HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Building a software system for teaching and learning English

VŨ MINH HOÀNG
hoang.vm176765@sis.hust.edu.vn

Major: Information Technology

Supervisor: Dr. Phạm Quang Dũng _____

Signature

School: Information and Communications Technology

HANOI, 08/2022

ACKNOWLEDGMENTS

First of all, I would want to express my sincere appreciation to Dr. Phạm Quang Dũng for his guidance, patience, and feedback since day one of this thesis.

Second, without the instructors at Hanoi University of Science and Technology who so kindly shared their knowledge and experience with me while I was a student there, I would not have been able to go on this trip.

I would also want to express my gratitude to all of my coworkers and friends, who made this project feasible, actively supported me while I worked on it, and provided me with invaluable experiences and evaluation.

I should not forget to give thanks to my parents at this point. I am so grateful to have such a supportive family, they helped me a lot growing up.

Lastly, I would like to give credits to myself for trying hard, and to all the improvements that I had working on this thesis.

ABSTRACT

In today's society, multilingualism is becoming more and more significant. Along with increasing your employability, learning a foreign language gives you the opportunity to communicate with others and gain knowledge of other cultures. Children are therefore encouraged to acquire a foreign language from a very young age. English is the most well-known foreign language in our nation. Children are learning English at a younger age as English is in many countries' educational systems. Traditional ways of learning English are important, but they are geographically and chronologically constrained, as well as it could usually get expensive.

Throughout the years, technological improvements have had an impact on English learning. With the rising availability of smartphones and computers, as well as communications technology, learners may access English learning resources anywhere and whenever they choose. Many applications for learning English on various platforms have been created and released since then. Each product has advantages and disadvantages, but only a handful of them can give users with a comprehensive experience for learning a new language.

In this thesis, we design and build a web application for users to teach and learn English. Our web application features dictionary, flashcards, lessons and exercises. In addition, the system allows administration, managing its users and contents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVIATIONS

Abriviation	Full Expression
API	Application Programming Interface
BSON	Binary Javascript Object Notation
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DB	Database
DI	Dependency Injection
GUI	Graphical user interface
HTML	HyperText Markup Language
IDE	Integrated development environment
JSON	JavaScript Object Notation
NoSQL	Not only Structured Query Language
OOP	Object-oriented programming
RAM	Random Access Memory
RDBMS	Relational database management system
REST	Representational state transfer
SQL	Structured Query Language
URL	Uniform Resource Locator
UX	User experience

CHAPTER 1. INTRODUCTION

1.1 Motivation

Multilingualism is becoming increasingly important in today's world. Aside from expanding job options, knowing a foreign language allows you to interact with people and learn about different countries, places, and lifestyles. The more skilled you are, the more effectively you can express yourself. Although it ranks second behind Mandarin in terms of total speakers, English is the most frequently used language, as it is spoken in more nations than any other.

English is the language of technology, particularly high-tech subjects such as computer science, genetics, and medicine. If you want to read documents in such disciplines, you'll generally have to do so in English. In other words, English is a necessary instrument for broadening and illuminating your worldview. Your access to the world of information will be limited if you do not speak English. Furthermore, once you have mastered English, you will have more opportunity to learn about other cultures through literature, as most popular international works are translated into English rather than other languages. The majority of material on the Internet is also in English.

More and more individuals are investing time to learning English as a second language these days. Many nations integrate English in their education systems, and children are learning English at an earlier and earlier age. Traditional methods of learning English have their importance, but they are geographically and chronologically limited, and also usually pricey.

Learning English has been influenced by technological advancements throughout the years. The emergence of the Electronic Dictionary in the early 2010s transformed the way individuals learn English. Replacing a bulky paper dictionary with a compact device that contains hundreds of thousands of words and includes capabilities like pronunciation allows students to be considerably more versatile. The increasing in availability of smartphones and laptops and telecommunication technology technologies let learner access English learning resources wherever and whenever they want. Since then, many applications for learning English on all platforms have been developed and introduced to the market.

Although each product has its own pros, very few of them can combine effectively between letting people freely choosing what to learn and guiding them to grow properly. Some products only focus on the dictionary features, while others

try to tie them strictly to lessons. For that reasons, the idea was to built an all-in-one English learning platform where people can access with a competitive pricing.

1.2 Objectives and scope of the graduation thesis

As mentioned in the section 1.1, products for English education are widely available on the market today. With over 1.75 billion [1] learner worldwide and the market size of 1.95 billion USD [2], it's not surprising that so many companies want to break into this market.

One of which is Duolingo, the biggest platform in 2022 for multilingual learning according to Statista [1]. With its popularity, its effectiveness in learning a new language is undeniable. Its multilingual aspect, although a huge plus, unfortunately draws the platform away from some essential student necessities, such as the dictionary feature. Since it is a product aimed at a global market, the contents and pricing are also not completely appropriate for Vietnamese people.

Looking into the domestic market, there are also English learning platforms from local companies. These products compete on lower pricing, however they fall short in terms of features. TFlat dictionary could be seen as an honorable mentions. While it is one of the richest dictionary for Vietnamese, its features are mostly focus on its own name "dictionary".

In this thesis, we design and build a web application for users to teach and learn English. Our web application features word games, dictionary, flashcards, lessons and exercises; administration.

1.3 Tentative solution

As mentioned in the previous section, our orientation to resolve the business issues is to build a web-based system for learning English.

The system can be divided into two parts: (i) Client-side application with which the users interact, (ii) Server-side application which stores data and executes business logic. For the characteristics of the data, which are tightly linked, a relational database is chosen to be our main database. There are also data not suitable for storing in relational database. In this case, a NoSQL database will be used to store them as form of documents. The detailed design for the system will be described in 3 and 4

The functions oriented for this project are as follow:

Authentication: The system will let learners create accounts and track their progress. The accounts will be divided into different roles, and each role has its own permissions.

Dictionary: Dictionary is one of the features that many English learning system are lacking. Having a dictionary integrated into the application prevent user switching to another platform.

Flashcards: Flashcards system are built for users to memorize their words of choice. The flashcards are tied heavily to the dictionary and the lessons that the platform provides.

Lessons exercises: Users can learn lessons and do exercises attached to them, therefore have a clearer route to growth.

Upgrade account: The system provides basic functionalities with free tier, and also let people access special contents with a competitive pricing subscription.

Administration: There will also be a page for administration with the ability to manage users, lessons and exercises. The page also provides current statistics of the platform.

Besides our main system, to create a dictionary, we need a data source for words. To tackle this problem, the solution we came up with is to crawl the data from different sources on the internet, then transform the data and store it in our relational database. Therefore, the system might have more control over the data and be easier to get new feature related to those data.

1.4 Thesis organization

The remaining sections of this graduation thesis report are structured as follows.

Chapter 2: Surveying will offer specifics about the current situation, evaluate various popular programs operating in the same sector that are already accessible on the market, and analyze business requirements.

Chapter 3: Methodology will present the topic's primary technologies as well as the environment that technology provides to the system.

Next, in Chapter 4: Experiment and Evaluation, we will study and develop each system function based on the needs assessed, as well as how to implement the system.

Chapter 5: Project Solution Contributions

Finally, Chapter 6 Conclusion will compare our application to the products discussed in Chapter 2, providing conclusions and orientations for future development areas.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

In Chapter 2, surveys will be provided on how English learning systems and platforms are currently being utilized. These surveys will include user and customer demands as well as a review of the various English learning products used in Vietnam and overseas. The functional and non-functional requirements for our platform in general will be covered later on.

2.1 Status survey

In this day and age, anybody may effortlessly learn any language at any time and from any location using a web or mobile application. Online language learning has properly dominated the industry in recent years. There are several websites on the internet that may help you get started. Among the various options for learning a new language, Duolingo[3], British Council[2] Learn English are a well-known platforms. There are also competitors from the domestic market, one of which is Monkey Junior[4]. We investigated these applications as the foundation for our project.

	Duolingo	**Monkey Junior**	**The British Council App**
Characteristics	Easily accessible, all contents can be accessed for free, allows learning multiple languages simultaneously	Wide variety of contents, focus mostly on children from the age of 6 to 10	This popular app is solely dedicated to teaching English grammar. Users can work from Beginner, Elementary, Intermediate and Advanced level.
Disadvantages	Beginners oriented, provides support best for English speakers, focus on vocabulary and phrases only	Only recommended for children, focus only on vocabulary and phrases	Not suitable for newcomers, focus heavily on courses, lack of interactive contents
Dictionary support	No	No	No
Flashcard	Yes	No	No
Vietnamese UI	Partially supported	Yes	No
Pricing	Plus: $6.99/month	2,499,000VND / lisence	Subscription: $69/month

Table 2.1: Comparing 3 platforms for learning English

From our survey, each platforms has their own strengths and weaknesses. Duolingo, despite being one of the most downloaded language learning apps in the world, still has its own drawbacks. Duolingo focus heavily on games and interactive contents, guiding learners toward phrases and vocabulary, which is very fun and suitable for beginners, but it lacks contents for more intermediate users. As a free-to-use platform, its approach to the market is still undeniably effective.

In contrast, British Council Learning English App provides mostly on courses and traditional English learning, focus on English grammar. With this platform, it is possible for learners to go from beginner to advanced level. It even provides courses for international English test like the IELTS. However, this product lacks the fun and enjoyment compared to Duolingo, and the pricing also on the heavier side. This site is also not suitable for Vietnamese to start learning English, since it has no support for Vietnamese language.

Monkey Junior is a good platform for children to learn English from Vietnam. With the wide variety of contents, creativeness, friendly UI, and appropriate one-time payment, it became one of the top choices when it comes to teach English to children. As much good as it is, the platform only stick with its young set of users.

While researching, we found out that all 3 platforms do not integrate dictionaries to their platform. Although there are many dictionary out there already in the market, integrate one inside our learning application will create a more cohesive experience. Therefore, letting people have access to everything in only one platform is our main priority.

To summarize, to provides best English learning experience for Vietnamese people, we will create our system navigating toward the Vietnamese market. Which means (i) the user interfaces should be in Vietnamese, (ii) the contents should be suitable for Vietnamese culture, and (iii) the pricing should be competitive. For the scope of this thesis, below are the features desired to be implemented:

- Searching for words using dictionary
- Flashcard system allows remembering words.
- Learn courses and do exercises
- Managing users, courses and more.

2.2 Functional Overview

2.2.1 General use case diagram

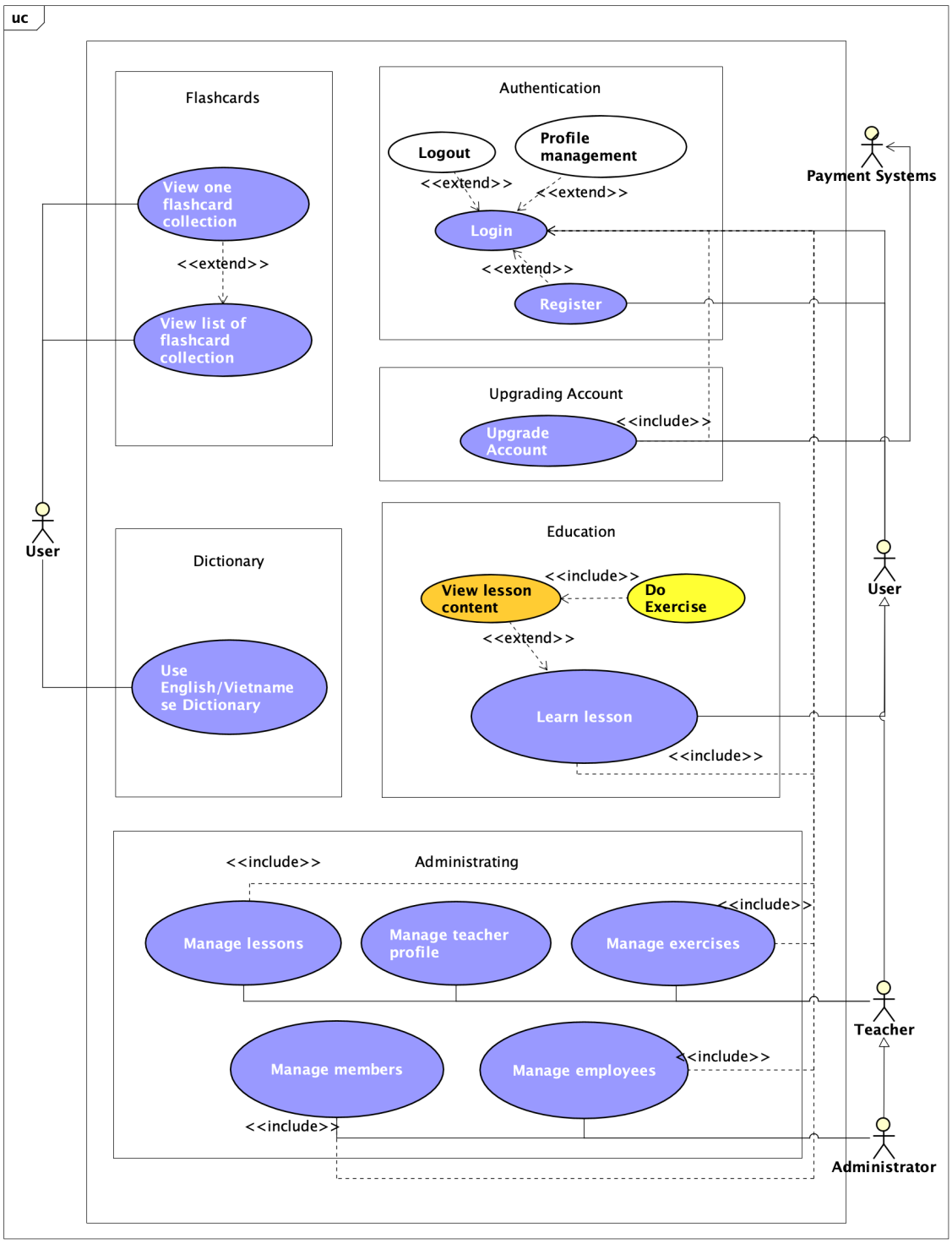

Figure 2.1: General use case diagram of the application

The system contains of 4 main actors.

User is the most basic actor, which represent a general person accessing our system. An user is able to search for words using the system dictionary, and memorize

them through flashcards. The actor also have access to lessons, exercises, and can upgrade account to unlock new contents and features.

Teacher acts as a supporter for students and manages the contents uploaded to the system, which includes exercises and lessons. Teacher also has all the ability of a User.

Administrator is in charge of decentralization and system management. Administrator can manage both User and Teacher, and has all the abilities.

Payment System is an business actor, which represents payment gateways for purchasing account upgrade.

2.2.2 Detailed use case "Use Dictionary"

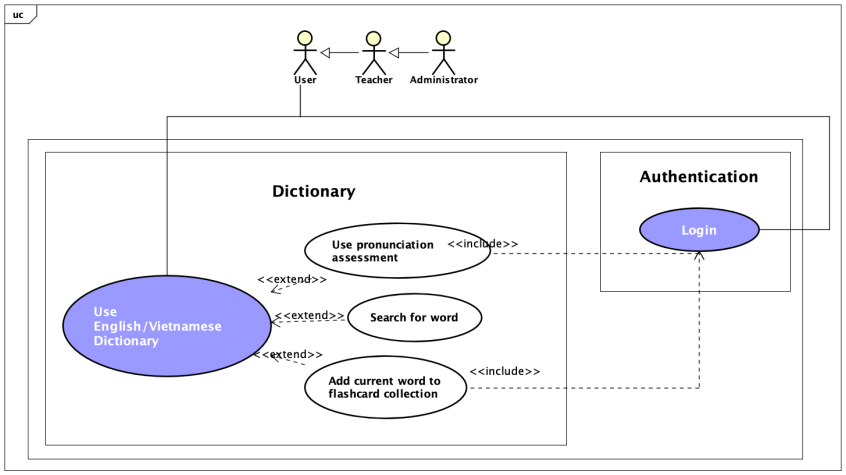

Figure 2.2: "Use Dictionary" detailed use case diagram

The use case "Use Dictionary contains of 3 main functions: (i)Looking for word translation, (ii)Add current word to a flashcard collection, and (iii)Assessing pronunciation. To use feature (ii) and (iii), user must be authenticated by the system.

2.2.3 Detailed use case "View Flashcard Collection"

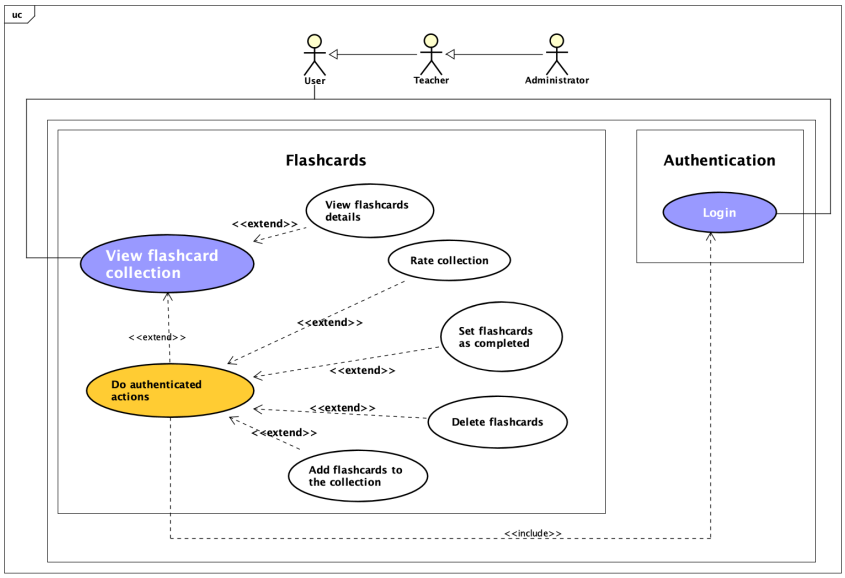

Figure 2.3: "View Flashcard Collection" detailed use case diagram

People can access the flashcard collections and their details anonymously, with only the permission to view. Once an user is logged in, he/she will be able to make changes to the flashcard collections which they created or are following. Furthermore, users may also rate the collections.

2.2.4 Detailed use case "Learn Lesson"

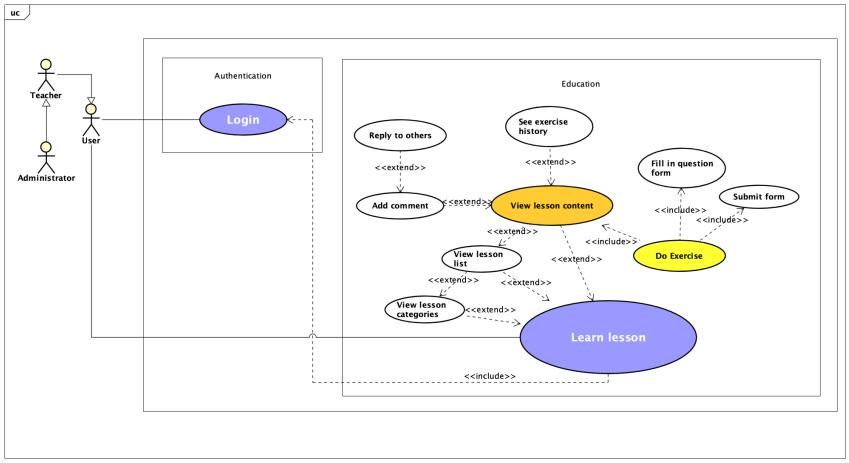

Figure 2.4: "Learn Lesson" detailed use case diagram

The "Learn Lesson" use case always require user to be authenticated. This use case provides the functions to view lessons in list or by category. From there, users can access the lesson content, comment their questions. Once the lesson is finished, exercise section for that lesson will be unlocked, and user will be able to fill and

submit the exercise. After finishing exercise, it is able to see attempts history, and redo the exercise.

2.2.5 Detailed use case "Upgrade Account"

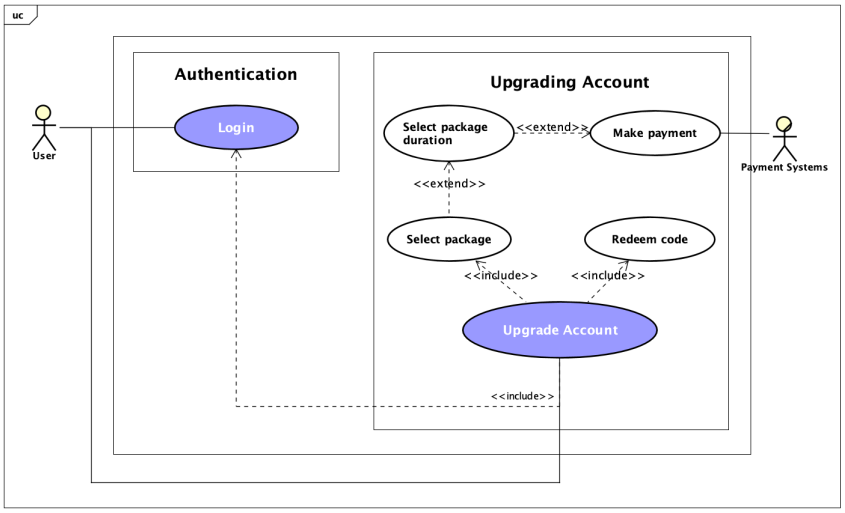

Figure 2.5: "Upgrade Account" detailed use case diagram

Similar to 2.4, this use case also require users to be logged in. The use case includes (i)selecting package, (ii)selecting package duration, (iii)making payment, and (iv)redeeming the codes.

2.2.6 Detailed use case "Manage Lesson"

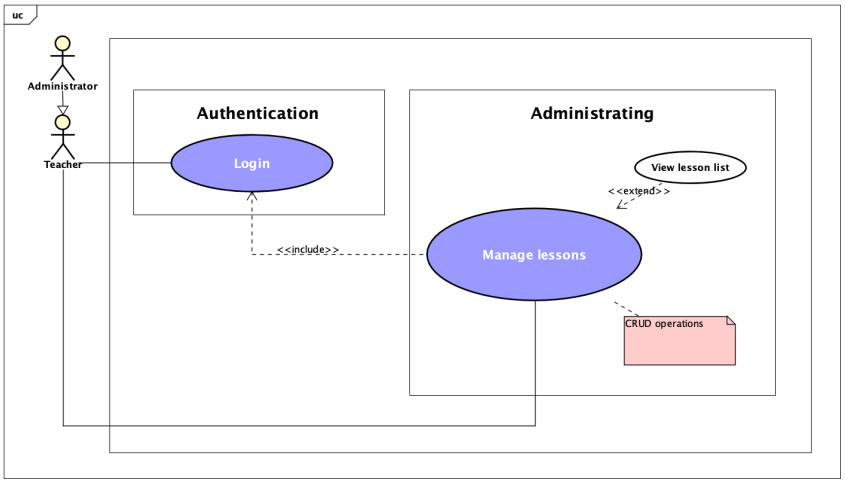

Figure 2.6: "Manage Lesson" detailed use case diagram

"Manage lesson" is the use case specified for the actor **Teacher**. The teacher actor's function in this use case is managing the list of lesson, creating, modifying, and deleting lessons. Login required.

2.2.7 Detailed use case "Manage Exercise"

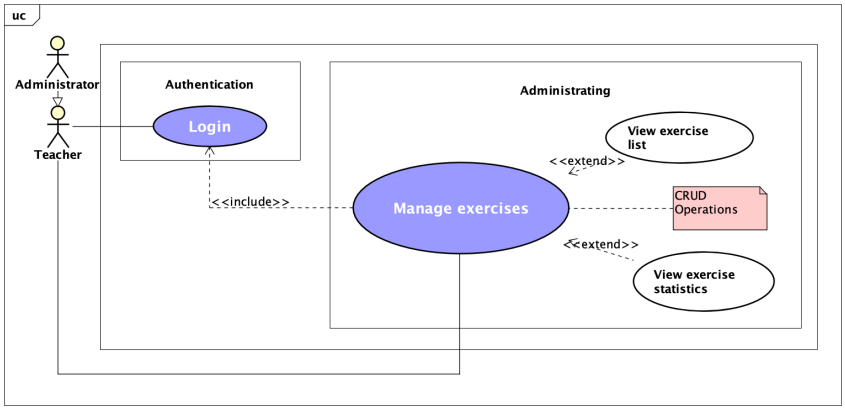

Figure 2.7: "Manage Exercise" detailed use case diagram

This use case is nearly identical with the section 2.2.6, with the functions of viewing exercise list and doing CRUD operations. The actor could also view exercises' statistics, which provide detail insight of the selected exercise.

2.2.8 Detailed use case "Manage Members"

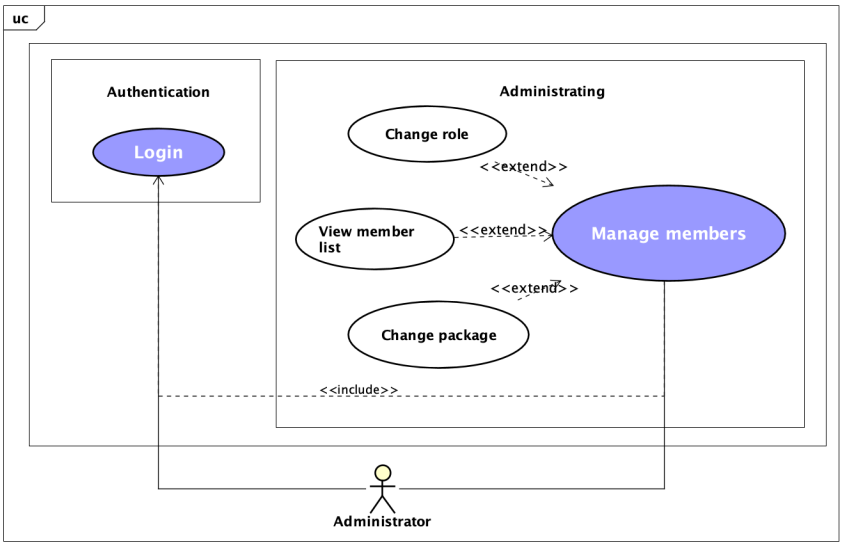

Figure 2.8: "Learn Lesson" detailed use case diagram

s

Administrator has the permission to manage members. An administrator can change privilege of a specific member, making it a Teacher or an Administrator. The actor can also change package information of each account, which related to contents and features unlocking or limitation.

2.3 Functional description

2.3.1 Description of use case "Login"

Brief Description: As a user, I want to login to the system to access more features and contents.

Input data: None

Output data: None

<table>
<tr><td>Use case ID</td><td colspan="3">001 | Use case name | Login</td></tr>
<tr><td>Actor</td><td colspan="3">User</td></tr>
<tr><td>Pre-condition</td><td colspan="3">User is not logged in</td></tr>
<tr><td>Main flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Click "Login"</td></tr>
<tr><td>2</td><td>System</td><td>Show login form</td></tr>
<tr><td>3</td><td>User</td><td>Fill in username and password</td></tr>
<tr><td>4</td><td>User</td><td>Click "Confirm"</td></tr>
<tr><td>6</td><td>System</td><td>Verify inputs</td></tr>
<tr><td>5</td><td>System</td><td>Verify account</td></tr>
<tr><td>6</td><td>System</td><td>Generate token for user</td></tr>
<tr><td>7</td><td>System</td><td>Notify user</td></tr>
<tr><td>8</td><td>System</td><td>Redirect to home page</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>6a</td><td>System</td><td>Alert invalid inputs</td></tr>
<tr><td>7a</td><td>System</td><td>Alert invalid account</td></tr>
</table></td></tr>
<tr><td>Post-condition</td><td colspan="3">System recorded lesson progress for user</td></tr>
</table>

Table 2.2: Description of use case "Login"

2.3.2 Description of use case "View Lesson Content"

Brief Description: As a user, I want to view lesson available on the website.

Input data: None

Output data: None

<table>
<tr><td>Use case ID</td><td>002</td><td>Use case name</td><td>View Lesson Content</td></tr>
<tr><td>Actor</td><td colspan="3">User</td></tr>
<tr><td>Pre-condition</td><td colspan="3">User is logged in and currently in lesson preview page</td></tr>
<tr><td>Main flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Click "Start lesson"</td></tr>
<tr><td>2</td><td>System</td><td>Check package permission</td></tr>
<tr><td>3</td><td>System</td><td>Get lesson contents</td></tr>
<tr><td>4</td><td>System</td><td>Bind lesson to view</td></tr>
<tr><td>5</td><td>User</td><td>Click "Finish lesson"</td></tr>
<tr><td>6</td><td>System</td><td>Check if lesson has flashcard collection</td></tr>
<tr><td>7</td><td>System</td><td>Ask user if they want to add to their collections</td></tr>
<tr><td>8</td><td>User</td><td>Click "Yes"</td></tr>
<tr><td>9</td><td>System</td><td>Add lesson flashcard collection to user’s collections</td></tr>
<tr><td>10</td><td>System</td><td>Record user lesson progress</td></tr>
<tr><td>11</td><td>System</td><td>Redirect to lesson preview and allow doing exercise</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>7a</td><td>User</td><td>Click "No"</td></tr>
<tr><td>7a1</td><td>System</td><td>Record user lesson progress</td></tr>
<tr><td>7a2</td><td>System</td><td>Redirect to lesson preview and allow doing exercise</td></tr>
</table></td></tr>
<tr><td>Post-condition</td><td colspan="3">System recorded lesson progress for user</td></tr>
</table>

Table 2.3: Description of use case "View Lesson Content"

2.3.3 Description of use case "Do Exercise"

Brief Description: As a user, I want to do exercises after finishing a lesson.

Input data: None

Output data: None

<table>
<tr><td>Use case ID</td><td colspan="3">003</td><td>Use case name</td><td colspan="2">Do Exercise</td></tr>
<tr><td>Actor</td><td colspan="6">User</td></tr>
<tr><td>Pre-condition</td><td colspan="6">User is logged in and had finished lesson</td></tr>
<tr><td>Main flow</td><td colspan="6"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Click Begin</td></tr>
<tr><td>2</td><td>System</td><td>Get Exercise and Questions information</td></tr>
<tr><td>3</td><td>System</td><td>Binding questions to view</td></tr>
<tr><td>4</td><td>User</td><td>Choose answer</td></tr>
<tr><td>5</td><td>System</td><td>Record all answers</td></tr>
<tr><td>6</td><td>User</td><td>Click “Submit”</td></tr>
<tr><td>7</td><td>System</td><td>Calculate exercise result and save attempt</td></tr>
<tr><td>8</td><td>System</td><td>Announce result</td></tr>
<tr><td>9</td><td>User</td><td>Press Confirm</td></tr>
<tr><td>10</td><td>System</td><td>Redirect to lesson preview and show history</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="6"></td></tr>
<tr><td>Post-condition</td><td colspan="6">System recorded attempt history for each submission</td></tr>
</table>

Table 2.4: Description of use case "Do exercise"

2.3.4 Description of use case "Upgrade Account"

Brief Description: As a user, I want to upgrade my account so I can have more contents and features.

Input data: None

Output data: None

<table>
<tr><td>Use case ID</td><td colspan="3">004 | Use case name | Upgrade account</td></tr>
<tr><td>Actor</td><td colspan="3">User</td></tr>
<tr><td>Pre-condition</td><td colspan="3">User is logged in</td></tr>
<tr><td>Main flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Click "Upgrade account"</td></tr>
<tr><td>2</td><td>System</td><td>Redirect user to upgrade page</td></tr>
<tr><td>3</td><td>User</td><td>Choose package and period</td></tr>
<tr><td>4</td><td>System</td><td>Show payment popup</td></tr>
<tr><td>5</td><td>User</td><td>Fill in payment card details</td></tr>
<tr><td>6</td><td>User</td><td>Click “Submit”</td></tr>
<tr><td>7</td><td>System</td><td>Send request to payment system to make purchase</td></tr>
<tr><td>8</td><td>Payment System</td><td>Verify purchase and send result</td></tr>
<tr><td>9</td><td>System</td><td>Send email with redeem code to user</td></tr>
<tr><td>10</td><td>System</td><td>Show result</td></tr>
<tr><td>11</td><td>User</td><td>Press Confirm</td></tr>
<tr><td>12</td><td>System</td><td>Redirect to redeeming screen</td></tr>
<tr><td>13</td><td>User</td><td>Redeem code</td></tr>
<tr><td>14</td><td>System</td><td>Verify code</td></tr>
<tr><td>15</td><td>System</td><td>Show result and redirect to login screen.</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3"><table>
<tr><th>No.</th><th>Actor</th><th>Action</th></tr>
<tr><td>8a</td><td>System</td><td>Receive payment failed</td></tr>
<tr><td>8a1</td><td>System</td><td>Show result</td></tr>
<tr><td>8a2</td><td>System</td><td>Close popup</td></tr>
<tr><td>14a</td><td>System</td><td>Show warning about invalid code</td></tr>
</table></td></tr>
<tr><td>Post-condition</td><td colspan="3">System recorded attempt history for each submission</td></tr>
</table>

Table 2.5: Description of use case "Upgrade account"

2.4 Non-functional requirement

To deliver a smooth and speedy user experience, the system must fulfill the following non-functional criteria in addition to the business requirements: (i) The application’s interface should be well-designed and get feedback from the system fast. (ii) Application components on the user’s site must be operated without logical errors or compromising the original user interface. (iii) The program must provide good performance as the number of users in the system grows, constantly ensuring that the system is not overloaded.

Chapter Summaries

Preliminary surveys on English Learning solutions from Vietnam and around the world were provided in Chapter 2. We had a better understanding of what

needed to be done to create a platform fit for the Vietnamese market as a result of the survey, and we drew out the functional and non-functional needs for our application. We will look at what was used to construct such a platform in the following chapter 3.

CHAPTER 3. METHODOLOGY

Chapter 2 looked at the application requirements and the specific requirements that the application must fulfill. In chapter 3, we will describe the technologies implemented in Front-end and Back-end and services used in the project.

3.1 Architectures

A good architecture is essential for developing scalable, modular, and maintained programs. Different architectures may differ in their features, but they always strive for the same goal: separation of concerns. And they all attempt to establish this separation by layering the application.

3.1.1 Clean Architecture

Clean Architecture is a software architecture designed to keep code under control while avoiding the tidiness that prevents anybody from modifying code once it has been released. The basic idea behind Clean Architecture is that application code/logic that is unlikely to change should be developed without any direct dependencies. So if we alter our framework, database, or user interface, the core of the system (Business Rules/ Domain) should not change. It means that external dependencies can be fully replaced.

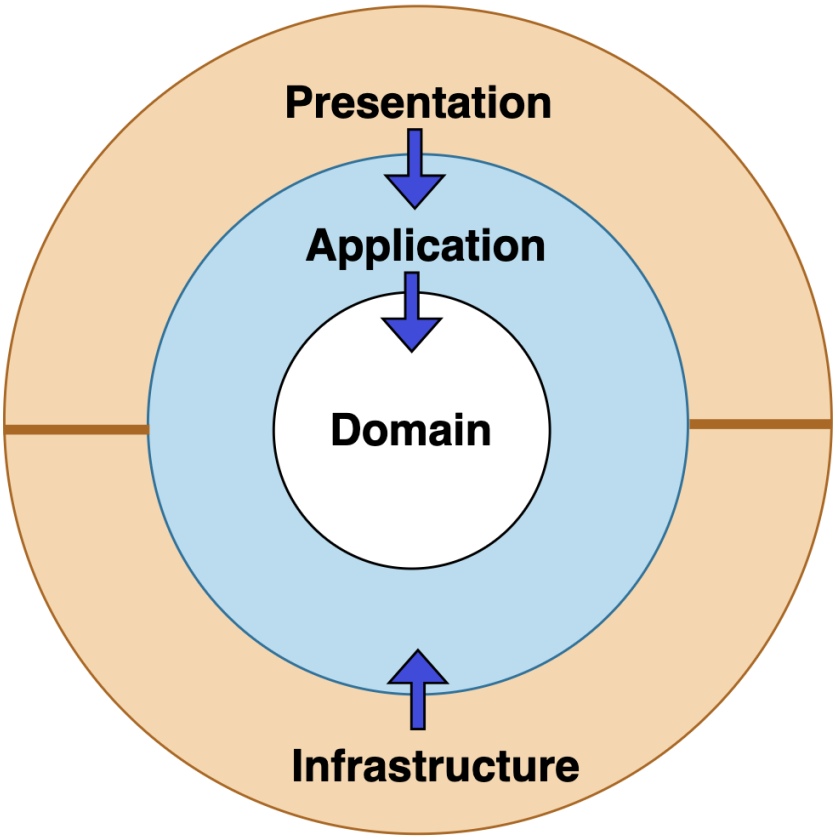

Figure 3.1: Clean architecture

The Domain and Application layers remain in the base of the design, known as the Core of the system, in Clean architecture. Enterprise logic is contained in the domain layer, whereas business logic is included in the application layer. Enterprise logic can be shared across several systems, however business logic is often employed just within the system. The core will be unaffected by data access or other infrastructure issues. And we can accomplish this by utilizing interfaces and abstraction within the Core and having other layers outside of the Core implement them.

All dependencies flow inwards in Clean Architecture, and Core is independent on any other layer. And the Infrastructure and Presentation layers are dependent on the Core.

The Advantages of Clean Architecture Database and framework independence Unrelated to the presentation layer. We may alter the UI at any moment without affecting the rest of the system or business logic. Highly testable, particularly the fundamental domain model and associated business rules.

The advantages of Clean Architecture:

- Database and framework independence

- Unrelated to the presentation layer. We may alter the UI at any moment without affecting the rest of the system or business logic.
- Highly testable, especially the fundamental domain model and associated business rules.

The detail application of Clean Architecture to our platform will be described in details in the Chapter 4.

3.1.2 Microservices Architecture

A Microservices architecture[5], known as Microservices, is an architectural paradigm based on a collection of independently deployable services. These services are self-contained, with their own business logic and database, and they serve a specific function. Each service is updated, tested, deployed, and scalable. Microservices isolate critical business challenges into discrete code bases. Microservices do not reduce complexity, but they do make it visible and controlled by breaking operations down into tiny processes that operate independently while contributing to the overall system. As a result, developers may focus on one microservice without worrying about the others. This centralized model implies that development cycles for several teams are shorter, and firms can get goods to market faster.

Microservices are usually associated with DevOps because they serve as the foundation for continuous delivery methodologies that allow teams to respond to changing customer requirements quickly. More businesses are moving away from monolithic approaches and toward microservices.

3.2 REST API

A REST API [6] (also known as a RESTful API) is an API (a set of definitions and protocols for developing and integrating application software) that follows to the constraints of the REST architectural style and allows interaction with RESTful web services. When a client uses a RESTful API to request a resource, the resource's REST is sent to the requester or endpoint. This information, or representation, is sent via HTTP in one of several formats: JSON (Javascript Object Notation), HTML, XML, Python, PHP, or plain text. This is a well-known mode of communication in web applications.

3.3 Front-end technologies

3.3.1 Basic technologies for web programming

The World Wide Web, the most popular platform utilized by 4.9 billion people worldwide, is the foundation around which our application is built. Therefore, we should have a fundamental grasp of the components that make up a web application

before delving further into the program's core.

3.3.2 HTML

HTML (Hyper Text Markup Language) [7] is a standard markup language for web pages established by the W3C (World Wide Web Consortium) that aids in the definition of a page's text structure. HTML elements are represented by tags, which are often expressed in pairs. HTML is something that a browser can comprehend and present to the user as an interactive interface.

3.3.3 CSS & SCSS

CSS (Cascading Style Sheets) [8] enables web browsers to render HTML text. CSS is a language that specifies the style of an HTML document. It makes the website more colorful and appealing. We may modify the font style, font size, color, and many other aspects of the webpage using CSS.

SCSS is a CSS superset used for client-side styling. SCSS is essentially a more advanced version of CSS that enables cleaner, more manageable style as well as a major increase in capability. Variables, operators, nested syntax, functions, and many more capabilities are available in SCSS. SCSS is our tool of choice for developing and styling the web client.

3.3.4 JavaScript & TypeScript

JavaScript [9] is a scripting language for creating interactive web pages. It follows client-side programming standards, meaning it operates in the user's web browser without requiring any resources from the web server.

TypeScript [10] is a JavaScript programming language from the current era. It is a statically built language for writing concise JavaScript code. TypeScript has static typing, classes, and an interface. Adopting Typescript for a big JavaScript project can result in more robust software that is readily deployed alongside a standard JavaScript application. TypeScript was also officially supported by the framework that we used, which will be described in the next section.

3.3.5 Angular

Even though a web application can be build by using only web vanilla technologies, it might be cumbersome to manage and maintain as your application grows. Therefore, many web framework and libraries, like React, Vue and Angular[11], has been developed to solve these issues, speed up the developing process, and also optimize performance of your applications.

Google's Angular is a JavaScript framework for creating Single Page Applications (SPAs) with JavaScript, HTML, and TypeScript. Angular offers built-in

capabilities for animation, http service, auto-complete, navigation, toolbar, menus, and so on. The code is written in TypeScript and compiles to JavaScript before being shown in the browser.

The architecture of an Angular application is based on a few core concepts. The core building elements of the Angular framework are Angular components arranged into NgModules. NgModules organize related code into functional groups; an Angular application is defined by a collection of NgModules. An application always contains at least one root module that facilitates bootstrapping, and it usually has many more feature modules.

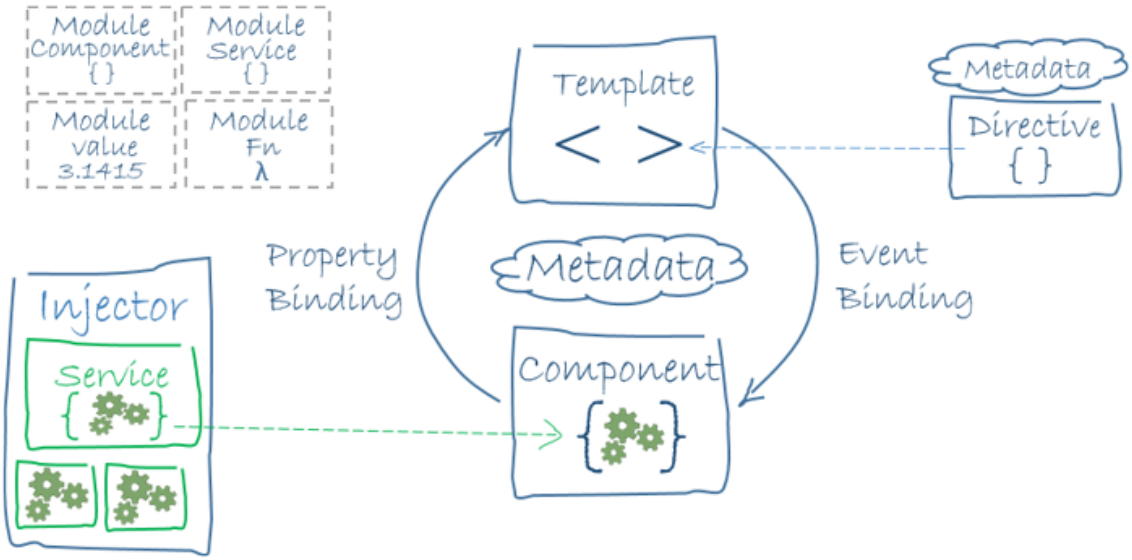

Figure 3.2: Angular communications[12]

Angular is also tightly coupled with the concept of Dependency Injection (DI). Components make use of services, which provide functionality that is not directly connected to views. Service providers can be injected as dependencies into components, making your code modular, reusable, and efficient.

3.3.6 Material UI

Material UI[13] is the most powerful and efficient tool for developing an application by incorporating designs and animations, as well as combining technological and scientific innovation. It is essentially a design language created by Google in 2014. It includes additional design and animations, a grid system, and shadows and lighting effects.

It can be used with any JavaScript frameworks and libraries, such as Angular and VueJS, to make the application more fantastic and responsive. Material UI is one of the leading React User Interface frameworks, with over 35,000 stars on

GitHub.

By using Material UI, the application user interface will be more elaborate and consistent, and provides better experience to users.

3.4 Back-end technologies

3.4.1 .NET 6.0

Microsoft .NET [14] is a software framework and development ecosystem designed to facilitate application engineering for online and desktop environments. It now supports the programming environment for most phases of software development, therefore it has a wide range of applications and use cases.

.NET is a suitable development option for firms seeking for a variety of capabilities in their software, such as desktop applications, cloud infrastructure support, and web-based services.

.NET is built on the object-oriented programming model (OOP). This technique divides data into objects (data fields) and utilizes class declarations to define the behavior and contents of the objects. This OOP development's modular structure allows developers to establish object interactions without having to handle their inner characteristics. In the long term, this simplifies development since the code is easier to test, more manageable, and responsive to problems.

Code written using .NET Core can support cross-platform application, which can run on Windows, macOS, and Linux. While the original.NET framework was not totally open and did not provide such cross-compatibility, the.NET Core includes fully open-source technology to guarantee the proliferation of use cases throughout the developer community. From C to Visual Basic, code created in.NET will operate on any supported operating system, allowing businesses to access a wide range of platforms without leaving the ecosystem.

One of the most significant advantages of.NET programming is its flexibility. It is simple to install as part of a program or separately. The platform's modular architecture incorporates all essential dependencies, making deployment as simple as copying a folder. Additionally, multiple.NET Core versions may operate on the same machine at the same time, making it simple to work on different projects while smoothly executing deployment.

With the latest version, .NET 6.0 simplifies the developing process, and brings better performance at the same time. The version 6.0 also listed as long-term support, which is suitable for building a stable system.

3.4.2 Authentication - JSON Web Token (JWT)

Authentication is the process of ascertaining whether or not someone or something is who or what they claim to be. Authentication technology controls system access by determining if a user's credentials match those in a database of authorized users or in a data authentication server. Authentication ensures safe systems, secure processes, and organizational information security in this way.

JSON Web Token (JWT)[15] is an open standard (RFC 7519) that offers a concise and self-contained method for securely sending information between parties as a JSON object. Because it has been digitally signed, this information can be checked and trusted. JWTs can be signed with a secret (using the HMAC algorithm) or a public/private key pair (using RSA or ECDSA).

Although JWTs can be encrypted to offer confidentiality between parties, we will concentrate on signed tokens. Signed tokens can validate the claims contained inside them, whereas encrypted tokens conceal those claims from third parties. When tokens are signed with public/private key pairs, the signature also confirms that only the party with the private key signed it.

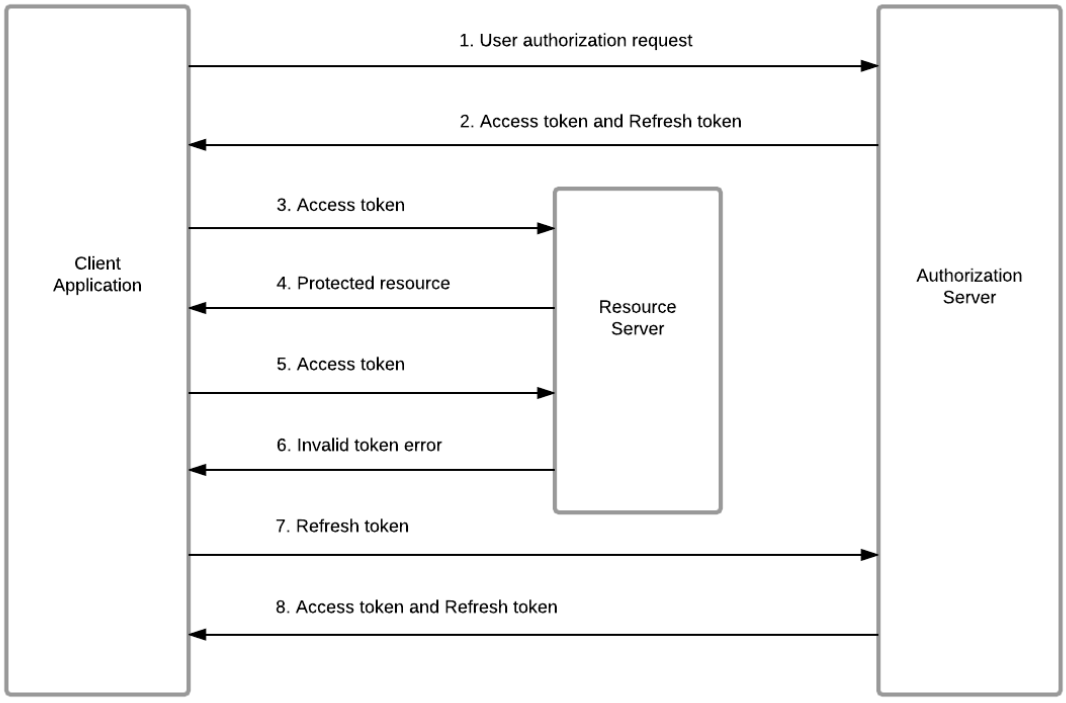

Figure 3.3: JWT authenticating process diagram

There are many token types, but the most common in JWT authentication are access tokens and refresh tokens.

The access token contains all of the information required for the server to determine whether or not the user or device may access the resource you are seeking. Typically, these are expired tokens with a short validity time.

A refresh token is used to produce a new access token. If the access token contains an expiration date, the user must authenticate again to acquire an access token once it expires. This phase can be avoided with a refresh token and a request to the API to obtain a new access token that allows the user to continue accessing the application resources.

Our application use JWT as the main method of authenticating user, with the use of both access token and refresh token.

3.4.3 Database

A database is a structured collection of data that is often stored and accessible digitally via a computer system. It allows for data storage and manipulation. In other words, databases are utilized by businesses to store, manage, and retrieve information.

In our application, there were 2 databases that we used - MySQL and MongoDB

a, MySQL

MySQL[16] is the most popular database in the world, ranking second only to Oracle Database. MySQL is the most common open source database currently in use. It was called after its founder's daughter My, and is notable for arranging data into one or more data tables in which data kinds are connected to one other. It is one of the most trustworthy and performative databases available. Because SQL is a programming language used to create, modify, and extract data from a relational database, these relationships assist structure data.

MySQL is a structured query language relational database. Relational databases are a form of database that employs a structure to identify and access data in relation to other data within the database.

Multiple storage engines, including InnoDB, CSV, and NDB, can be used with MySQL to store and access data. For improved performance and durability, MySQL is also capable of data replication and table partitioning. Users using MySQL do not need to learn any new SQL commands in order to access their data.

Developed in C and C++, MySQL runs on more than 20 different operating systems, including Mac, Windows, Linux, and Unix. Numerous data types, including signed or unsigned integers of lengths 1, 2, 3, 4, and 8 byte(s), FLOAT, DOUBLE, CHAR, VARCHAR, BINARY, are supported by RDBMS, which also supports

massive databases with millions of records.

For security, MySQL uses an encrypted access and password system that allows server-based verification. MySQL clients can connect to the MySQL Server using a number of protocols, including TCP/IP on any platform. MySQL also supports a number of client and utility programs, command line programs, and administrative tools such as MySQL Workbench. Not simply from a data standpoint, but also from a development one, MySQL is dependable. It is established, regularly updated with patches, and supported by a strong development community. Compared to more recent, less developed RDBMS choices, this makes it a secure option.

b, MongoDB

The most well-known NoSQL database, MongoDB[17], is a free, document-oriented database. Non-relational is what "NoSQL" refers to. It means that MongoDB offers a completely alternative mechanism for data storage and retrieval and is not based on the relational database structure that resembles a table. The name of this storage format is BSON (similar to JSON format).

Tabular data is stored in SQL databases. This data is kept in a preset data model that isn't very flexible for the modern, rapidly expanding real-world applications. Applications today are more social, interactive, and networked than ever before. Applications are storing and accessing data at increasing and higher speeds.

Due to their architecture, relational database management systems (RDBMS) are not the best option for handling massive data because they are not horizontally scalable. The database will hit a scaling limit if it uses a single server. NoSQL databases offer better performance and are more scalable. With its adaptable document model, MongoDB, a NoSQL database, boosts productivity by adding evermore servers.

Document Oriented: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk, etc.

Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time. **Scalability**: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data

chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database.

Replication and High Availability: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.

Aggregation: Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

Because of the flexibility of MongoDB, it was used in our application to store more complex data such as lessons content or exercises' questions.

3.4.4 Message broker and RabbitMQ

In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain. Message brokers provide communication and coordination for these distributed applications.

A message broker (also known as an integration broker or interface engine) is an intermediary computer program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Message brokers are elements in telecommunication or computer networks where software applications communicate by exchanging formally-defined messages

RabbitMQ [18] functions as a message broker, accepting and forwarding messages. Consider it like a post office: if you put your mail in a post box, you can be sure that the letter carrier will eventually deliver it to your receiver. RabbitMQ is a post box, a post office, and a letter courier in this example. The primary distinction between RabbitMQ and the post office is that it takes, stores, and forwards binary blobs of data messages rather than paper.

Let assume you now have a web service that must receive a large number of requests every second while ensuring that not a single request is dropped. And instead of being locked because it is processing the previous request, your web service is always ready to take a new request. So the goal is to queue them between the web service and the processing service. This will ensure that the two processes are fully independent of one another. Furthermore, when the number of requests becomes exceedingly enormous, the queue will hold them all without missing any.

Further details about implementation of RabbitMQ in our system will be found

later in chapter 5.

Chapter Summaries

This chapter outlines provided you with chosen technologies, what are they, and why we used them. In chapter 4, we will take a look at our system design.

CHAPTER 4. EXPERIMENT AND EVALUATION

In this Chapter 4, we will show the overall design, detailed design, and method of deploying the application based on the findings of the current condition survey as well as the deep business analysis of all aspects in the application provided in 2.

4.1 Architecture design

4.1.1 Software architecture selection

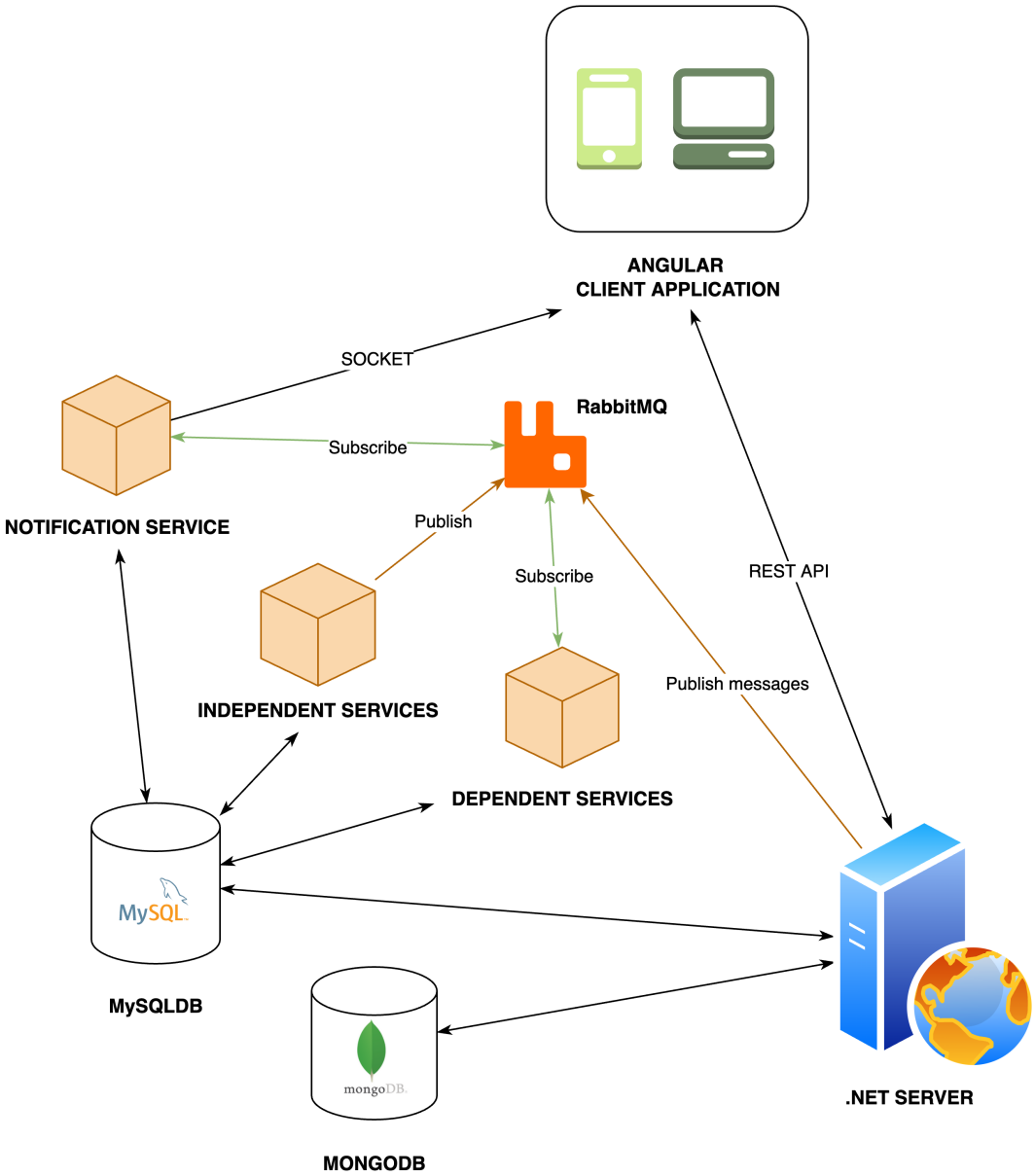

Figure 4.1: System architecture

Our system architecture is illustrated in figure 4.1. Our web application communicates with server through REST API. For tasks required real-time communi-

cations, client will connect to services through network socket.

Our server is based on the simplified version of microservices architecture, in which most of the main business logic will be handled by our main .NET 6.0 server, while specific tasks will be executed by our services (or workers).

Our main server, which is implemented with .NET 6.0, used the Clean Architecture (section 3.1.1). Detailed application for our main server with Clean Architecture is describe more in section 4.1.3.

There are two main types of workers in our system, the independent ones, which decides all is logic independently, and the dependent ones, which need to receive in formations from other services to be able to do their duties. In order to communicate with each others, we used RabbitMQ as a message broker which mentioned in section 3.4.4. By using this, our services can send and receive information, without having to deal with others' business logic.

For storing data, our system used MySQL as the main database, storing most information for our application. Where as MongoDB is used in addition to store document type data. Both databases are described in details in section 4.2.3

4.1.2 Overall design

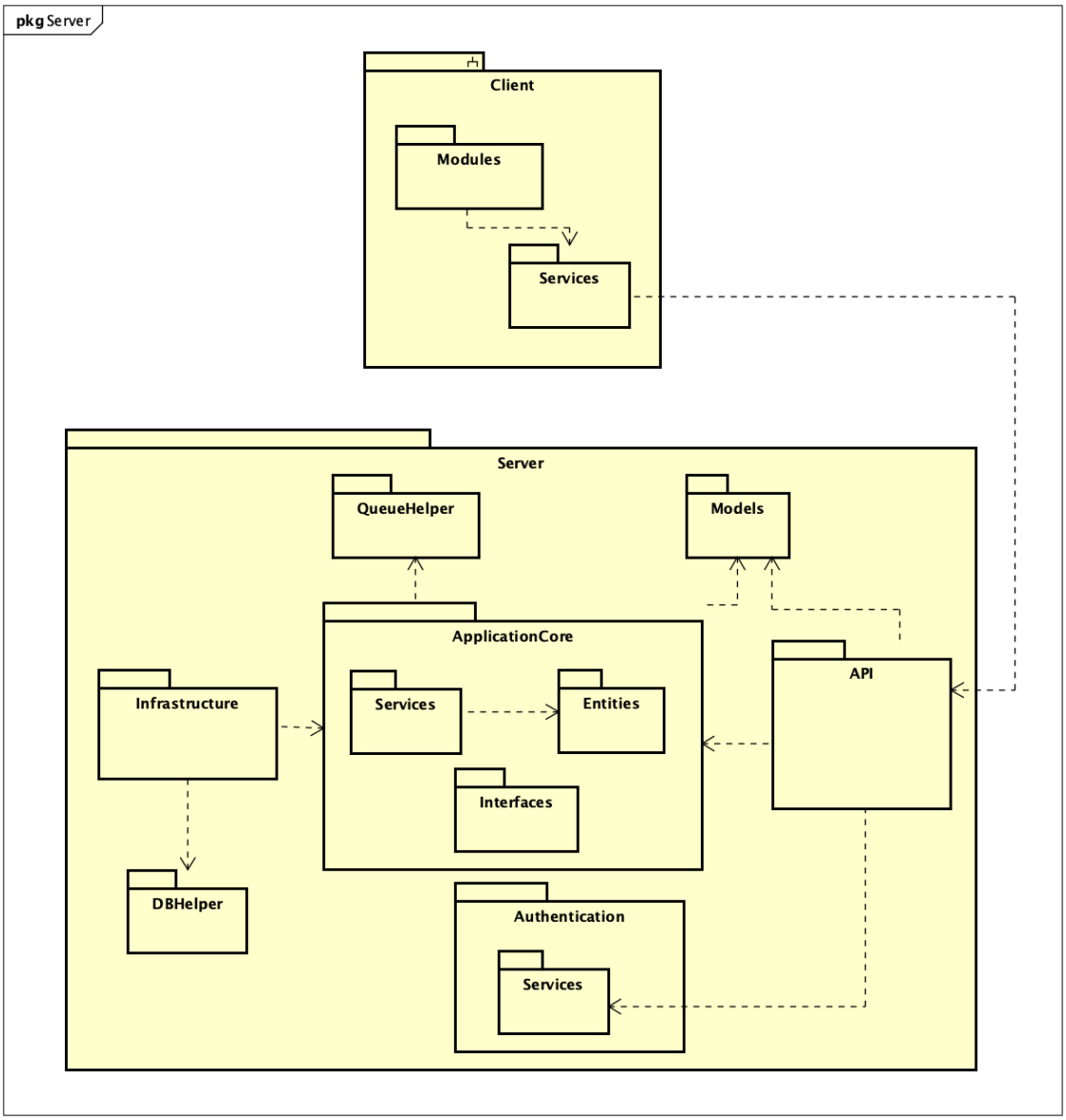

Figure 4.2: Overall package design

Our package design consists of two main components: **Client** and **Server**

The client application is designed with Angular 14, which divides the app into modules. Each module may contain Angular essential components that compose pages in our application. To connect with the server, the client must send an request using the Services package. Services manage application states and send HTTP requests to the main server.

The server is built using the Clean Architecture, which was mentioned in the section 3.1.1.

The ApplicationCore package represents the Application and Domain compo-

nent in the Figure **??**. The Services package is responsible for executing business logic. The Entities package contains all database entities for the whole system. Interfaces defines rules for communication between packages. Interfaces and Entities packages combined become the Domain from the Figure **??**, and Services stand for Application component. As the name implies, ApplicationCore is the heart of our server, on which most other packages in the system depend.

API and Infrastructure packages stand at the outer layer of the architecture. Both packages depend on ApplicationCore. Infrastructure is used for communicating with the database, and it use the DBHelper package for establishing connections. API package handle incoming requests from the client. Beside using ApplicationCore for executing business logic, API package also use the Authentication package to authenticate.

Next, Models package contains shared models between package, which are not related to the Entities package.

Finally, QueueHelper package is used for accessing the MessageBroker of the system. The package also used by others subsystem in order to communicate with the main server.

For specific tasks, there will be independent services, also known as workers, to support the main server. These workers will be described more in the Chapter 5.

4.1.3 Detailed package design

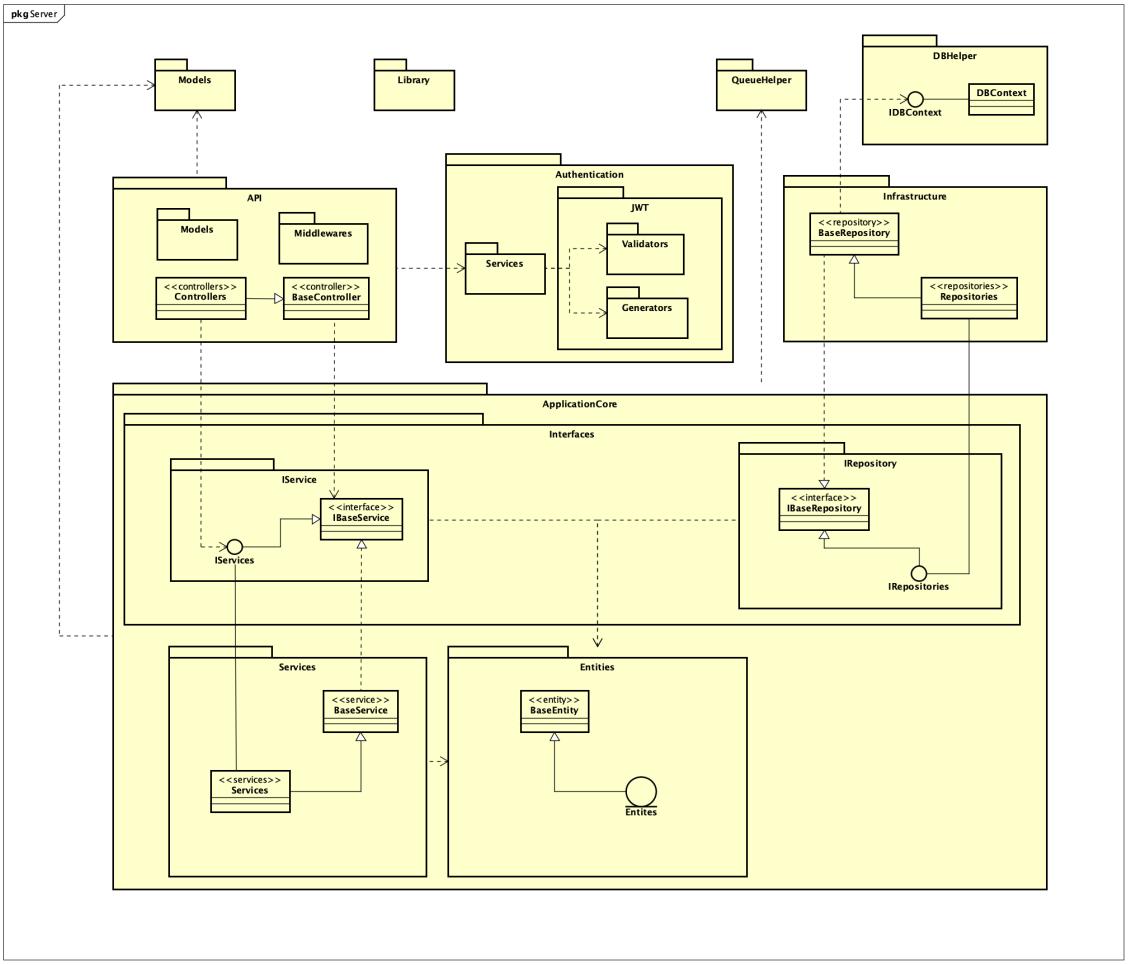

Figure 4.3: Detailed package design diagram

In the section 4.1.3, we have describe about how the system will depend on the ApplicationCore package. The Interfaces package contains 2 packages IService and IRepository, each of which contains interfaces for corresponding classes.

ApplicationCore implements classes for the IService package, while class implementation for IRepository package will be implemented in Infrastructure package. By keeping both service and repository interfaces in the same space, all the business logic and actions are controlled by the ApplicationCore. The Entities package contains a BaseEntity class which holds common attributes and actions. Since both the API package and Infrastructure package refer to ApplicationCore, they can access both the Interfaces and the Entities.

To implement the architecture, using Dependency Injection (DI) is compulsory. In API packages, request handlers are known as Controllers. In order to communicate with ApplicationCore, a controller need to inject the service class instance through interface using DI. The same is true for Infrastructure; in order for services

to obtain instances of repositories classes, the system will inject the appropriate class instance that corresponds to the repository interfaces. Using DI, a class might be updated or replaced without requiring the entire system to be fixed.

The repository classes use DBHelper package, also by using DI, to get the connection context instance for the databases. In our system, there are contexts for both MongoDB and MySQL.

Authentication package is similar to ApplicationCore, but only process logic related to the authenticating tasks. It contains Validators and Generators classes for JWT authentication.

4.2 Detailed design

4.2.1 User interface design

a, Monitor specifications

supports resolutions for desktop displays with aspect ratios of 4:3, 16:9, 16:10, or 21:9 with the minimum resolution of 1280x720px. For the scope of the thesis, only desktop screens are supported, there is no native support for mobile screen at the time of this writing. The software can display 8-bit color spectrum, which is standard for a website.

b, User Interface Illustration

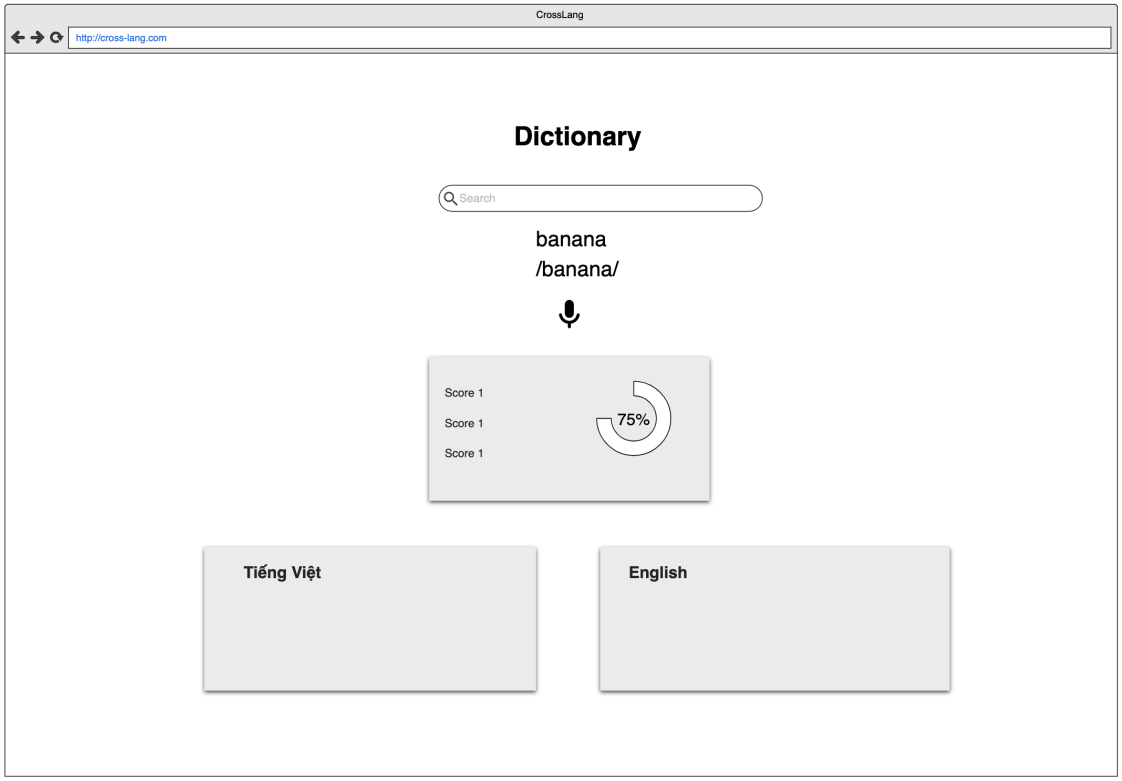

Figure 4.4: Dictionary screen mockup

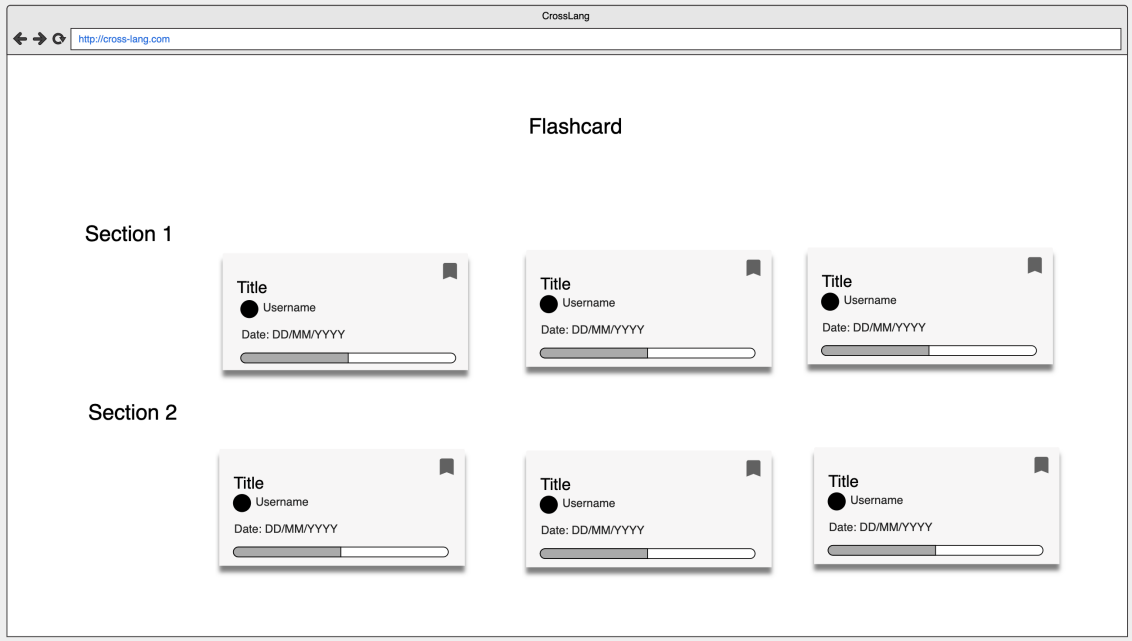

Figure 4.5: Flashcard collection list screen mockup

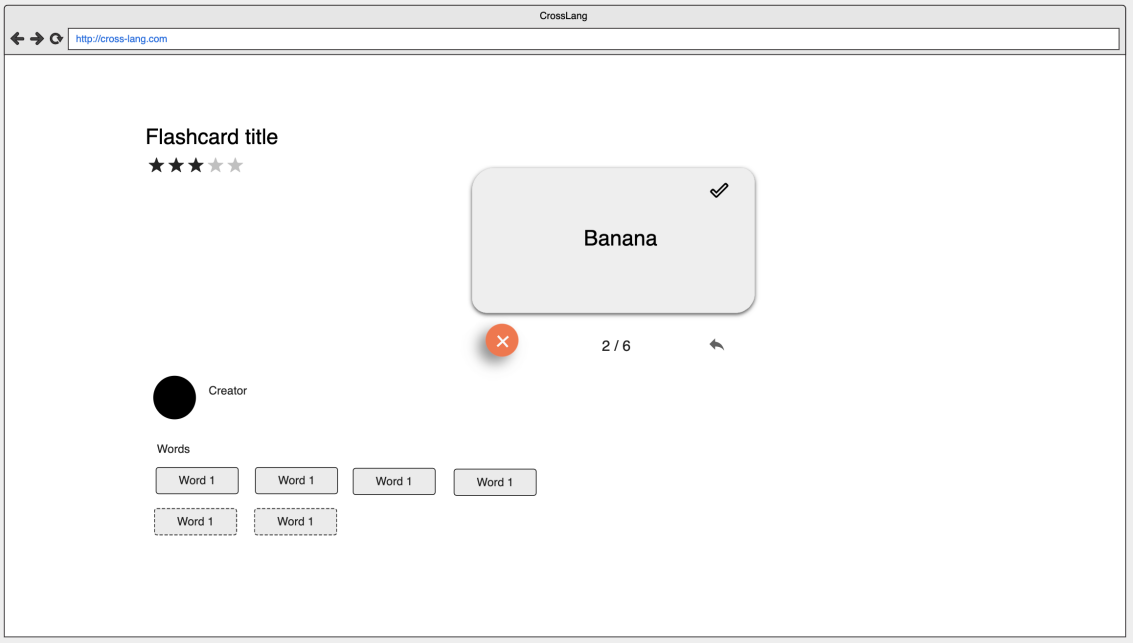

Figure 4.6: Flashcard collection details mockup

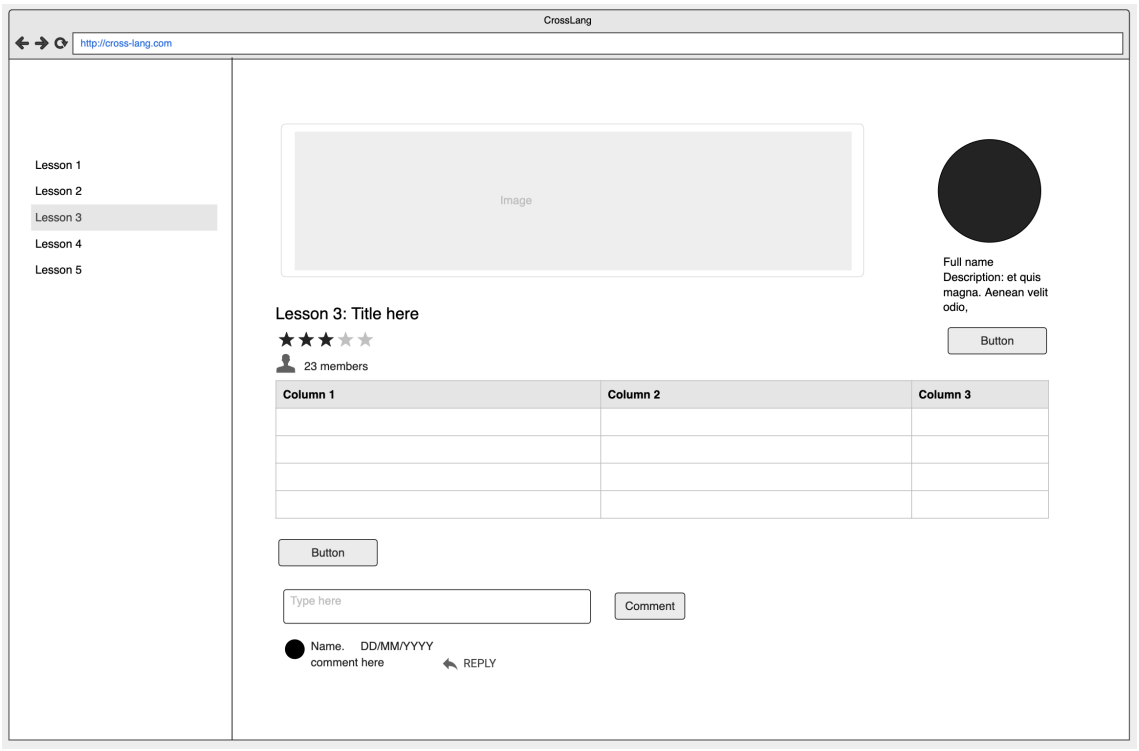

Figure 4.7: Lesson preview screen mockup

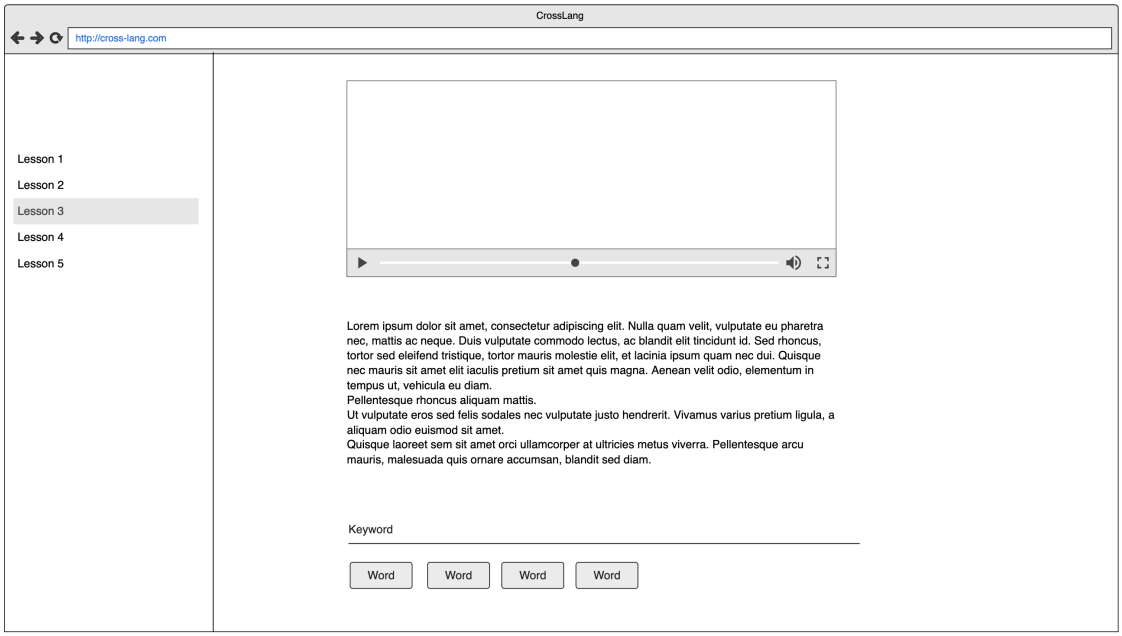

Figure 4.8: Lesson detail screen mockup

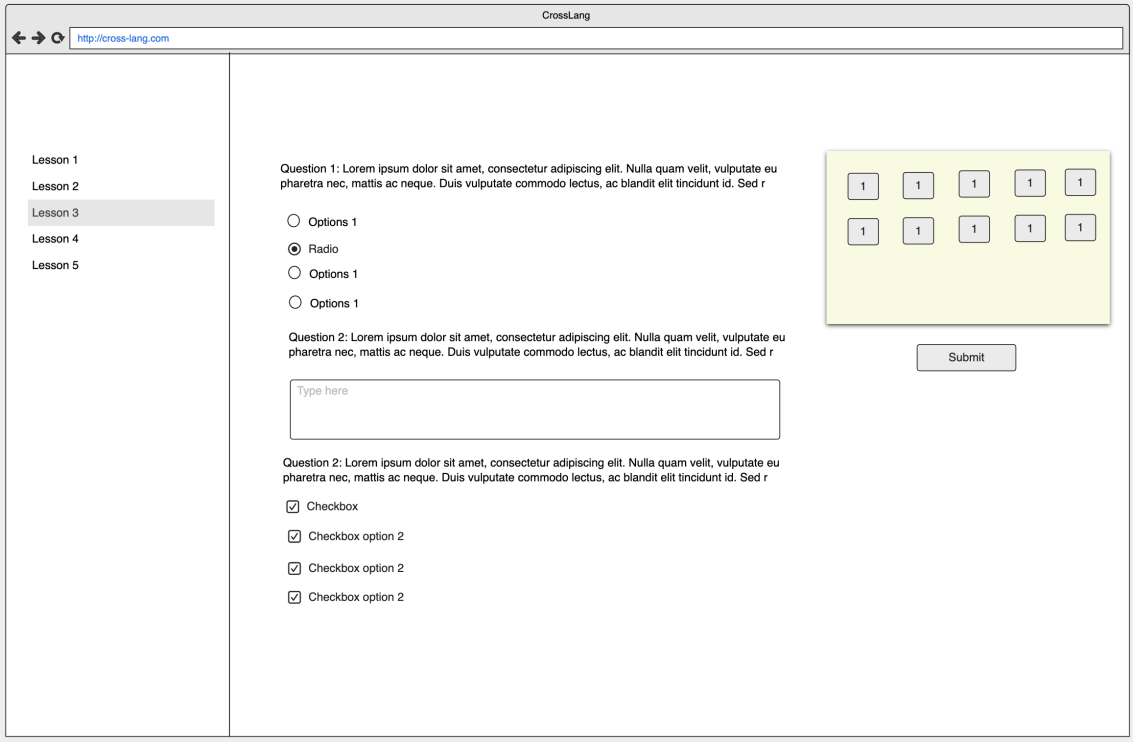

Figure 4.9: Do exercise screen mockup

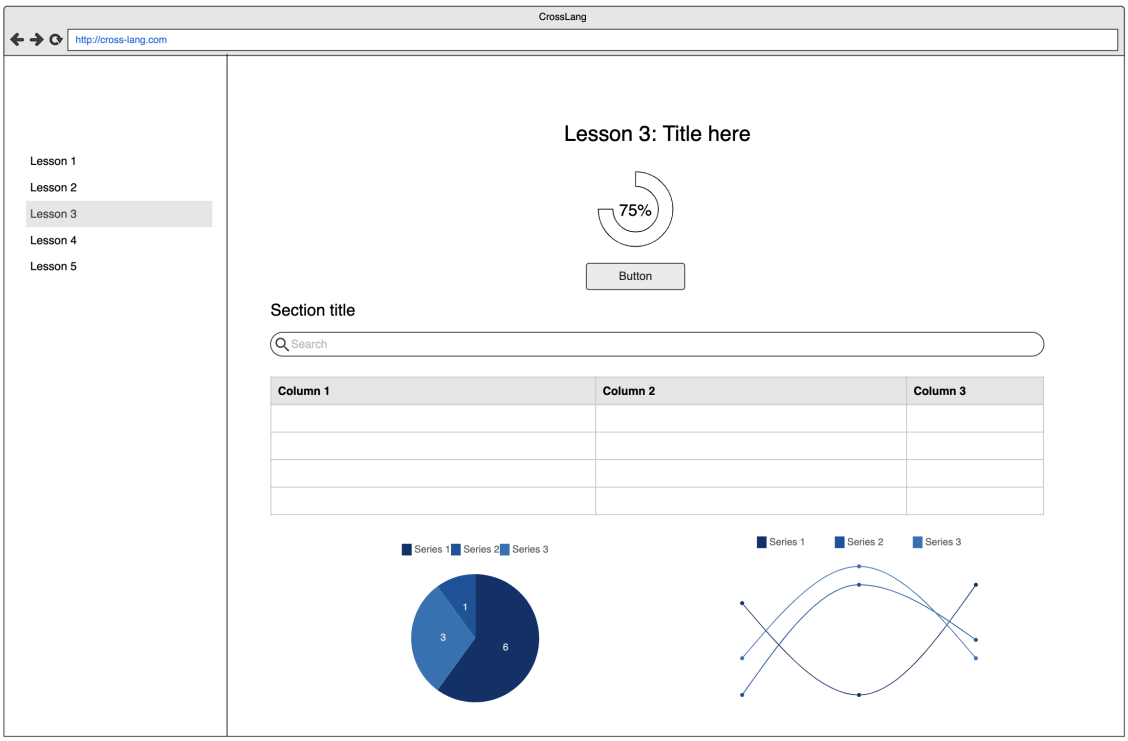

Figure 4.10: Exercise overview screen mockup

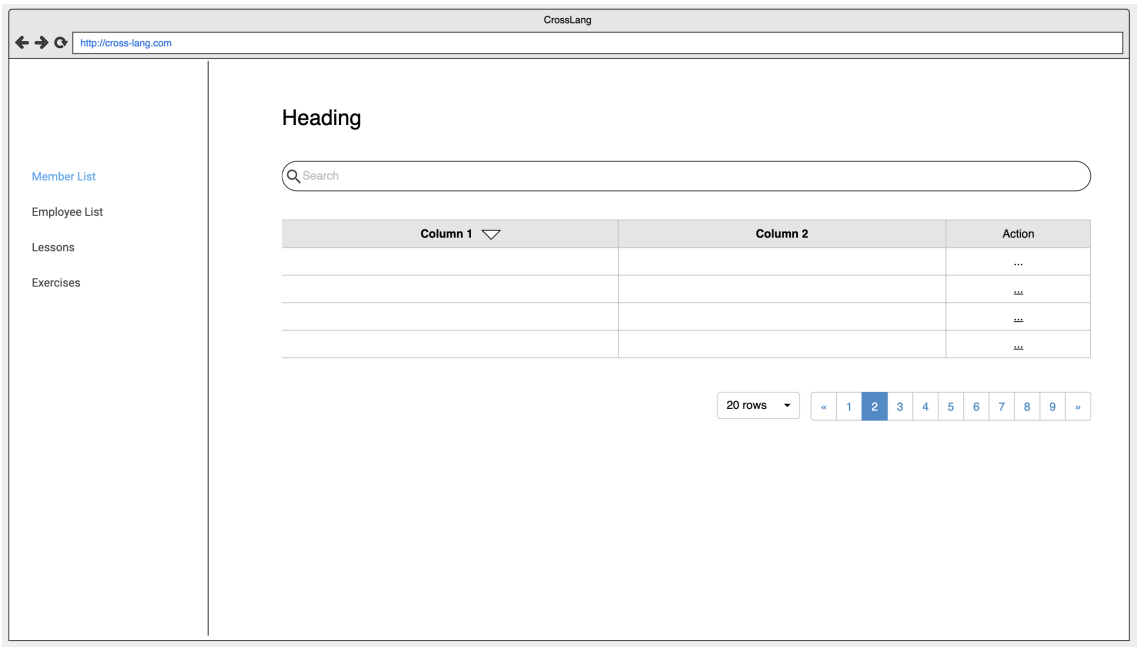

Figure 4.11: List for admin pages mockup

4.2.2 Layer design

In this section, we will have a proper look at three base classes that forms application's main flow.

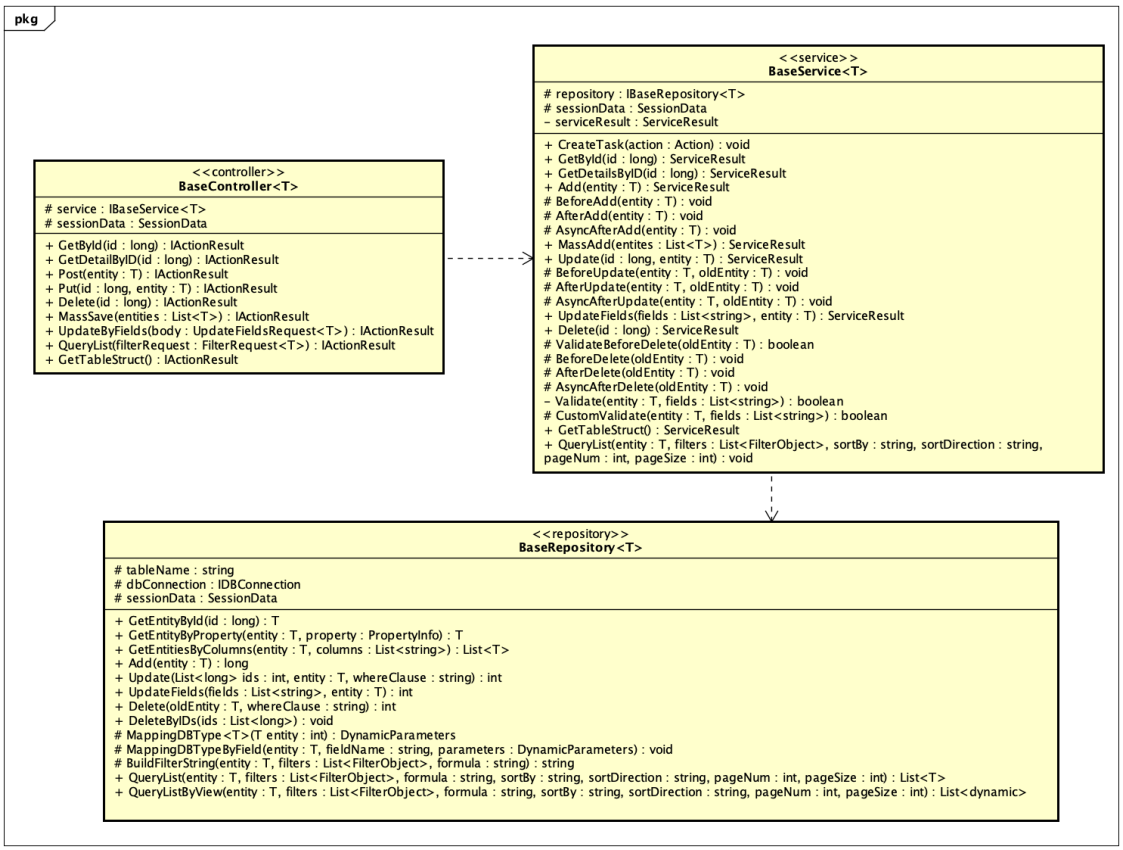

Figure 4.12: Base classes interactions

a, BaseController class

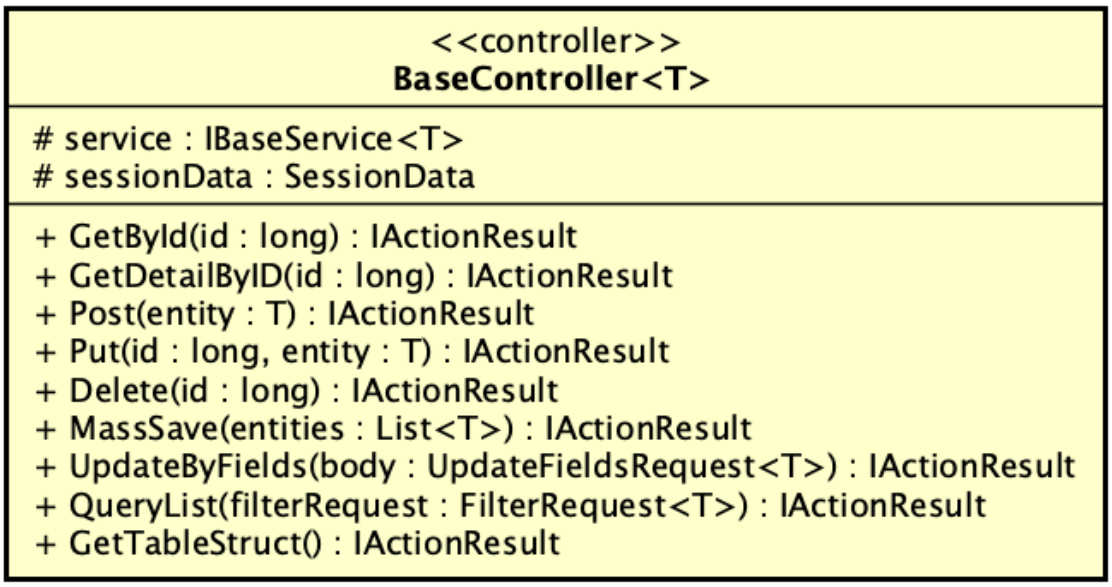

Figure 4.13: Design of BaseController class

Attributes

No	Name	Data Type	Description
1	service	IBaseService<T>	The service that processes business logic.
2	sessionData	SessionData	Contains authenticated user information

Table 4.1: Attributes of BaseController class

Operations

No	Name	Return Type	Description
1	GetById	IActionResult	Get record by id
2	GetDetailByID	IActionResult	Get record details by id
3	Post	IActionResult	Add new record
4	Put	IActionResult	Edit record
5	Delete	IActionResult	Delete record
6	MassSave	IActionResult	Add multiple record
7	UpdateByFields	IActionResult	Update data of fields
8	QueryList	IActionResult	Get list data
9	GetTableStruct	IActionResult	Get table structure for list

Table 4.2: Operations of BaseController class

Parameters

No	Name	Description
1	id	ID of record
2	entity	The information required of type T
3	entities	List of entity
4	body	Contains information of fields' name and their new value
5	filterRequest	information about filtering a list

Table 4.3: Parameters of BaseController class

b, BaseService class

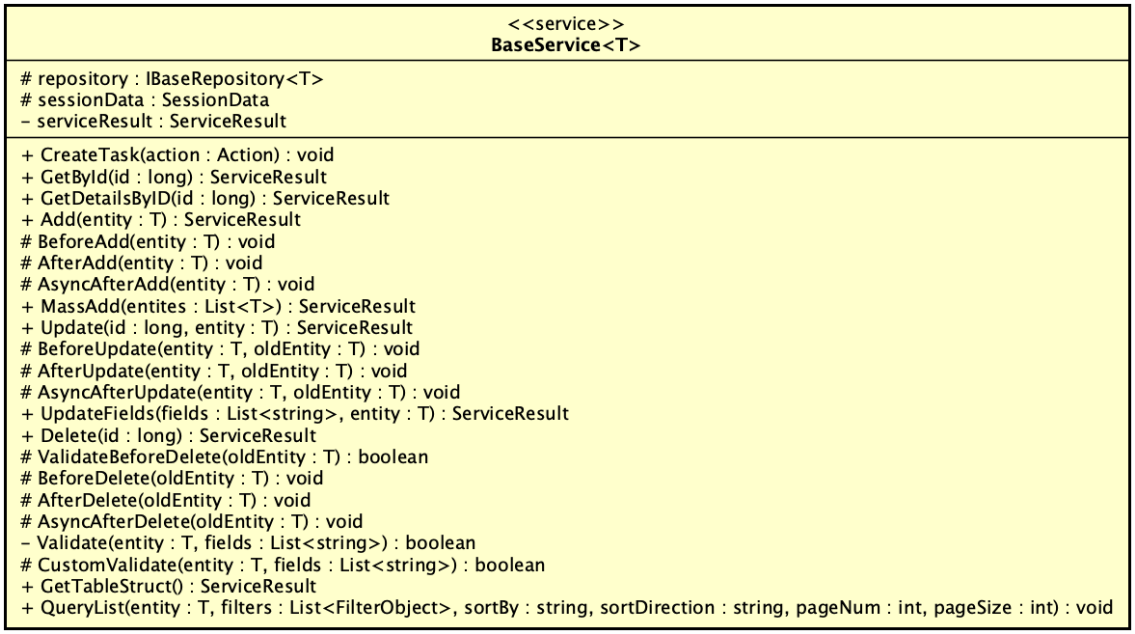

Figure 4.14: Design of BaseService class

Attributes

No	Name	Data Type	Description
1	repository	IBaseRepository<T>	The main repository that communicates with database.
2	sessionData	SessionData	Contains authenticated user information
3	serviceResult	ServiceResult	Response of the service

Table 4.4: Attributes of BaseService class

Operations

No	Name	Return Type	Description
1	CreateTask	void	Create new task with correct context
2	GetById	ServiceResult	Get record by id
3	GetDetailsById	ServiceResult	Get record details by id
4	Add	ServiceResult	Add new record
5	BeforeAdd	void	Process data before add
6	AfterAdd	void	Process logic after adding completed
7	AsyncAfterAdd	void	Process logic after adding completed asynchronously
8	MassAdd	ServiceResult	Add multiple records
9	Update	ServiceResult	Update record
10	BeforeUpdate	void	Process data before update
11	AfterUpdate	void	Process data before update completed
12	AsyncAfterUpdate	void	Process data before update completed asynchronously
13	UpdateFields	ServiceResult	Update fields
14	Delete	ServiceResult	Delete record
15	ValidateBeforeDelte	boolean	Check if record can be deleted
16	BeforeDelete	void	Process data before delete record
17	AfterDelete	void	Process data after delete record
18	AsyncAfterDelete	void	Process data after delete record asynchronously
19	Validate	boolean	Check if entity is valid
20	CustomValidate	boolean	To be overwritten
21	GetTableStruct	ServiceResult	Get table structure for list
22	QueryList	ServiceResult	Get record list

Table 4.5: Operations of BaseService class

Parameters

No	Name	Description
1	id	ID of record
2	entity	The information required of type T
3	entities	List of entity
4	action	Action to be executed
5	oldEntity	Entity before updated or deleted
6	fields	List of field
7	filters	Filter options selected
8	sortBy	Field to be sorted
9	sortDirection	Sort direction
10	pageNum	Page number
11	pageSize	Page size

Table 4.6: Parameters of BaseService class

c, BaseRepository class

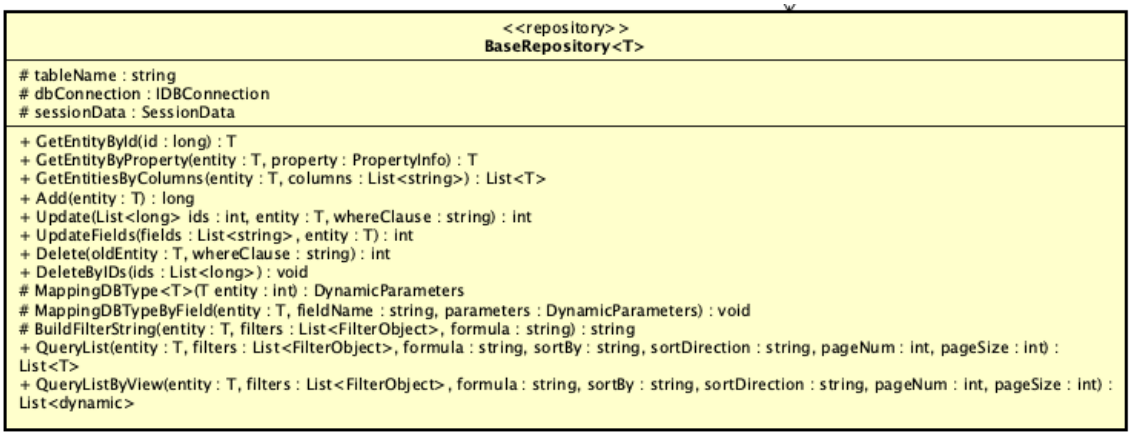

Figure 4.15: Design of BaseRepository class

Attributes

No	Name	Data Type	Description
1	repository	IBaseRepository<T>	The main repository that communicates with database.
2	sessionData	SessionData	Contains authenticated user information
3	dbConnection	IDBConnection	Connection to the database

Table 4.7: Attributes of BaseRepository class

Operations

No	Name	Return Type	Description
1	GetById	T	Get record by id
2	GetDetailsById	T	Get record details by id
3	GetEnitiesByColumns	List<T>	Get records by column conditions
4	QueryList	List<T>	Get list records
5	QueryListByView	List<dynamic>	Get list records by DB view
6	Add	long	Add new record
7	Update	int	Update record
8	UpdateFields	int	Update fields
9	Delete	int	Delete record
10	DeleteByIDs	void	Delete record by IDs
11	MappingDBType	DynamicParameters	Map query parameters with correct types
12	MappingDBTypeByField	void	Map query parameters with correct types of selected fields
13	BuildFilterString	string	Build where clause

Table 4.8: Operations of BaseRepository class

Parameters

No	Name	Description
1	id	ID of record
2	ids	ID list
3	entity	The information required of type T
4	entities	List of entity
5	oldEntity	Entity before updated or deleted
6	action	Action to be executed
7	fields	List of field
8	filters	Filter options selected
8	formula	Formula to combine filters
10	sortBy	Field to be sorted
11	sortDirection	Sort direction
12	pageNum	Page number
13	pageSize	Page size

Table 4.9: Parameters of BaseRepository class

To have better understanding about classes design, below are two sequence diagrams for the "Learn lesson" and "Do Exercise" flow.

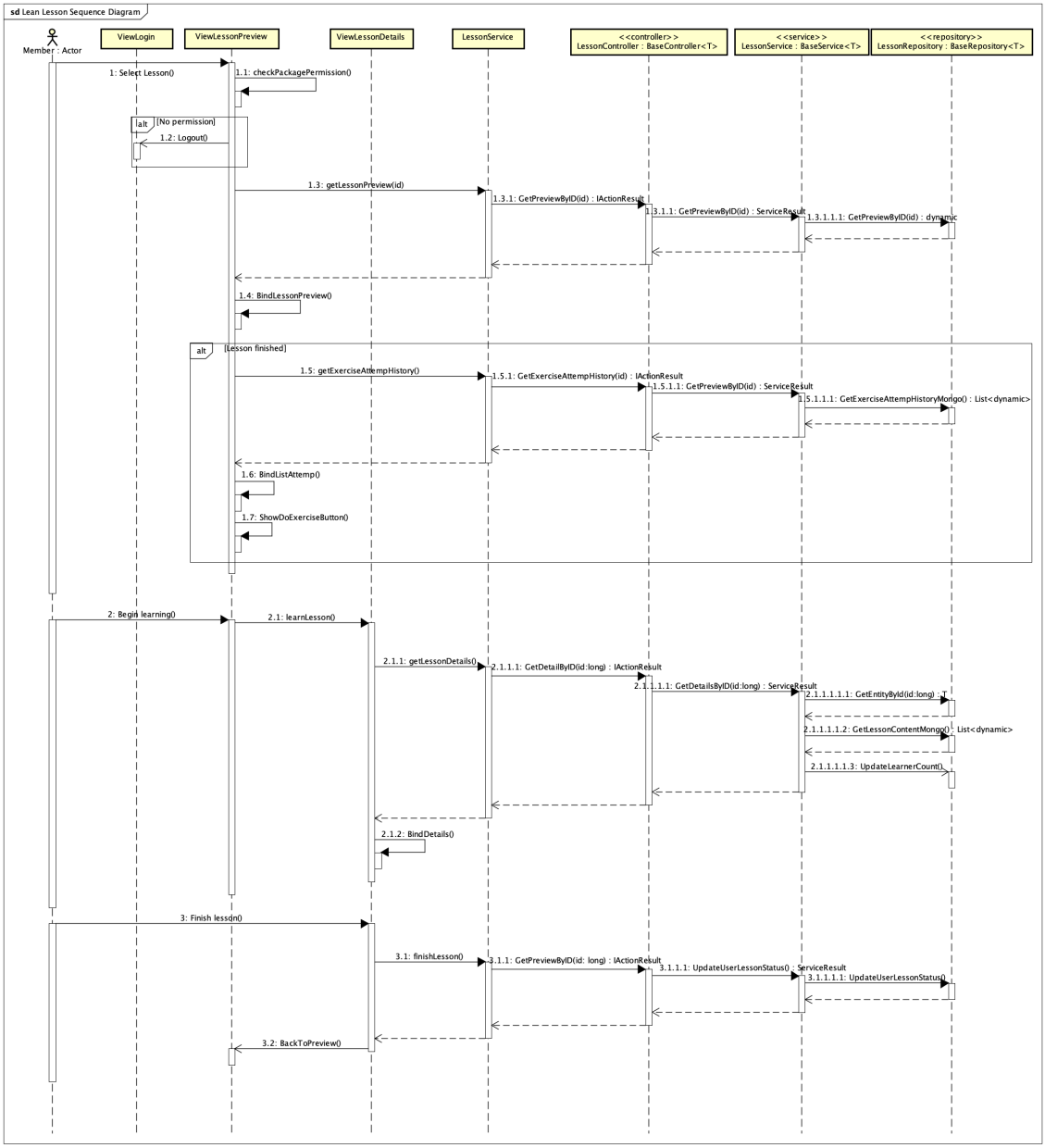

Figure 4.16: "Learn lesson" Sequence Diagram

Figure 4.16 describes the steps when a member learn a lesson. The data flow will move from client application, through API calls, accessing the server. The server transfers from LessonController to LessonService, then accesses the DB using LessonRepository.

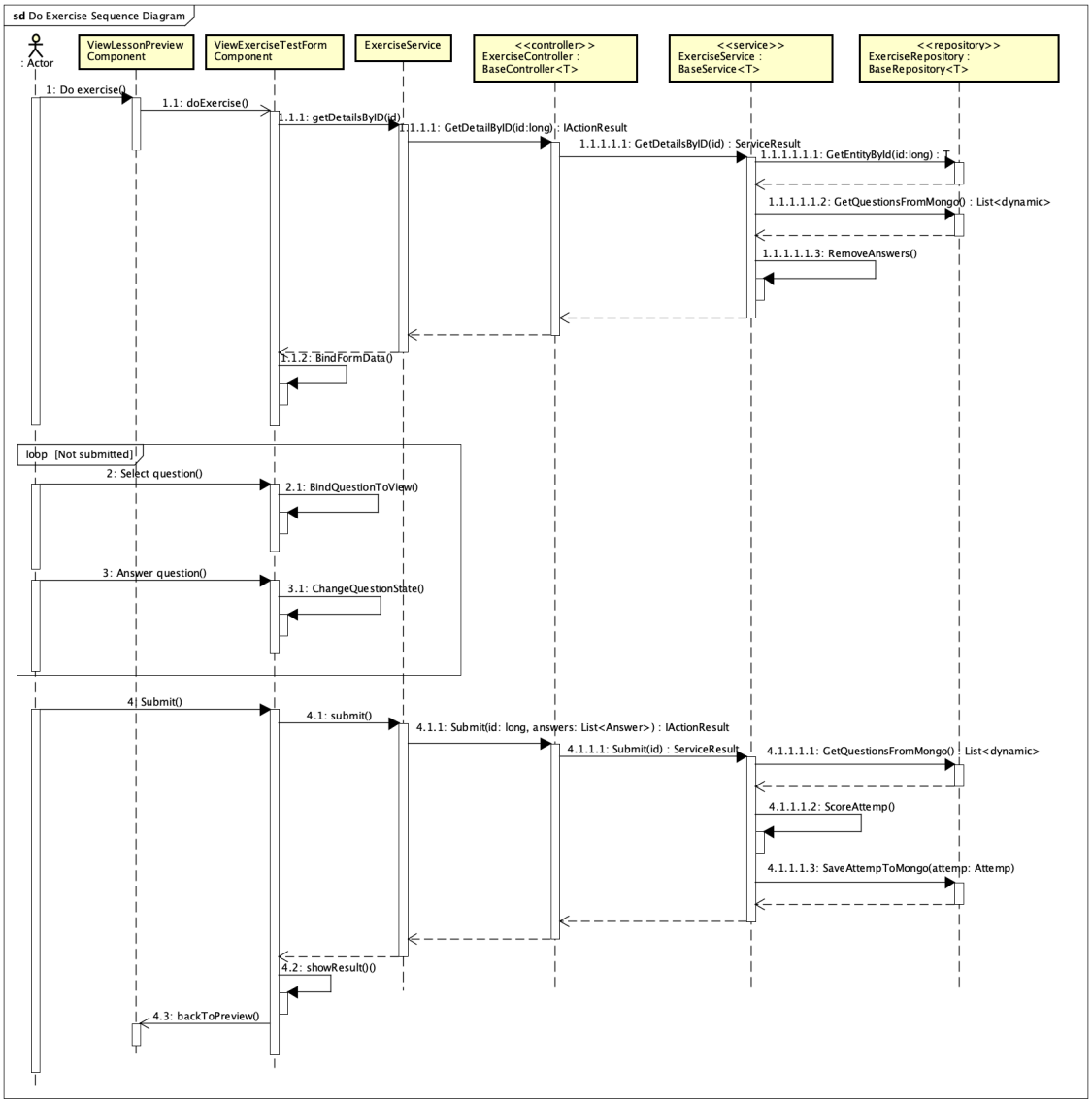

Figure 4.17: "Do exercise" Sequence Diagram

Figure 4.17 describes the steps when a member start doing an exercise.

4.2.3 Database design

a, Entity Relationship Diagram

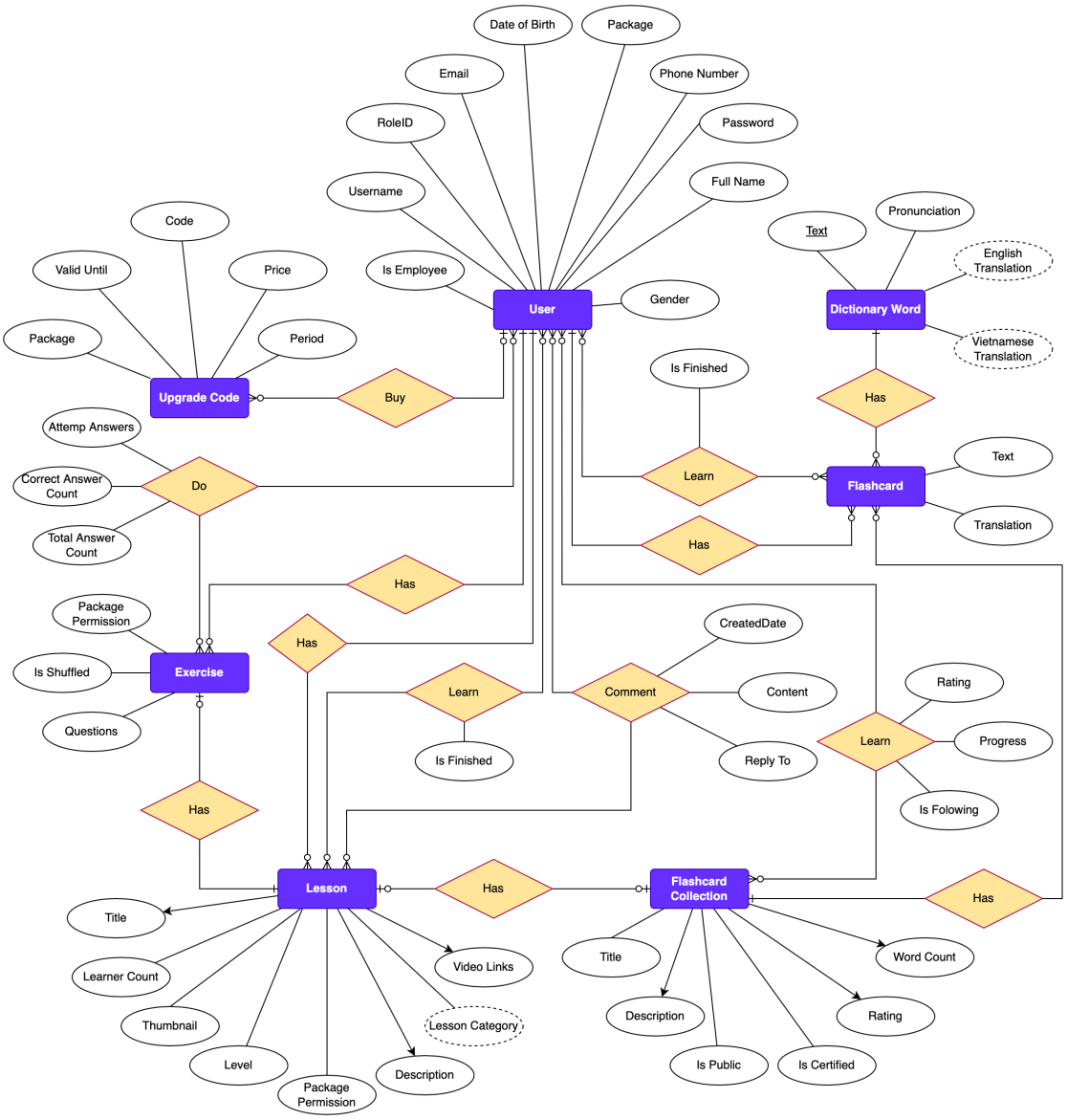

Figure 4.18: Entity Relationship Diagram

The object organizational structure of the system is represented by the E-R diagram.

An User can learn multiple Lesson, and vice versa; Each will create a corresponding record with finished value. A Lesson might contain an Exercise. When User do the exercise, the system also create an attempt record. An User with specified roles can create Exercises and Lessons. Users can also comment on multiple lessons.

Flashcard contains one Dictionary Word. A Flashcard can be created by only one User, and must belong to a Flashcard Collection. An user can learn many dif-

ferent flashcards, and a Flashcard might be learned by multiple Users. Therefore, the learning status of user with flashcards is stored separately in the system.

Flashcard Collection might be created manually by the User or automatically created when creating a Lesson. User learning Flashcard Collection is n-n relation, with the information of progress status.

An User can upgrade package by purchasing and redeeming Upgrade Code. Each upgrade code has the information of package level and valid period.

b, Database Design Implementation

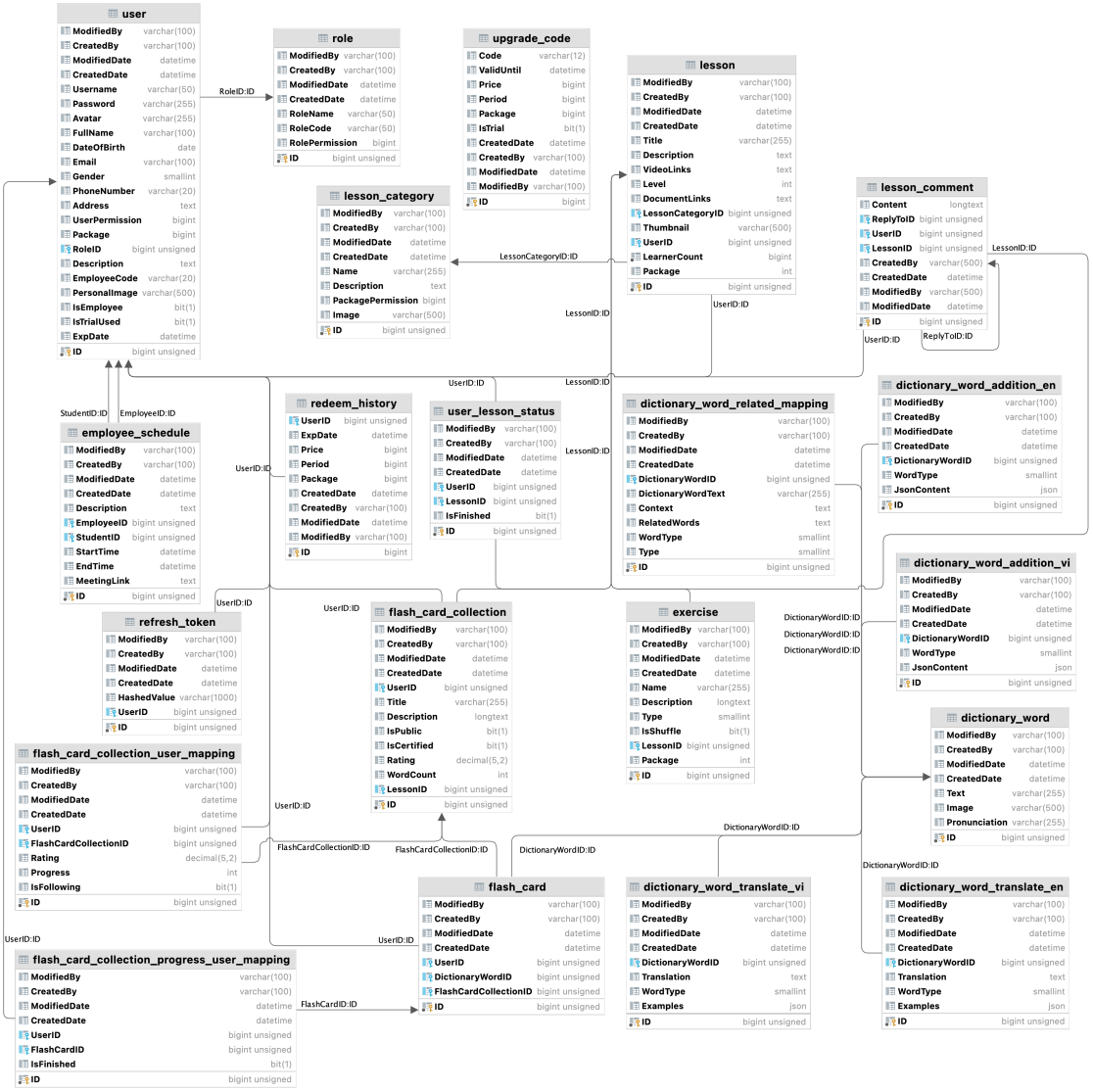

Figure 4.19: MySQL Database Design Implementation

From ER Diagram in figure 4.18, each entity, is given its own table. Each n-n relationship between entities is corresponding to a mapping table in the database,

while other relationships are stored as a attributes in either tables.

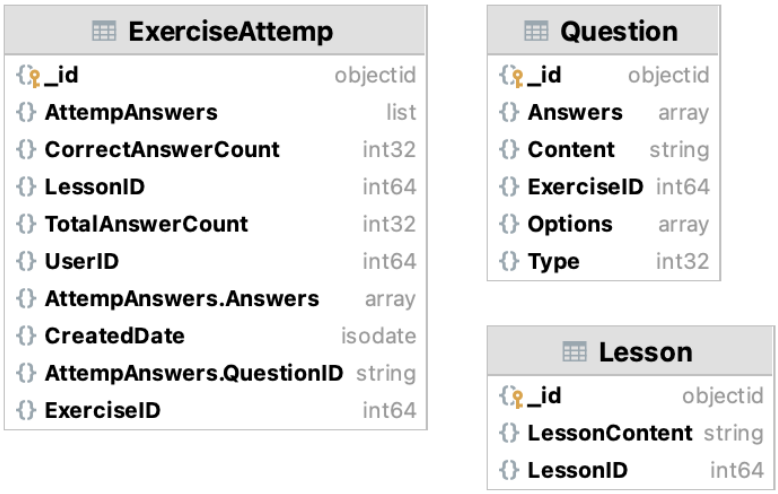

Figure 4.20: MongoDB Design Implementation

MongoDB (Figure 4.20) only contains 3 collection. The data stored in these collection are complicated and not suitable for using MySQL database. For example. a lesson's content include base64 encoded data of images, which can get over the maximum size of an MySQL TEXT variable. Therefore, storing those kinds of data in MongoDB will be a better option.

4.3 Application Building

4.3.1 Libraries and Tools

Tool	URL	Purpose
Visual Studio	https://code.visualstudio.com/	IDE
Visual Studio Code	https://visualstudio.microsoft.com/	IDE
Postman	https://www.postman.com/	API testing tool
Drawio	https://app.diagrams.net/	Diagram drawing tool
Astah UML	https://astah.net/	Diagram drawing tool
Angular 14	https://angular.io/	Front-end development framework
TypeScript	https://www.typescriptlang.org/	Programming language
@angular/material 14.0.2	https://material.angular.io/	UI library
angular-highcharts 14.1.5	https://www.npmjs.com/package-/angular-highcharts/	Charts drawing library
Quill 1.3.7	https://quilljs.com/	Rich text editor library
.NET 6.0	https://docs.microsoft.com/en-us/-dotnet/	Back-end development framework
Microsoft.Cognitive-Services.Speech 1.22.0	https://docs.microsoft.com/en-us/-azure/cognitive-services/speech-service/	Cognitive SDK
Newtonsoft.Json 13.0.1	https://www.newtonsoft.com/json/	JSON support for .NET
MongoDB.Bson 2.16.1	https://www.mongodb.com/docs/-drivers/csharp/	BSON support for .NET
MySQLConnector 2.1.10	https://mysqlconnector.net/	Library for MySQL connecting
MongoDB.Driver 2.16.1	https://www.mongodb.com/docs/-drivers/csharp/	Library for MongoDB connecting
Dapper 2.0.123	https://github.com/DapperLib/-Dapper	Library for Micro-ORM support
RabbitMQ.Client 6.4.0	https://www.rabbitmq.com/-dotnet.html	Library for connecting to RabbitMQ
MySQL 8.0.29	https://www.mysql.com/	Database
MongoDB 5.0.9	https://www.mongodb.com/	Database
RabbitMQ 3.10.5	https://www.rabbitmq.com/	Message broker
Github	https://github.com/	Source code repository

Table 4.10: List of tools and frameworks used

4.3.2 Achievement

The application was built from the start. Therefore, most features mentioned in previous sections are implemented by me. Due to the time limitation, the payment feature was postponed. The program will skip the logic of payment section. Chapter 5 will describe the implementations and constructions in detail.

The finished application is packaged together including client application, main server and workers.

Source code size	3.5GB
Compress size	908MB
No. packages of server	11

Table 4.11: Application Information

4.3.3 Illustration of main functions

a, Dictionary

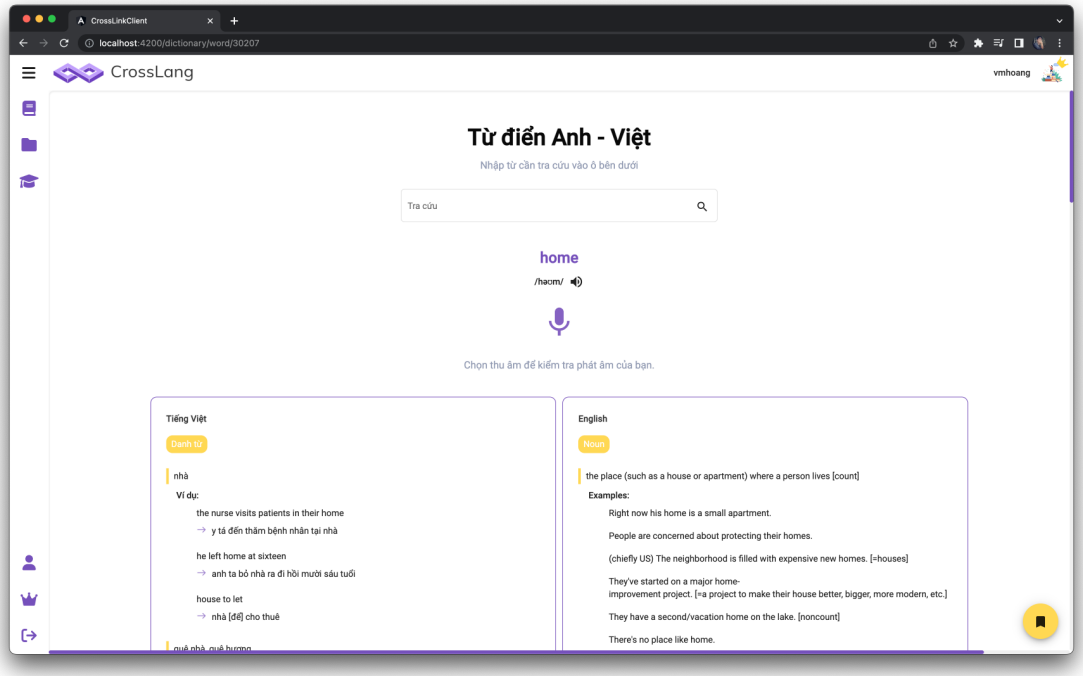

Figure 4.21: Dictionary screen

The Dictionary let user search English word and get result in Vietnamese and English. In addition, there is a float button at right bottom corner for quick adding new flashcard.

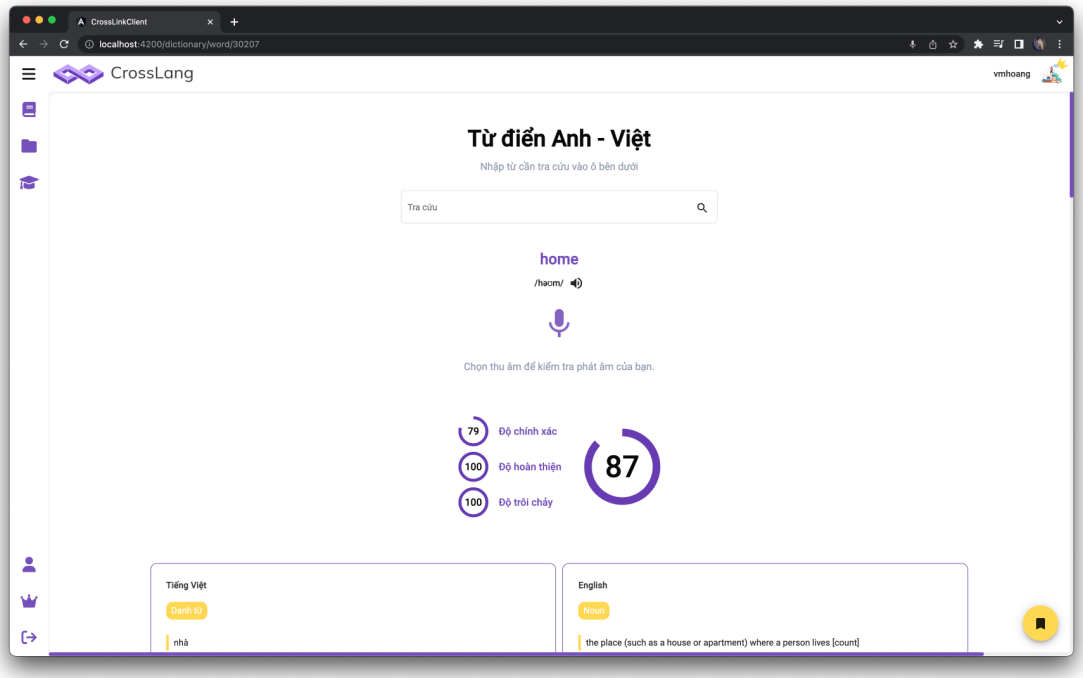

Figure 4.22: Dictionary pronunciation assessment screen

The system allow all-level user using dictionary pronunciation assessment feature. The result will be displayed as in figure above.

b, Flashcards

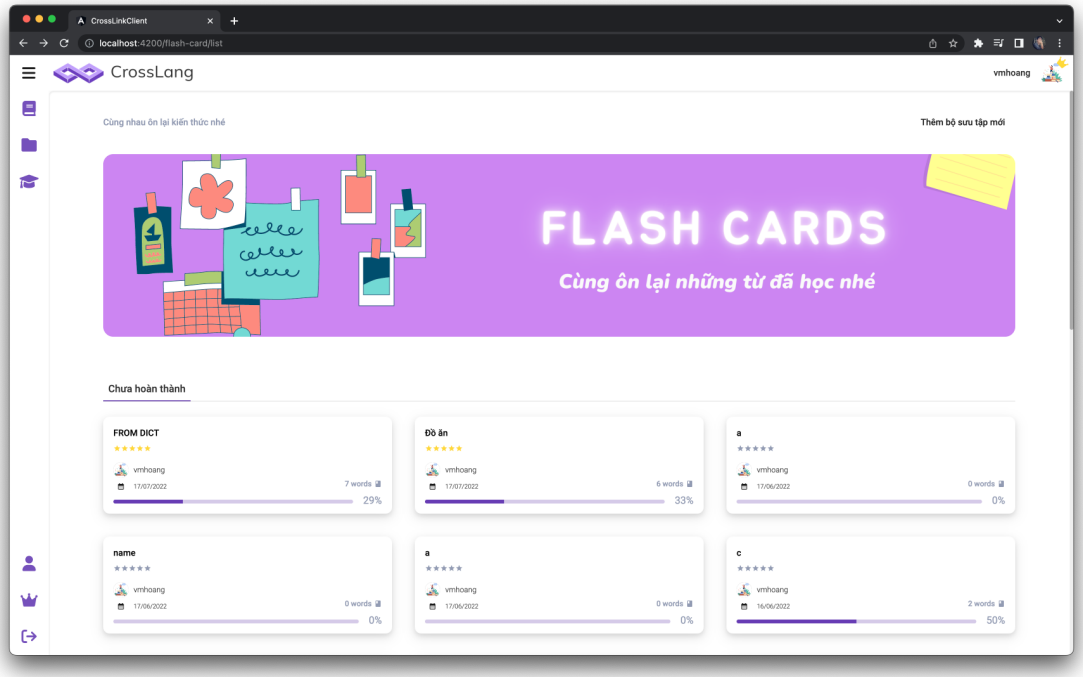

Figure 4.23: Flashcard collection list screen

The flashcard collection list screen contains both finished and unfinished collections. At this screen, users are allowed to create a new collection for themselves.

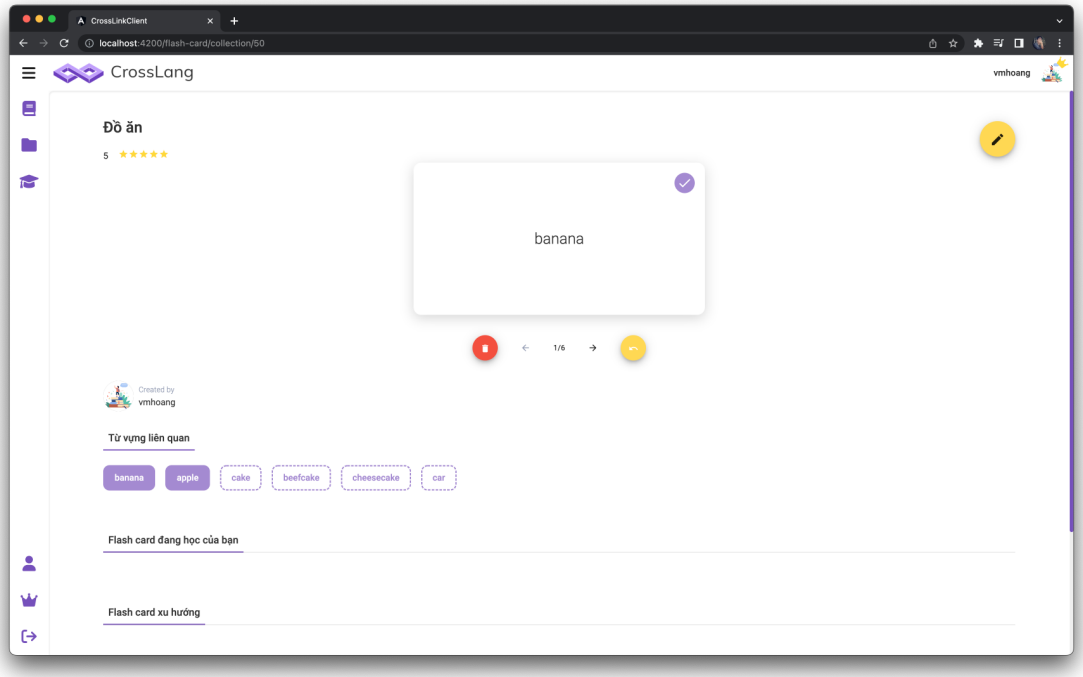

Figure 4.24: Flashcard collection details screen

Flashcard collection detail includes its flashcards. Users can mark a flashcard as completed or remove it. Users can also add new words to this collection.

c, Lessons

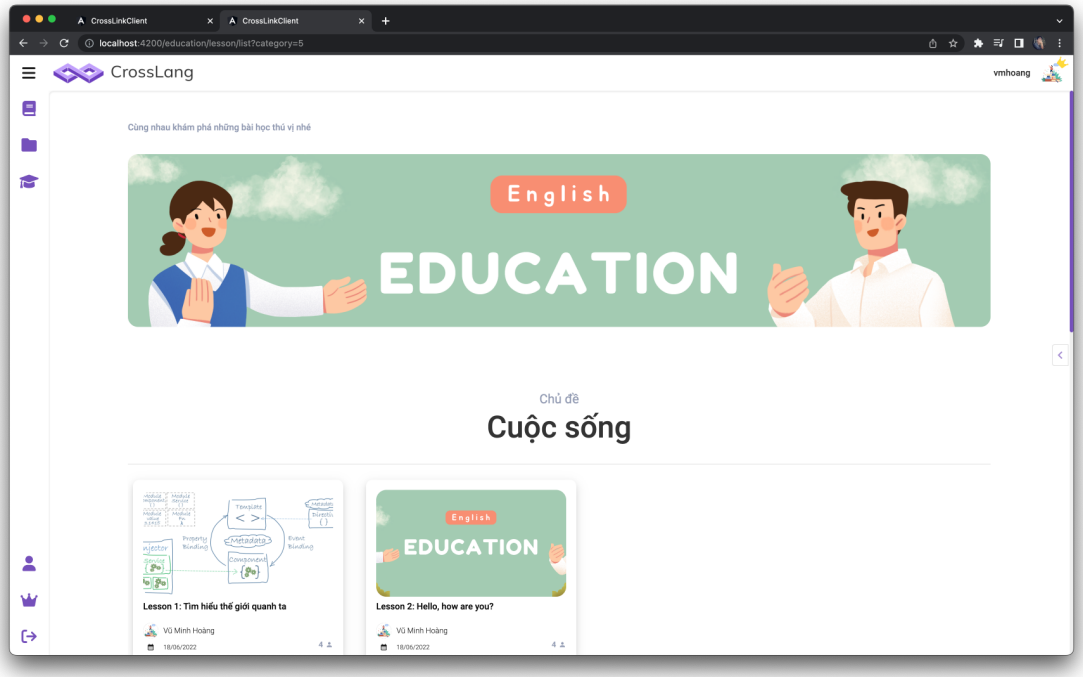

Figure 4.25: Lesson list screen

Lessons from different categories, in-progress and finished lessons are displayed here, in Lesson list screen.

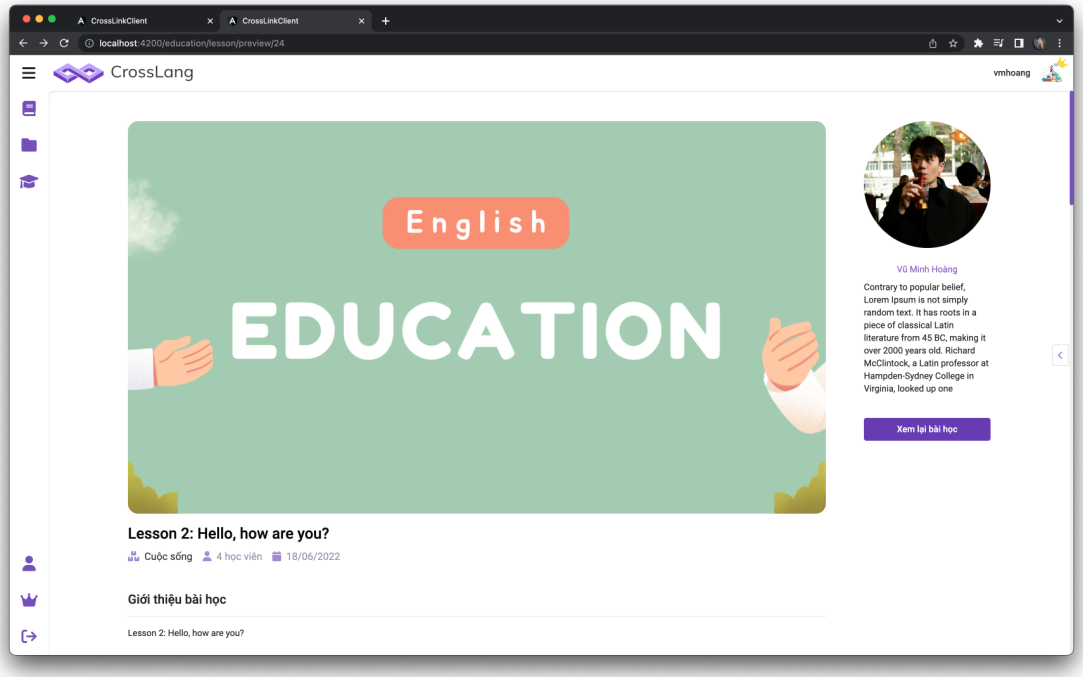

Figure 4.26: Lesson preview screen (1)

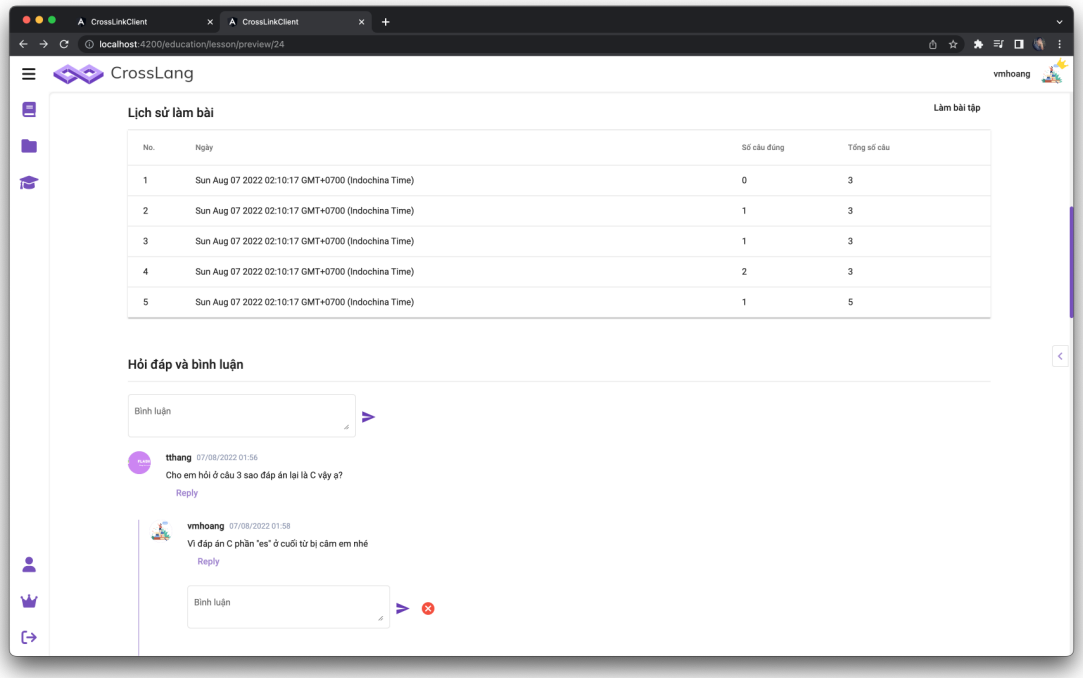

Figure 4.27: Lesson preview screen (2)

Lesson preview screen displays the overall information of the lesson and its creator. In addition, it shows attempt history and comment section for questions and answers.

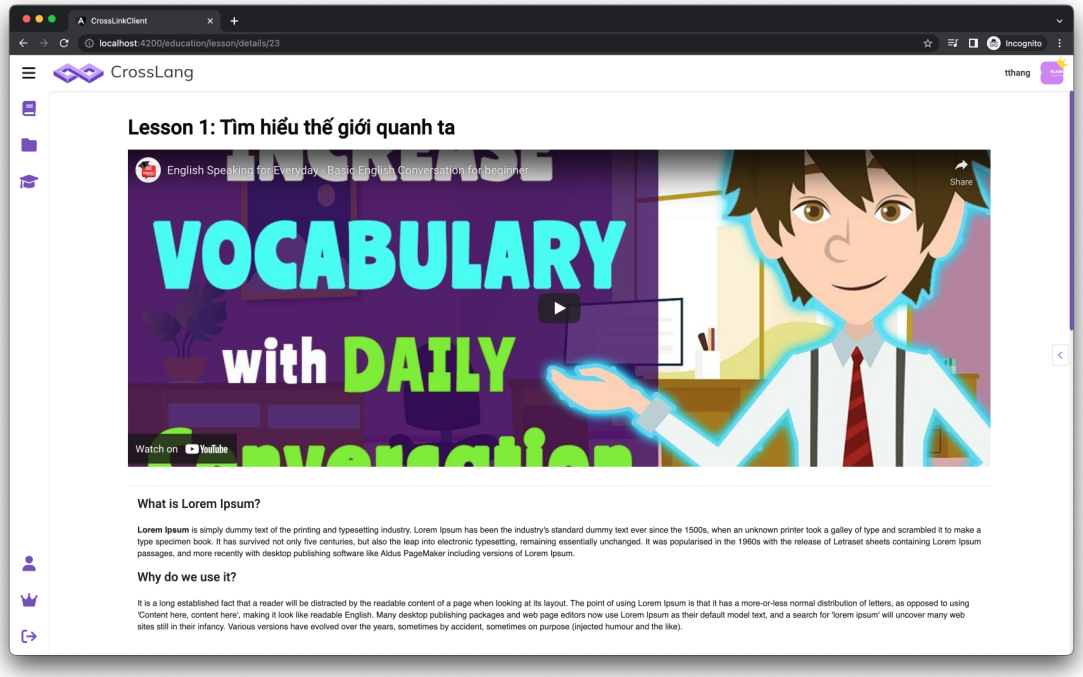

Figure 4.28: Lesson detail screen (1)

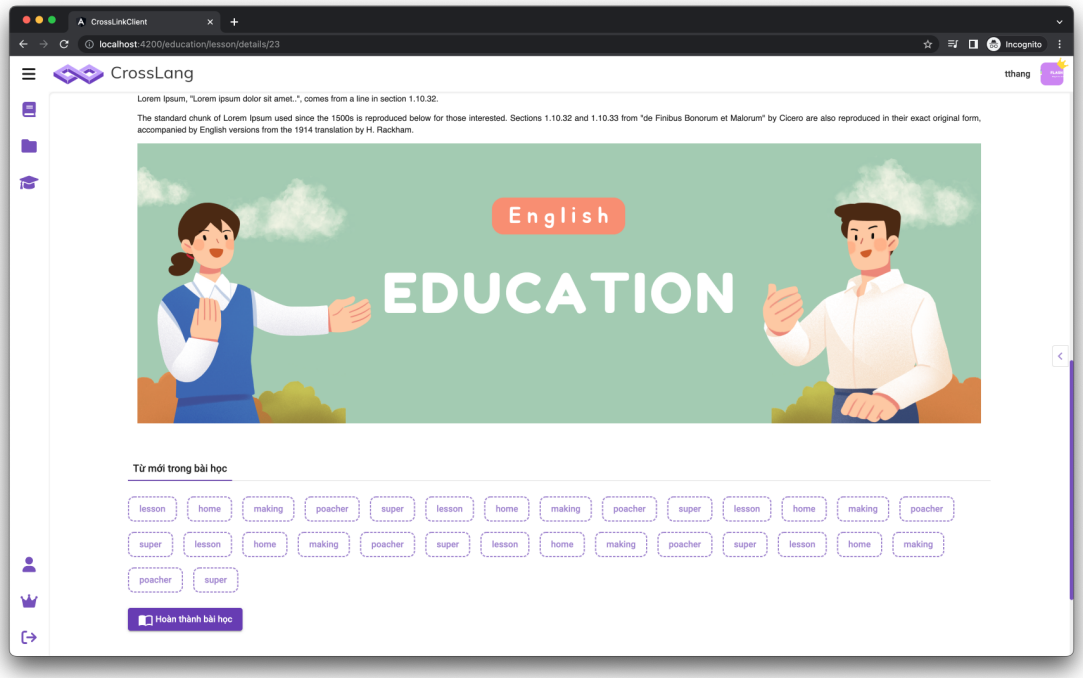

Figure 4.29: Lesson detail screen (2)

Lesson detail screen displays content of the lesson in detailed. It includes new words in the lesson and user can finish the lesson after learning.

d, Exercise

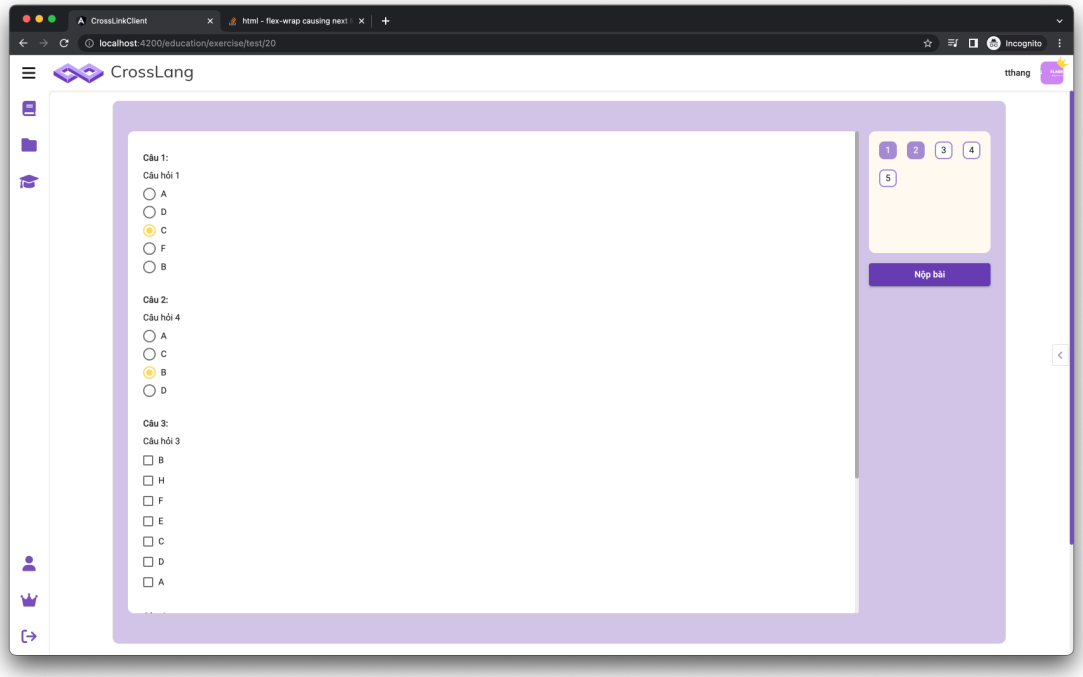

Figure 4.30: Do exercise screen

Do exercise screen has question palette on the right and the main content of exercise on the left. The system will display a dialog with the result of the exercise after submitting.

e, Upgrade Account

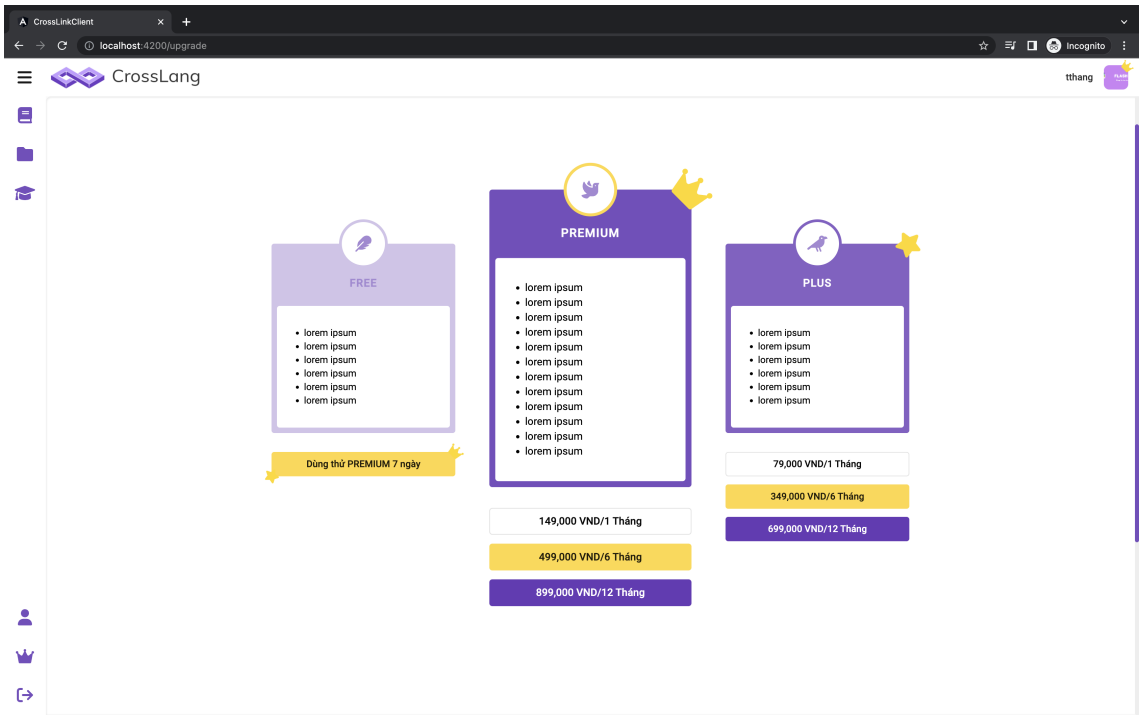

Figure 4.31: Upgrade screen

The upgrade screen displays three packages available in the system: Free, Premium and Plus.

f, Admin

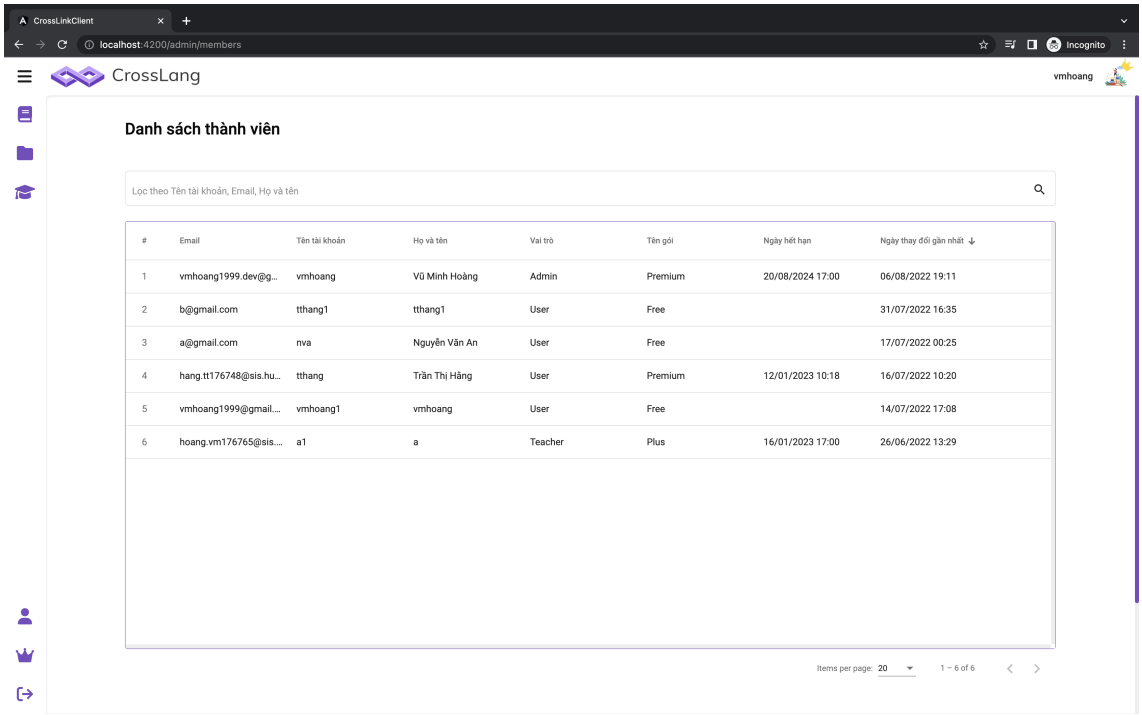

Figure 4.32: Member list screen

Member list is displayed in form of a table with search by username, email and full name function.

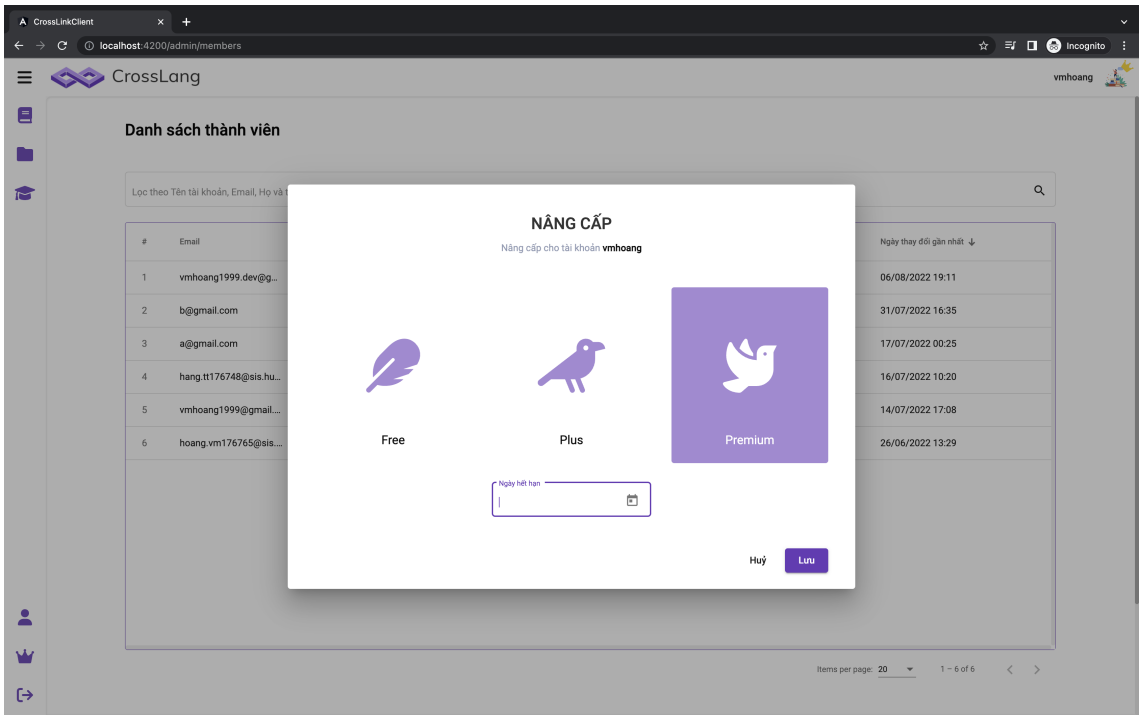

Figure 4.33: Upgrade package pop-up

Admin has permission to change users' package with an expiration date.

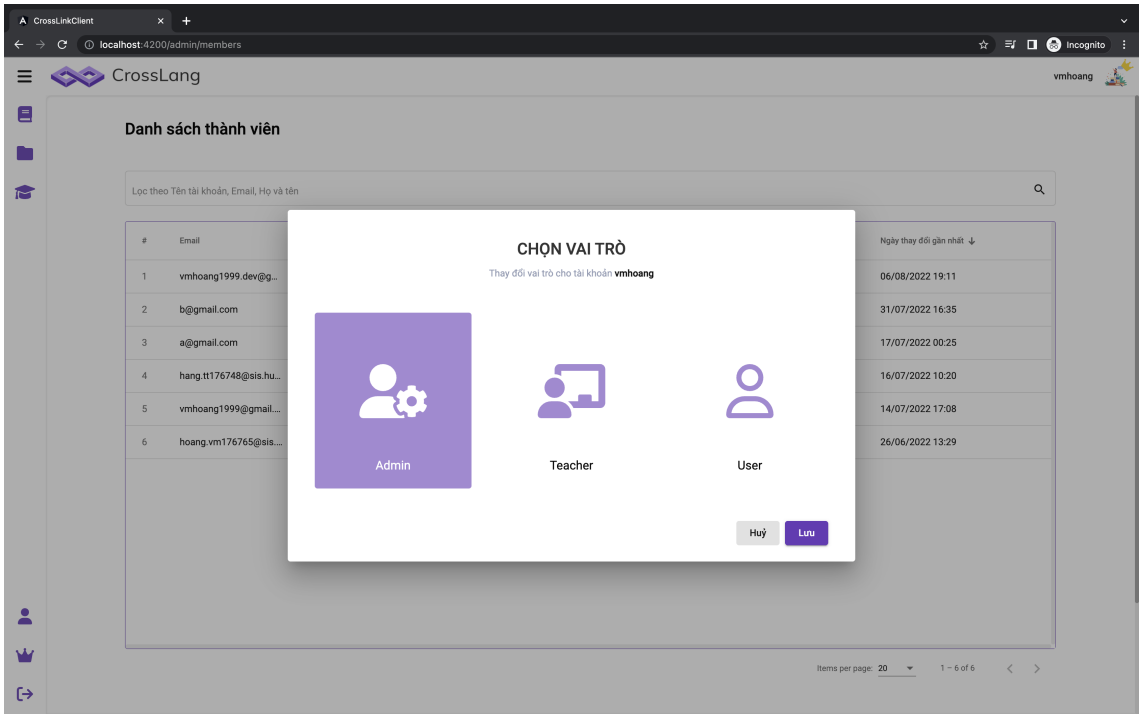

Figure 4.34: Upgrade role pop-up

Admin can also change users' role from basic user to teacher or admin.

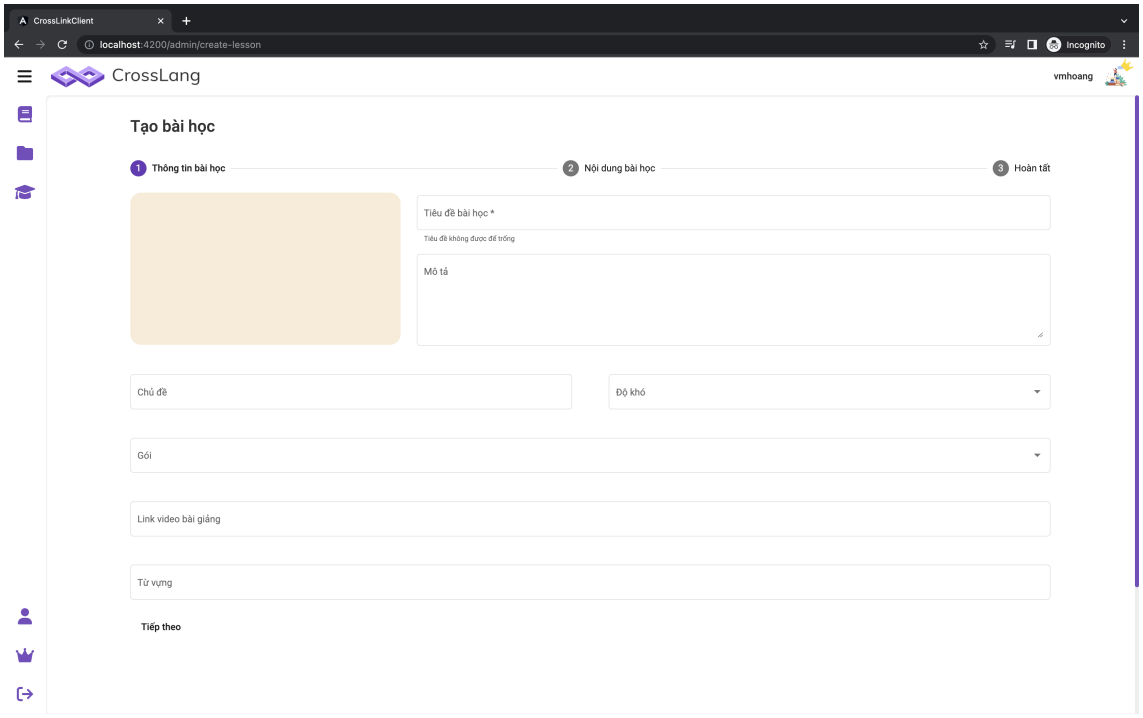

Figure 4.35: Create lesson screen (1)

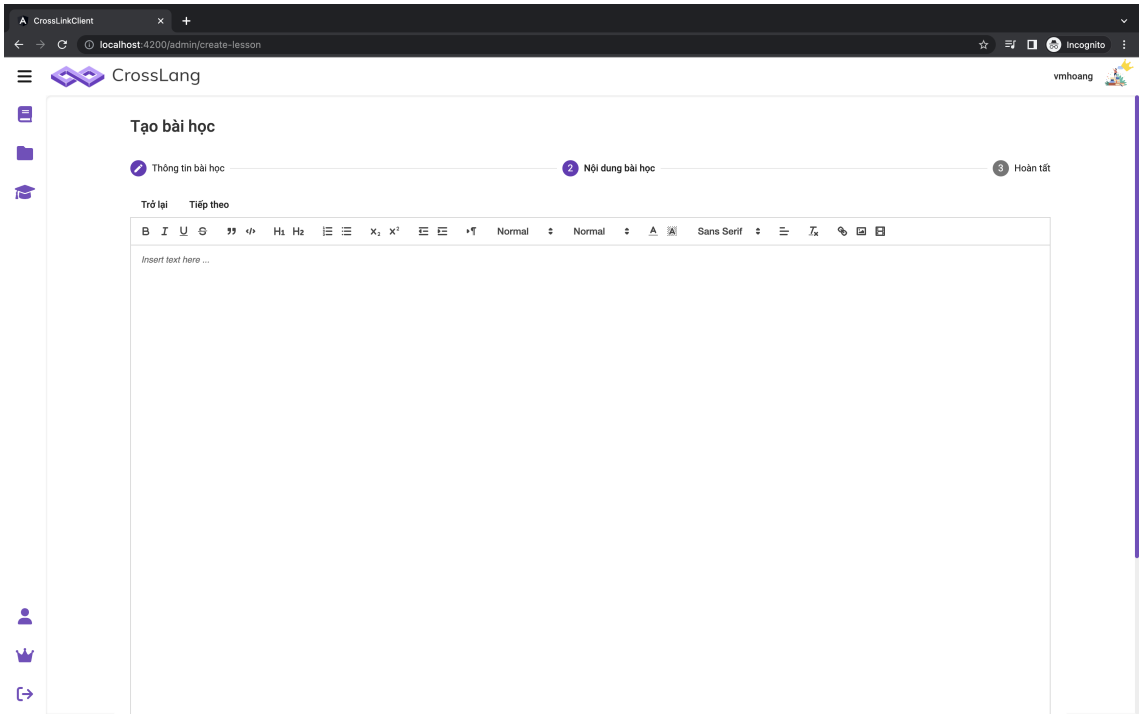

Figure 4.36: Create lesson screen (2)

Teachers and admins can create lessons by following two main steps: Fill out the (1) overall information of the lesson (figure 4.35) and (2) the lesson's content (figure 4.36)

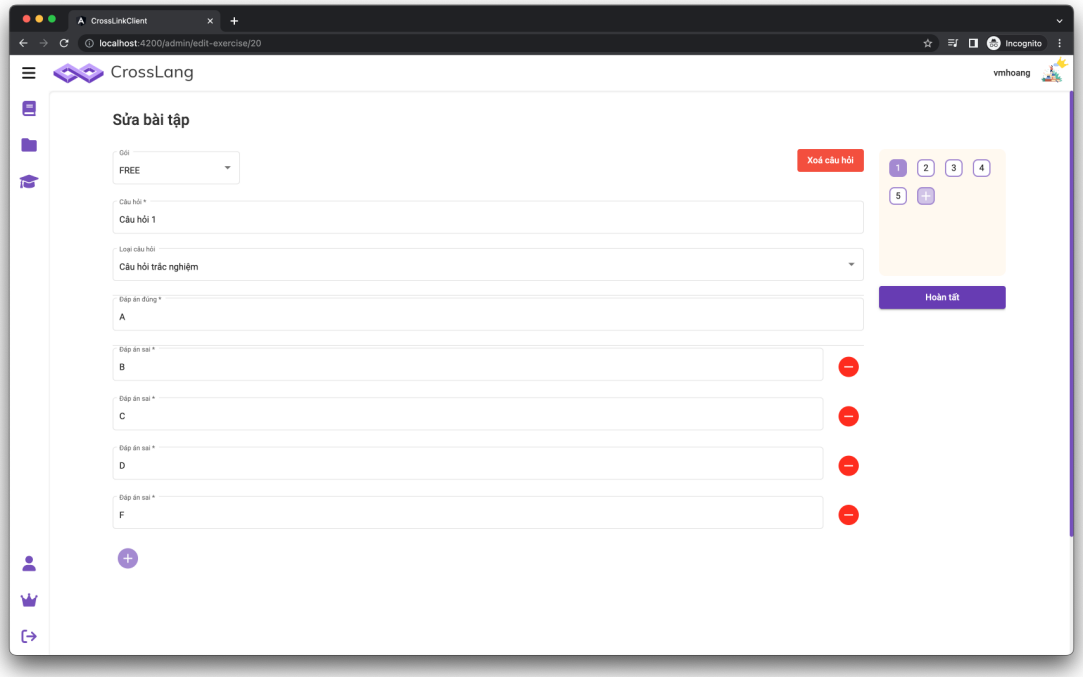

Figure 4.37: Create exercise screen

Teachers and admins can create exercises with three types: multiple choices with

one correct answer or multi-correct answers and fill in the blank.

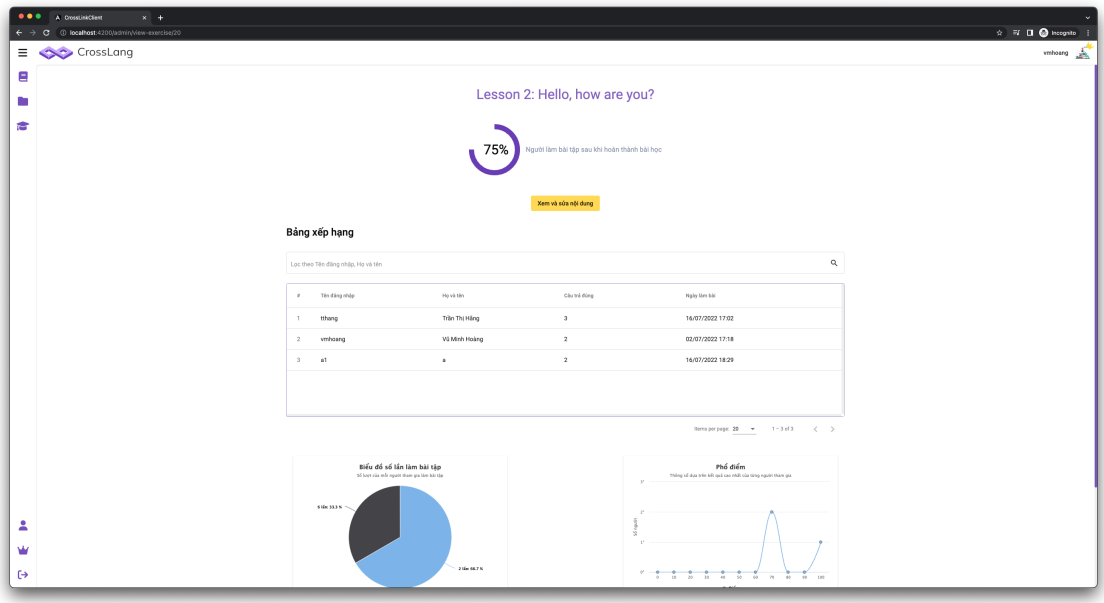

Figure 4.38: Exercise overview screen

Teachers and Admin can view an exercise statistics with these information: exercise score ranking, rate of exercise participants over all enrolled students, pie chart of retry rates and score spectrum.

4.4 Testing

To illustrate the testing processes, in this section, we will take a look at test cases of upgrading account and doing exercise functions.

4.4.1 Test cases for "Upgrade account"

Input data

No	Name	Value
1	User01	Username: vmhoang Package: Free Free trial used: False
2	User02	Username: tthang Package: Plus Free trial used: True

Table 4.12: Data for test case "Upgrade account"

No.	Input	Expected Result	Result
T01	Click "Upgrade account"	Redirect to pricing view	PASS
T02	User01 click "Free trial"	Open payment popup	PASS
T03	User01 types Card number: 1 EXP: 05/12 CCV: 123	Show invalid card warning	PASS
T04	User01 enters valid card and click "Accept"	Open payment gateway service in new tab	FAILED
T05	-	Show confirmation to the user	PASS
T06	User02 click "Free trial"	Show warning trial used	PASS
T07	User02 select "Premium 12 months"	open payment popup	PASS
T08	User02 enters valid card and click "Accept"	Show dialog email with code sent	PASS
T09	User02 check emails	System sent email with upgrade code	PASS
T10	User02 click "Redeem code"	Navigate to redeeming code page	PASS
T11	User02 types invalid code	Show warning	PASS
T12	User02 types received code	Show upgrade success	PASS
T13	-	Redirect to login page	PASS

Table 4.13: Test cases for "Upgrade account"

As mentioned in section 4.3.2, the integration with a payment gateway service was not implemented. Therefore, the test case T04 failed.

4.4.2 Test cases for "Do exercise"

Input data

No	Name	Value
1	User01	Username: vmhoang Package: Free Free trial used: False
2	User02	Username: tthang Package: Plus Free trial used: True
3	Lesson01	Package permission: Plus User01 unfinished User02 finished
4	Lesson02	Package permission: Free User01 finished User02 finished

Table 4.14: Data for test cases "Do exercise"

No.	Input	Expected Result	Result
T01	User01 clicks Lesson01's "Begin exercise"	Show permission warning	PASS
T02	User01 clicks Lesson02's "Begin exercise"	Show warning lesson not finished	PASS
T03	User02 clicks Lesson01's "Begin exercise"	Redirect to exercise form	PASS
T04	-	Questions of exercise displayed properly on the screen	PASS
T05	User02 answers a question	Light up corresponding question	PASS
T06	User02 removes or deletes the answer	Light down corresponding question	PASS
T07	User02 selects "Premium 12 months"	open payment popup	PASS
T08	User02 clicks "Submit"	Show confirm dialog	PASS
T09	User02 clicks "Confirm"	Show result popup	PASS
T10	-	Redirect to lesson preview screen	PASS
T11	-	Show new attempt in the history section	PASS

Table 4.15: Test cases for "Do exercise"

4.5 Deployment

The platform is deployed and tested using localhost environments.

For client-side, we ran the application for testing with instructions provided in the project's Github repository[1].

For server-side, first, we need to start the main server. For email and notification related features, we need to fire up our notifying and emailing workers. In addition, we build the service for checking accounts' expired date. All the source code for deploying can be found at the server's repository[2]. RabbitMQ for message broking message are setup using Docker image following the official RabbitMQ documentation [18]. Queue setup will be done automatically once the workers are started.

For database, we used MongoDB 5.0.9 and MySQL 8.0.29.

Our system was tested and ran properly on 2 configurations:

- Configuration 1:
  - Operating System: Windows 11 21H2

[1]https://github.com/mHoang99/CrossLang-Client
[2]https://github.com/mHoang99/CrossLang-Server

  - Processor: Ryzen 7 4800U (x86)
  - RAM: 16GB
  - Storage: 1TB
- Configuration 2:
  - Operating System: macOS Monterey 12.5
  - Processor: Apple M1 (ARM)
  - RAM: 16GB
  - Storage: 256GB

Chapter Summaries

This chapter has covered system architecture, application development, and deployment in depth. We also presented the components, use cases, user interface, database design, and tools we used during the software development life cycle. Lastly, we provided the project's achievements, some test cases, and application deployment information. The following chapter will summarize my contributions to this project, as well as specific challenges and solutions we used to confront them.

CHAPTER 5. SOLUTION AND CONTRIBUTION

According to the design and development of the application described in Chapter 4, in this chapter, we would like to discuss some challenging difficulties that we encountered while building the program, as well as the remedies we devised to increase product quality and user experience efficiency.

5.1 Designing and implementing the base classes for main server

5.1.1 Problem

From the chapter 4, we have mentioned about how the system implements the Clean Architecture, and how the interfaces can help our application easy to maintain, thus expandable. As the application grow, there will be similar business logic that needs to be executed similarly, but not in the exact same way. Base classes are used in Object Oriented Programming (OOP) to take advantages of the Inheritance mechanism. Many class can derived from the base class, and inherit all the public and protected properties and methods from the parent class. One question is how we can get base classes that can cover the basic flows, without needing the child classes to overwrite completely the related methods in order to modify the business logic. Another one is about how we can optimize and prevent performance penalties, since the performance of base classes affects heavily to the whole system.

5.1.2 Solution

Normal base classes, even though can deal many different cases, are not very flexible. My system is built on an entity oriented basis. Therefore each class usually attached directly to an Entity type. For example, the LessonService class is responsible for executing lesson related business logic. If we use a normal base class, it is nearly impossible to deal with all entity types, which make the class much less useful and inheritable.

Therefore, we decided to implements base classes by using generic class. With this approach, all the methods of base classes now can be abstracted through a generic class **T**. At this point, our base classes can used with any Entity types, as long as it is inherited from the BaseEntity class.

Below is an example of applying generic principle into implementing the base classes.

```
/// <summary>
/// Base Service
/// </summary>
/// <typeparam name="T"></typeparam>
/// CREATEDBY: VMHOANG (25/02/2022)
public class BaseService<T> : IBaseService<T> where T : BaseEntity
{
    #region Fields
    /// <summary>
    /// repository kết nối DB
    /// </summary>
    protected IBaseRepository<T> _repository;
    protected IHttpContextAccessor _httpContextAccessor;
    protected SessionData _sessionData;


    #endregion

    #region Properties
    protected ServiceResult serviceResult;

    #endregion

    #region Constructor
    public BaseService(IBaseRepository<T> repository,
        IHttpContextAccessor httpContextAccessor,
        SessionData sessionData)
    {
        try
        {
            _repository = repository;
            _httpContextAccessor = httpContextAccessor;
            _sessionData = sessionData;
            serviceResult = new ServiceResult
            {
                SuccessState = true,
            };
        }
        catch (Exception ex)
        {
            throw new ServiceException(
                ex.Message,
                UnexpectedErrorResponse(ex.Message)
            );
        }
    }
    #endregion
```

Figure 5.1: Generic class example

```
public ServiceResult Add(T entity)
{
    try
    {
        entity.EntityState = Enums.EntityState.ADD;

        //validate entity
        var isValid = Validate(entity);
        if (!isValid)
        {
            return serviceResult;
        }

        BeforeAdd(ref entity);

        serviceResult.SuccessState = true;
        var newID = _repository.Add(entity);
        serviceResult.Data = newID;

        entity.ID = newID;

        AfterAdd(ref entity);

        AsyncAfterAdd(entity, _httpContextAccessor.HttpContext);

        //Không tác động được bản ghi
        if (int.Parse(serviceResult.Data.ToString()) <= 0)
        {
            serviceResult = RowAffectingUnexpectedFailureResponse();
        }

        return serviceResult;
    }
    catch (Exception ex)
    {
        throw new ServiceException(
            ex.Message,
            UnexpectedErrorResponse(ex.Message)
        );
    }

}
```

Figure 5.2: Generic class method example

In the example is the Add method of BaseService class. If we want to declare a new service class, LessonService for example, we will let the new class extends BaseService<Lesson>, with the Lesson class is a child of BaseEntity. At this point, the Add method will take Lesson instead of T for executing, which is suitable for our requirements.

Taking in the Add methods again, even though code logic for saving a new instance to our database might be the same across the application, business logic might be different. For example, validation before adding an lesson and an exercise

have both similarities and differences. The answer is straightforward: inside base class methods, there will be custom methods that allow child classes to override, allowing them to adjust the flow without completely rebuilding those routines. Child classes can use the BeforeAdd, AfterAdd, and AsyncAfterAdd methods to customize the Add record flow. These approaches assist to dramatically boost development speed, when a fully new business requirement can be satisfied simply by customizing and overwriting methods.

Restful APIs are provided by our server to the client application. A request with slow response time not only degrades the user experience, but it also has a significant impact on server resources. The longer a request takes to execute, the more CPU, RAM and other resources will be used, which decreases the number of request can be handled at once by the server.

For requests that used to get infrequently changed data, we implemented caching. By using caching, server could respond instantly without querying the database or executing slow procedures. Using caching also mean that we have to clear and update the cache properly, ensure the integration of data.

Not every business logic requires real-time execution. Add new tasks for non-real-time requirements to minimize server response time. For example, after creating a lesson, we might wish to make a flashcard collection based on that lesson. Because the information used to create the flashcard collection is not necessary for the response, this operation may be classified as non-real-time. As a result, performing this in a separate task and allowing the response to be provided to the client significantly increases server response time. Functions like AsyncAfterAdd or AsyncAfterUpdate called inside a separate task, let the child classes to do things outside and not affecting the main flow.

5.1.3 Result

The server code is structured in a clean, extendable, and maintainable manner. With the addition of generic classes, our system has become entity-centric. A new business flow can be built by simply extending and overwriting the methods of base classes.

Caching and asynchronous processing aid in reducing server response time, particularly for slower queries, resulting in a substantially better experience for consumers.

5.2 Implementing authentication

5.2.1 Problem

Identifying users across courses is required in a learning system. Json Web Tokens (JWT) are one of the most widely used methods of authentication. The system may authenticate the user using JWT without storing the token itself. This also implies that once an access token is produced, anyone having that token gets full access to the system as the original user until it is expired. This increases the danger of losing a token to attackers.

5.2.2 Solution

A possible way is to create a blacklist of tokens on the server. The main advantage of JWTs is that they speed up session verification. We will lose that benefit if we keep a blacklist/deny list and have to query it with every API call.

Another solution, which is implemented in for my system, is to use short life-span access tokens and refresh tokens. To overcome the risk of losing access token, they will have short life-spans. Therefore, if an attacker can get access to an user's access token, it will be invalidated automatically in a very short amount of time. However, using a short life-span token alone will cause the client application to be unauthorized constantly, and the user will have to re-login over and over again to get new access token.

A refresh token, which is stored in the server, is used for generating new access token. At this point, once the old access token becomes invalid, a refresh token will be sent to the server in order to getting the new access token, and this process will repeat constantly until the user has logged out. Comparing When Comparing storing refresh tokens and the access token blacklist, the blacklist must be accessed for every request to the server, whereas the refresh token table only needs to be accessed for generating new access tokens, retaining all of the advantages in terms of both security and speed of access tokens. A refresh token could also be invalidated easily by deleting it from the server.

Implementing JWT authentication with refresh token, the server generates and returns the access token along with the refresh token when user first login. Refresh tokens are stored inside MySQL database. Client will use the access token to communicate with the server, until it get the response status of 401 - Unauthorized. At this stage, the response goes through a network interceptor. The interceptor take the refresh token, make a request to the server in order to get new access token, then try again with the unauthorized request. If the refresh token is invalid, the server will response with status 401, and user will be logged out.

```
catchError(error => {
    if (error.status === 401) {
        return this.refreshToken().pipe(
            exhaustMap((res) => {
                console.log(res);

                const userData = JSON.parse(localStorage.getItem('userData'));

                this.authService.handleAuthentication(
                    userData.email,
                    userData.id,
                    userData.username,
                    userData.employeeId,
                    userData.avatar,
                    res.Data.AccessToken,
                    userData._refreshToken,
                );

                let newModifiedReq = req.clone(
                    {
                        //Thêm auth vào header
                        headers: req.headers.append(
                            'Authorization',
                            `Bearer ${res.Data?.AccessToken}`
                        ),
                        //Thay url
                        url: req.url.includes("https://") ? `${req.url}` : `${this.baseUrl}${req.url}`
                    });
                return next.handle(newModifiedReq)
            }),

            catchError(e => {
                if (e.status == 401) {
                    this.authService.logout();
                }
                return throwError(e);
            }));
    }
```

Figure 5.3: Client request interceptor handling

5.2.3 Result

I successfully integrated JWT authentication with refresh token to my system, increase the system security, while keeping all the advantages of using access tokens.

5.3 Finding data for the English-Vietnamese dictionary

5.3.1 Problem

As mentioned in section 2.1, to overcome the weaknesses of some existing products on the market, our application will provide an English - Vietnamese dictionary. The persist problem is how we can find words for our dictionary.

5.3.2 Solution

We may incorporate the search engine of free dictionaries into our programs. Although taking this route is quick and easy, it eliminates our ability to connect our features to the lexicon. The data that is not saved in our system is tough to modify for other features.

The ideal situation would be to store all of the structured data for terms in our database. There are suppliers who will let us use their dictionary data for our application in exchange for a charge. That approach is suitable from a functional and legal standpoint. However, for the scope of this thesis, a workaround would be to collect data from various online sources and use it to create a sample dictionary for the system.

First, we need to collect the list of available English words. After researching, we found a free source code from Github[19], which give us a list of **370101** English words.

Then, we used Python and Scrapy for getting data for words inside the list. We tried scrapping data on 2 dictionary website: Laban Dictionary and VDict. The common characteristic of this website is that they do not use Restful API to fetch word data requests to the server, but using server-side rendering. Therefore, we can only access the data through the HTML rendered on the browser. The spider using to scrap these 2 website determine words mostly based on HTML classes attached to each elements. By analyzing and experiencing, we gathered words into a JSON file as following.

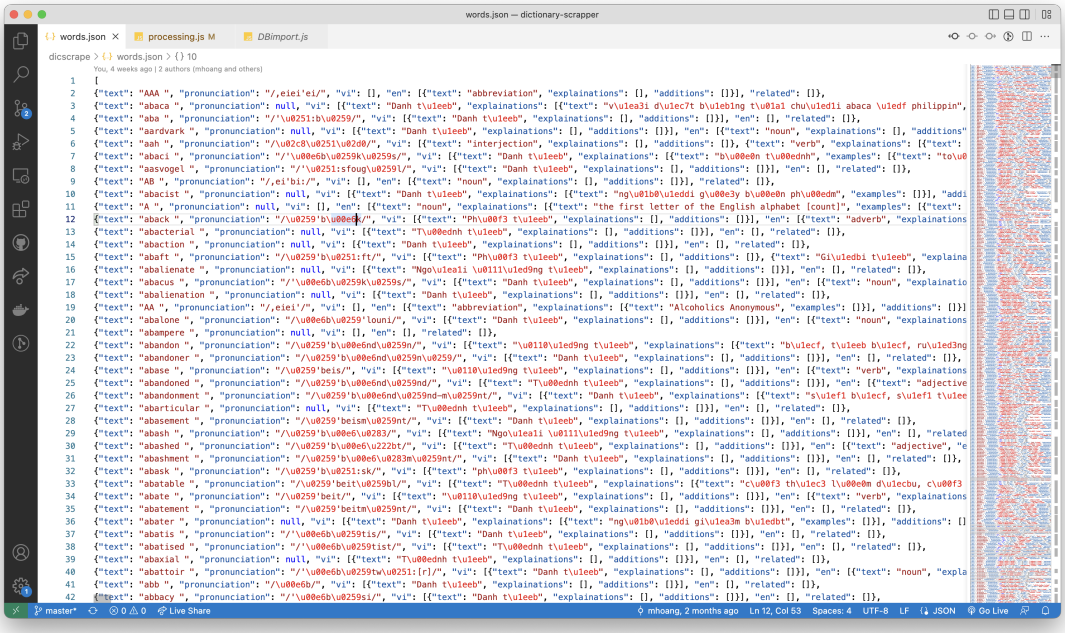

Figure 5.4: Scrapping result

The data now contains the word, pronunciation, Vietnamese and English translation of the word, and additional information. The next step is find a way to transform it into usable data.

We have to standardize gathered data before storing it in our database. Looking at the data, we can see that there were redundant white spaces, and duplicated records. After removing duplication, we generate a new word list, which only contains words that has data.

words.json | processing.js M | DBimport.js

JS > processing.js > ...

You, 41 seconds ago | 1 author (You)

```
1  var fs = require('fs');
2      You, 4 weeks ago • Update Profile + Admin
3  const dic = require(`../dicscrape/words.json`);
4
5  var a = dic.map(x => JSON.stringify(x));
6
7  //remove duplication
8  var b = new Set(a)
9
10 var newDic = Array.from(b).map(x => JSON.parse(x));
11
12 var newWordList = {}
13
14 //generate word list
15 newDic.map((x) => x.text).forEach(x => {
16     newWordList[x.trim()] = 1;
17 });
18
19
20 //processed file
21 fs.writeFile('processed.json', JSON.stringify(newDic), 'utf8', (err) => { })
22
23 //processed word list
24 fs.writeFile('words_dictionary_processed.json', JSON.stringify(newWordList), 'utf8', (err) => { })
```

Figure 5.5: Script for processing data

Database for dictionary is designed using 6 tables.

Figure 5.6: Dictionary tables

The dictionary_word table is the center table, storing the word text and the pronunciation. The dictionary_word_translate tables are used for translation data, containing the word type and the meaning. The dictionary_word_translate stores the additional translation, such as phrases. Finally, the The dictionary_word_related_mapping is for synonyms and antonyms.

Our mission is to transform the JSON data that we have into records of these tables. For doing this task, we created a JavaScript project for manipulating the processed input data. For each word, we generate corresponding insert queries into the tables. After inserting, words will become available inside our dictionary.

5.3.3 Result

After gathering words, our system dictionary now contains over **75000** words. The reason why we could not hit a higher mark is because the 2 chosen dictionaries lack many words. Moreover, the source file for words was not containing compound words, hence could not cover a bigger range of English.

The number of 75000 words is not ideal, but still perfectly good for developing our demo system.

5.4 Pronunciation recognition and assessment

5.4.1 Problem

For every language learner, knowing how to correctly pronounce a word is very important. To compete with other products on the market, an indispensable feature for English learning platforms is pronunciation assessing.

5.4.2 Solution

With the limitation of this Thesis, we decided to go with a third party service that could help me deal with this problem. There are many paid services like Speechace can provide us with such features. After researching, we found out that Microsoft has the Azure Cognitive Services, which provides many speech-to-text features. Since, we have free education Azure server, and they also native SDK for C development, we decided to go with it.

First, we have to setup the environment for our Azure Cognitive Services. By following the documentation, we managed to run the service on our Azure server. The service will provides the information to connect with our server through SDK.

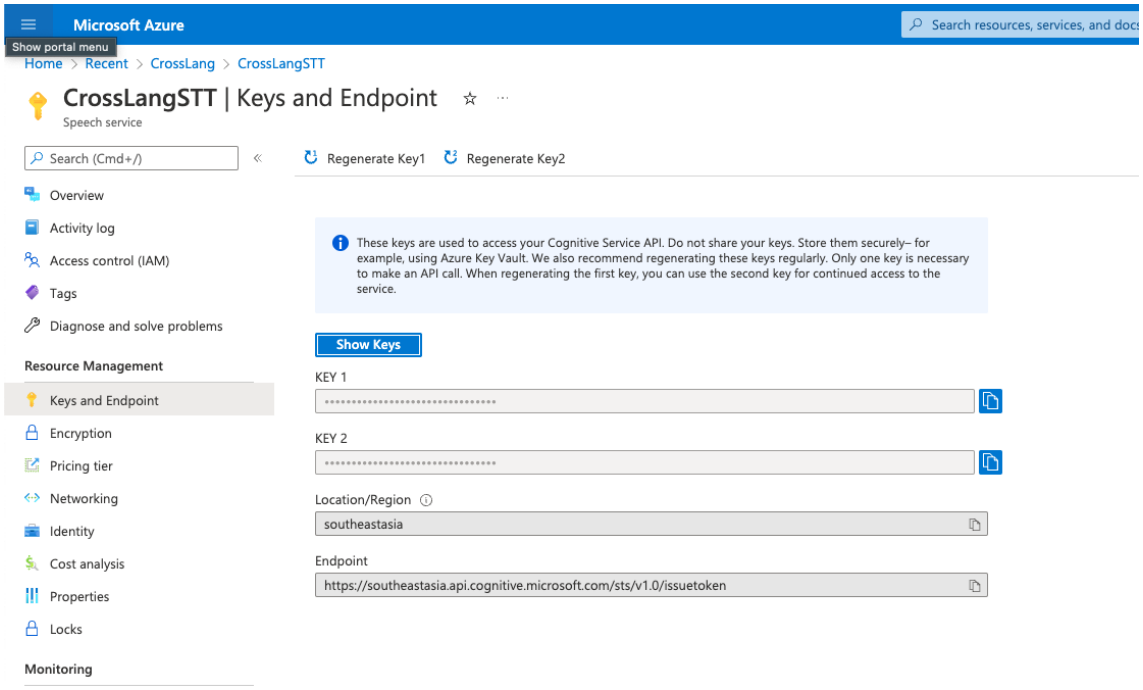

Figure 5.7: Azure Cognitive Services keys and endpoints

Next, we constructed an API that integrates and configures the given SDK with the keys shown in Figure 5.8, delivering requests to the service. To evaluate pronunciation, we must submit a word and a wav audio file. As a result, we establish another API for clients to contribute pronunciation files, which we subsequently utilize to grade tasks.

The client program allows users to record their own voice. One issue is that browsers do not support recording wav audio files. Fortunately, we discovered a JavaScript module named "extendable-media-recorder-wav-encoder," which enables wav extension support. We were able to record the audio in the proper format and transfer it to the server thanks to its support. The sole remaining step is to provide evaluation results to the users.

5.4.3 Result

The pronunciation assessment features was successfully integrated to the system, providing users a way to effectively self improve their English skills.

5.5 Reducing server loads with microservices and RabbitMQ

5.5.1 Problem

Our system is currently quite monolithic. Too many tasks allowed in one process puts heavy loads on the server, and raises the possibility that one job may crash the entire system.

5.5.2 Solution

To improve the system's reliability and performance, it is a good idea to use microservices to handle complex and asynchronous operations. An individual microservice is a service that usually exists only for a single purpose, is self-contained and independent of other instances and services.

Microservices or modules are decoupled from each other but still able to communicate. Using message brokers is one way of communication. A message broker acts as a middleman for various services. They can be used to reduce loads and delivery times of web application servers by delegating tasks that would normally take up a lot of time or resources to a third party that has no other job. For our applications, we implemented RabbitMQ as the message broker.

Consider the email sending task: an email will be sent once the user successfully upgrades the account. If we place the sending email code in our main server, it will be sent whenever a user's account is upgraded. The server's resources will be diverted to the process of sending email, reducing its ability to handle new requests. Not to mention, if the email server does not respond, our main server will suffer greatly, even though this is not a real-time operation.

Going toward our solution, we created a worker for sending email only named "CrossLang.Worker.Email".

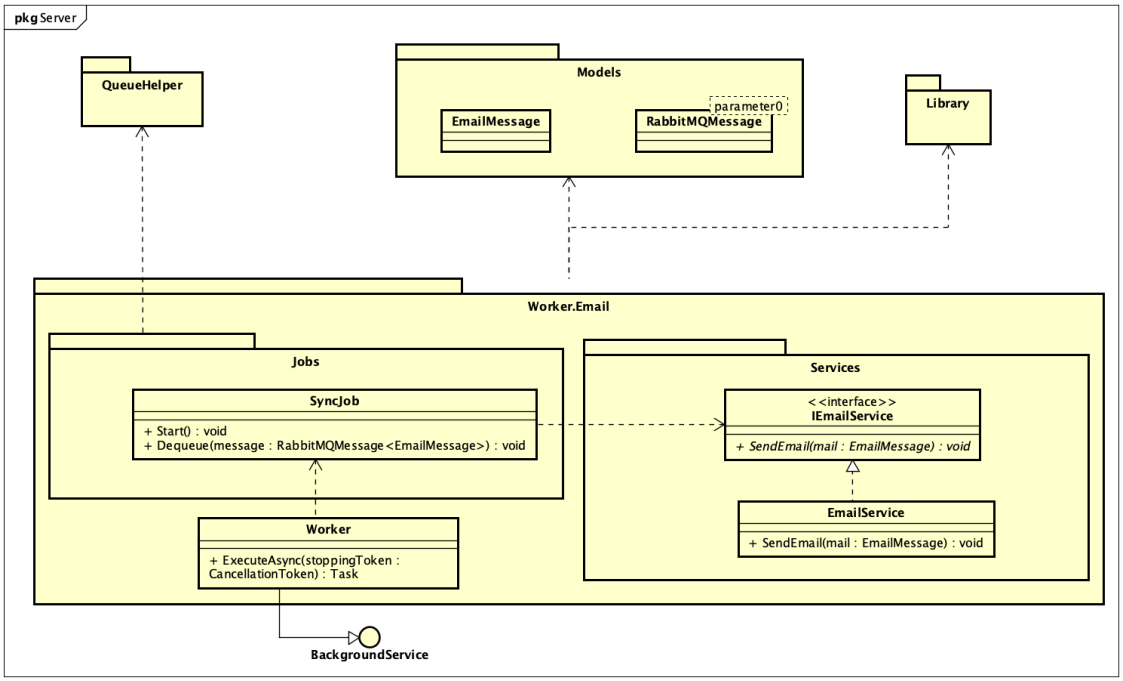

Figure 5.8: CrossLang.Worker.Email class diagram

Worker class extends BackgroundService to be able to run as a background service on host machine. The worker calls Syncjob.Start() method to begin dequeuing the message queue. This worker communicates with our main server using RabbitMQ on a publishing/consuming basis, where our main server will be the publisher, and the worker will be the subscriber. At this point, instead of sending the email directly, our main server will publish a structured message to the "Email" queue on RabbitMQ. The worker, as the subscriber, dequeueing and executing sending email logic on the message received.

```
namespace CrossLang.QueueHelper
{
    public class RabbitMQMessage<T>
    {
        public long? UserID { get; set; }
        public T Body { get; set; }
    }
}
```

Figure 5.9: RabbitMQMessage class

An example message send through our queue contains the information about id

of the sent user, and the body with generic type T. In our example, messages will be instance of class RabbitMQMessage<EmailMessage> and the Body attribute will be of type EmailMessage as following.

```
namespace CrossLang.Models
{
    public class EmailMessage
    {
        public string To { get; set; }

        public string Subject { get; set; }

        public string Body { get; set; }
    }
}
```

Figure 5.10: Body of the of RabbitMQMessage<EmailMessage> class

If there is any problem with our worker, the main server will stay unaffected, and continue to operate normally, and all the messages that were not dequeued will still be kept by RabbitMQ. Therefore, once the worker wakes up, it will continue to execute not processed messages, without needing to publish those message again. Furthermore, one queue can be subscribed by more than one worker. Multiple instance of the same worker can run at once to speed up the dequeuing process.

We used this approach to manage account subscriptions as another job. The database tables should be routinely checked for expired accounts. This operation should be done in a separate process so as not to interfere with our main server. Although communicating with the main server is not necessary for this task, it does include sending emails to users notifying them that their accounts have expired. We let this process run as a cron job in a separate worker. With the aid of RabbitMQ, this worker may still publish a message to the "Email" queue, allowing the Email worker to carry out its duties. This illustration demonstrates how the Email worker may be used for several processes at once. By incorporating this approach into our system, we can improve both platform scalability and performance.

5.5.3 Result

As stated, we were able to effectively implement RabbitMQ into our system as a message broker. Our server was divided up into smaller services that are more

quickly scalable and prepared for future growth.

Chapter Summaries

My contributions to the thesis are summarized in this chapter. In addition, we highlighted the difficulties we experienced during the process of development, along with my solutions. The final chapter, chapter 6, will include my reflections on the project and future works.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The fundamental platform met the requirements for a base service for online English learning, according to the results of implementing Graduation Thesis. We successfully created a system that allows anyone to learn English. The code was designed for ease of maintenance and scalability. At current stage, the system has successfully integrated a simplified version of the microservices.

Our platform currently offers users a dictionary that allows them to search for English words, observe definitions and usage in both Vietnamese and English, and assess their pronunciation. Furthermore, the site has a flashcard system for learning new words. A user might even learn lessons from other categories and practice exercises after finishing a lesson. Teachers and administrators have their own section for managing content and users.

6.2 Future work

Building a heavy platform like this requires all different kinds of resources. The features that we made are just fundamental compared to the whole picture.

First, our plan was to finish the uncompleted features. For example, the current system lacks a payment gate for making transaction. Without this feature, our system would not be able to earn profits.

Then, the application will update new E-Learning standards, new types of online learning, and develop in accordance with those features and requirements. In addition, we also had plan to integrate the Notification feature for various functions such as account upgrading notification, expiration date reminder, and so on. Some types of practice will also be incorporated. I'm preparing to build some games to make learning more fascinating, dynamic, and entertaining. Lastly, we want to implement a live classroom feature in which students might study face-to-face with teachers according to the schedule they set up.

REFERENCE

[1] L. Ceci, *Language learning apps - statistics facts*. [Online]. Available: https://www.statista.com/topics/8425/language-learning-apps/#topicHeader__wrapper (visited on 07/28/2022).

[2] British Council, *The english effect*. [Online]. Available: https://www.britishcouncil.org/sites/default/files/english-effect-report-v2.pdf (visited on 07/28/2022).

[3] *Duolingo - product overview*. [Online]. Available: https://www.duolingo.com/ (visited on 07/28/2022).

[4] *Product overview*. [Online]. Available: https://monkey.edu.vn/san-pham/monkey-junior (visited on 07/28/2022).

[5] Chandler Harris, *Microservices vs. monolithic architecture*. [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (visited on 07/20/2022).

[6] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] *Html: Hypertext markup language*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML (visited on 06/30/2022).

[8] *Css: Cascading style sheets*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS (visited on 06/30/2022).

[9] *Javascript*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 06/30/2022).

[10] *Typescript documentation*. [Online]. Available: https://www.typescriptlang.org/docs/ (visited on 06/30/2022).

[11] *Introduction to the angular docs*. [Online]. Available: https://angular.io/docs (visited on 06/30/2022).

[12] *Introduction to angular concepts*. [Online]. Available: https://angular.io/guide/architecture (visited on 07/02/2022).

[13] *Angular material*. [Online]. Available: https://material.angular.io/ (visited on 06/30/2022).

[14] *What's new in .net 6*. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6 (visited on 06/30/2022).

[15] *Introduction to json web tokens*. [Online]. Available: https://jwt.io/introduction (visited on 07/20/2022).

[16] *Mysql documentation*. [Online]. Available: https://dev.mysql.com/doc/ (visited on 06/30/2022).

[17] *Welcome to the mongodb documentation*. [Online]. Available: https://www.mongodb.com/docs/ (visited on 06/30/2022).

[18] *Documentation*. [Online]. Available: https://www.rabbitmq.com/documentation.html (visited on 06/10/2022).

[19] *List of english words*. [Online]. Available: https://github.com/dwyl/english-words (visited on 04/02/2022).