Contents

[CHAPTER 2. INTRODUCTION 3](#_Toc27503118)

[CHAPTER 3. THEORIES AND TECHNOLOGIES 4](#_Toc27503119)

[3.1. Overview of application’s architecture 4](#_Toc27503120)

[3.2. Theories and Technology 4](#_Toc27503121)

[3.2.1. Node.js 4](#_Toc27503122)

[3.2.2. Express.js 5](#_Toc27503123)

[3.2.3. What is the purpose of it with Node.js? 5](#_Toc27503124)

[3.2.4. React 6](#_Toc27503125)

[3.2.5. MongoDB 6](#_Toc27503126)

[3.2.6. Mongoose 9](#_Toc27503127)

[3.2.7. MVC Framework 9](#_Toc27503128)

[Following are the components of MVC − 9](#_Toc27503129)

[ Model 10](#_Toc27503130)

[ View 10](#_Toc27503131)

[ Controller 10](#_Toc27503132)

[Version control 10](#_Toc27503133)

[3.2.8. Git 10](#_Toc27503134)

[Github 11](#_Toc27503135)

[CHAPTER 4. ANALYSIS AND DESIGN 11](#_Toc27503136)

[4.1. Analysis 12](#_Toc27503137)

[4.1.1. Use case diagram 12](#_Toc27503138)

[Use case specification 22](#_Toc27503139)

[Login 22](#_Toc27503140)

[4.2. Models relationships 40](#_Toc27503141)

[4.3. Activity diagrams 41](#_Toc27503142)

[4.3.1. Using writing service (for User) 41](#_Toc27503143)

[4.3.2. Checking writing (for Admin and Mod) 41](#_Toc27503144)

[4.3.3. Create new user (for Admin) 43](#_Toc27503145)

[4.3.4. Create new Blog 44](#_Toc27503146)

[4.3.5. Delete blog 45](#_Toc27503147)

[4.3.6. Create new Podcast 46](#_Toc27503148)

[4.3.7. Delete podcast 47](#_Toc27503149)

[CHAPTER 5. IMPLEMENTATION AND EVALUATION 47](#_Toc27503150)

[5.1. Overview 47](#_Toc27503151)

[5.1.1. Environment 47](#_Toc27503152)

[5.2. Demonstration 49](#_Toc27503153)

[CHAPTER 6. CONCLUSION AND FUTURE WORKS 54](#_Toc27503154)

[6.1. Achieved results 54](#_Toc27503155)

[6.2. Future works 55](#_Toc27503156)

# INTRODUCTION

English is the language of science, of aviation, computers, diplomacy, and tourism. Knowing English increases your chances of getting a good job in a multinational company within your home country or for finding work abroad. It’s also the language of international communication, the media and the internet, so learning English is important for socialising and entertainment as well as work!

Beside of that, nowadays with the variation of English certificates such as TOEIC, TOEFL, IELTS…students who want to get certificate really need one truthful source to learn English. So that, I decided to design a system that can help students to improve their language skills base on what I experienced. For improving Listening skills, the key is the time you spend to just listen on the specific topic, podcast is my favorite source, so I will implement this section in my website. In addition, my system will have one section for sharing tips to enhance reading, listening, speaking and writing skills. For the learners who want to get IELTS certificate, Writing section is the rough one, they need someone who has experience to check the mistakes in there writing, so I also have a section named “Writing Service” for checking writing.

Because, there are people - who don’t have the conditions to use a computer, or the software on computer is too complex to use for ordinary users, so that requiring an application with the following features:

* Allowing users to listen to the podcasts
* Providing cool tips and tricks on all the sections you will in IELTS test
* Using writing service to help you realize your faults in your writing

With the above requirements, I have implemented the project entitled: Teacher Luke – Website system for IELTS practicing

# THEORIES AND TECHNOLOGIES

## Overview of application’s architecture

Throughout the project, I used MVC architecture to build the architecture of the system, it helps design and maintenance easier. Also, to manage the versions, Git and GitHub are a great helper. In the next section, I will introduce the components and tools used in the project.

## Theories and Technology

### Node.js

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In most applications, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.



Figure 1.1. Node.js implementation

This is in contrast to today's more common concurrency model, in which OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks. Because nothing blocks, scalable systems are very reasonable to develop in Node.js.

Node.js is similar in design to, and influenced by, systems like Ruby's [Event Machine](https://github.com/eventmachine/eventmachine) and Python's [Twisted](https://twistedmatrix.com/trac/). Node.js takes the event model a bit further. It presents an [event loop](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/) as a runtime construct instead of as a library. In other systems, there is always a blocking call to start the event-loop. Typically, behavior is defined through callbacks at the beginning of a script, and at the end a server is started through a blocking call like EventMachine::run(). In Node.js, there is no such start-the-event-loop call. Node.js simply enters the event loop after executing the input script. Node.js exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

Node.js being designed without threads doesn't mean you can't take advantage of multiple cores in your environment. Child processes can be spawned by using our [child\_process.fork()](https://nodejs.org/api/child_process.html#child_process_child_process_fork_modulepath_args_options) API, and are designed to be easy to communicate with. Built upon that same interface is the [cluster](https://nodejs.org/api/cluster.html) module, which allows you to share sockets between processes to enable load balancing over your cores

### Express.js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It's the most popular framework as of now (the most starred on NPM).



Figure 1.2

### What is the purpose of it with Node.js?

That you don't have to repeat same code over and over again. Node.js is a low-level [***I/O***](http://en.wikipedia.org/wiki/Input/output) mechanism which has an HTTP module. If you just use an HTTP module, a lot of work like parsing the payload, cookies, storing sessions (in memory or in [***Redis***](http://en.wikipedia.org/wiki/Redis_(data_store))), selecting the right route pattern based on [***regular expressions***](http://en.wikipedia.org/wiki/Regular_expression) will *have* to be re-implemented. With Express.js, it is just there for you to use

### React

**React** (also known as **React.js** or **ReactJS**) is a JavaScript library for building [user interfaces](https://en.wikipedia.org/wiki/User_interfaces). It is maintained by [Facebook](https://en.wikipedia.org/wiki/Facebook) and a community of individual developers and companies.

React can be used as a base in the development of [single-page](https://en.wikipedia.org/wiki/Single-page_application) or mobile applications, as it is optimal for fetching rapidly changing data that needs to be recorded. However, fetching data is only the beginning of what happens on a web page, which is why complex React applications usually require the use of additional libraries for [state management](https://en.wikipedia.org/wiki/State_management), [routing](https://en.wikipedia.org/w/index.php?title=Web_library&action=edit&redlink=1), and interaction with an API, Redux React Router and axios are examples of such libraries.

* Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

* Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

* Learn Once, Write Anywhere

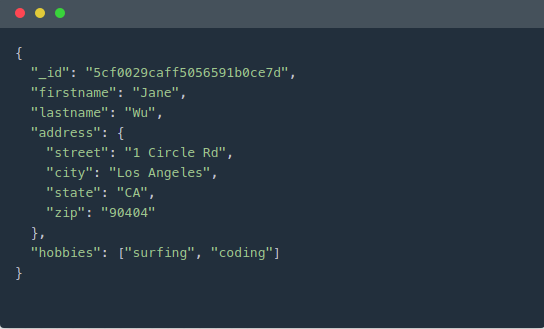
We don’t make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

### MongoDB

MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era. No database makes you more productive.

MongoDB is a document database, which means it stores data in JSON-like documents. We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.

* Rich JSON Documents
* The most natural and productive way to work with data.
* Supports arrays and nested objects as values.
* Allows for flexible and dynamic schemas.

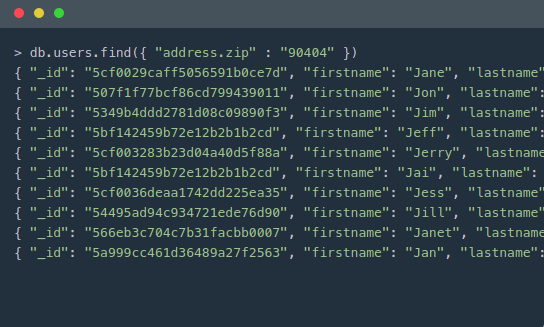


* + - * Powerful query language

Rich and expressive query language that allows you to filter and sort by any field, no matter how nested it may be within a document.

Support for aggregations and other modern use-cases such as geo-based search, graph search, and text search.

Queries are themselves JSON, and thus easily composable. No more concatenating strings to dynamically generate SQL queries.

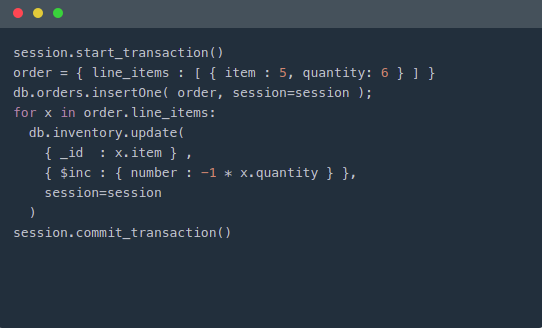


All the power of a relational database, and more...

Full ACID transactions.

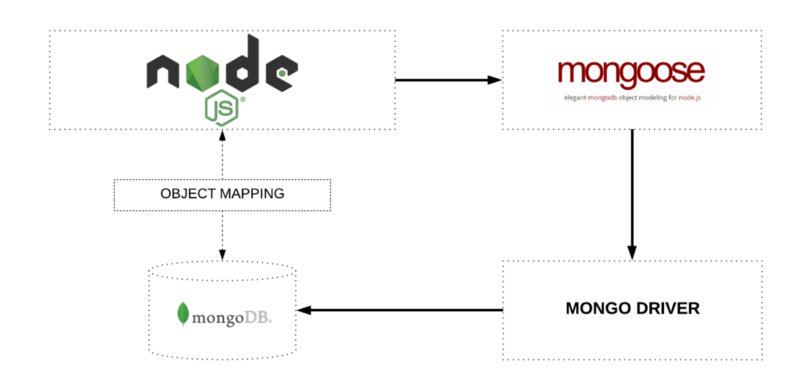
Support for joins in queries.

Two types of relationships instead of one: reference and embedded.



### Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

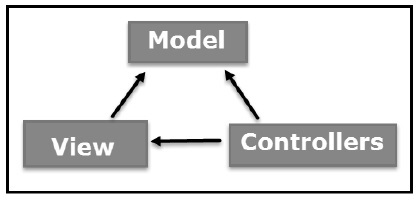


### MVC Framework

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects***.***

**MVC Components**

Following are the components of MVC −



* + - Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

* + - View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

* + - Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

Version control

Version control helps developers track and manage changes to a software project’s code. As a software project grows, version control becomes essential. Take WordPress…

At this point, WordPress is a pretty big project. If a core developer wanted to work on one specific part of the WordPress codebase, it wouldn’t be safe or efficient to have them directly edit the “official” source code.

Instead, version control lets developers safely work through branching and merging.

With branching, a developer duplicates part of the source code (called the repository). The developer can then safely make changes to that part of the code without affecting the rest of the project.

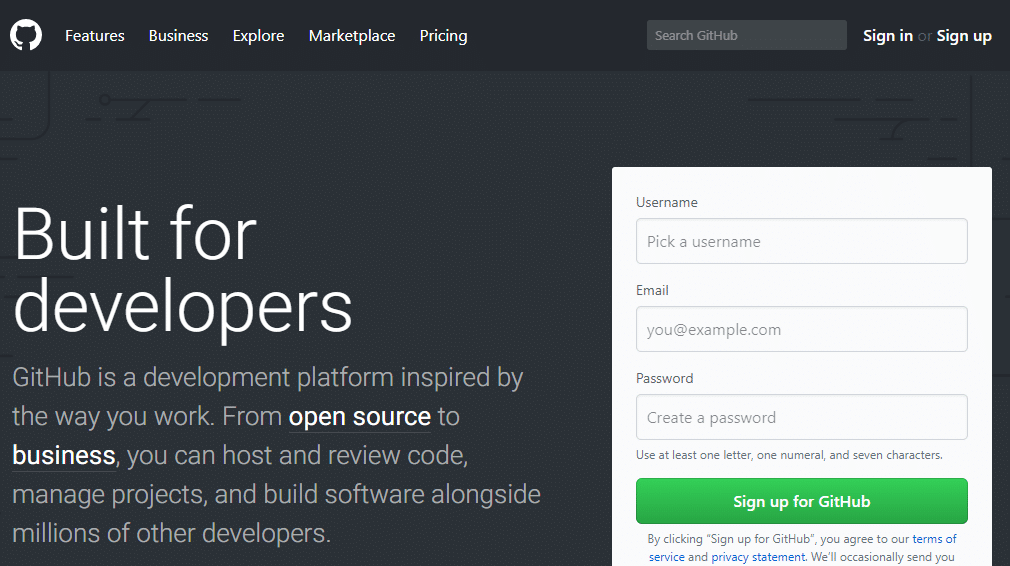
Then, once the developer gets his or her part of the code working properly, he or she can merge that code back into the main source code to make it official.

All of these changes are then tracked and can be reverted if need be.

### Git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system, it is aimed at speed, data integrity, and support for distributed, non-linear workflows [11].

Github



GitHub is a for-profit company that offers a cloud-based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

GitHub’s interface is user-friendly enough so even novice coders can take advantage of Git. Without GitHub, using Git generally requires a bit more technical savvy and use of the command line.

GitHub is so user-friendly, though, that some people even use GitHub to manage other types of projects – [like writing books](http://braythwayt.com/2015/01/29/how-i-write-books-with-github-and-leanpub.html).

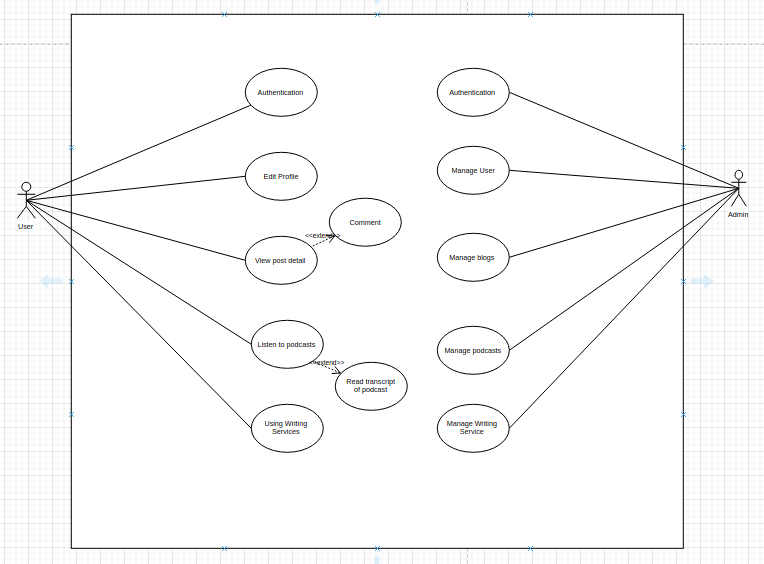
Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

# ANALYSIS AND DESIGN

## Analysis

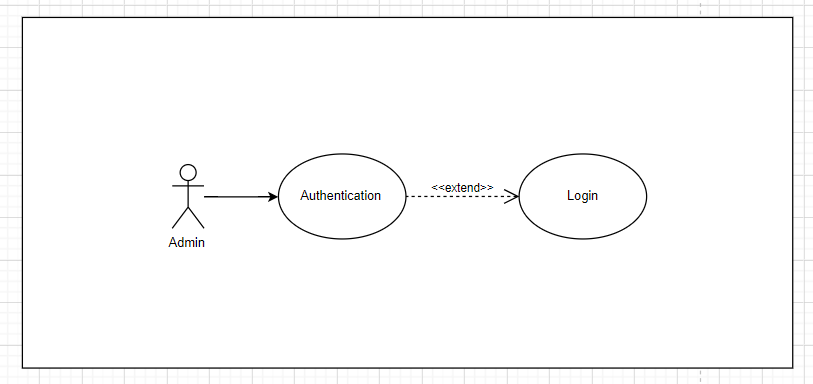
### Use case diagram

#### Overall use case

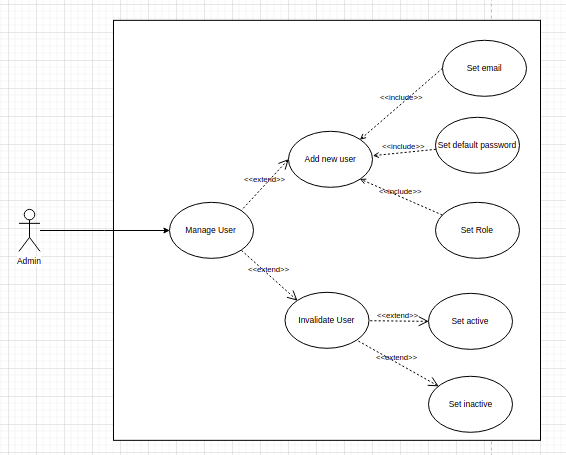


#### Admin & Mod

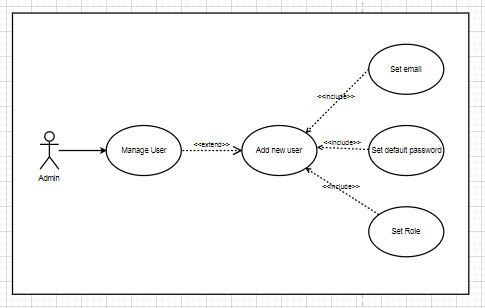
* Use case for authentication



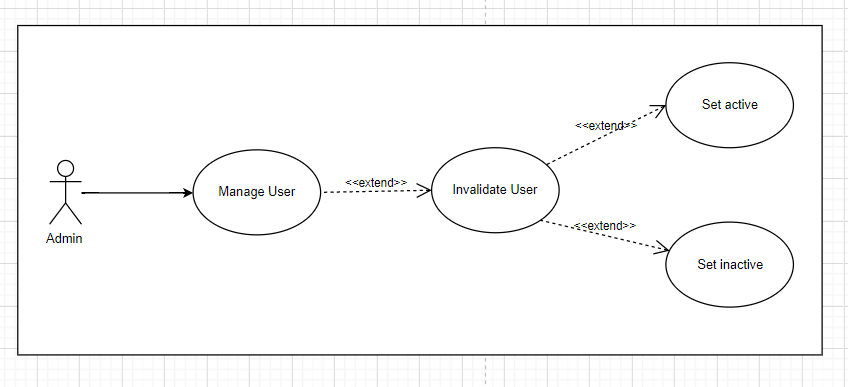
* Use case for managing User



* Use case for adding new User



* Use case for invalidate User



* Use case for managing the podcasts



Figure 2.3 Use case for managing the podcast

* + Use case for adding podcast

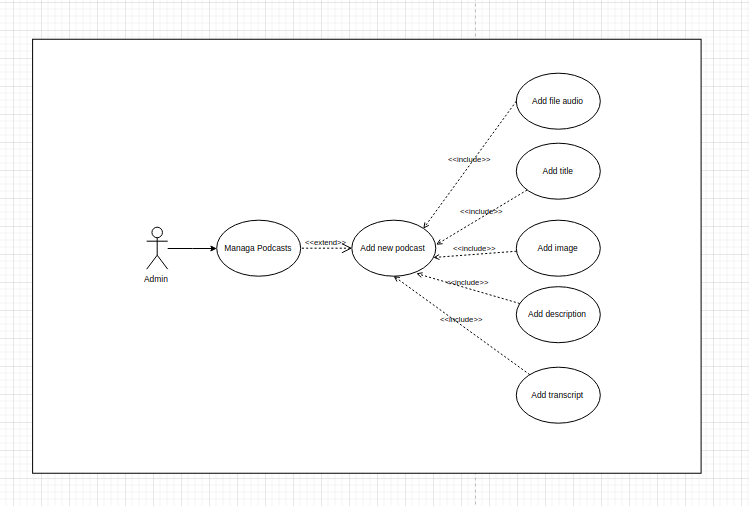


Figure 2.4 Use case for adding **podcasts**

* + Use case for editing podcast.

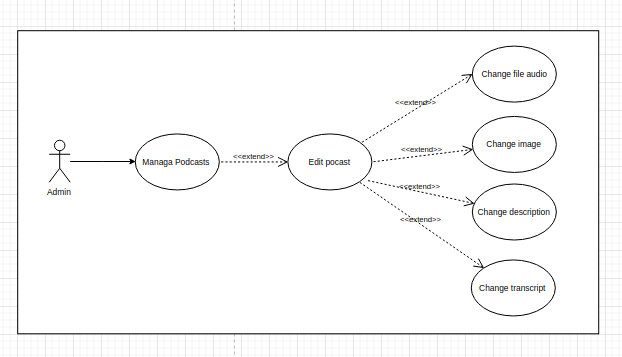


Figure 2.5 Use case for editing podcast

* + Use case for deleting podcast

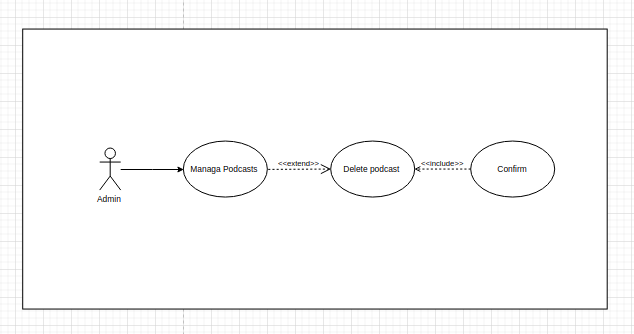
**

Figure 2.6 Use case for deleting podcast

* Use case for manage writing service

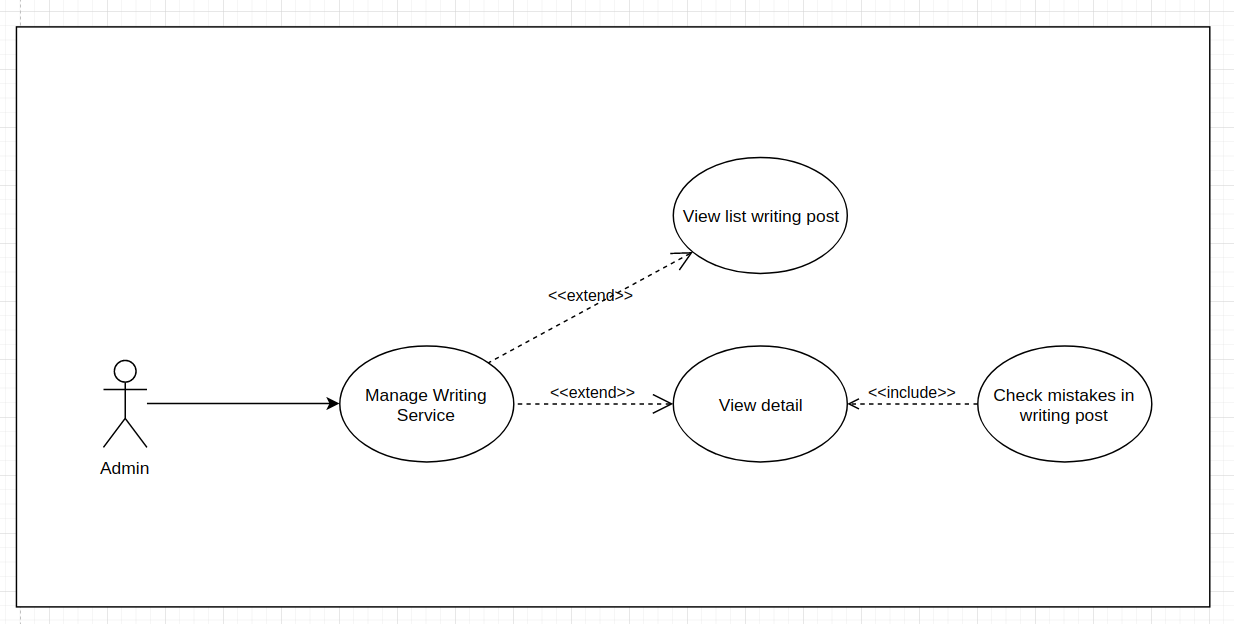


Figure 2.7 Use case for managing writing services

* Use case for checking mistakes

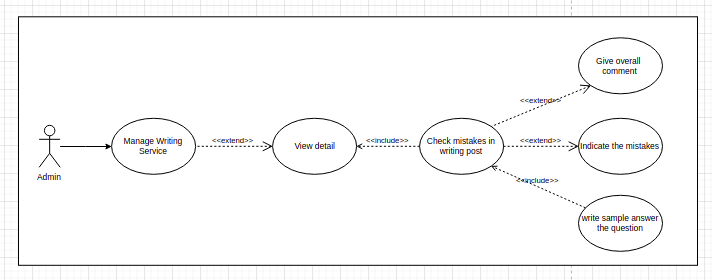


Figure 2.7 Use case for checking mistakes.

* Use case for managing blogs.

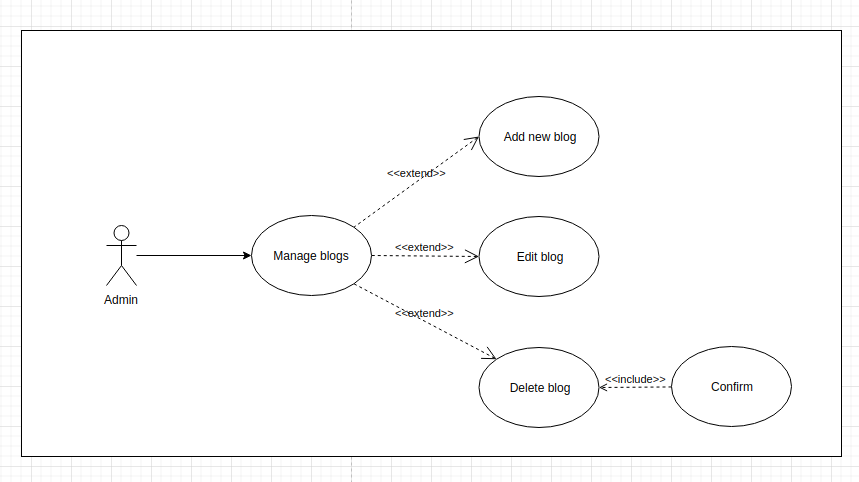


Figure 2.8 Use case for managing blogs

* Use case for adding new post.

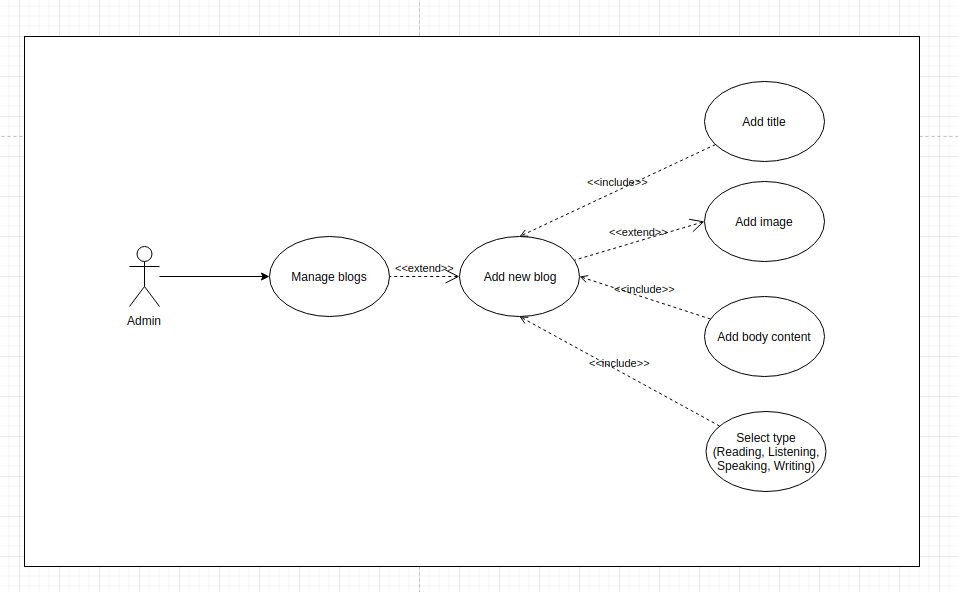


Figure 2.9 Use case for adding **new post**

* Use case for editing post

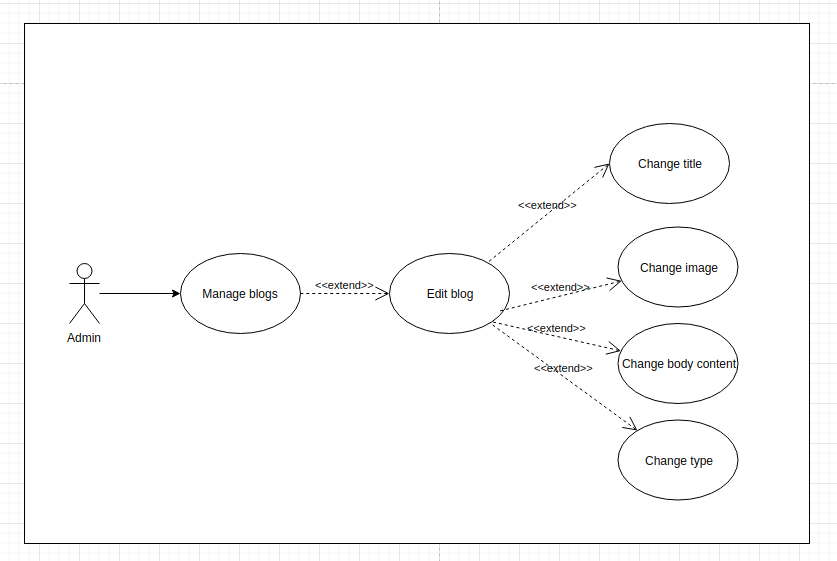


Figure 2.10 Use case for editing post

* Use case for deleting post

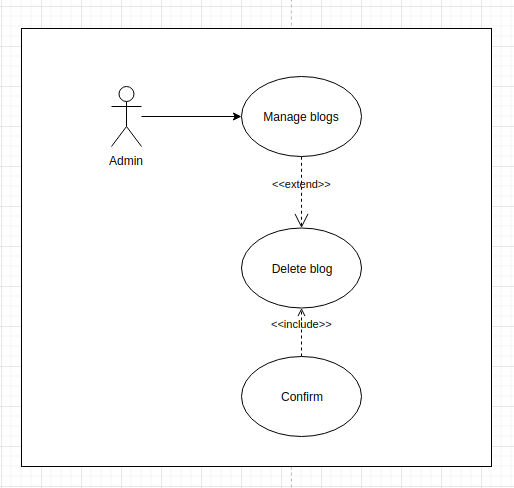


Figure 2.11 Use case for **deleting post**

#### User

* Use case for managing writing service

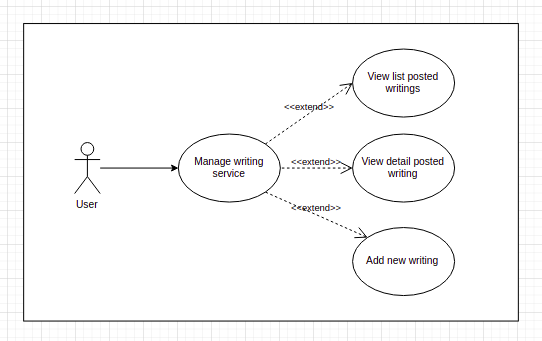


Figure 2.12 Use case for **managing writing services**

* Use case for adding new writing post

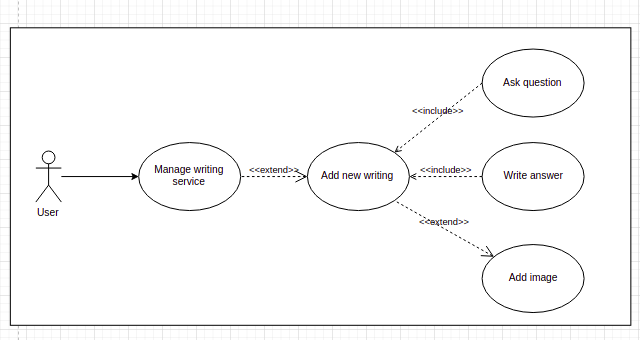


Figure 2.13 Use case for **adding new writing post**

Use case specification

Login

Table 2‑1

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC - 01** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for logging into the app | | |
| Pre – conditions |  | | |
| Post – conditions | Actor can view profile and use writing service, comment... | | |
| Flow of events |  | Actor inputs | System response |
| 1 | Actor clicks button “Login” on Navbar |  |
| 2 |  | System show login screen. |
| 3 | Actor fills in the email field and password field, then click button “Login” |  |
| 4 |  | System redirect user to the home page of the app, post a message “Login successfully” |

* Change profile

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC - 02** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for changing profile | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions | Actor has successfully updated his/her profile | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor click the button “Profile” on the Navbar |  |
| 2 |  | System shows actor’s profile:  avatar, name, email... |
| 3 | Actor change avatar, name, address... |  |
| 4 | Actor clicks the “Save information” button |  |
| 5 |  | System redirect user to the home page with message “Update profile successfully” |

* View list of blogs

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 03** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for viewing list of blogs | | |
| Pre – conditions |  | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Navbar |  |
| 2 |  | System redirect user to blogs page |
|  | 3 | Actor clicks type buttons on the top left of the pages |  |
|  | 4 |  | System shows the posts according to type |

* View blog detail

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for viewing the detail of blog | | |
| Pre – conditions | Actor has viewed the page show list of blogs | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Navbar |  |
| 2 |  | System redirects user to the page show list blogs |
| 3 | Actor clicks the title of one of the blogs |  |
| 4 |  | System redirects user to detail page |

* Comment on blog

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 05** | | |
| Actors | User | | |
| Brief description | This use case for commenting on blog | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Navbar |  |
| 2 |  | System redirects user to the page show list blogs |
| 3 | Actor clicks the title of one of the blogs |  |
| 4 |  | System redirects user to detail page |
| 5 | User writes content in the comment section and press button “Enter” |  |
|  | 6 |  | System shows your recent comment |

* View list of podcasts

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 06** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for viewing the list of podcasts | | |
| Pre – conditions |  | | |
| Post - conditions | System shows the list of podcasts | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcast” on the Navbar |  |
| 2 |  | System redirects user to the page show list podcasts |

* Listen to the specific podcast

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 06** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for viewing the detail of podcasts | | |
| Pre – conditions | Actor is standing on the page listing podcasts | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcast” on the Navbar |  |
| 2 |  | System redirects user to the page show list podcasts |
|  | 3 | Actor clicks on the title of one of the podcast showing on the screen |  |
|  | 4 |  | System redirects user to the detail page of podcast |
|  | 5 | Actor clicks the button “Play” on the audio tag |  |
|  | 6 |  | System plays the podcast |

* View writing service section

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | User, Admin, Mod | | |
| Brief description | This use case for viewing the writing service section | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Writing Service” on the Navbar |  |
| 2 |  | System redirects user to the page show list posted writing |

* Add new writing

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | User | | |
| Brief description | This use case for adding the | | |
| Pre – conditions | Actor has viewed the page show list of blogs | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Writing Service” on the Navbar |  |
| 2 |  | System redirects user to the page show list posted writings |
| 3 | Actor clicks the “Post new writing” button |  |
| 4 |  | System redirects user to the page adding new writing |
|  | 5 | Actor fills the question section, the body section…  Then click ‘send’ button |  |
|  |  |  | System show the message notify sent successfully |

* View list of users

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin | | |
| Brief description | This use case for viewing the list of users | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Users” on the Left Side Bar |  |
| 2 |  | System shows the list of Users in the main section |

* Activate/Inactivate user

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin | | |
| Brief description | This use case for activating/inactivating users | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Users” on the Left Side Bar |  |
| 2 |  | System shows the list of Users in the main section |
| 3 | Actor clicks the switch to activate/inactivate the user |  |
| 4 |  | System shows message changed successfully |

* Add new user

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin | | |
| Brief description | This use case for adding new user | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Users” on the Left Side Bar |  |
| 2 |  | System shows the list of Users in the main section |
| 3 | Actor clicks the button “Add new” on the left top of the main section |  |
| 4 |  | System show the drawer on the right side |
|  | 5 | Actor fill the information of the new user in the fields and click Save Button |  |
|  | 6 |  | System show message added successfully |

* View list of podcasts

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin | | |
| Brief description | This use case for viewing list of podcasts | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcasts” on the Left Side Bar |  |
| 2 |  | System shows the list of Podcasts in the main section |

* Add new podcast

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for adding new podcast | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcasts” on the Left Side Bar |  |
| 2 |  | System shows the list of Podcasts in the main section |
|  | 3 | Actor clicks the button “Add new” |  |
|  | 4 |  | System redirect actor to the add podcast page |
|  | 5 | Actor fills all the fields and click “Save” button |  |
|  |  |  | System post a message notify succeed and redirect actor to the page listing podcasts |

* Delete podcast

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for deleting podcast | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcasts” on the Left Side Bar |  |
| 2 |  | System shows the list of Podcasts in the main section |
|  | 3 | Actor clicks the button “Delete” on the row of the podcast |  |
|  | 4 |  | System show the popup message wants the actor the confirm the action |
|  | 5 | Actor clicks “yes” button |  |
|  | 6 |  | System delete the podcast and show the message deleted successfully |

* Edit podcast

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for editing podcast | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Podcasts” on the Left Side Bar |  |
| 2 |  | System shows the list of Podcasts in the main section |
|  | 3 | Actor clicks the button “Edit” on the row of the podcast |  |
|  | 4 |  | System redirect actor to the edit page |
|  | 5 | Actor fills the fields which actor wants to update |  |
|  | 6 |  | System update the podcast and show the message updated successfully |

* View list of blogs

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for viewing list of blogs | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Left Side Bar |  |
| 2 |  | System shows the list of Blogs in the main section |

* Add new blog

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for adding new blog | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Left Side Bar |  |
| 2 |  | System shows the list of Blogs in the main section |
|  | 3 | Actor clicks the button “Add new” |  |
|  | 4 |  | System redirect actor to the add blog page |
|  | 5 | Actor fills all the fields and click “Save” button |  |
|  |  |  | System post a message notify succeed and redirect actor to the page listing blogs |

* Delete specific blog

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for deleting blog | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Left Side Bar |  |
| 2 |  | System shows the list of Blogs in the main section |
|  | 3 | Actor clicks the button “Delete” on the row of the podcast |  |
|  | 4 |  | System show the popup message wants the actor the confirm the action |
|  | 5 | Actor clicks “yes” button |  |
|  | 6 |  | System delete the blog and show the message deleted successfully |

* Edit blog

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for editing blog | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Blogs” on the Left Side Bar |  |
| 2 |  | System shows the list of Blogs in the main section |
|  | 3 | Actor clicks the button “Edit” on the row of the blogs |  |
|  | 4 |  | System redirect actor to the edit page |
|  | 5 | Actor fills the fields which actor wants to update |  |
|  | 6 |  | System update the blog and show the message updated successfully |

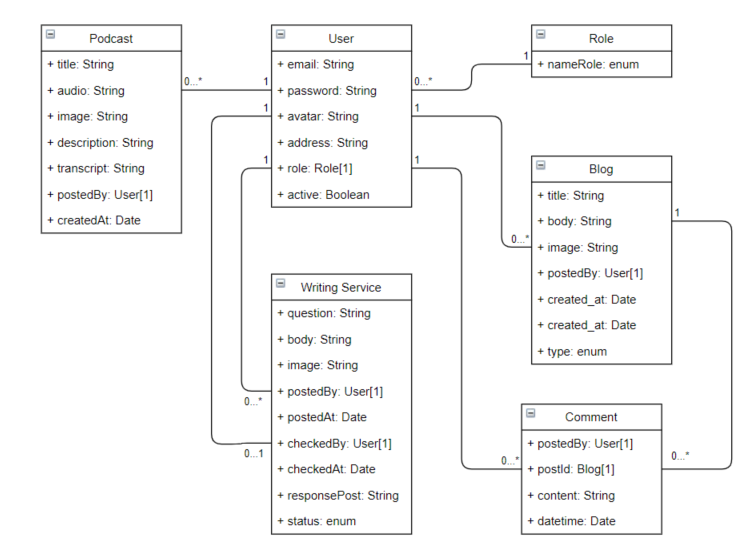
* View the list of writing need to be checked

|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for viewing the list of writing services | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Services” on the Left Side Bar |  |
| 2 |  | System shows the list of Writings in the main section |

* Check writings

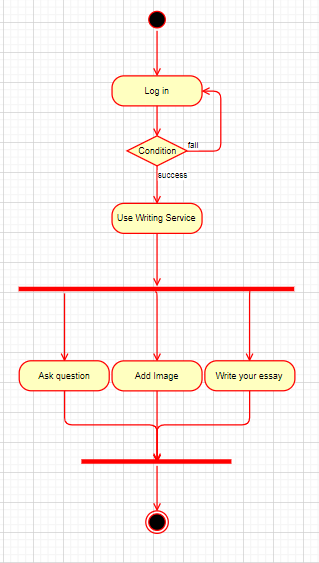
|  |  |  |  |
| --- | --- | --- | --- |
| Use case ID | **UC – 04** | | |
| Actors | Admin, Mod | | |
| Brief description | This use case for checking writings | | |
| Pre – conditions | Actor has logged in the app | | |
| Post - conditions |  | | |
| Flow of events |  | Actor input | System response |
| 1 | Actor clicks the button “Services” on the Left Side Bar |  |
| 2 |  | System shows the list of Writings in the main section |
|  | 3 | Actor clicks the “Check writing” Button |  |
|  | 4 |  | System shows the question and writing of user |
|  | 5 | Actor fills in the overall comment, mistakes and sample section, then click  “Submit” button |  |
|  |  |  | System shows the message notify successfully |

## Models relationships

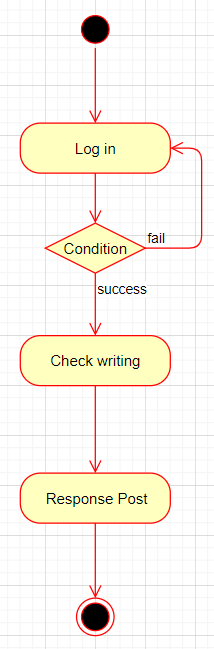


## Activity diagrams

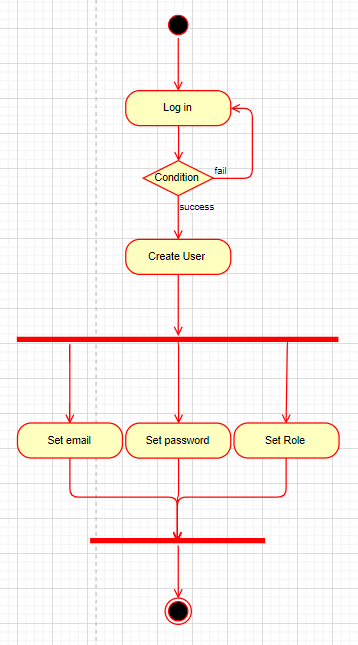
### Using writing service (for User)



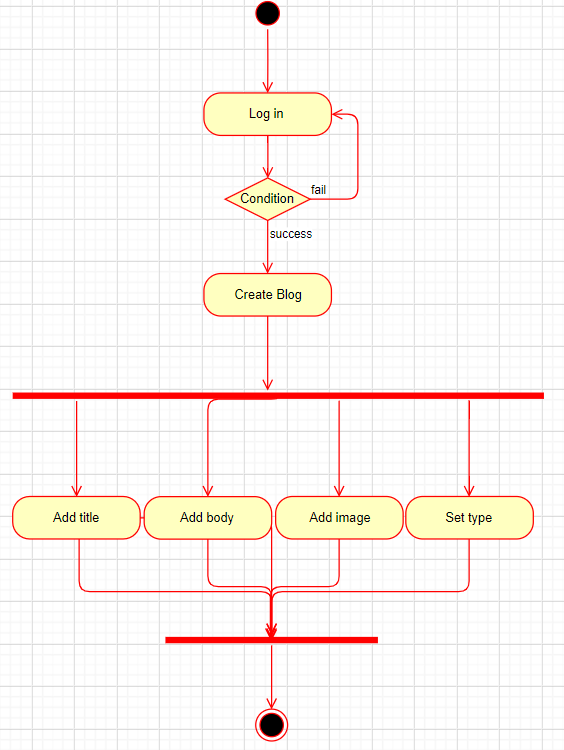
### Checking writing (for Admin and Mod)



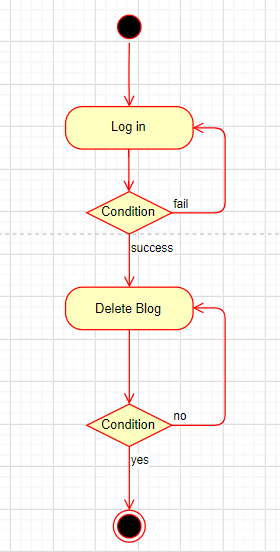
### Create new user (for Admin)



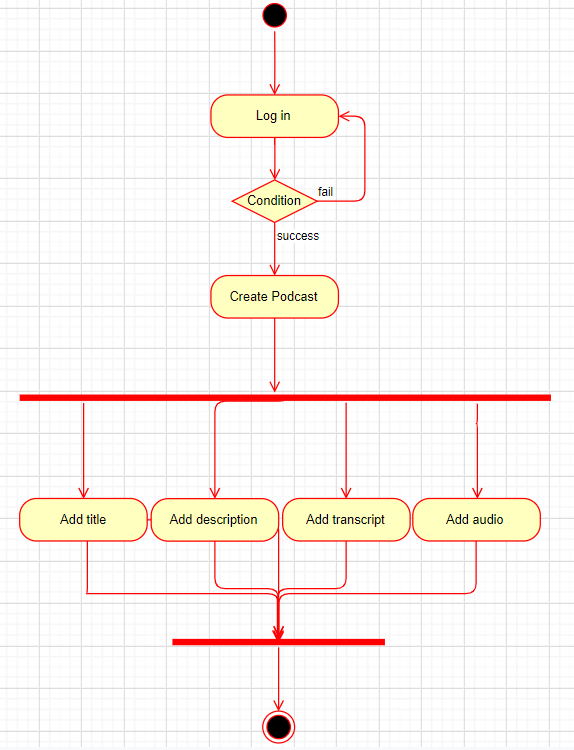
### Create new Blog



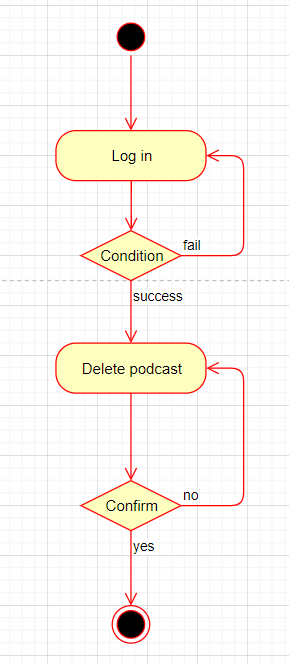
### Delete blog



### Create new Podcast



### Delete podcast



# IMPLEMENTATION AND EVALUATION

## Overview

### Environment

My application was developed on Visual Studio Code. It’s a light editor comes with lots of extension help developers to improve productivity.

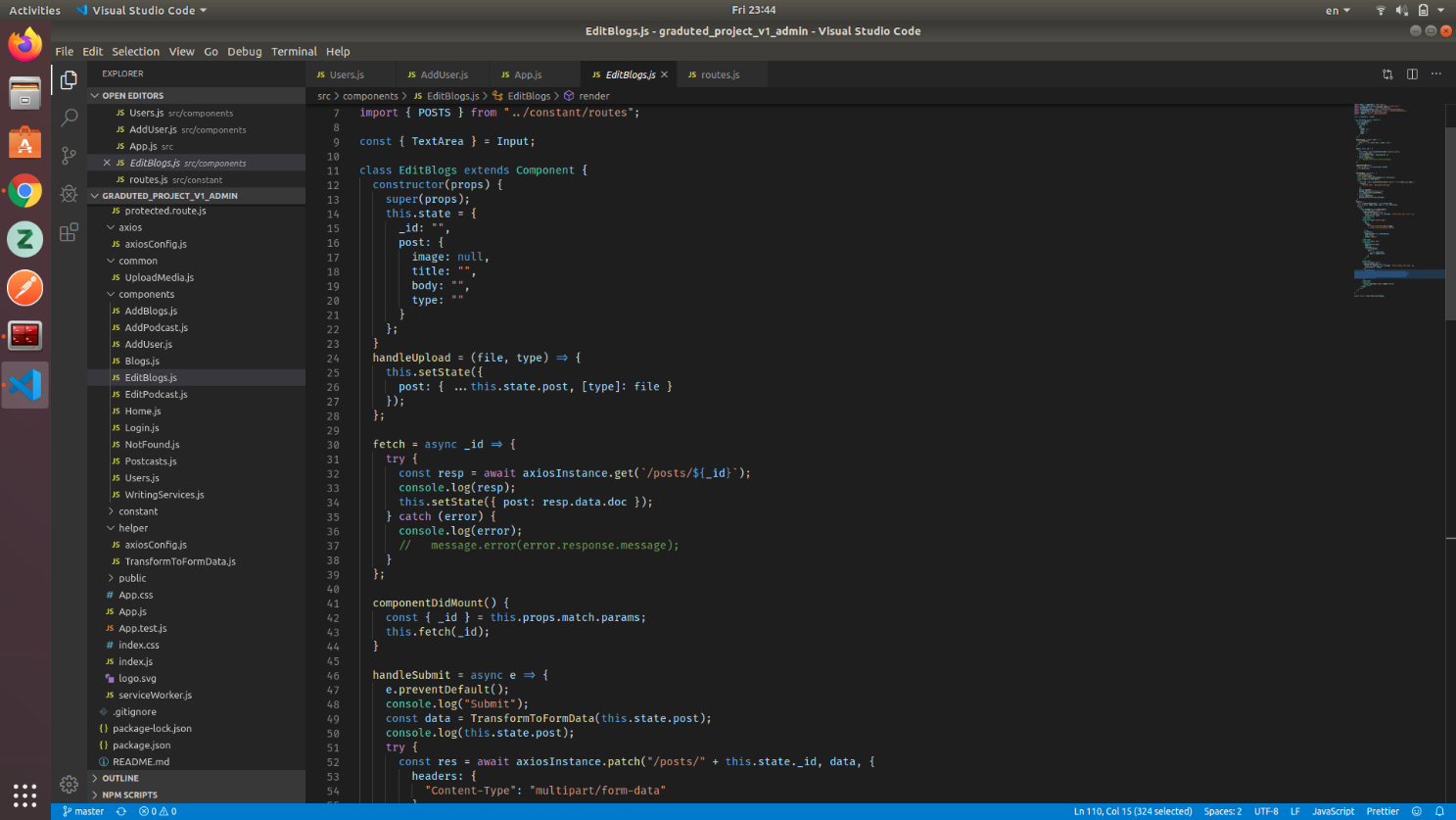


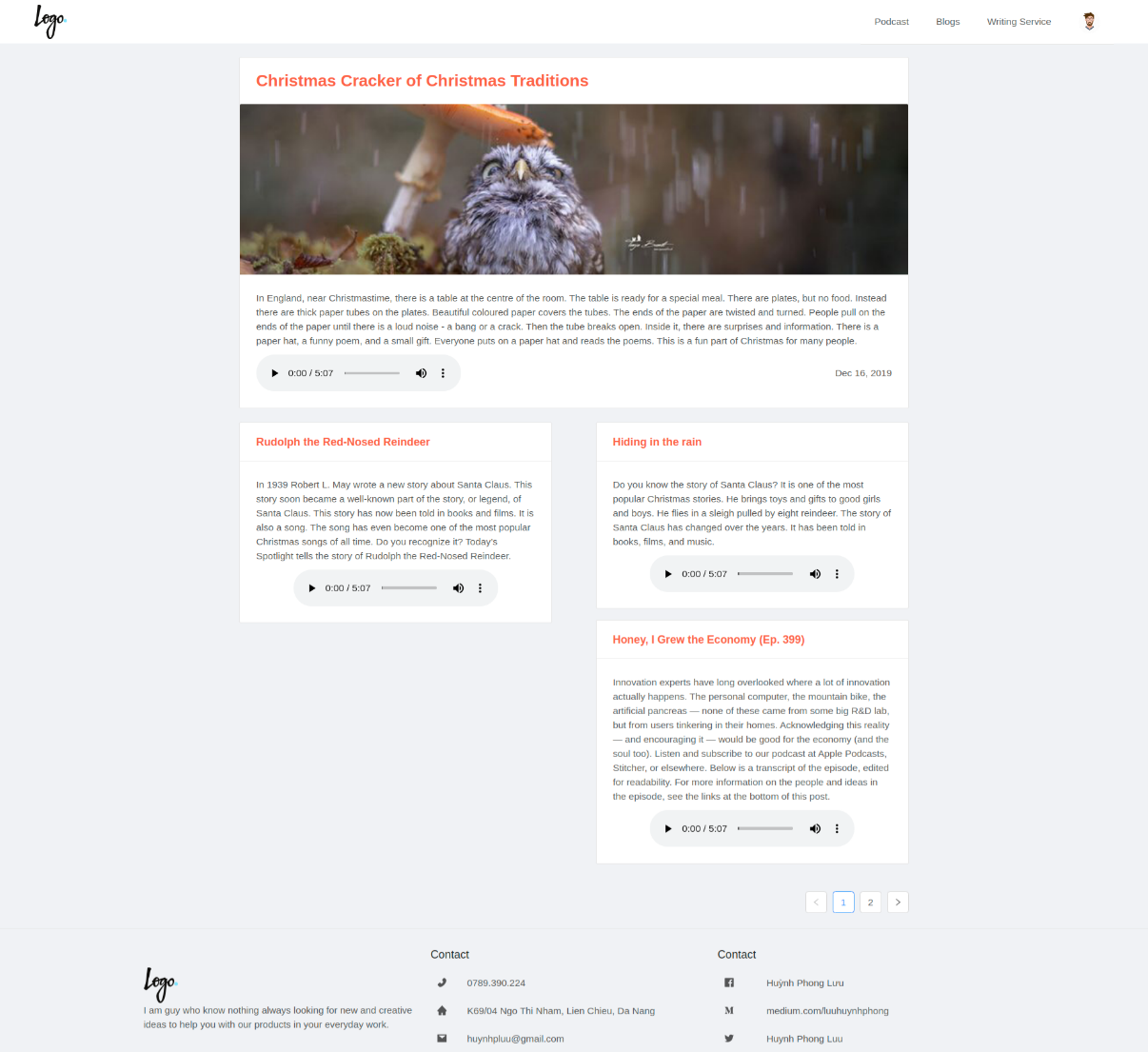
Figure 3.1 Android Studio Screenshot

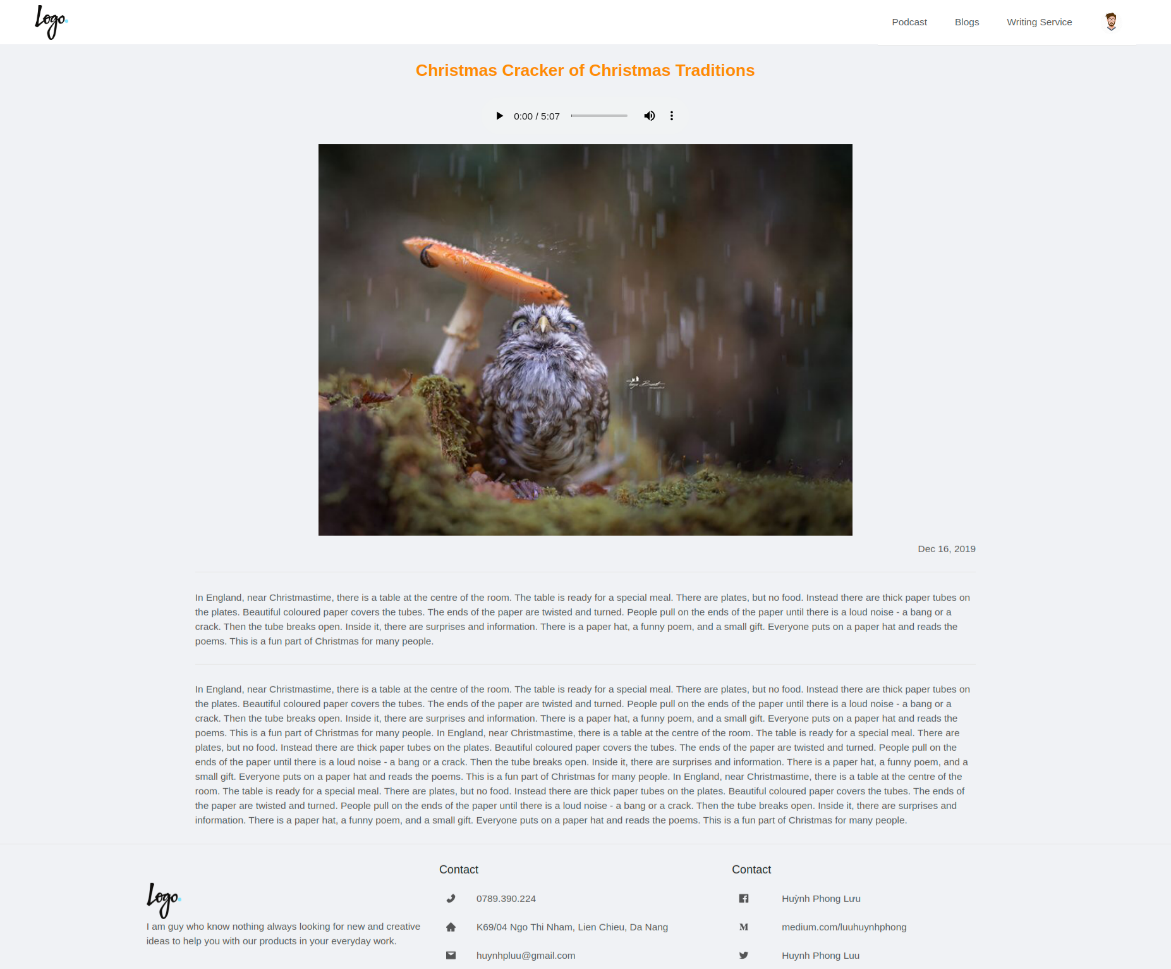
## Demonstration

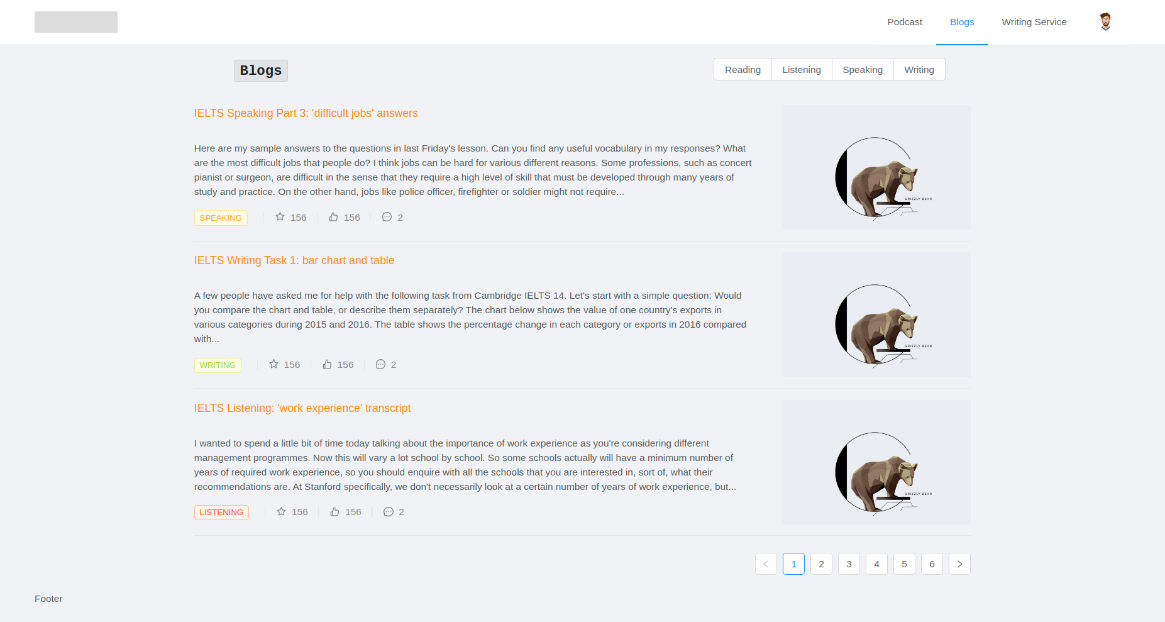
****

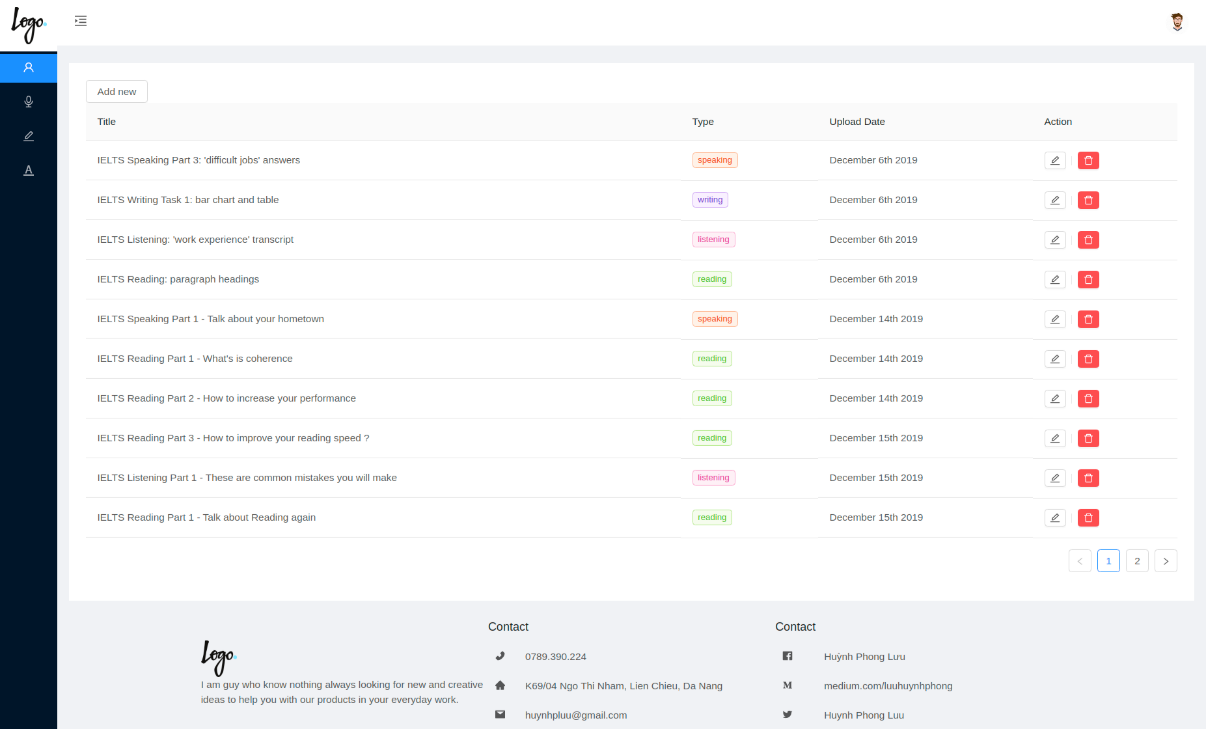
Figure 3.3 Login screen.

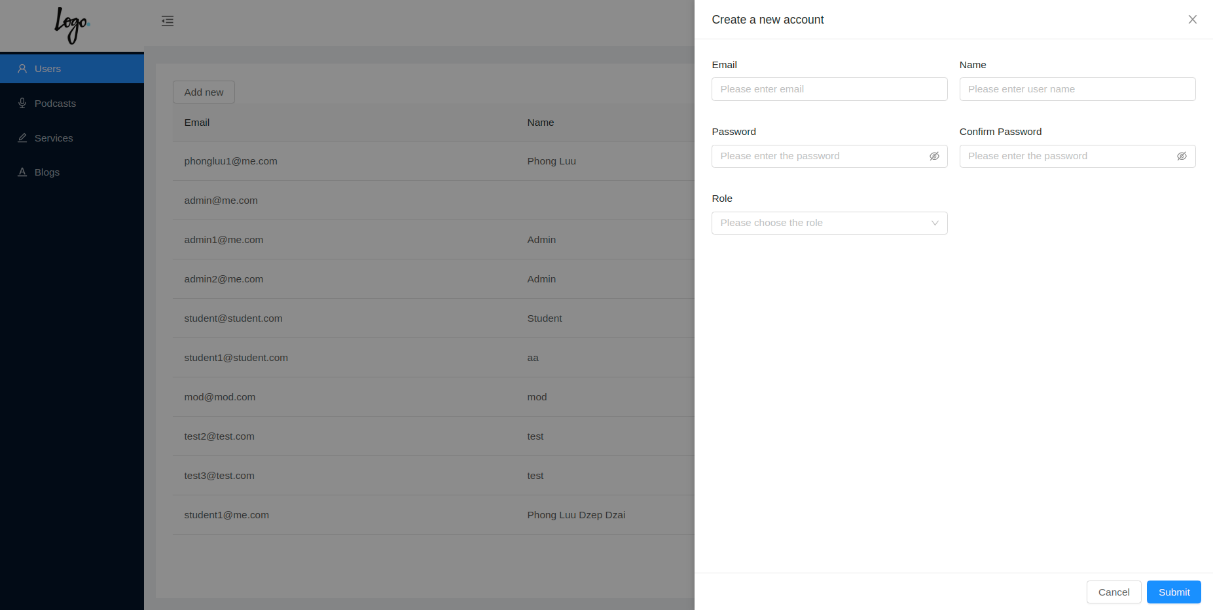


****









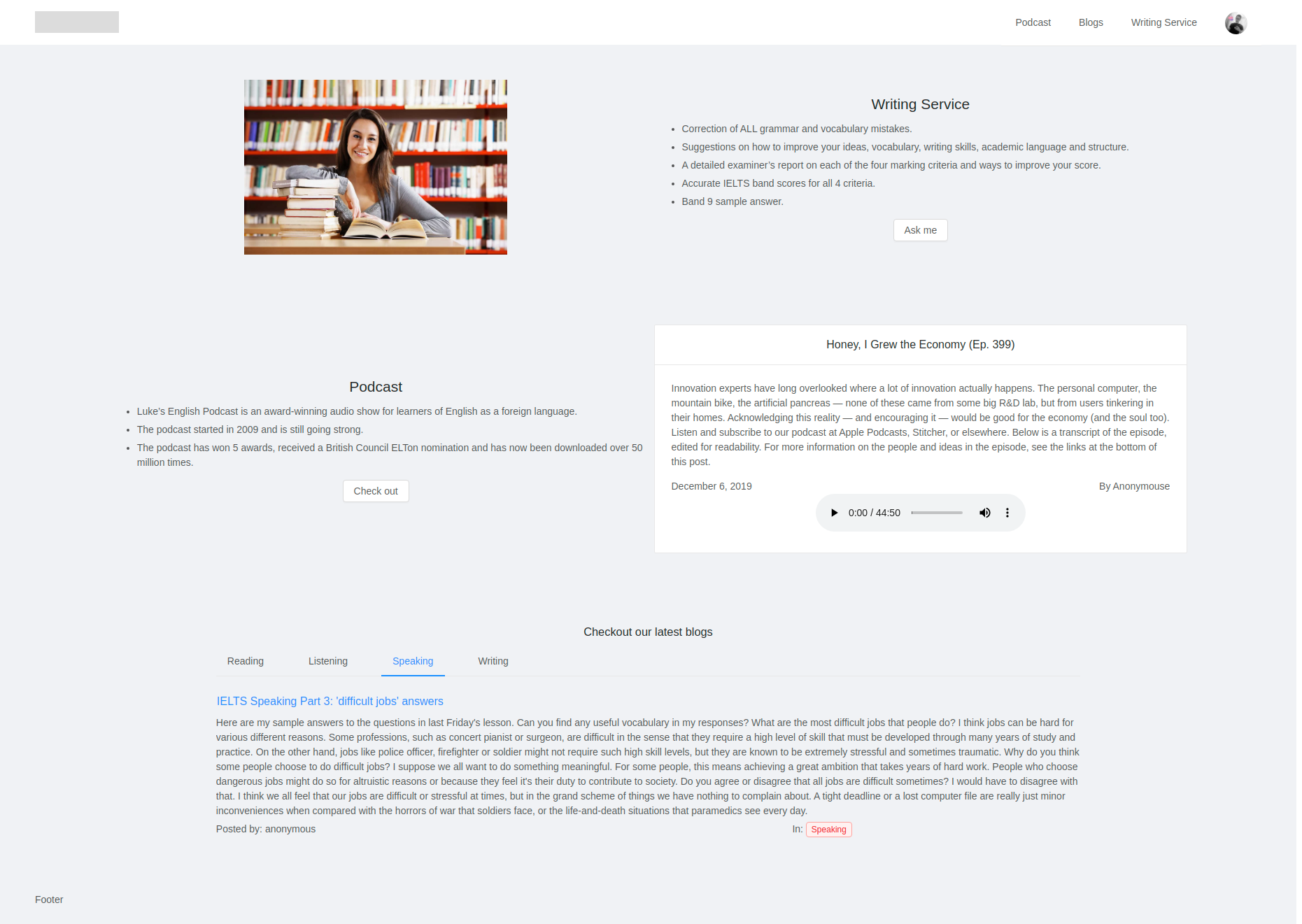
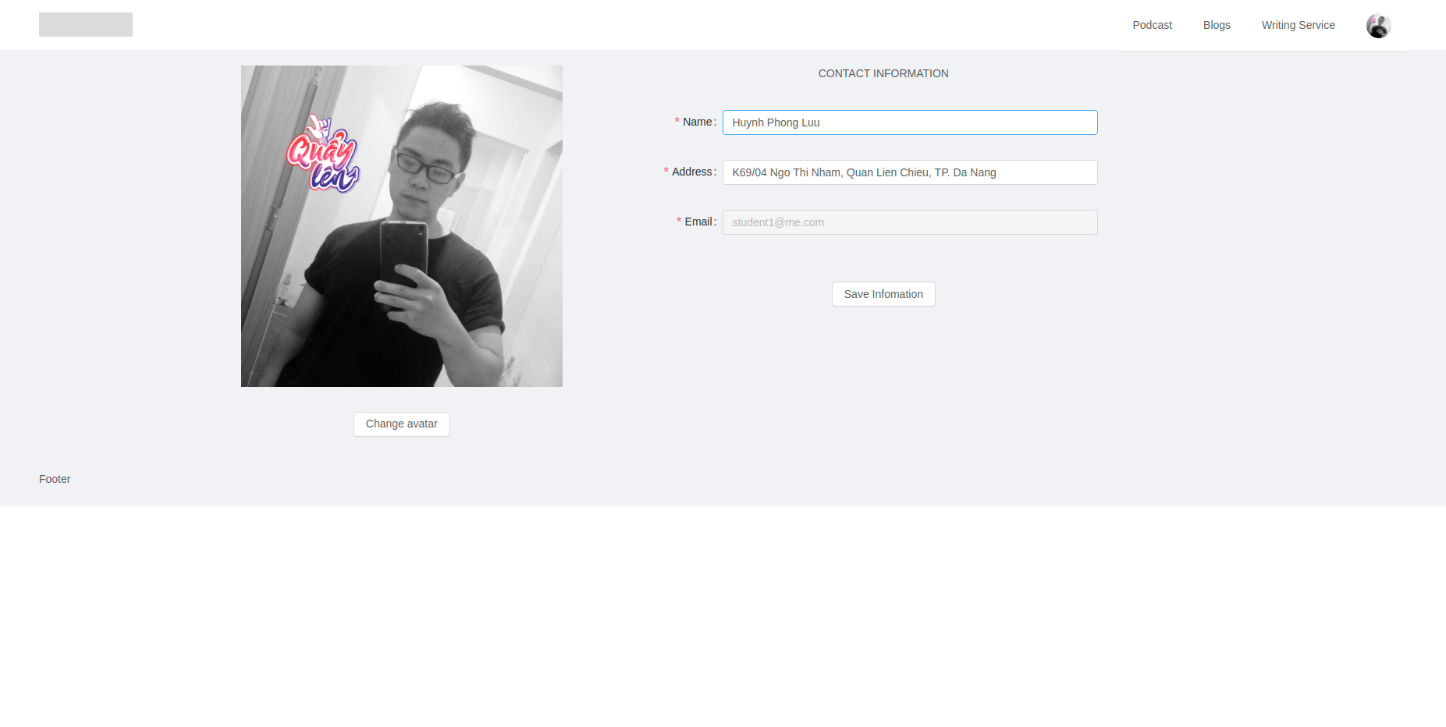


Figure 3.4 Home Screen.

Figure 3.4 Profile Screen.

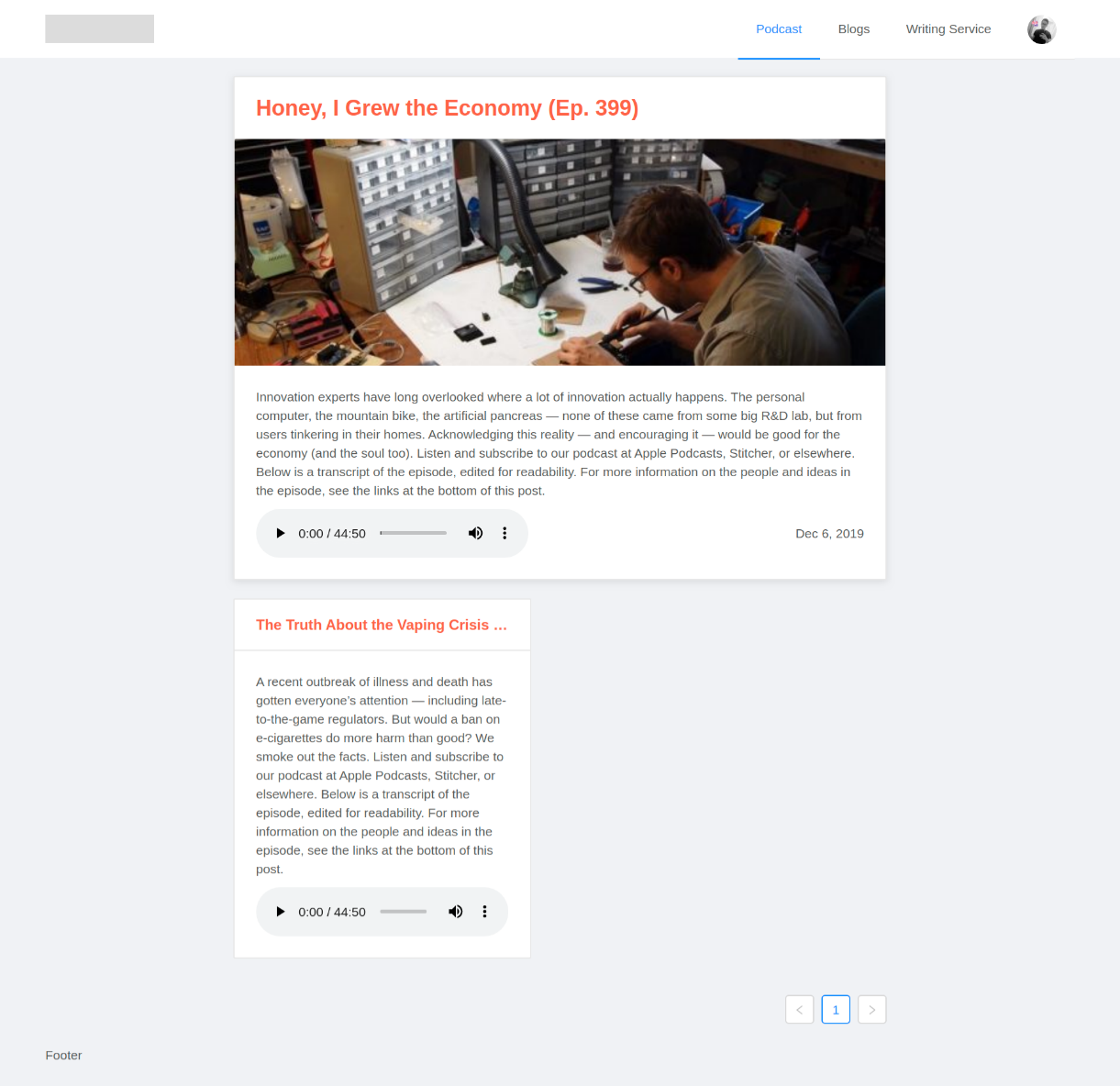


Figure 3.5 Podcasts List Screen

# CONCLUSION AND FUTURE WORKS

## Achieved results

Firstly, I had useful knowledge through working process in the enterprise environment, learned about teamwork skills and resolve the problem as the plan. In addition, I have a good plan to make sure process is stable and be on time.

About technique, I learned the basics of the NodeJS, ExpressJS, ReactJS, MongoDB, GitHub and how to create complete products.

The product was able to make good using and meet the basic demands of users, such as reading blogs, listening to the podcast and using writing services. However, there are still many difficulties, risks and challenges for the product. I will solve them as soon as possible.

## Future works

* Improve performance
* Redesign for user friendly UI
* Apply payment method for user who wants to use Writing Services
* Add some new features such as chatting, like, react to the post…