. The test results are displayed in the above picture.

```
@foreach (var q in Model.QuestionResponses)
{
    var isCorrect = q.UserAnswer == q.CorrectAnswer;
    <div class="answer-container">
        <div style="margin: 20px;"><b>Q:</b> @q.Question.QuestionString</div>
        <div id="button-line-answer"></div>
        <div class=@(isCorrect ? "correct-answer" : "incorrect-answer")>
            <div>
                <strong>Your Answer:</strong>
                @q.UserAnswer
            </div>

            <div>
                <strong>Correct Answer:</strong>
                @q.CorrectAnswer
            </div>

        </div>
    </div>
```

Figure 3.48 - Test results highlighting implementation

For each displayed answer, it is checked if the answer is correct. If the answer is correct, a special CSS class named *correct-answer* is assigned to the container of the answer, else the class *incorrect-answer* is assigned. Consequently, the custom CSS classes define the highlighting of a particular answer.

Once all the subchapters and afferent tests are solved, the *Student* will promote the course. A promoted course will be moved to the *CourseArchive* submenu, where the end-user can see a list of all the courses he/she finished. From this submenu, the course content of the course can be revised through the Syllabus option, and leave a review through the Reviews button.

In Figure 3.45 one last option remains to be explained. The Create a New Chat option allows the *Student* to request the help of a *HelpingStudent*, if he/she has a difficult time understanding the current lecture.

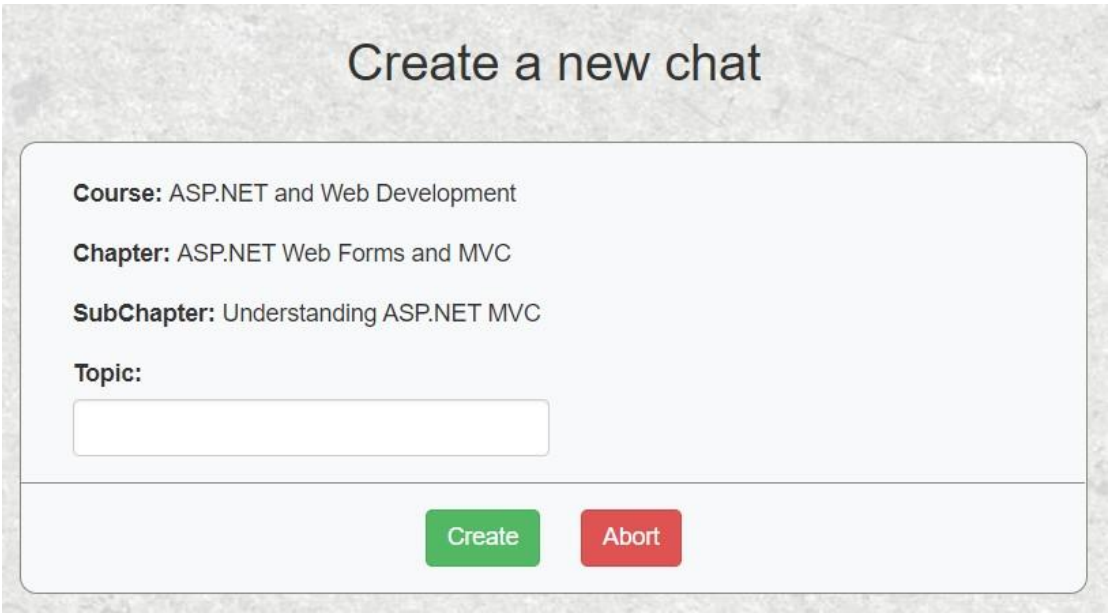

Figure 3.49 - Create a new chat page

Figure 3.49 presents the form allowing the *Student* to make a help request. The user must provide a short description of the topic he does not understand. Once the chat request is created, it will be visible in the *MyChats* submenu.

The *MyChats* submenu contains the list of all chats the *Student* has ever created. The chats are separated into two categories, Active Chats and Archived Chats. Irrespective of the classification, each chat contains information about the topic of the chat, the course, chapter and subchapter that it addresses, and the users involved in it.

The first category, Active Chats, contains all the chats that still need the attention of a *HelpingStudent*. For each chat in this section, the student can open the chat window to start working on solving his misunderstandings.

The second category, Archived Chats, contains all the chats that served their purpose. The chat history of each chat can be reviewed, if the *Student* ever needs to do so.

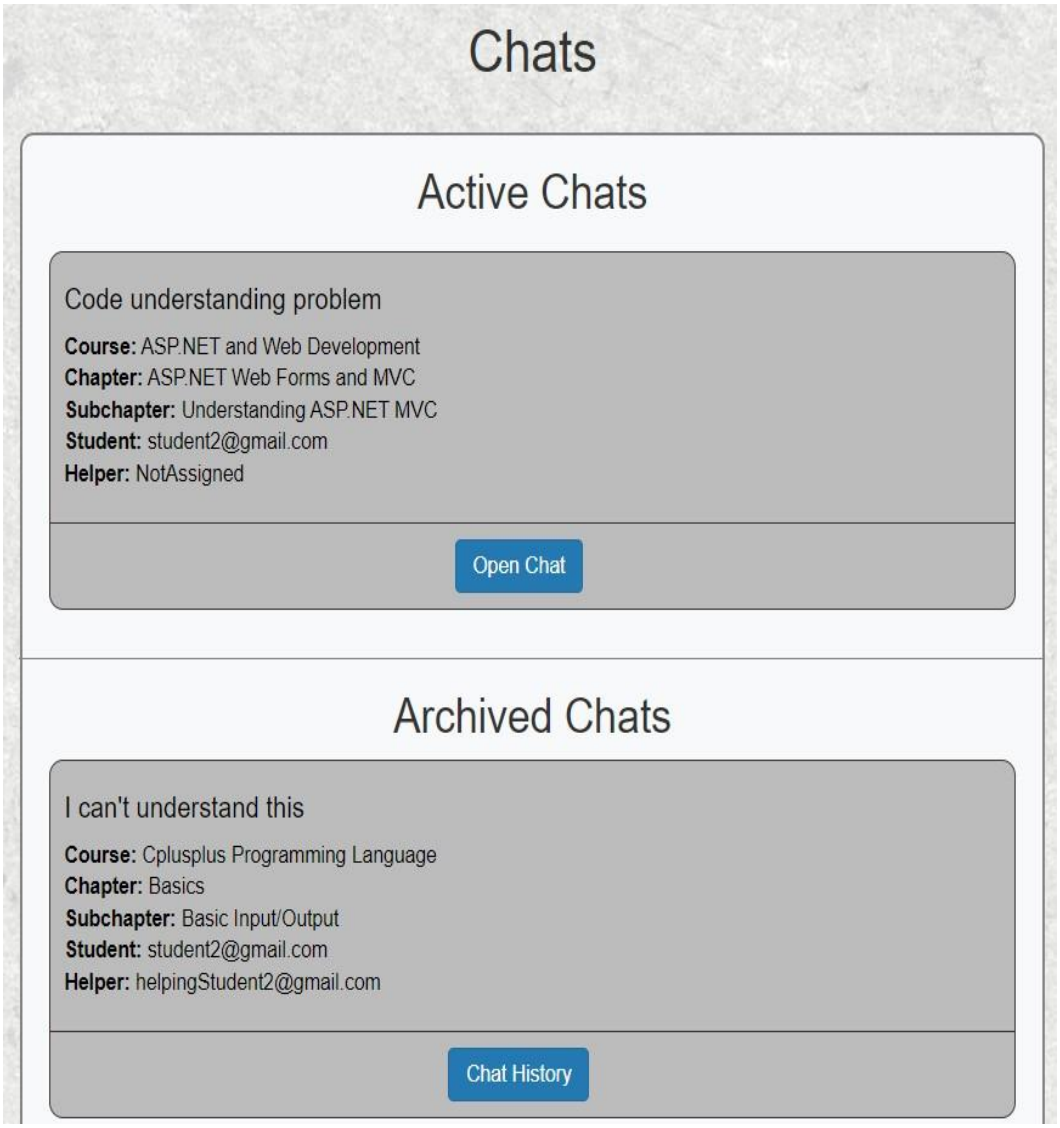

Figure 3.50 - MyChats submenu

The Open Chat option, available for active chats, will open the chat window. The newly opened chat window can be observed in Figure 3.51. The *Student* can start describing his problem in the chat, even if no *HelpingStudent* has yet joined the chat. This can be achieved by writing the desired message in the *Type Message* section and pressing the Send button. Once a message is sent, it will appear in the chat window. On the opposite side of the chat the messages from the *HelpingStudent* will appear, with a different color. For each particular message, the time it was sent is displayed. Additionally, each message can be edited or deleted if the user so desires.

Once the *Student* responds to the new messages, he/she can return to the *MyChat* submenu through the *Chats* option. The last option available on this page is declaring the issue for which the chat was created solved. The *IssueSolved* button serves this purpose. Once a chat is declared closed, it will appear in the Archived Chats section of the *MyChats* submenu. Additionally, the *HelpingStudent* will receive twenty Points, the platform currency. The Points can be used to unlock special features on the website.

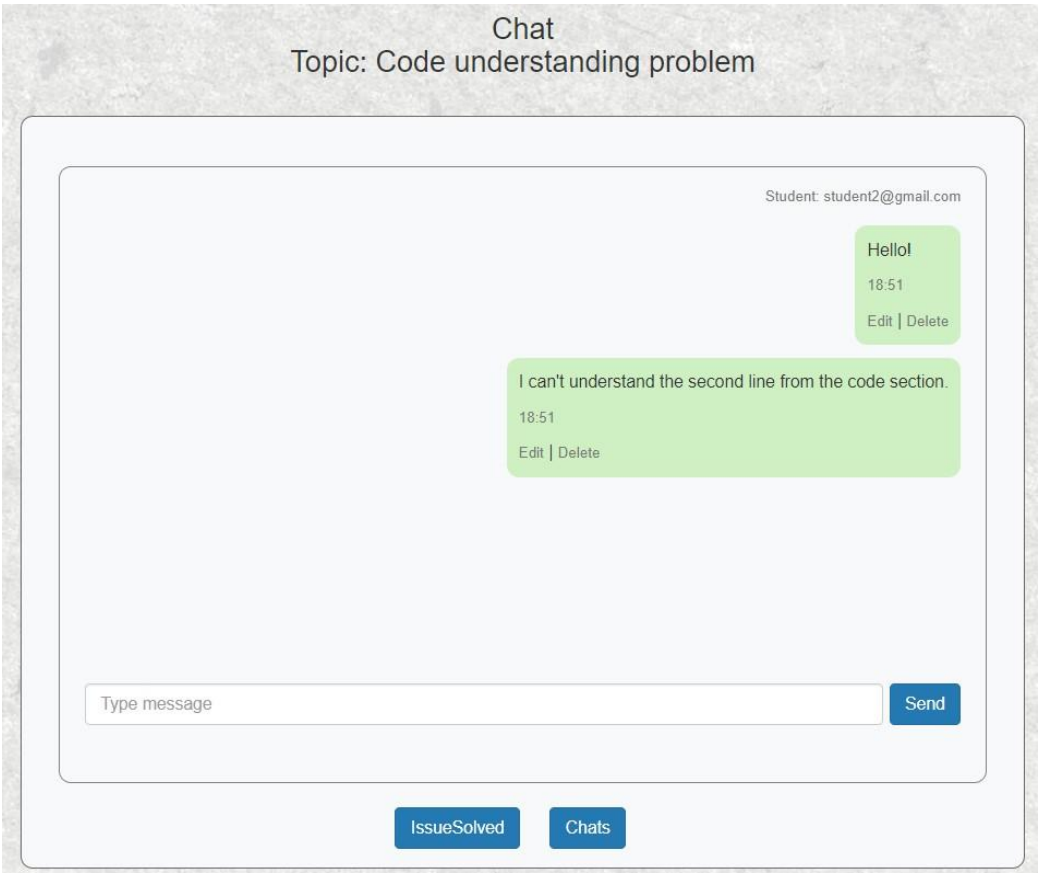

Figure 3.51 - One-to-one chat window

The next feature presented is located under the *HelpingStudentApplication* submenu. On this page, the *Student* can express his interest in helping his fellow users by joining the *HelpingStudent* community.

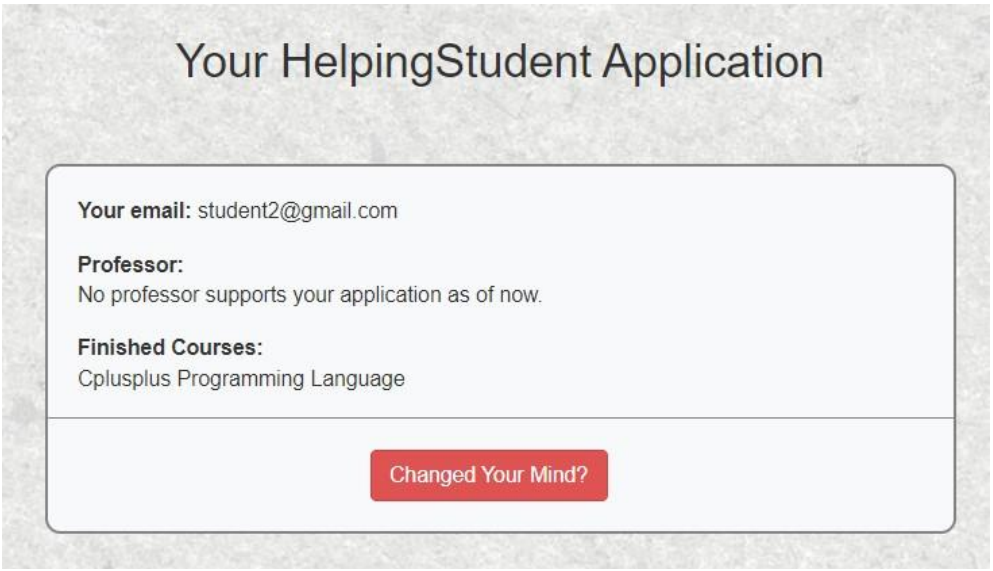

Figure 3.52 - HelpingStudent application page

Figure 3.52 showcases the previously mentioned feature. The page contains information about the email of the user, and a list of all the courses the *Student* has finished up to the moment of the application.

In the particular case of Figure 3.52, the *Student* just registered his promotion request. The next step is for a *Professor* to evaluate the application and recommend the user for promotion. At any point, the promotion request can be canceled by selecting the *Changed Your Mind?* option available at the bottom of the page.

Next, the storage mechanism of the finished courses will be presented. A list of all the IDs of the courses in question is created. The created list is serialized and saved in the database in a string field, under the *HelpingStudentApplication* table. The code performing the serialization can be observed in Figure 3.53. The foreach loop iterates through all the courses the user is enrolled in and verifies if he/she completed the course. In the end, the list is serialized and then saved in the database.

```
    foreach (var enrollment in enrolledStudentInCourses)
    {

        if (enrollment.CompletedCourse)
        {
            finishedCoursesId.Add(enrollment.CourseId);
        }
    }
    var finishedCourses = SerializeFinishedCourses(finishedCoursesId);
    return finishedCourses;
}
```

Figure 3.53 - Finished course serialization

When the list of finished courses is needed, the list of IDs is deserialized. For each particular id, the respective course is found and sent to the view. This approach is meant to provide a more memory-efficient way to store the information, and avoid adding a new relationship to the already complex existing database.

The presentation continues with features common to all logged-in users and will be presented only once. The features presented from now on until the end of this subsection are available for *Student*, *HelpingStudent*, *Professor*, and *Admin*.

The functionalities are found under the *Account* submenu, illustrated in Figure 3.54. On this page, the user can view particular data about the state of his/her account. The email, number of Points, the user-role are mentioned, and the Username are mentioned.

The *Student* can perform two main actions regarding the information presented. He/she can change the *Username* associated with the account through the *Edit Username* option, or change the password of the account by selecting the Change Your Password button.

It is important to mention that a newly created account will have no Username associated. Not having an Username set-up does not impact the functionalities of the platform, as it serves as a customization element.

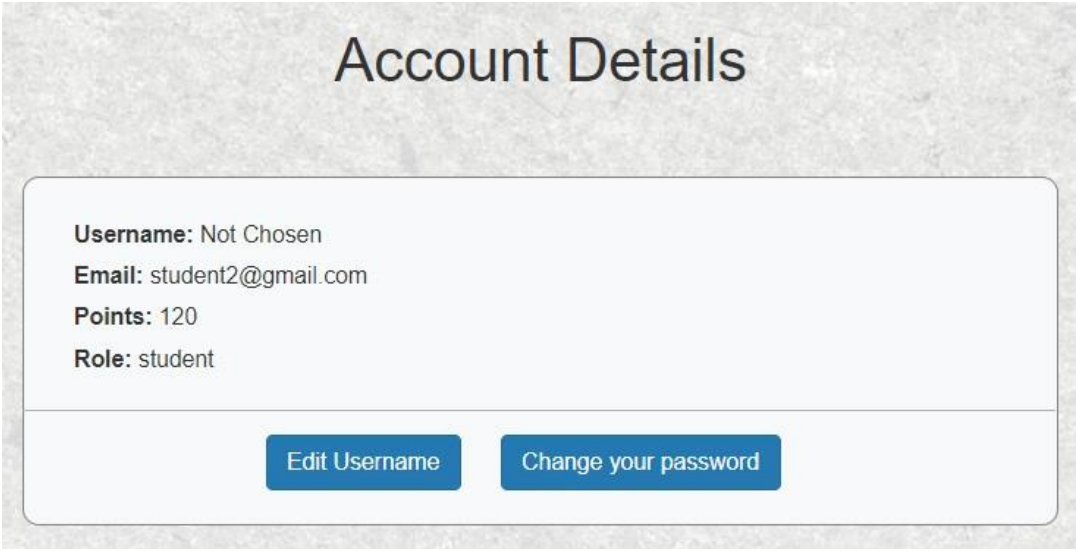

Figure 3.54 - Account submenu

The Edit Username option will redirect the user to the page presented in Figure 3.55. A short summary of the account is provided, and the user can choose a new username in the editable text field. At any point, the operation can be canceled by choosing the Abort option.

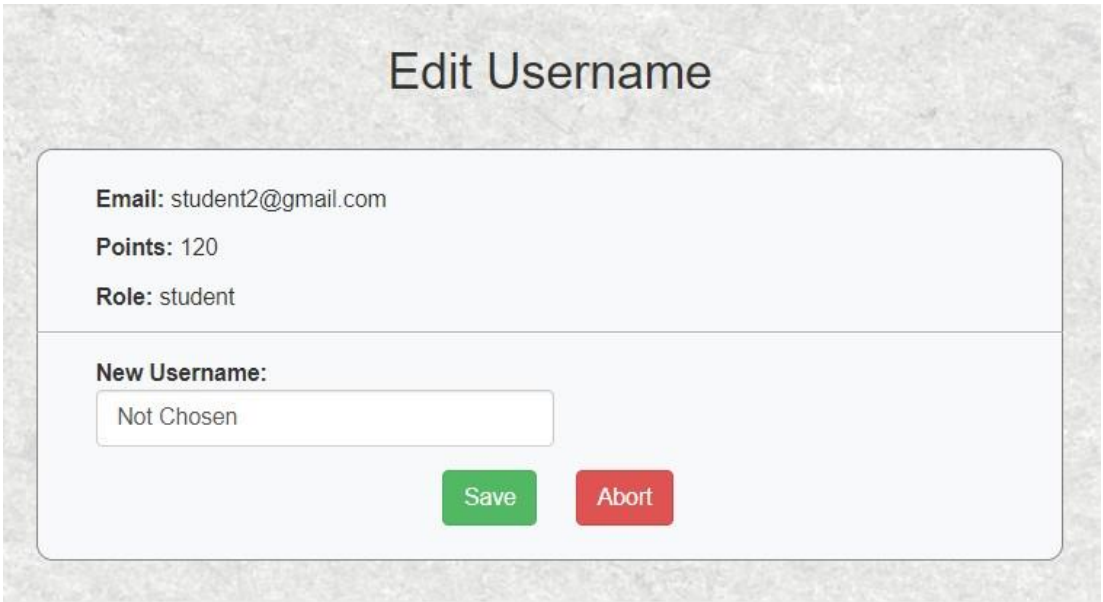

Figure 3.55 - Edit Username window

The remaining option present in Figure 3.54 has a similar design to the *Edit Username* page. In the *Change Password* window, the user is asked to provide the current password of the account for security reasons. Next, the user can choose a new password, confirm it and proceed with the changes.

3.3.3.3 HelpingStudent module

An important starting mention for the presentation of this module is that the facilities described in section 3.3.3.2 are also available to this user type. The reason for

this is the fact that the *HelpingStudent* represents a *Student* with more experience, that has the desire to help his colleagues and is also rewarded for doing so.

To enable the user to do his job, two additional submenus are provided, *ChatRequests* and *HelpStudents*.

The first submenu, *ChatRequests*, showcases a list of all the active chat requests. The chats listed on this page have no *HelpingStudent* assigned. Figure 3.56 presents this window. For each chat request, the user has two options. The first one is to open the conversation through the Open Chat option and see potential messages left by the *Student*. The second option is to join the chat and help the *Student* through the Help Student button.

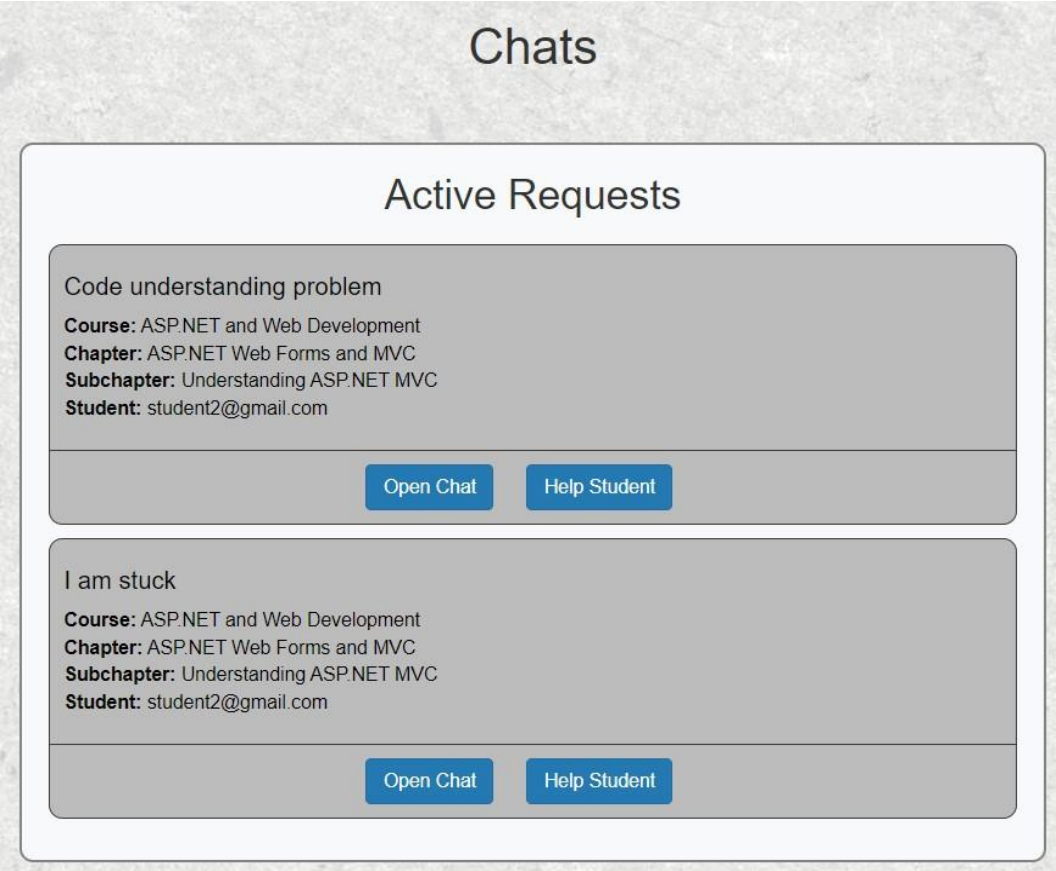

Figure 3.56 - ChatRequests window

Once the user joins a chat, the conversation will be shown under the submenu *HelpStudents*. The second submenu is similar to the *MyChats* submenu, presented in Figure 3.50. The essential distinction lies in the kind of chats presented within each submenu.

In the *HelpStudents* submenu, the discussions presented are those in which the current user acts as a helper, whereas in the *MyChats* submenu, the dialogues are designed to assist the account holder, that is, the *Student*. The last distinction is that for each active chat, the *HelpingStudent* can leave the chat if he/she so desires.

3.3.3.4 *Professor module*

The main objective of this module is to provide reliable information for the *Students* to learn from. Multiple features are at the disposal of this user to make this possible.

The analysis will start from the landing page associated with this user type, showcased in Figure 3.57.

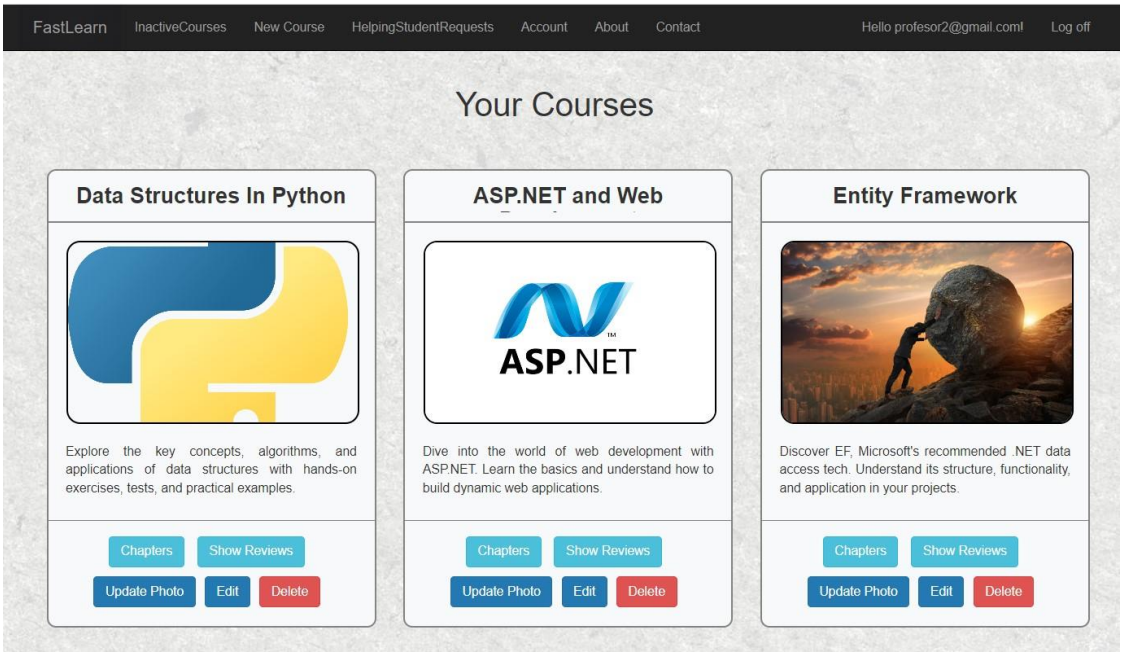

Figure 3.57 - Professor module landing page

On this page, all the approved courses owned by the account holder are listed. For each particular course, five main options are available: Chapters, Show Reviews, Update Photo, Edit and Delete.

The Show Reviews option lists the reviews left by the *Students* for a particular course, and was presented in section 3.3.3.2.

The Edit button redirects the user to the window presented in Figure 3.58. On this page, the course owner can update the name of the course and the Course Description. Several restrictions must be respected when changing the information of the course. For instance, the Course Description must not be empty, and the text length must be between one hundred and twenty and one hundred and fifty characters. If these conditions are not respected, an error message will be displayed. On the other hand, if the restrictions are respected, the user can change the course information through the Save Changes button. At any point, the user can cancel this procedure by selecting the Abort option.

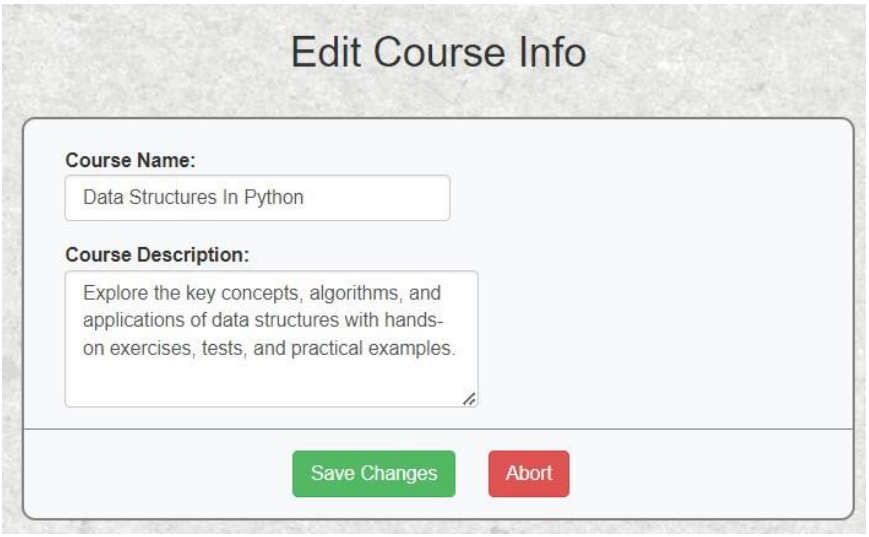

Figure 3.58 - Edit course window.

The implementation of the above-mentioned restrictions can be observed in Figure 3.59. In the controller, the conditional statements check for the validity of the input. In case an error is detected, an error message is attached to the Course Description field through the command presented in line 125.

```
117             if (course.CourseDescription.IsEmpty())
118             {
119                 ModelState.AddModelError("CourseDescription", "Can't be empty");
120             }
121             else
122             {
123                 if(course.CourseDescription.Length < 120 || course.CourseDescription.Length > 150)
124                 {
125                     ModelState.AddModelError("CourseDescription", "The description of the course must be between 120 and 150 characters long.");
126                 }
127             }
128             if (ModelState.IsValid)
129             {
130                 courseDb.UpdateCourse(course);
131                 return RedirectToAction("GetUserCourses", new { id = course.CourseId });
132             }
133             return View(course);
134         }
```

Figure 3.59 - Edit course restriction implementation

Once the error is detected, the model with the newly added error is returned to the view and displayed. The new model is used by the view, and the instruction presented in Figure 3.60 at line 88 displays the error.

```
84
85              <div class="form-group" style="margin-left: 0px; margin-right: 0px;">
86                  <b>Course Description: </b>
87                  @Html.EditorFor(model => model.CourseDescription, new { htmlAttributes = new { @class = "form-control text-area" } })
88                  @Html.ValidationMessageFor(model => model.CourseDescription, "", new { @class = "text-danger" })
89              </div>
```

Figure 3.60 - Edit view code used to display the new errors.

The next presented option, *Delete*, will open a page containing a single course card, like the ones in Figure 3.58. Instead of the five buttons present in the previously mentioned figure, two options are available. The *Delete* and *Abort* buttons. The first button will remove all the existing data regarding the course, while the second one allows the *Professor* to cancel the deletion process.

The fourth feature present in the current submenu is changing the image of the course through the Update Photo button.

Figure 3.61 - Update course photo page

In the newly opened window, a preview of how the course looks at the moment is presented. If the user so desires, the new image is uploaded. The first step is to open the Choose File navigation window and navigate to the location of the new image. To save the changes, the Upload Image option must be selected.

The implementation of the above-mentioned features is presented in Figure 3.62.

```
[HttpPost]
0 references
public ActionResult UpdateCoursePhoto(HttpPostedFileBase file, int courseId)
{
    var course = courseDb.GetCourse(courseId);
    if (file != null && file.ContentLength > 0)
    {
        string folderName = course.CourseName;
        string extension = Path.GetExtension(file.FileName);
        string fileName = "CourseImage" + extension;
        string dirPath = Path.Combine(Server.MapPath("~/Content/Images/Courses"), folderName);

        if (!Directory.Exists(dirPath))
        {
            Directory.CreateDirectory(dirPath);
        }

        string path = Path.Combine(dirPath, fileName);

        string urlPath = VirtualPathUtility.ToAbsolute(Path.Combine("~/Content/Images/Courses", folderName, fileName));
        course.ImagePath = urlPath;
        file.SaveAs(path);
        courseDb.UpdateCourse(course);
    }

    return RedirectToAction("GetUserCourses");
}
```

Figure 3.62 - Upload image controller

Figure 3.62 presents the logic behind saving and updating the main image of a course. In simple terms, the file is locally saved on the server in a dedicated folder named after the course. The path is then saved in the database. To display the course image, the instruction is showcased in Figure 3.63.

```
<img src="@Url.Content(Model.ImagePath)" class="card-img-top" alt="Course Image" width="440" height="260">
```

Figure 3.63 - Instruction used to display the course image

The last button available in Figure 3.57, *Chapters*, opens a window displaying a list of all the chapters of a certain course. The newly opened page is shown in Figure 3.64.

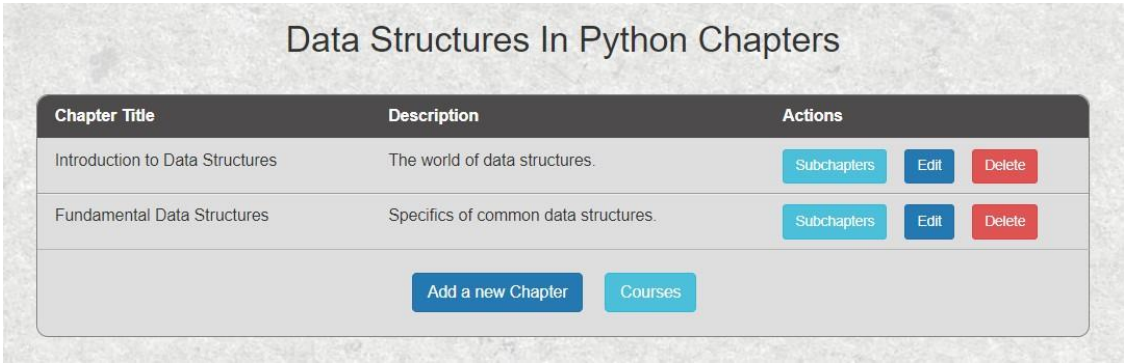

Figure 3.64 - Chapter list page

The user has a few options. For each chapter, the user can edit, delete, or view the subchapters afferent to the respective chapter. Additionally, a new chapter can be added.

The first feature that will be presented is editing a chapter. Selecting the *Edit* button opens the page presented in Figure 3.65.

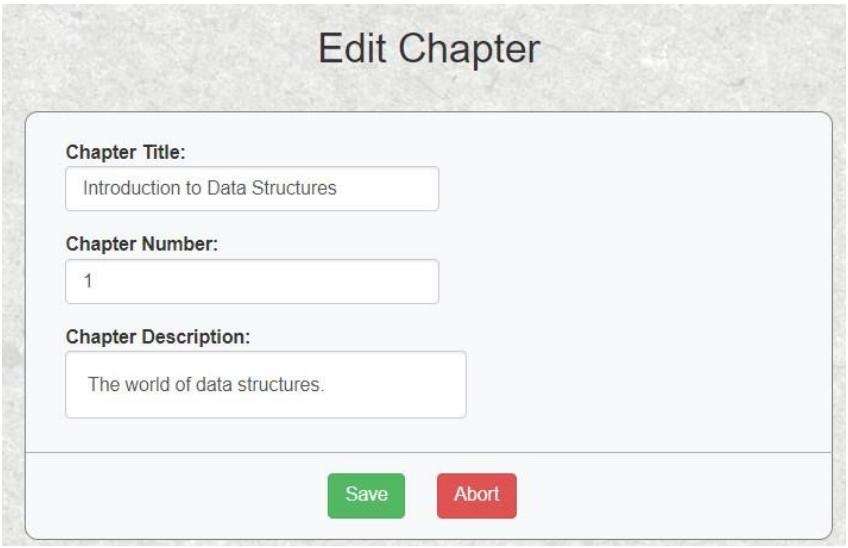

Figure 3.65 - Edit chapter page

On the Edit Chapter page, the user can edit the chapter title and the chapter description. Additionally, the order in which the chapter is displayed when going through the course can be updated via the Chapter Number field.

The next option available is the Delete button. Upon selecting this feature, the form presented in Figure 3.66 will open.

Figure 3.66 - Delete chapter form

The Add a new Chapter option present in Figure 3.64 allows the course owner to create a chapter. Opting for this feature will open the form presented in Figure 3.67.

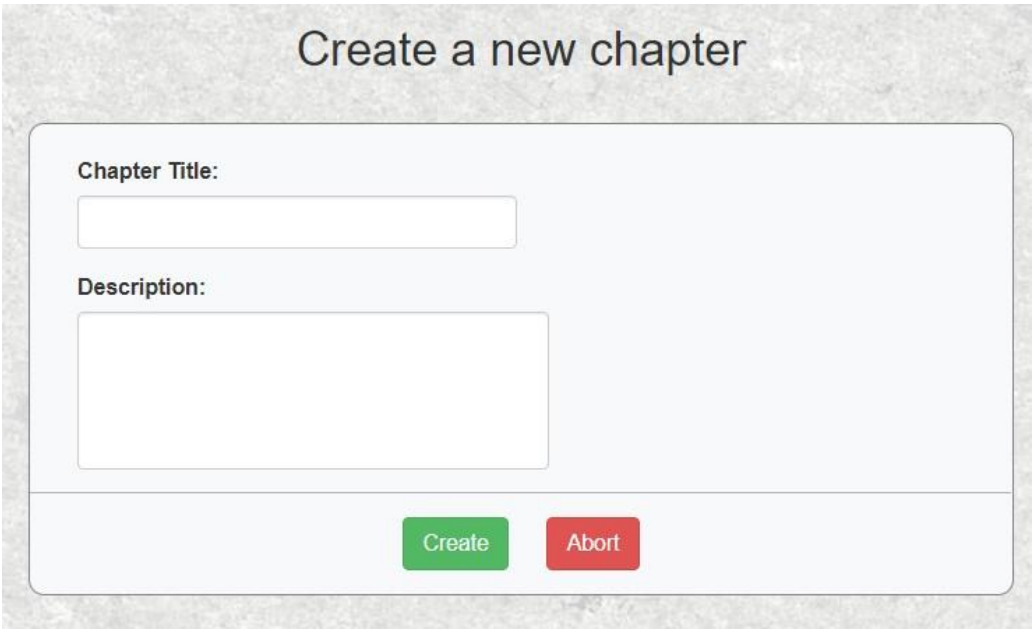

Figure 3.67 - Create a new chapter page

In the following paragraphs, features similar to the add, edit, and delete pages will be presented. Only special elements will be showcased and explained regarding these three operations.

The SubChapters button present in Figure 3.64 opens the page presented in Figure 3.68.

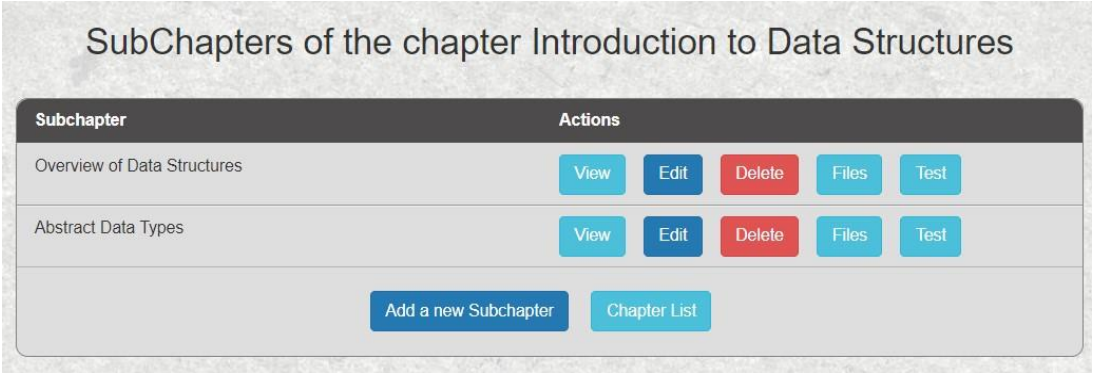

Figure 3.68 - Subchapter list page

The Delete option is identical to the correspondent feature from the chapter page and thus will not be further elaborated on.

The Add a New Subchapter button will open the page presented in Figure 3.69.

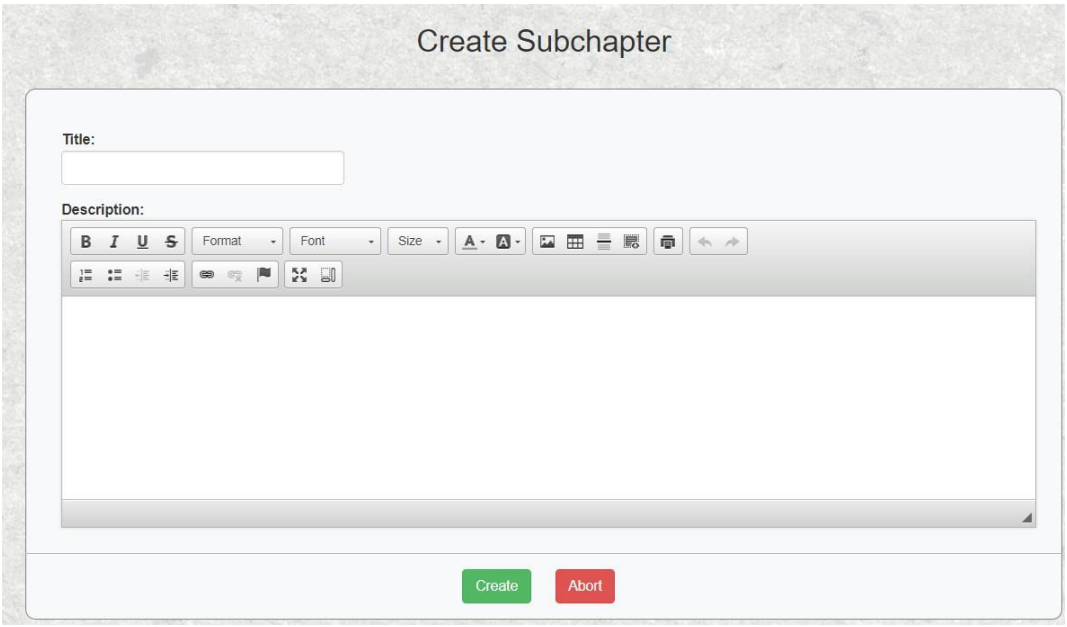

Figure 3.69 - Create subchapter page

From the above figure, the Description section stands apart from the other edit pages. The Description field represents the space where the *Professor* will add the content of the subchapter. To enable this feature, a reliable and robust test editor is provided. In the text editor, the *Professor* can use a wide set of tools, such as bolding the text, adding a table, or a code snippet. For the comfort of the user, the text editor can be expanded on the entire screen to enhance the content creation experience.

To implement the presented text editor, several steps are needed. The first step is to install the CKEditor package using the NuGet Package Manager. The second step is to include the code snippets presented in Figure 3.70, and Figure 3.71 must be included in the source code of the Create a New Chapter page.

```
@section scripts{
    <script src="~/Scripts/ckeditor/ckeditor.js"></script>
    <script>
        CKEDITOR.replace('SubchapterDescription', {
            extraPlugins: 'codesnippet',
            codeSnippet_theme: 'default',
            toolbar: [
                { name: 'basicstyles', items: ['Bold', 'Italic', 'Underline', 'Strike'] },
                { name: 'styles', items: ['Format', 'Font', 'FontSize'] },
                { name: 'colors', items: ['TextColor', 'BGColor'] },
                { name: 'insert', items: ['Image', 'Table', 'HorizontalRule', 'CodeSnippet'] },
                { name: 'document', items: ['Print'] },
                { name: 'clipboard', items: ['Undo', 'Redo'] },
                '/',
                { name: 'paragraph', items: ['NumberedList', 'BulletedList', 'Outdent', 'Indent'] },
                { name: 'links', items: ['Link', 'Unlink', 'Anchor'] },
                { name: 'tools', items: ['Maximize', 'ShowBlocks'] }
            ],
        });
    </script>
}
```

Figure 3.70 - CKEditor configuration script

Figure 3.70 presents the configuration script of the text editor. A high degree of flexibility is provided to the developer, as adding new toolbar items will expand the functionalities available to the user.

```
<div class="form-group" style="padding-bottom: 0px;">
    <b>Description: </b>
    @Html.TextAreaFor(model => model.SubchapterDescription, new { htmlAttributes = new { @class = "ckeditor form-control" } })
    @Html.ValidationMessageFor(model => model.SubchapterDescription, "", new { @class = "text-danger" })
</div>
```

Figure 3.71 - Subchapter description instruction

It is mandatory to use the TextAreaFor instruction in order to have a functional text editor. Figure 3.71 showcases how the instruction was implemented.

A final but important mention regarding the text editor is that the SubchapterDescription field and the project must be configured to accept raw HTML code. To ensure the security of the application, custom security measures must be employed in the mentioned field.

The Edit option present in Figure 3.68 is highly similar to the Create a new subchapter page. To implement this option, the same steps used for the Create page must be used.

The *View* option allows the *Professor* to preview how the content of the subchapter will be displayed to the *Student*. This feature was showcased in section 3.3.3.2.

The Files option opens a list of all the materials available for that subchapter. On this page, the user can provide additional materials to help the *Student* understand the content of the View page. The *Professor* has the ability to add, download and remove any file he/she considers relevant.

The Test button present in Figure 3.68 allows the management of the test that corresponds to the current subchapter. The options are presented in Figure 3.72.

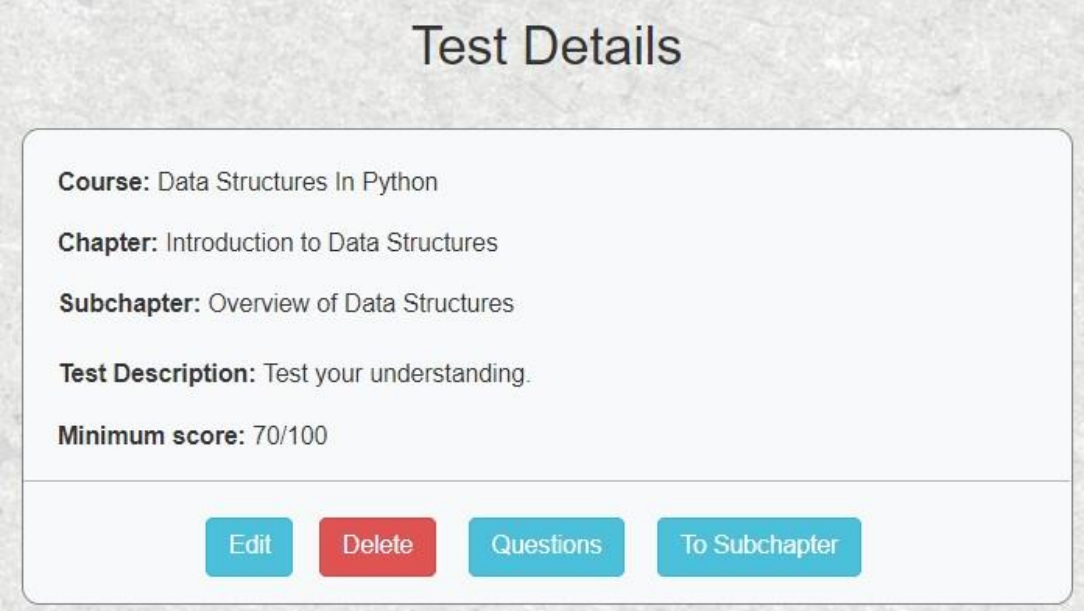

Figure 3.72 - Test page

The Edit and Delete are similar to the correspondent functionalities in the chapter page, thus will not be further explained. The Questions button will open a list of all the questions defined for a certain test. In this window, the user can add, edit, delete a question, and define the answers for each question. The Question is similar to the list page in Figure 3.64. For each question, the list of defined answers can be opened. Only one answer is permitted to be correct per question.

Next, the *New Course* submenu is presented in Figure 3.73. Several restrictions regarding the fields to be completed must be respected. The Course Name must be unique, and the Course Description must have between one hundred and twenty and one hundred and fifty characters. Not abiding to these rules will not allow the creation of the course.

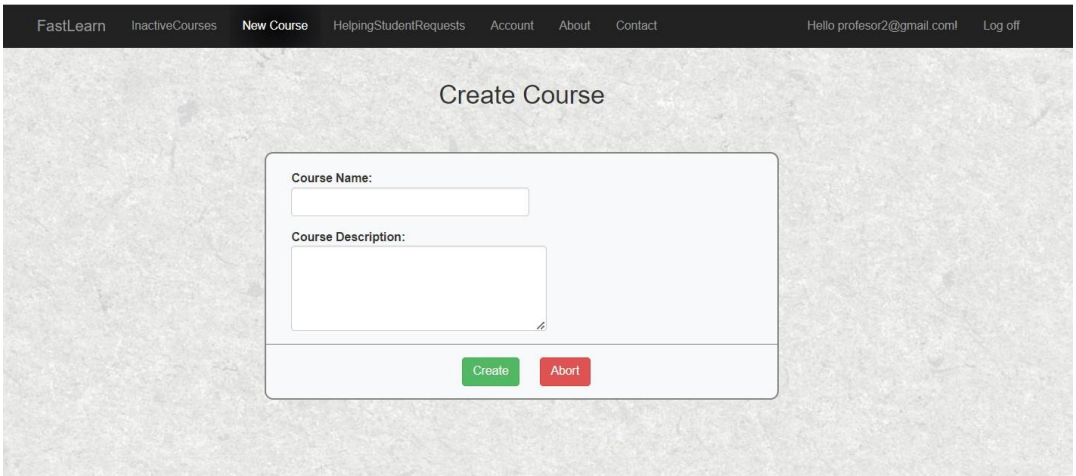

Figure 3.73 - Create course submenu

Once a course is created, the course needs to be approved by the admins of the platform. The course can be found under the *InactiveCourses* submenu. A view of the submenu can be observed in Figure 3.74.

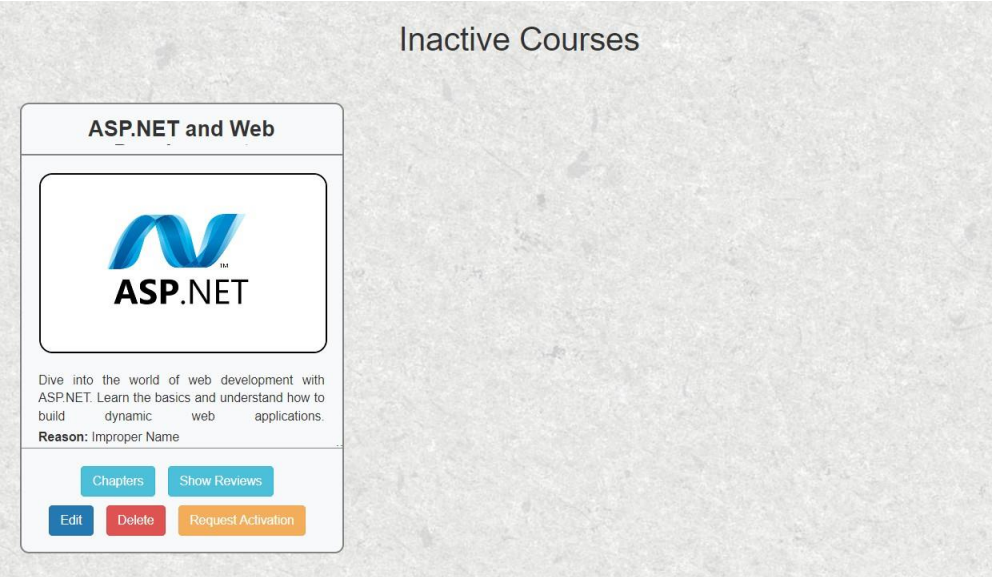

Figure 3.74 - InactiveCourses submenu

This *InactiveCourses* submenu hosts the courses that are not ready to be delivered to the end user, for one reason or another. On this page, the *Professor* can use the tools described for the landing page and solve the issue with the course. Once the issue is resolved, the *Professor* can make a review request. If the examining *Admin* decides the problem is solved, he/she will enlist the course on the platform.

The last feature this user has access to is located in the *HelpingStudentRequests* submenu. On this page, the Professor can choose a *HelpingStudent* application and review it. A view of the submenu is presented in Figure 3.75.

Figure 3.75 - HelpingStudentApplication submenu

The user can choose to assess the application by selecting the Support Application option. On the newly opened page, presented in Figure 3.76, the user is presented with a list of all the courses the *Student* completed until the application was made. By clicking on the name of a course, the *Syllabus* page for that specific course will be opened, allowing the *Professor* to examine the preparation of the *Student*. If the assessment is successful, the *Professor* can recommend the *Student* for promotion through the Support button.

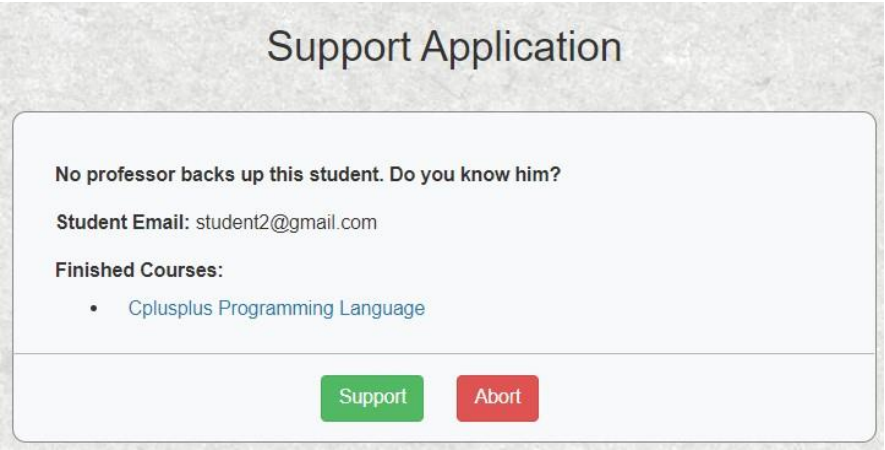

Figure 3.76 - Support Application page

3.3.3.5 Admin module

The last module of this application has the role of ensuring the smooth functioning of *FastLearn*. The presentation starts with the landing page of this module, showcased in Figure 3.77.

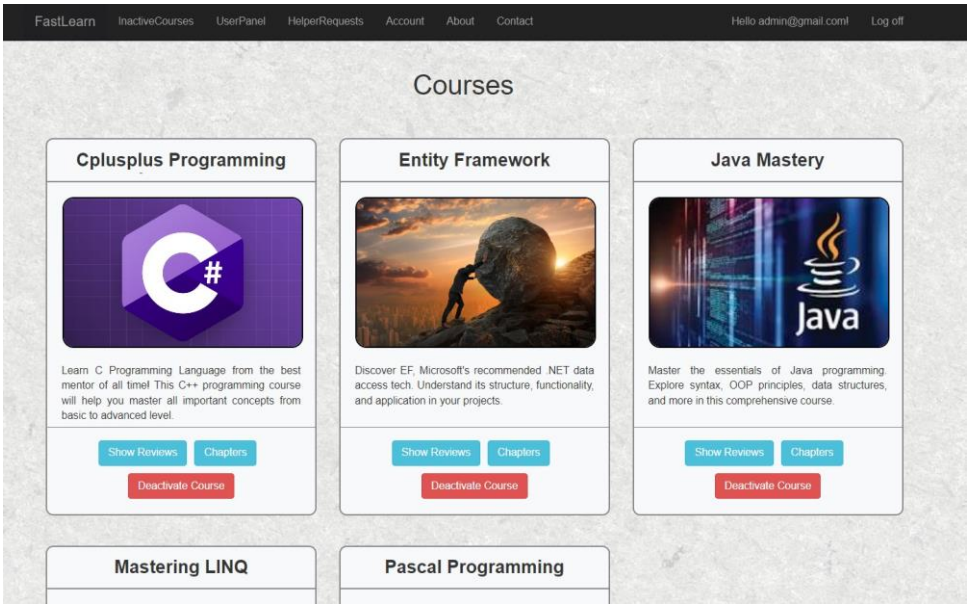

Figure 3.77 - Admin landing page

The default page designed for this user type contains a list of all the courses that are currently listed on the platform. The Chapters button enables a similar functionality as the analog button described for the *Professor* module. The main difference is that the *Admin* is cannot manage the content of a course, only to view it. Consequently, any option to alter the content of the course that is described in the flow described in section 3.3.3.4 is not present.

The Show Reviews page allows the user to view the reviews of a certain course. For each review, the administrator can opt to activate or deactivate it. Once an end-user leaves a review, it must be approved by the *Admin* in order to be displayed on the platform.

The last feature available in this submenu is to put a course under review through the Deactivate Course option. Selecting the option will open the window presented in Figure 3.78. The *Admin* must provide a reason for the course deactivation.

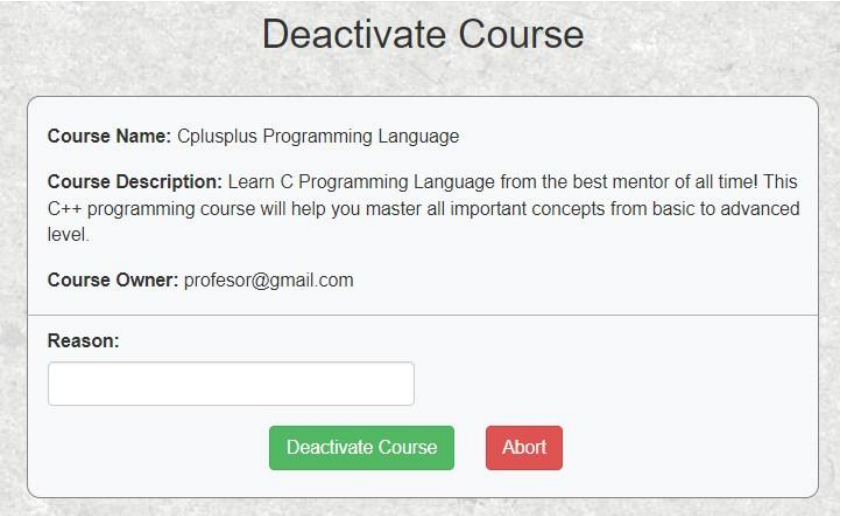

Figure 3.78 - Deactivate Course page

The next feature discussed is located under the *InactiveCourses* submenu. The submenu page is displayed in Figure 3.79.

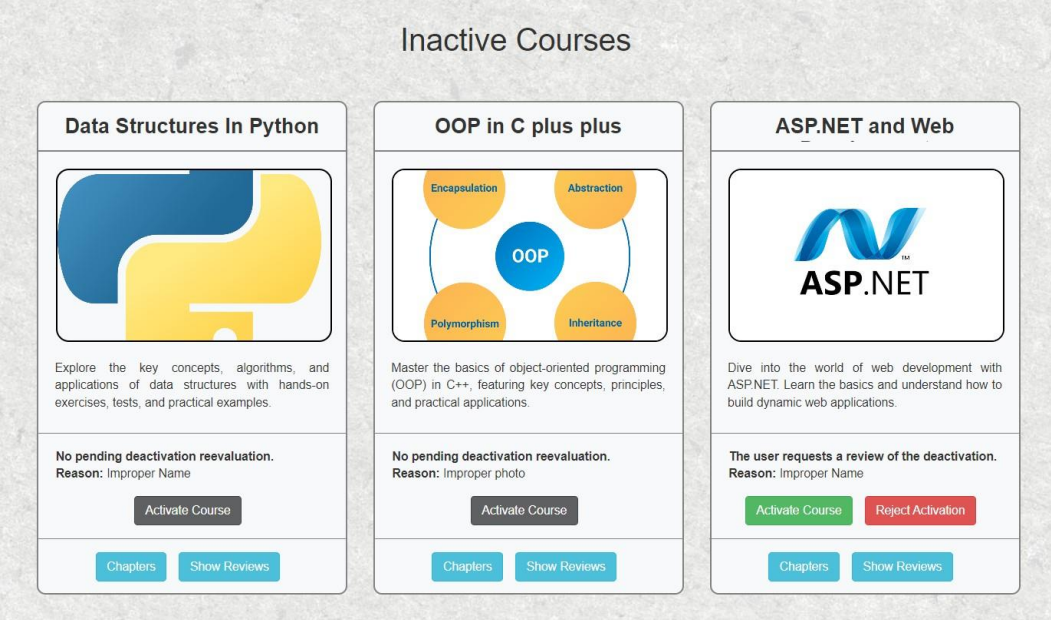

Figure 3.79 - Inactive course list for Admin

Among the options present in Figure 3.79, only the Activate Course and the Reject Activation options are not covered yet. The first option, Activate Course, opens a similar page to Figure 3.78, which allows the user to re-instantiate a course if the initial issue was solved. On the other hand, the Reject Activation option will reject the activation request sent by the *Professor* and will keep the course under review.

The next feature presented is the *HelperRequests* submenu. The newly opened page contains a list of all the active *HelpingStudent* applications, as can be observed in Figure 3.80. The *Admin* has two options, to Approve the request or to *Reject* the request through the Reject button. The *Admin* is the final examiner for a *HelpingStudent* application. The *Admin* must check the background of the *Student*, such as possible warnings, and determine if the user is suited for promotion.

Figure 3.80 - Helper Application page

The last submenu dedicated to this submodule is called UserPanel. Selecting the UserPanel toolbar button will open the page illustrated. This page contains a list of all the registered users in the platform.

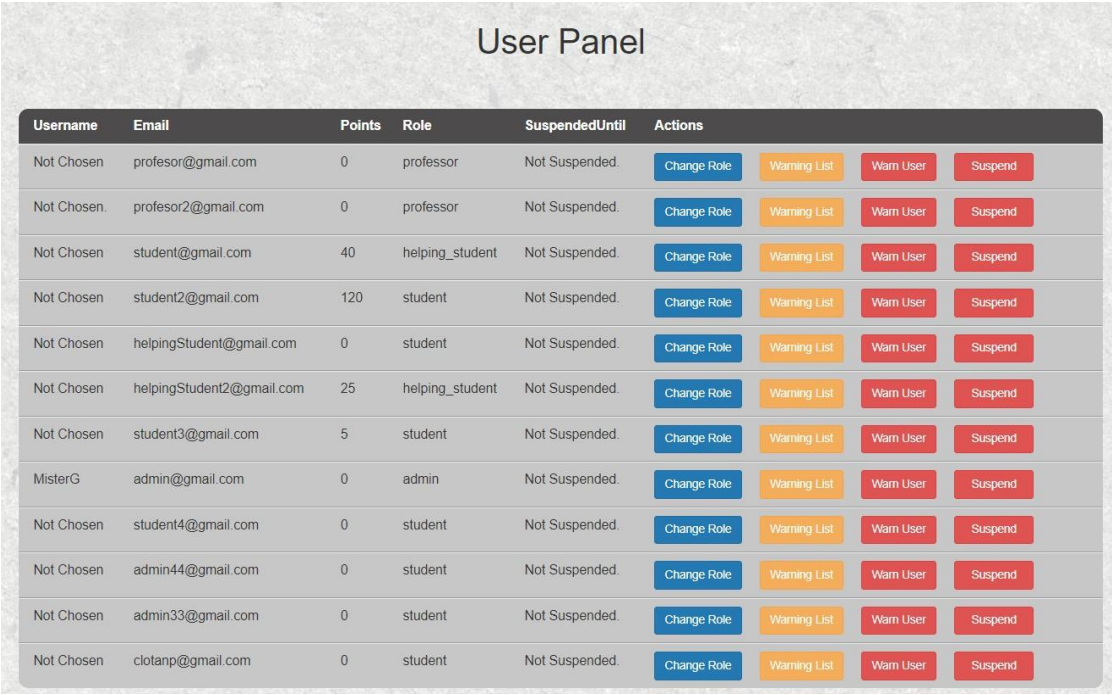

Figure 3.81 - UserPanel submenu

In the current submenu, a wide variety of features are at the disposal of the *Admin* to ensure *FastLearn* functions as intended. The *Admin* may receive different reports regarding users not using the platform as intended, through the address provided in the Contact submenu of the application. After assessing the report, the *Admin* can demote, warn or suspend the user in question if needed.

The Change Role option opens the page shown in Figure 3.82. This functionality can be used both to promote or demote a user, depending on the case.

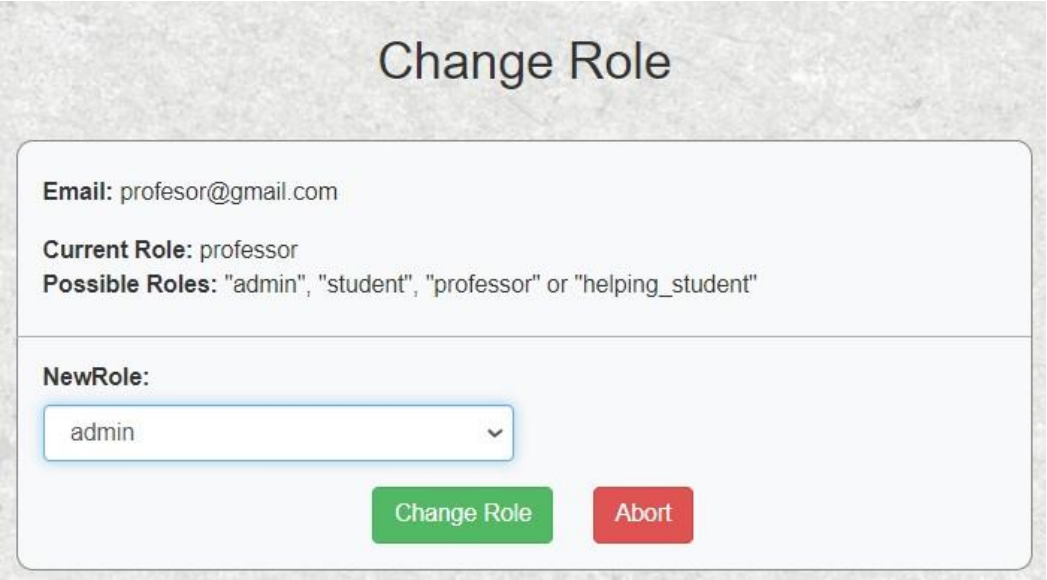

Figure 3.82 - Change Role page

The next presented option is called *Warn User*. This option represents the least severe disciplinary action the *Admin* can use. When a user accumulates three warnings, his account will be suspended for three days. Viewing the warning list of a particular user is possible through the Warning List button. By selecting this option, the page illustrated in Figure 3.83 opens. A list of the current warnings the user has will be displayed. The *Admin* can remove a certain warning through the Remove Warning option if deemed appropriate.

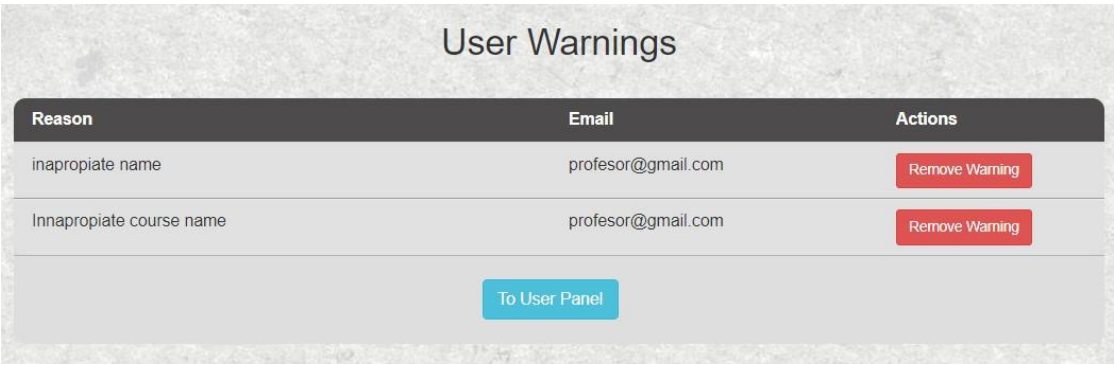

Figure 3.83 - User Warnings page

The last and final feature is the option to Suspend or lift the suspension of a certain user. To suspend a user, the Suspend option is available, and the page displayed in Figure 3.84 will open. A valid date for the suspension end date must be chosen. The only condition is for the date to be in the future.

If, at any point, it is concluded that the suspension of a user should be lifted, the *UnSuspend* option can be used.

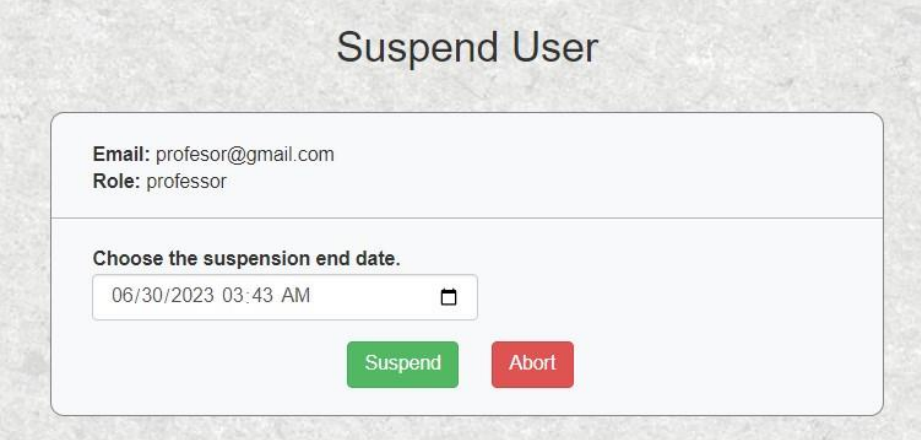

Figure 3.84 - Suspend User page

3.3.4 Testing and Validation

The last step in developing a reliable application is thorough testing to ensure that all the features work as intended.

The responses to the tests conducted below may be dependent on the machine they are tested on. The tests were performed on the developer's computer, having the following specifications: ASUS FX503VD, Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.90 GHz, 16.0 GB RAM, having a Windows 10 Pro Operating System. The tests were run on two different browsers, Google Chrome and Opera, to eliminate potential browser-related issues.

The test cases were elaborated by taking into account potential error-generating scenarios. The testing was conducted manually by the developer. Some of the test cases that were analyzed are listed in the table below.

No.	Tested functionality	Testing case	Expected result	Result
1	Role assignment at signup	Create a new account and check the role assigned by default.	The account will receive the Student Role.	Passed
2	Login	Log into the platform with a suspended account.	The login request is denied.	Passed
3	Course enrollment	The Student tries to enroll in a course.	The course is moved to the MyCourses submenu.	Passed
4	Progress tracking	The Student passes the test afferent to a subchapter.	The next time the Student resumes the learning process, the next subchapter will be displayed.	Passed
5	Subchapter materials download	The user tries to download one of the additional files provided for a subchapter.	The file will be downloaded without any errors.	Passed

6	Chat creation	The Student creates a new chat.	The chat is visible in the ChatRequests submenu(available for the HelpingStudent)	Passed
7	Message	Editing and Deleting a message.	The user is able to edit or delete a chat message he/she sent.	Passed
8	Message	Send a message into a conversation without a HelpingStudent assigned	The message is sent, and can be viewed in the history of the chat	Passed
9	Course management	The Professor edits or deletes a course he owns	The desired changes will be saved in the database.	Passed
10	Course creation	The Professor tries to create a course. The name of the course is not unique.	The course will not be created and an error will be displayed.	Error
11	Subchapter content formating	The Professor tries to edit the content of a subchapter (SubChapterDescription field) .	The Professor can use all the formating options provided by the text editor, and the format will be consistent when the subchapter content is displayed.	Passed
12	Closing a chat	The Student considers that the goal of the chat was reached and tries to close the chat.	The chat is moved to the ArchivedChats section. No more messages can be sent in that conversation. The HelpingStudent is rewarded with twenty Points.	Passed

Among the tested functionalities, an error was identified when trying to create a new course, namely, the name of the course was not imposed to be unique. The occurrence of this fault would result in confusing experiences for the end-users, so it was mandatory to be resolved. The issue was resolved by creating a new method that checks if the potential name for the course already exists or not. If it already exists, an error is transmitted to the view through the ModelState functionality. The way the new method is used to address the error is showcased in Figure 3.85.

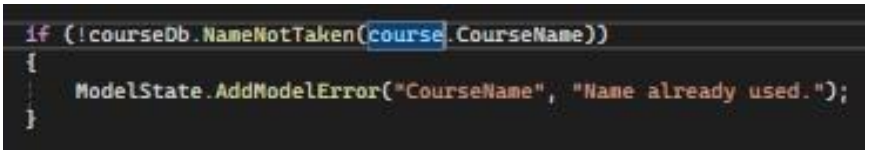

Figure 3.85 - Unique course name error fix

4 Conclusions

4.1 Achievements

The end goal of the *FastLearn* web application was to provide an intuitive online platform that enhances the learning experience of its users. In pursuit of this purpose, the application attempts to bring comprehensive learning tools and collaborative spaces for effective learning to the table.

To achieve the proposed objectives in Chapter 1.2 and implement the theme of this thesis, *FastLearn*, ample research to determine the suitable technologies was conducted. The results of the research are described in Chapter 3.1.2.

To facilitate the implementation process of the application, a Code-First approach was used. At the end of the implementation phase more than 15 separate domain classess, each having their own associated logic were created to support the functionalities of *FastLearn*.

The final web application has an easy-to-use interface, used by five different types of users, described in Chapter 3.1.1. Compared to similar platforms platforms that aspire to fulfill similar goals, *FastLearn* is distinguished by several features:

- A dedicated user-role, HelpingStudent, which is rewarded for helping their colleagues in need.
- One-to-one interactive learning environment between a user in need of help and a user that already masters the respective concept.
- A promotion system allowing a *Student* to become a *HelpingStudent* based on the user's expertise.
- A streamlined, two-tier promotion assessment process conducted by Professors for expertise verification and *Admin* for user background check.
- Easy-to-use User Interface that allows new users to immediately grasp how to utilize the facilities of the platform.

4.2 Future work

A good developer should always have foresight and think of ways to match the rapid pace at which the digital world advances. The last subchapter of this diploma thesis addresses this topic.

To ensure the application's longevity, the end-user's needs must always be decisive when determining what new functionalities will be added to the application. Some of the improvements that satisfy the above goals are:

- A search bar function that will allow the users to search for a specific course.

- Creating dedicated filters that enable the user to search for a course by the subject.
- Introducing an advanced course recommendation system based on the previous interests of the user.
- Adding a multi-language support system to enable coverage across wide geographical locations.
- Delivering a broader set of rewards for the HelpingStudent.
- Simplify the registration process by allowing the users to register on the platform using Google or Facebook accounts.
- Improve the user experience on the platform by collaborating with professionals from the UI design domain.
- Implementing a dedicated issue reporting system instead of using the official email addresses of the administrators.
- Transitioning the existing services into microservices ensures the application's smooth scaling.

To conclude, *FastLearn* was designed using a Use-case Driven Design approach, prioritizing the needs of its users. This approach allows the platform to grow alongside its *Students* to offer the environment needed for efficient learning.

5 References

[1] SQLZOO website [Online]. Available: https://sqlzoo.net/wiki/SQL Tutorial, Accessed: April 4, 2023

[2] GeeksForGeeks website [Online]. Available: https://www.geeksforgeeks.org/, Accessed: April 5, 2023

[3] Udemy website [Online]. Available: https://www.udemy.com/, Accessed: April 5, 2023

[4] Coursera Website [Online]. Available: https://www.coursera.org/, Accessed: April 5, 2023

[5] „ASP.NET MVC Pattern“ [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc, Accessed: 15 May, 2023

[6] Jessica Wilkins, „MVC Architecture -What is a Model View Controller Framework?“, 2021, [Online]. Available: https://www.freecodecamp.org/news/mvc-architecture-what-is-a-model-view-controller-framework/, Accessed: May 15, 2023

[7] Sushant Kumar, „Software Architecture and Architectural Patterns“, 2022, [Online]. Available: https://www.scaler.com/topics/software-engineering/software-architecture/, Accessed: May 15, 2023

[8] Microsoft, „What is .Net?“, Available: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet, Accessed: May 17, 2023

[9] „A tour of the C# language“ [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/, Accessed: May 17, 2023

[10] „What is Entity Framework?“ [Online]. Available: https://www.entityframeworktutorial.net/what-is-entityframework.aspx, Accessed: May 17, 2023

[11] „What is Code-First“ [Online]. Available: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx, Accessed: May 17, 2023

[12] Partech Media, „Implement it in .NET?“, 2021 [Online]. Available: https://www.partech.nl/nl/publicaties/2021/05/basics-of-autofac-how-to-implement-it-in-asp-net-application, Accessed: May 25, 2023

[13] „Dependency Injection in ASP.NET Core“, 2021, [Online]. Available: https://www.tektutorialshub.com/asp-net-core/asp-net-core-dependency-injection/, Accessed: May 25, 2023

[14] Yaakov Chaikin, „What is HTML?“, 2017, [Online]. Available: https://clearlydecoded.com/what-is-html, Accessed: May 25, 2023

[15] „JavaScript - Overview“, [Online]. Available: https://www.tutorialspoint.com/javascript/javascript_overview.htm, Accessed: May 25, 2023

[16] Anna Tomanek, „Open source text editing for your website with CKEditor“, 2023. [Online]. Available: https://opensource.com/article/23/4/website-text-editor-ckeditor, Accessed: May 25, 2023