ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Business**		
Unit number and title	**Unit 30: Application Development**		
Submission date	3/3/2023	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Do Huu Duy	**Student ID**	GCC200018
Class	GCC0903	**Assessor name**	Luong Hoang Huong
Student declaration I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.			
		Student's signature	huuduy

Grading grid

P4	P5	P6	M3	M4	M5	D2	D3

❒ **Summative Feedback:**	❒ **Resubmission Feedback:**

Grade:	**Assessor Signature:**	**Date:**

Internal Verifier's Comments:

Signature & Date:

Assignment Brief 2 (RQF)

Higher National Certificate/Diploma in Computing

Student Name/ID Number:	
Unit Number and Title:	**Unit 30: Application Development**
Academic Year:	**2021 – 2022**
Unit Assessor:	**Hoang Nhu Vinh**
Assignment Title:	**Application development with design diagrams and code**
Issue Date:	**01 April 2021**
Submission Date:	
Internal Verifier Name:	
Date:	

Submission Format:

Format:

- An individual report document in PDF

Submission

- Students are compulsory to submit the assignment in due date and in a way requested by the Tutor.
- The form of submission will be a soft copy posted on http://cms.greenwich.edu.vn/.
- Remember to convert the word file into PDF file before the submission on CMS.

Note:

- The individual Assignment *must* be your own work, and not copied by or from another student.
- If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style.
- Make sure that you understand and follow the guidelines to avoid plagiarism. Failure to comply this requirement will result in a failed assignment.

Unit Learning Outcomes:

LO3 Work individually and as part of a team to plan and produce a functional business application with support documentation.

LO4 Evaluate the performance of a business application against its Software Design Document and initial requirements

Assignment Brief and Guidance:

Assignment scenario (continued from Assignment 1) Your team has finished the analysis and design for the system. Next task is development of the system.

Tasks:

After the presentation about your design (from Assignment 1), you need to create a formal questionnaire that effectively reviews your business application, problem definition statement, proposed solution and development strategy. This formal questionnaire should be answered by your colleagues. For any new insights, ideas or potential improvements to your system you need to evaluate and justify the reasons why you have chosen to include (or not to include) them as part of this business application. Based on the feedback of your colleagues, amend the design if needed.

Next task is to develop the business application based on the design, chosen technologies and methodology. When the application is fully built and tested, you need to review its performance against the Software Requirement Specificationn, analyze the factors that influence its performance and use them to undertake a critical review of the design, development and testing stages of your application. Conclude your review by reflectively discussing your previously identified risks. You should evaluate the strengths and weaknesses of your business application and fully justify opportunities for improvement and further development.

To conclude, your report document should include:

- Peer review section (questionnaire and answers, your reflection on the feedback)
- Development section (how you develop and test the application, what is the result)
- Review section (review, analyse and critical evaluate your application)

Your team needs to prepare a demo based on this report for the final demonstration.

The working application must also be demonstrated.

Learning Outcomes and Assessment Criteria (Assignment 2):			
Learning Outcome	Pass	Merit	Distinction
LO3	**P4** Create a formal questionnaire that effectively reviews your business application, problem definition statement, proposed solution and development strategy. Use this questionnaire as part of a peer-review and document any feedback given. **P5** Develop a functional business application based on a specified business problem.	**M3** Interpret your peer-review feedback and identify opportunities not previously considered. **M4** Develop a functional business application based on a specific Software Design Document with supportive evidence of using the preferred tools, techniques and methodologies.	**D2** Evaluate any new insights, ideas or potential improvements to your system and justify the reasons why you have chosen to include (or not to include) them as part of this business application.
LO4	**P6** Review the performance of your business application against the Problem Definition Statement and initial requirements.	**M5** Analyze the factors that influence the performance of a business application and use them to undertake a critical review of the design, development and testing stages of your application. Conclude your review by reflectively discussing your previously identified risks.	**D3** Critically evaluate the strengths and weaknesses of your business application and fully justify opportunities for improvement and further development.

Table of Contents

Table of Figures

Table of Tables

I. Peer Review and Feedback Analysis

1. Questionnaire about FPT Book Store Application

Table 1. Questionnaire table

No.	Question	Who	Date	Answer	Who	Date
1	Is the website compatible for each device used, such as a phone or laptop?	Mr. John (FPT bookstore owner and our customer)	21/2/2023	Yes, it is	Tran Chi Huynh	21/2/2023
2	How long does it take for the site to resolve an activity?	Mr. John (FPT bookstore owner and our customer)	21/2/2023	The web can solve an operation in 3 seconds	Do Huu Duy	21/2/2023
3	Can the administrator intervene and edit the information of other users?	Mr. John (FPT bookstore owner and our customer)	21/2/2023	With the Admin role, they can manage Category and Account. With manage account, the admin can view, lock, and unlock the Owner's account	Do Huu Duy	21/2/2023
4	Can the website run on different browsers?	Mr. John (FPT bookstore owner and our customer)	21/2/2023	Yes, it is. The web can run on some browser such as Opera, Google, and Microsoft Edge	Tran The Tien	21/2/2023
5	Can the website be supported to log in and log in by third parties such as Facebook, Zalo, Google account?	Mr. John (FPT bookstore owner and our customer)	21/2/2023	Our website currently does not apply this function and we	Do Huu Duy	21/2/2023

				will develop this function in future		
6	*Does the website have the ability to reply to messages with AI?*	*Mr. John (FPT bookstore owner and our customer)*	*21/2/2023*	*Now, our website hasn't performed this feature yet. We will research and develop this feature in future*	*Tran The Tien*	*21/2/2023*
7	*The Owner role of the website, what can they perform in the web?*	*Mr. John (FPT bookstore owner and our customer)*	*21/2/2023*	*With the Owner role, they can manage Categories, Publishers, Orders, and Books of the store. They can perform some CURD function for each management task*	*Do Huu Duy*	*21/2/2023*
8	*Is it possible to cancel orders after they are placed on the website?*	*Mr. John (FPT bookstore owner and our customer)*	*21/2/2023*	*Yes, customers can cancel orders after they are placed on the website.*	*Tran Chi Huynh*	*21/2/2023*
9	*Can customer accounts be deleted or locked by the admin?*	*Mr. John (FPT bookstore owner and our customer)*	*21/2/2023*	*No, the admin has not have the right to delete and lock customer accounts on the website.*	*Tran Chi Huynh*	*21/2/2023*
10	*Does the admin have the right to add, lock, and unlock the owner account?*	*Mr. John (FPT bookstore owner and our customer)*	*21/2/2023*	*Yes, the admin has the right to add, lock, and unlock the owner account.*	*Tran The Tien*	*21/2/2023*

2. Interpret peer-review feedback

2.1. Question 1

Question: *Is the website compatible for each device used, such as a phone or laptop?*

Answer: *The website can be compatible for each device separately such as laptop or smart phone*

Explain: *When we use the laptop to access the website, the interface of the website will be responsive for this device about resolution and when we use the devices have the smaller resolution like smart phone, the web's interface also will be responsive about resolution*

2.2. Question 2

Question: *How long does it take for the site to resolve an activity?*

Answer: *Our website will meet the non-function requirement about the time perform a function do not exceed 3 seconds*

Explain: *When we perform a function such as login, the action will perform do not exceed 3 seconds*

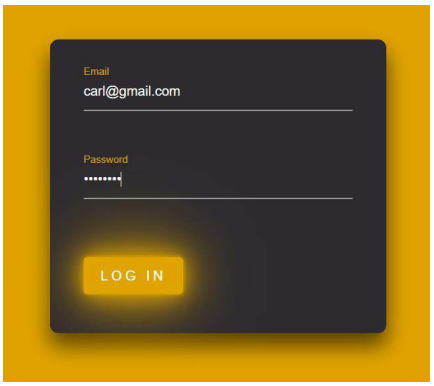


Figure 1. Figure that present for the question 2

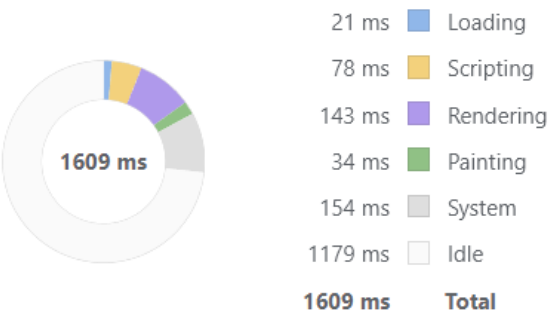


Figure 2. Figure that present for the question 2

Now, we can see the time to perform the login function will take 1609ms.

2.3. Question 3

Question: *Can the administrator intervene and edit the information of other users?*

Answer: *With the administrator role, they can view, lock, and unlock the owner's account*

Explain: *In this application the admin will allow management account. They can view all the accounts of the owner and they can lock or unlock the account of each owner.*

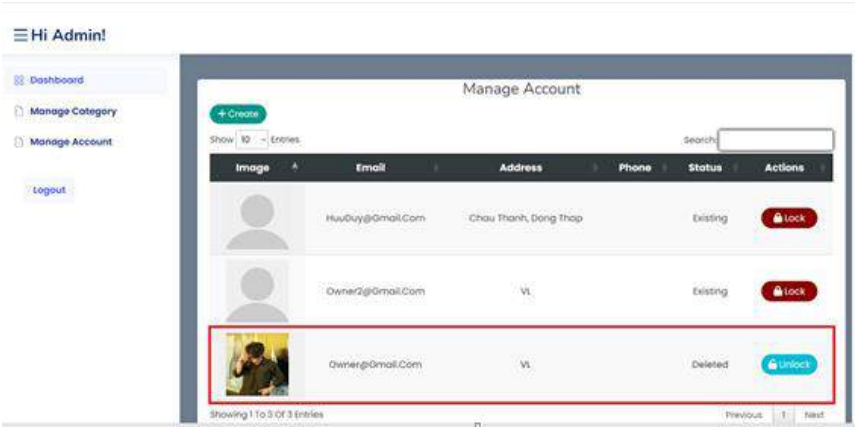


Figure 3. Figure that present for the question 3

The account that has been locked.

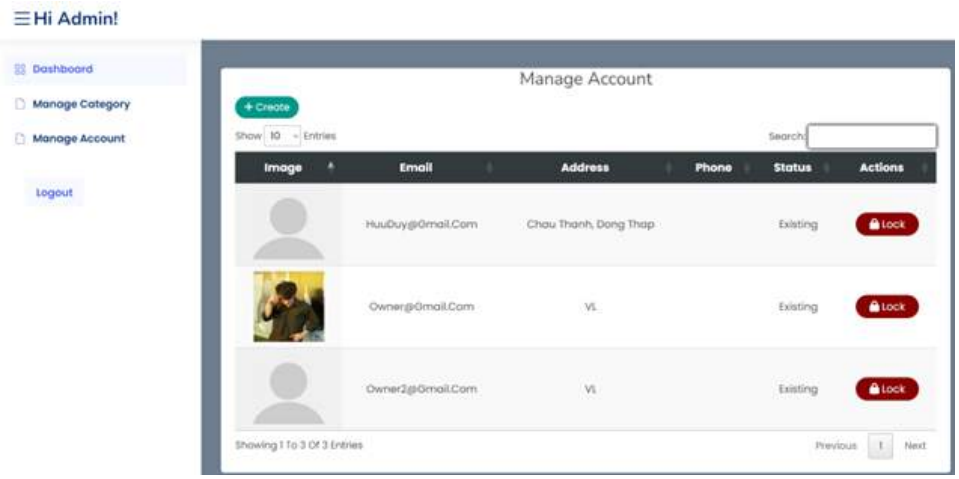


Figure 4. Figure that present for the question 3

Accounts that have not been locked yet.

2.4. Question 4

Question: *Can the website run on different browsers?*

Answer: *Our website can run in some browser such as Opera, Chrome, and Microsoft Edge*

Explain: *We can use one of three browser such as Opera, Chrome, and Microsoft Edge to access the website*

Run on Opera browser.

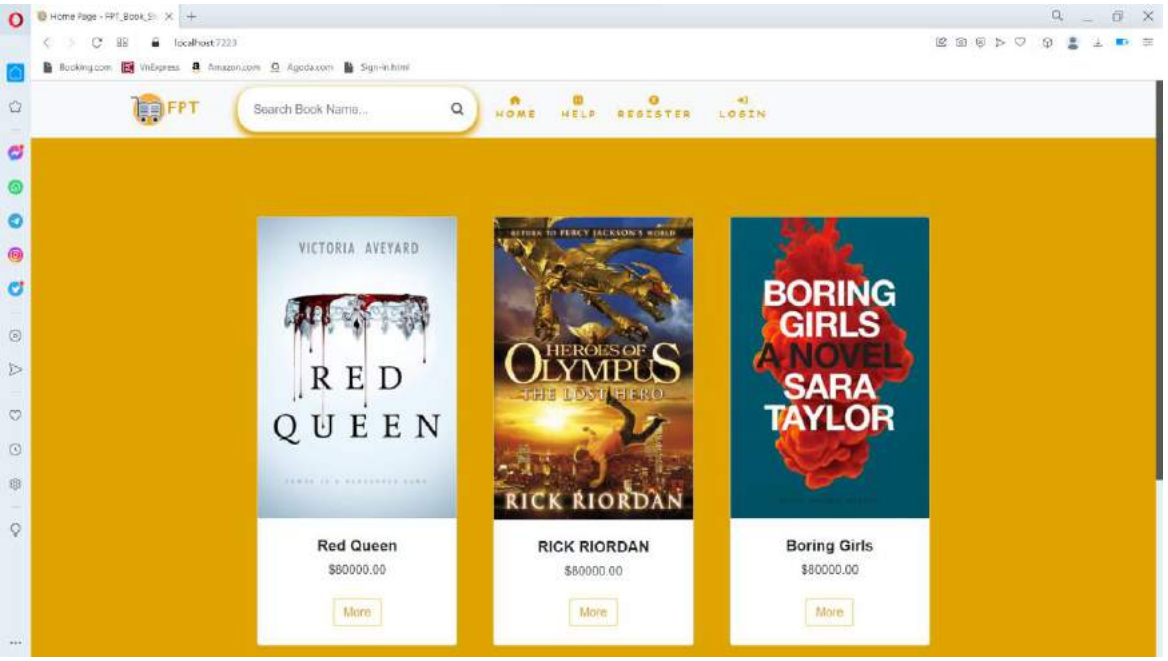


Figure 5. Figure that present for the question 4

Run on Chrome browser.

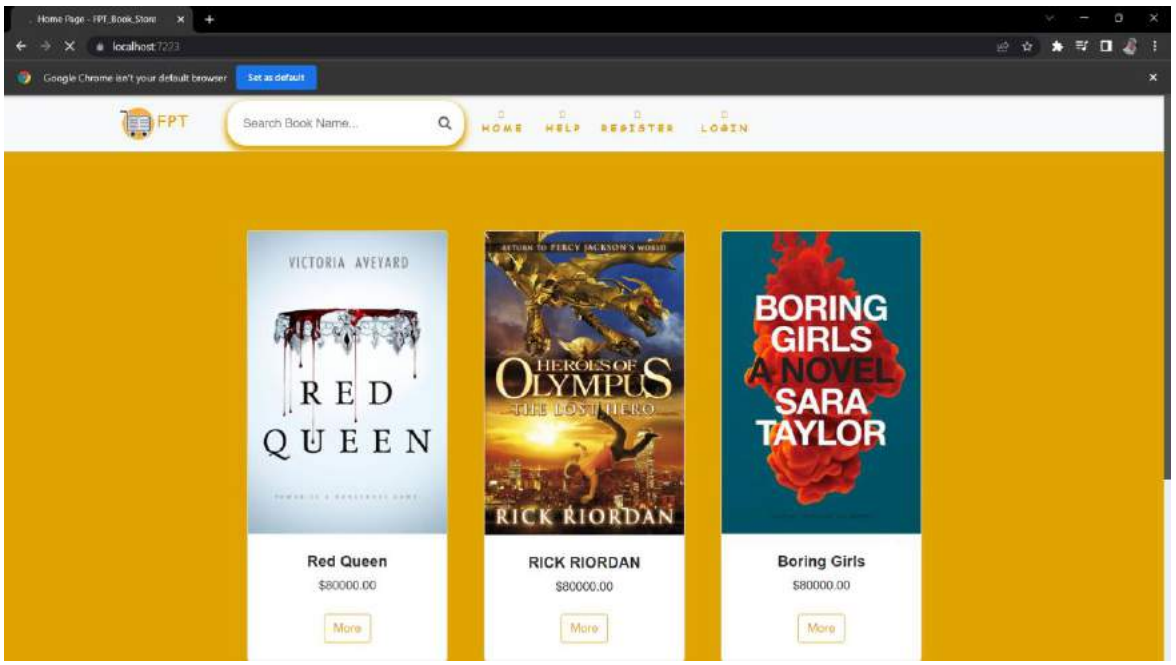


Figure 6. Figure that present for the question 4

Run on Microsoft Edge.

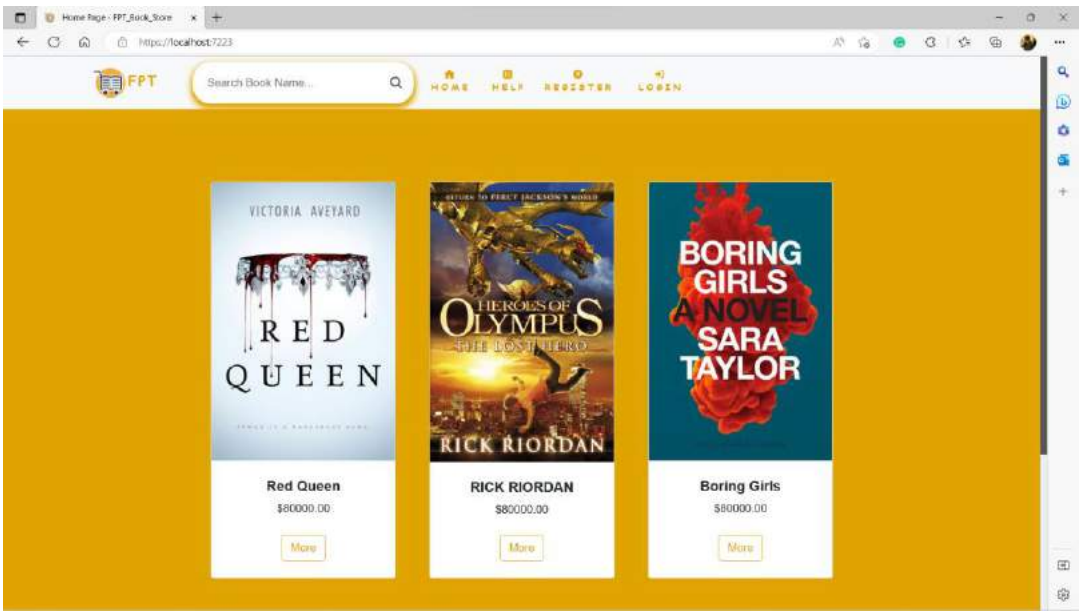


Figure 7. Figure that present for the question 4

2.5. Question 5

Question: *Can the website be supported to log in and log in by third parties such as Facebook, Zalo, Google account?*

Answer: *Our website does not support login by the as Facebook, Zalo, Google account?*

Explain: *When we want to login to the website, we must register an account before and then we will login with the account that just register*

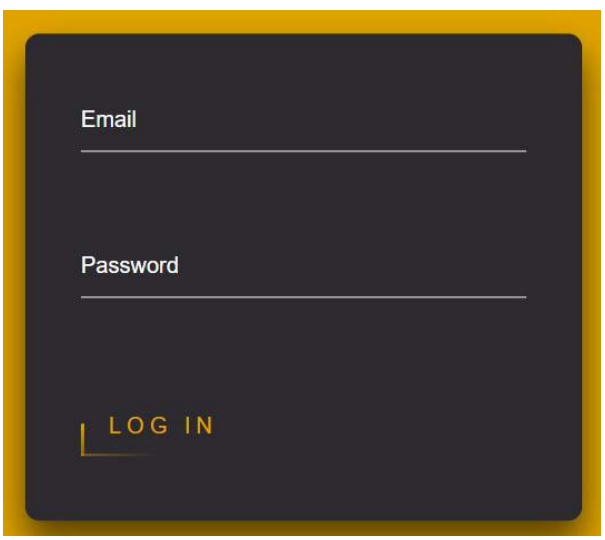


Figure 8. Figure that present for the question 5

We can see the login form requires us to enter the email and password of the account to login. Do not have any third parties account like Facebook, Zalo, Google account to support login.

2.6. Question 6

Question: *Does the website have the ability to reply to messages with AI?*

Answer: *Our website doesn't have the ability reply to message with AI*

2.7. Question 7

Question: The Owner role of the website, what can they perform in the web?

Answer: With the owner role, they can perform to manage the categories, books, orders, and publishers

Explain: In the owner role, the owner will login to their account and perform managing books, categories, publishers, and orders. They can add, update, and delete books, categories, and publisher management function. With the order management, they can confirm the customer's orders.

These actions will be presented in part 3.1.7, 3.1.8, 3.1.10, and 3.1.11 of part 3.1. Demonstration the application

2.8. Question 8

Question: *Is it possible to cancel orders after they are placed on the website?*

Answer: *The customer can cancel their orders after they ordered*

Explain: *The customer can access the Order page to view all of their orders that they ordered and they can cancel any orders that they want, but with the orders that shipped, they cannot cancel those orders.*

This action will be presented in part 3.1.14 of part 3.1. Demonstration the application

2.9. Question 9

Question: *Can customer accounts be deleted or locked by the admin?*

Answer: *The admin cannot delete and lock the customer's account, they just allow lock and unlock the owner's account, so the customer's account will not be deleted or locked by admin*

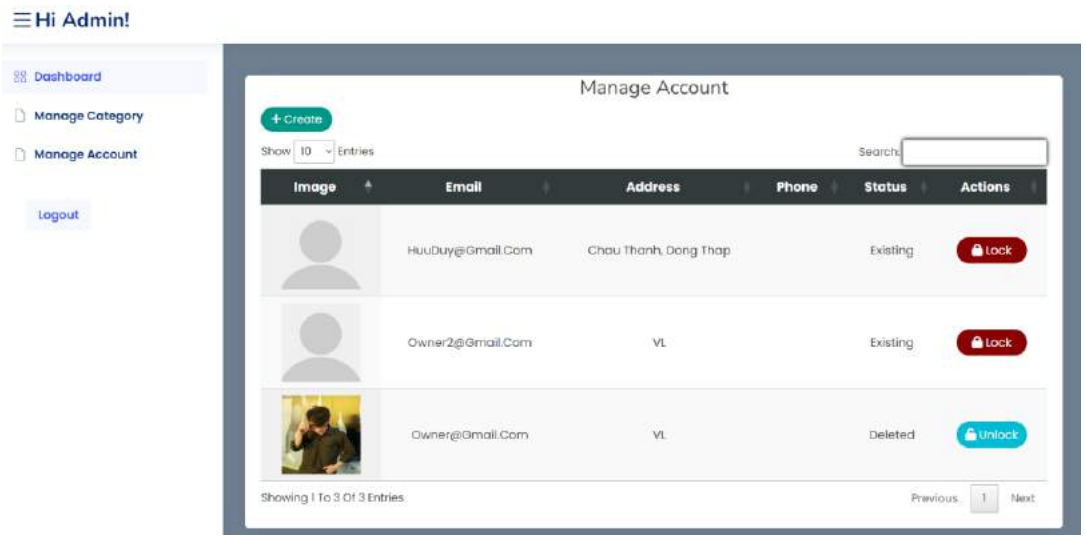


Figure 9. Figure that present for the question 9

We can see, the admin just can manage accounts of owner and they just can lock and unlock the owner's account.

2.10. Question 10

Question: *Does the admin have the right to add, lock, and unlock the owner account?*

Answer: *With the admin, they can manage the account of the owner, so they can lock and unlock the owner's account. In addition, they can add a new account for the owner, admin, or user.*

This action will be presented in part 3.1.12 of part 3.1. Demonstration the application

3. Evaluate any new insights, ideas or potential improvements

After building the application, I feel our website interface is simple and the web's interface is not too impressive. In addition, we built this website by the .Net Core framework with the C# programming language and building the website with the pure code that does not use API or Ajax. I suggest that should use API to perform the web because in the learning process I feel the API is useful. With API, we can build HTTP services: URIs, request/response headers, caching, versioning, content formats and can be hosted in the application or on IIS quickly. Besides, API is used mostly on desktop applications, mobile applications and website applications. This will help the website become friendly and professional when using numerous devices with different resolutions to access. Moreover, API is open source, supports full RESTful functionality, used by any client that supports XML, Json.API also fully supports MVC components. These are reasons that I suggest building a website with API to build and develop the web with cross-platform, faster.

In this project, we use the IIS server to deploy the website, this is cloud hosting. When we deploy the website to cloud hosting, the web will be safe and secure. Cloud hosting will help our website improve any server problems such as hacking, failure of the hardware or the system overload. Using cloud hosting we can back up the data avoid getting risk that unexpected. Cloud hosting helps us deploy the FPT_BOOK_STORE website safely and efficiently.

To sum up, our website is built like what we expected and did in assignment 1 from diagram, tools, technical. However, in the future we will improve and expand our website in some functions such as login with third parties, replying to message by AI, and editing interface nicer and more professional.

II. Application Development

1. Develop a functional business application

1.1. Develop Tools

- *Visual Studio Code*

According to the (Uzayr, 2021), Microsoft developed the open-source Visual Studio Code (VS Code) code editor for Windows, Linux, and macOS. Support for debugging, syntax highlighting, automatic code completion, snippets, code reorganization, and embedded Git are a few of the widely used standard features. Users can increase the functionality of the project overall by installing additional extensions, changing the design theme, preferences, and keyboard shortcuts.

Figure 10. Visual Studio Code

- *SQL Server*

According to the (Shanmugam, 2022), Relational database management system Microsoft SQL Server (RDBMS). In plainer terms, it is a piece of software created to manage databases, which serve as data repositories for other programs. One of the most often used databases is SQL Server, which is utilized in a variety of tasks such as analytics, business intelligence, and online transaction processing. The main purpose of SQL Server is to manage and store data in databases. SQL Server, nevertheless, now offers more than simply a database. With SQL Server, Microsoft has included a number of tools, including data management, business intelligence (BI), and analytics capabilities.

Figure 11. SQL Server

1.2. Technique

- *HTML*

According to the (Goyal, 2022), HTML is a markup language, it uses simple tags to format and mark up content. These tags, like HTML, are enclosed in curly braces. Almost all tags also have a closing tag. The html> tag tells the browser that an HTML document has been started, and the /html> tag indicates the end of the HTML document. Any code written inside these two tags is sent to the browser. Browsers then display the content contained in the body tag. HTML specifies the format in which web items should be displayed. To view your content, save your file with the .html or .htm extension, and then run it by selecting the option to

open it in any browser. HTML is one of the best options for developing a website or website for a small business or a growing business.

Figure 12. HTML

- *CSS*

According to the (Reddy, 2019), The look and feel of a web page are controlled by CSS. The color of the text, the font style, the distance between paragraphs, the size and arrangement of columns, and other elements can all be changed using CSS.

Figure 13. CSS

- *Framework Bootstrap*

Responsive web design is now a reality thanks to Bootstrap. It enables a website or app to recognize the size and orientation of the visitor's screen and automatically adjust the display. The mobile-first strategy presupposes that employees' main tools for accomplishing their work are smartphones, tablets, and task-specific mobile apps. Bootstrap provides UI elements, layouts, JavaScript tools, and an implementation

framework to handle the design requirements of those technologies. The program is offered both precompiled and as source code.

1.3. Framework and Programming language

- *.Net Core*

According to the (tutorialsteacher, 2023), A new iteration of Microsoft's free, open-source, general-purpose programming platform called.NET Framework is called.NET Core. It is a cross-platform framework that works with Linux, macOS, and Windows.

- *C#*

According to the (GeeksforGeeks, 2019), A versatile, contemporary, and object-oriented programming language is C#. The European Computer Manufacturers Association (ECMA) and the International Standards Organization both gave their approval for its development by Microsoft under the direction of Anders Hejlsberg and his team inside the.Net program (ISO). Version 7.2 of C#, one of the languages for Common Language Infrastructure, is currently available. For users who are familiar with C, C++, or Java, C# is simple since it shares many syntactical similarities with Java.

1.4. Deployment

- *IIS Server*

According to the (Morris, 2022), Running on the Windows OS and Microsoft.NET platform is an IIS web server. IIS can be used with Mono to operate on Linux and Mac computers. It has been extensively utilized in production for many years because it is adaptable and reliable. The most recent is version 10. After installation, your browser will display this welcome page. A single web platform called IIS Web Server unifies IIS, FTP services, PHP, ASP.NET, and Windows Communication Foundation (WCF). Because it automatically isolates applications, comes with a pre-configured sandbox, and has a smaller server footprint, you can use it to host your websites and services with the highest level of security. To speed up your website, it also incorporates dynamic caching and improved compression. Developers can add unique modules to the modular platform to increase its capabilities.

IIS supports the following protocols:

- *Hypertext Transfer Protocol (HTTP)*
- *Hypertext Transfer Protocol Secure (HTTPS)*
- *File Transfer Protocol (FTP)*
- *File Transfer Protocol Secure (FTPS)*
- *Simple Mail Transfer Protocol (SMTP)*
- *Network News Transfer Protocol (NNTP)*

2. Source control

- *GitHub*

According to the (Kinsta, 2022), A for-profit organization called GitHub provides a service for hosting Git repositories on the cloud. In essence, it makes it much simpler for both individuals and teams to utilize Git for collaboration and version control. Because of GitHub's user-friendly design, even newbie programmers can benefit from Git. Without GitHub, utilizing Git typically necessitates a little more command-line experience and technical know-how. However, because GitHub is so user-friendly, some individuals even use it to handle different kinds of projects, including writing books. Additionally, anyone may join and host a public code repository on GitHub for no cost, which is why open-source projects are particularly fond of it.

In this project, we use GitHub to manage the source code of the application.

Figure 14. Github

3. Presentation

3.1. Demonstration the application

3.1.1. Register function

In this function, user needs to enter all field of the register form to perform the register an account.

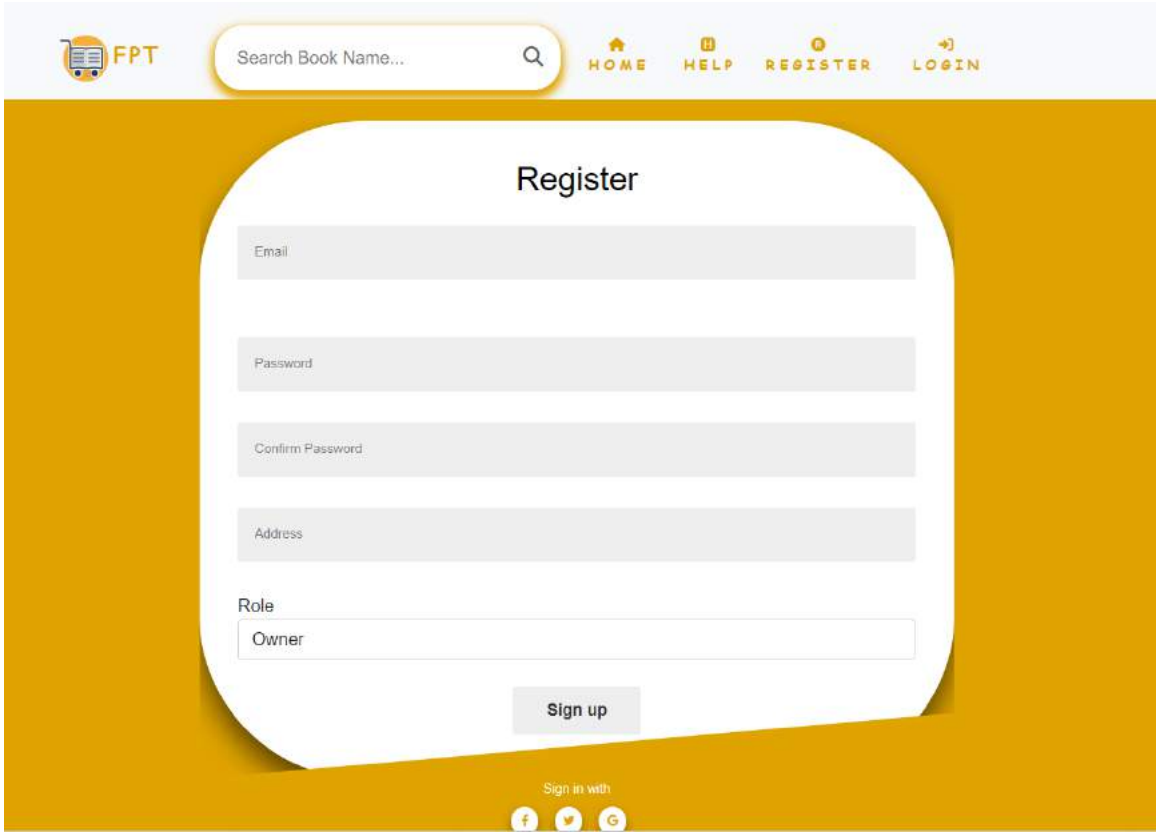


Figure 15. Register interface

After registering successfully, the web will pick user up to the home page.

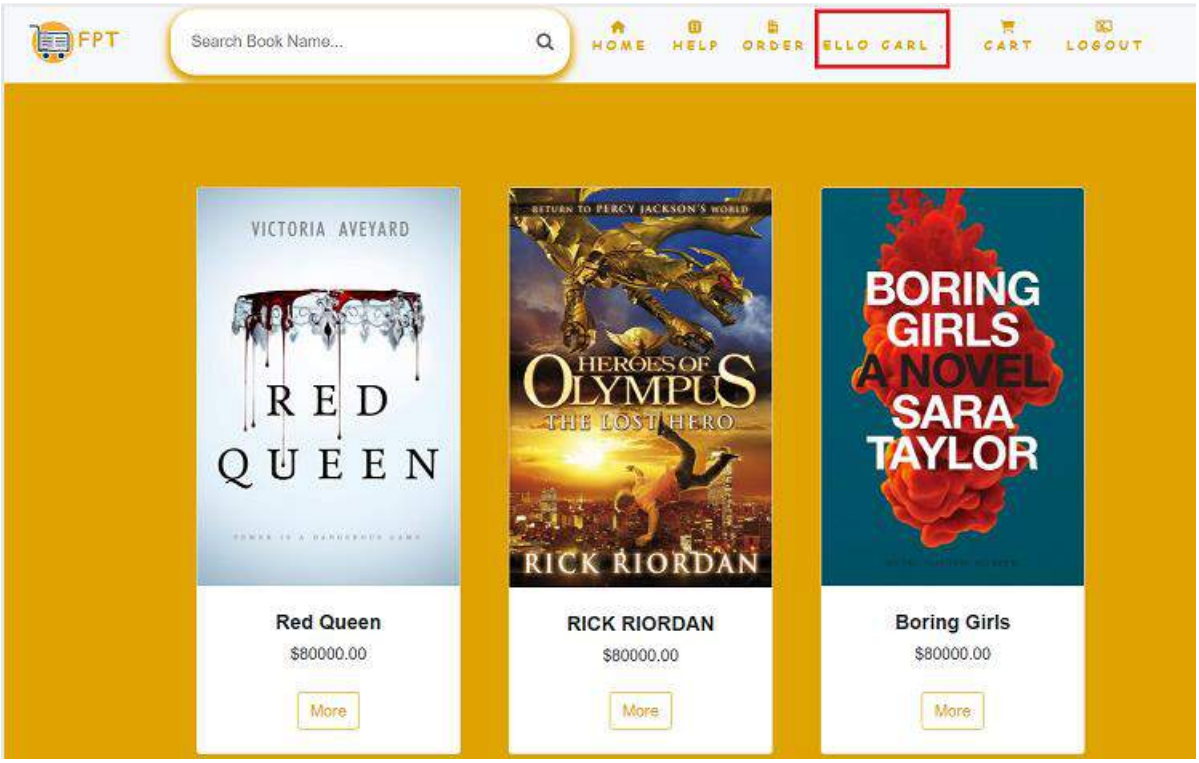


Figure 16. Home interface

3.1.2. Login function

After registering successfully, the user that has an account can login with their account. The user will enter the email and password, then click on the LOG IN button to login.

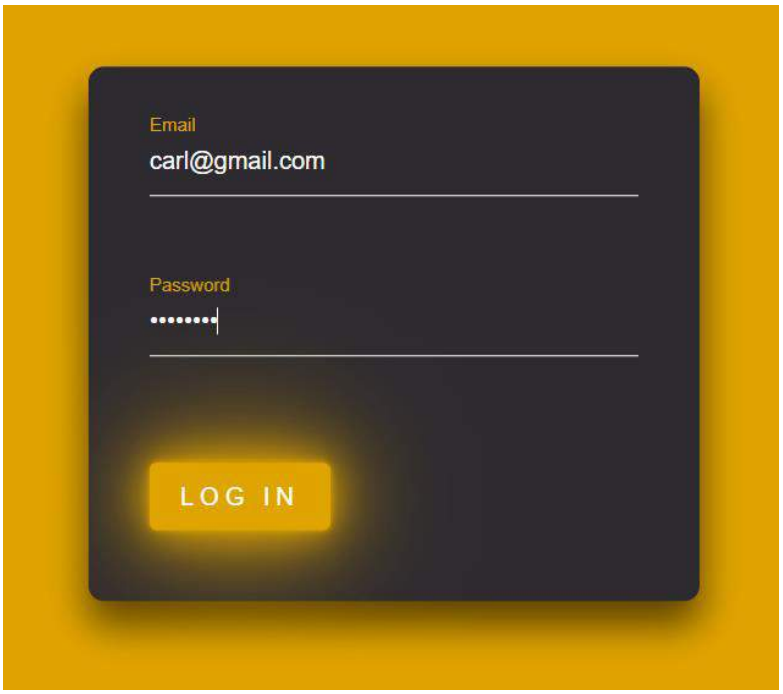


Figure 17. Login interface

3.1.3. View Book Detail function

After login successfully, the web will pick user up to home page and in the home page, user will view the book and they can click on "More" button to view book's detail.

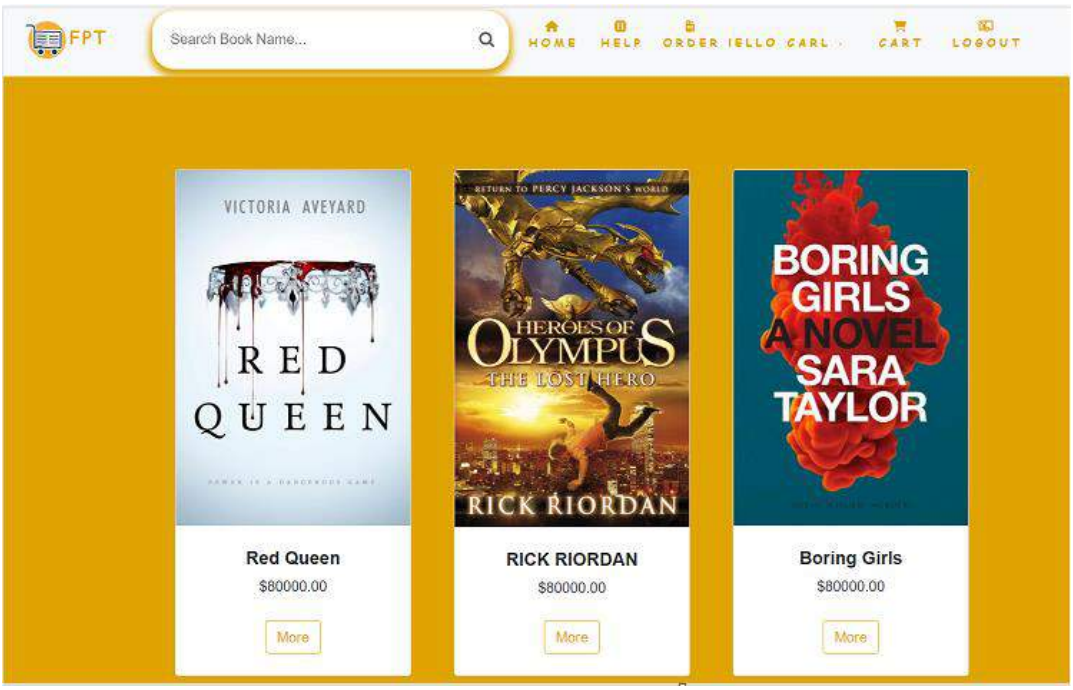


Figure 18. Home interface

After click on the More button, the book's detail page will be displayed, and user can view detail of the book.

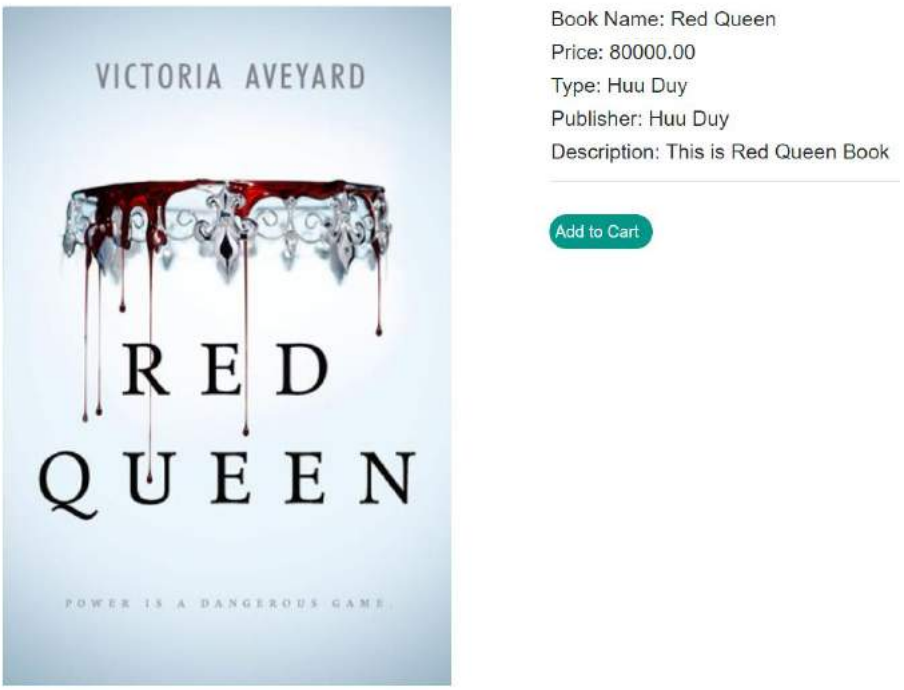


Figure 19. Book Detail interface

3.1.4. Add to Cart function

In order to add the book to the cart, they can click on the "Add to Cart" button in book's detail page

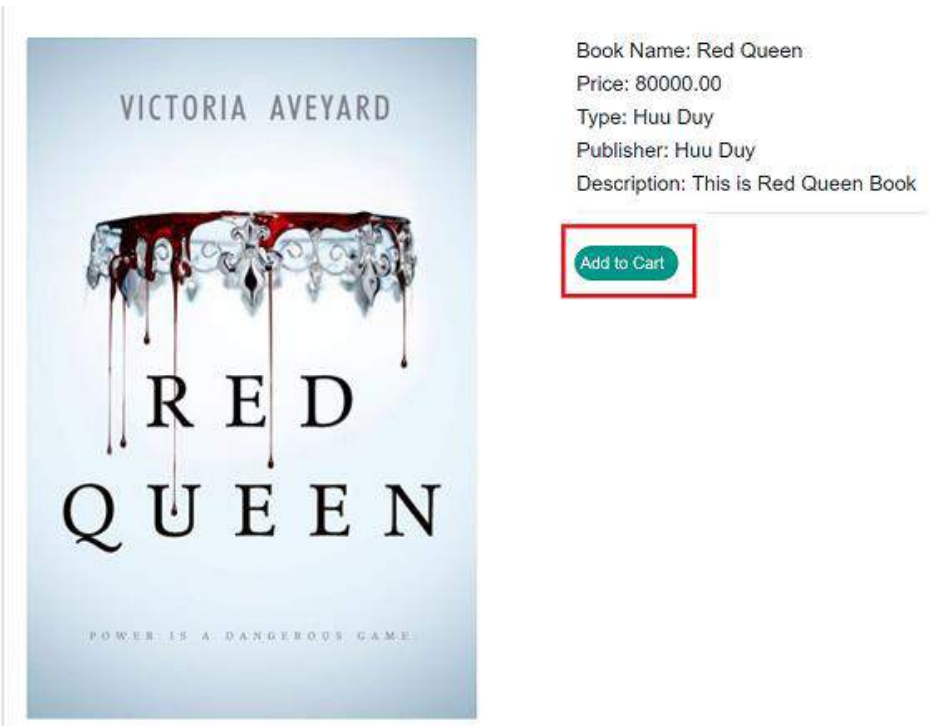


Figure 20. Book Detail interface that presents for add to cart function

After clicking on the "Add to Cart" button the product will be in their cart.

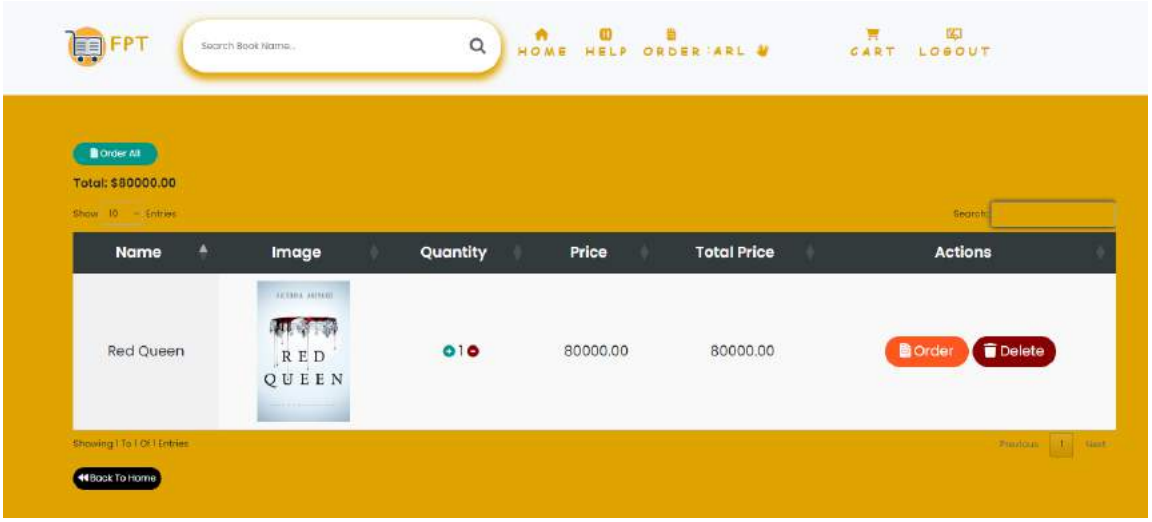


Figure 21. Cart interface

3.1.5. Order one

In the user's cart, they can order one product that they want by clicking on the order button in each product. But the user needs to go to their profile to update the phone number, then they can order.

After updating the phone number, now they can order.

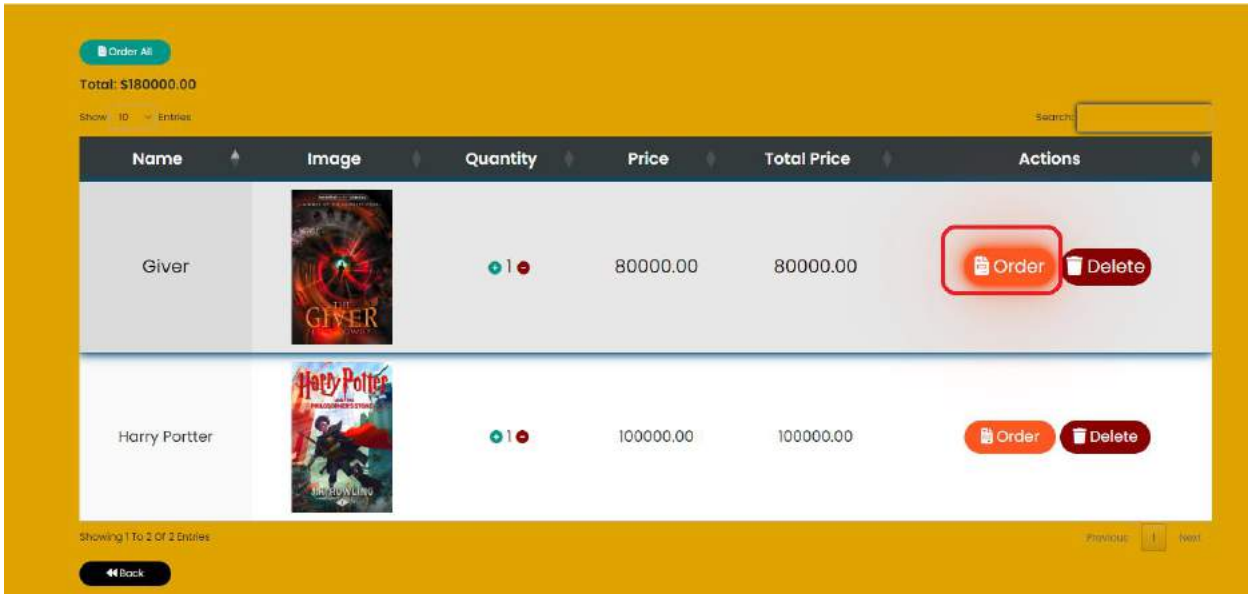


Figure 22. Cart interface that present for order one function

When a user click on the Order button the confirm box will be display and ask the user sure to order, now if the user wants to order they need to click on Ok button to perform order, but if they don't want, they can click on Cancel button.

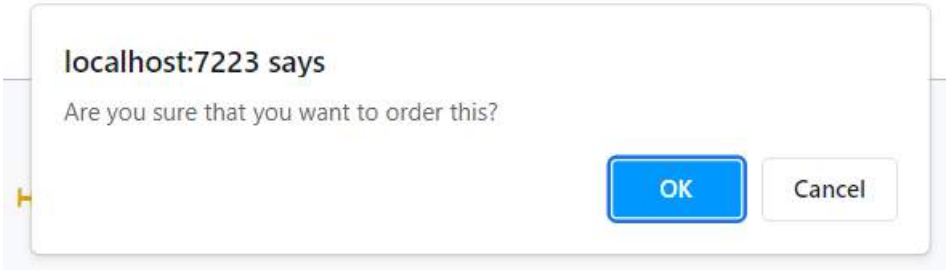


Figure 23. Message alert box

When order successfully the web will display a message "Order Is Successfully!"

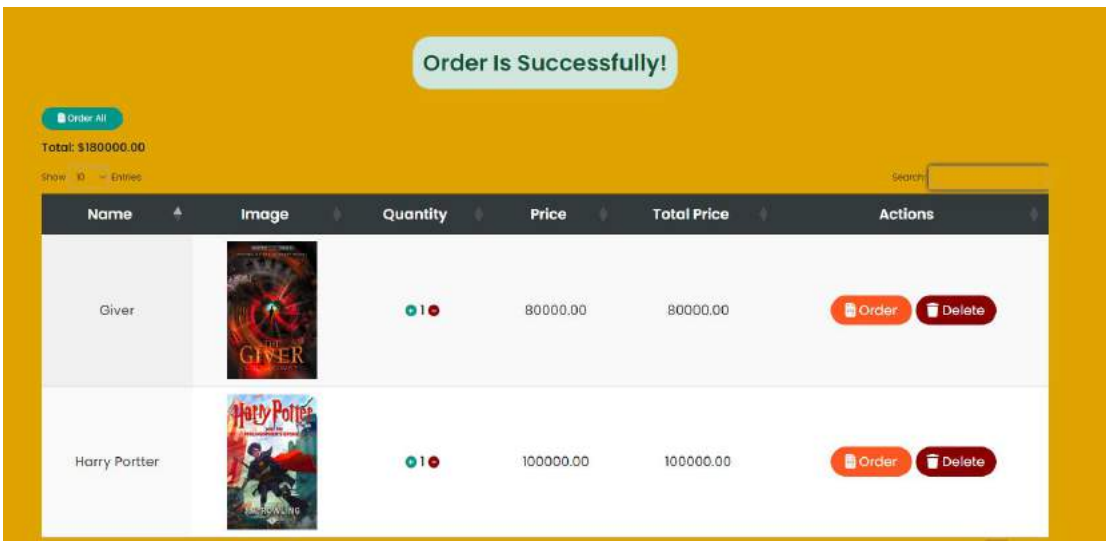


Figure 24. Cart interface after order one successfully

3.1.6. Order All

In the case, user wants to order all of product that have in their cart, they can click on the "Order All" button to order all product in their cart.

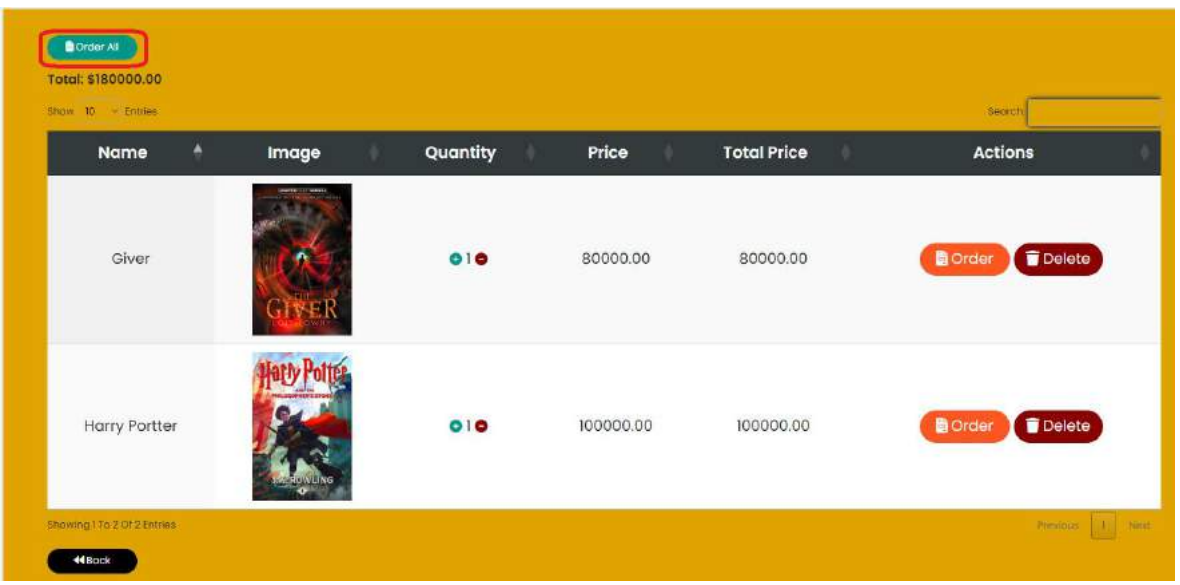


Figure 25. Cart interface that present for order all function

When ordering successfully, the web also will display a message "Order Is Successfully!"

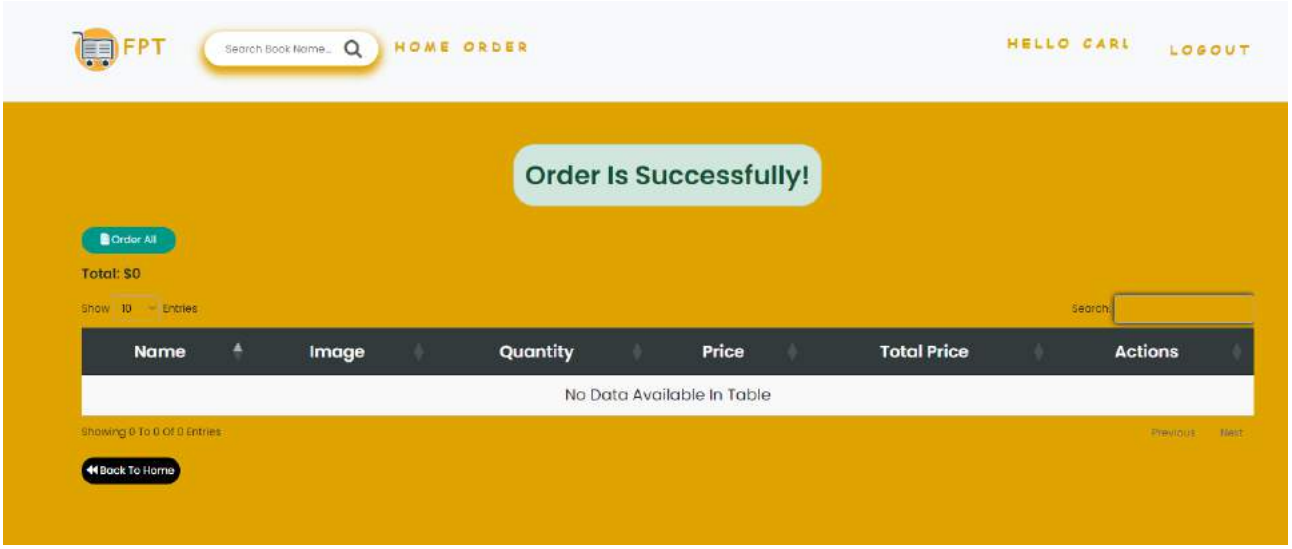


Figure 26. Cart interface after order all successfully

3.1.7. Manage Book function

In the manage book function the owner can add, update, and delete the book that they want. In order to perform this function, the owner needs to login in their account and the owner dashboard will be displayed.

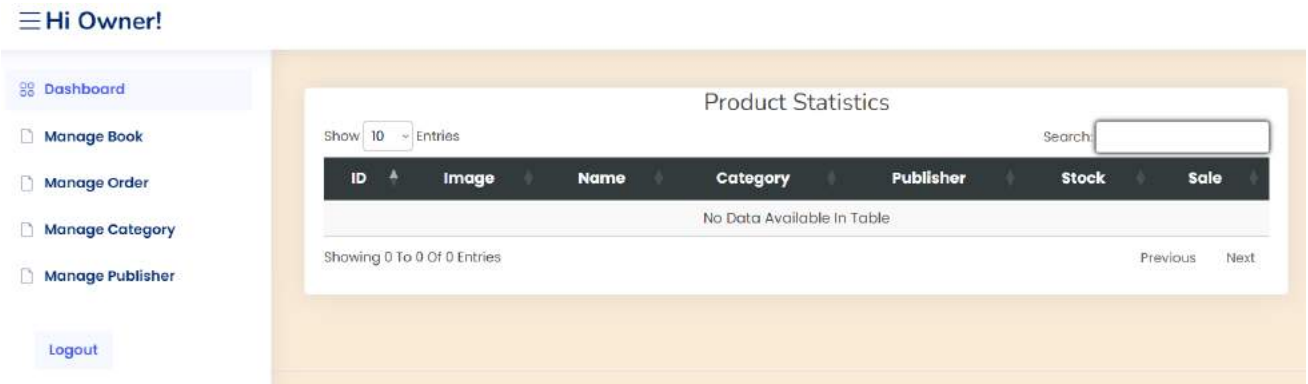


Figure 27. Dashboard interface of Owner role

Now, the owner needs to click on Manager Book to perform manage books

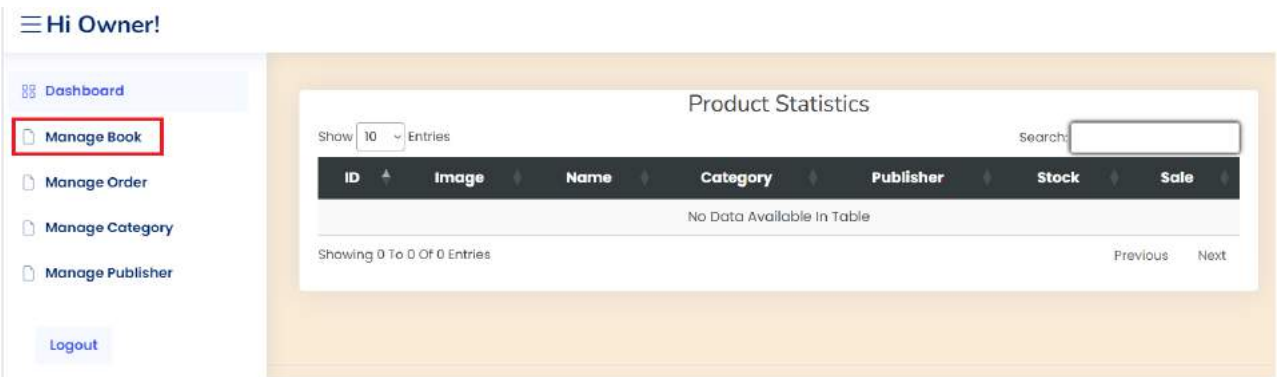


Figure 28. Dashboard interface that present for manage book

When clicking on the Manager Book, the manager book page will display

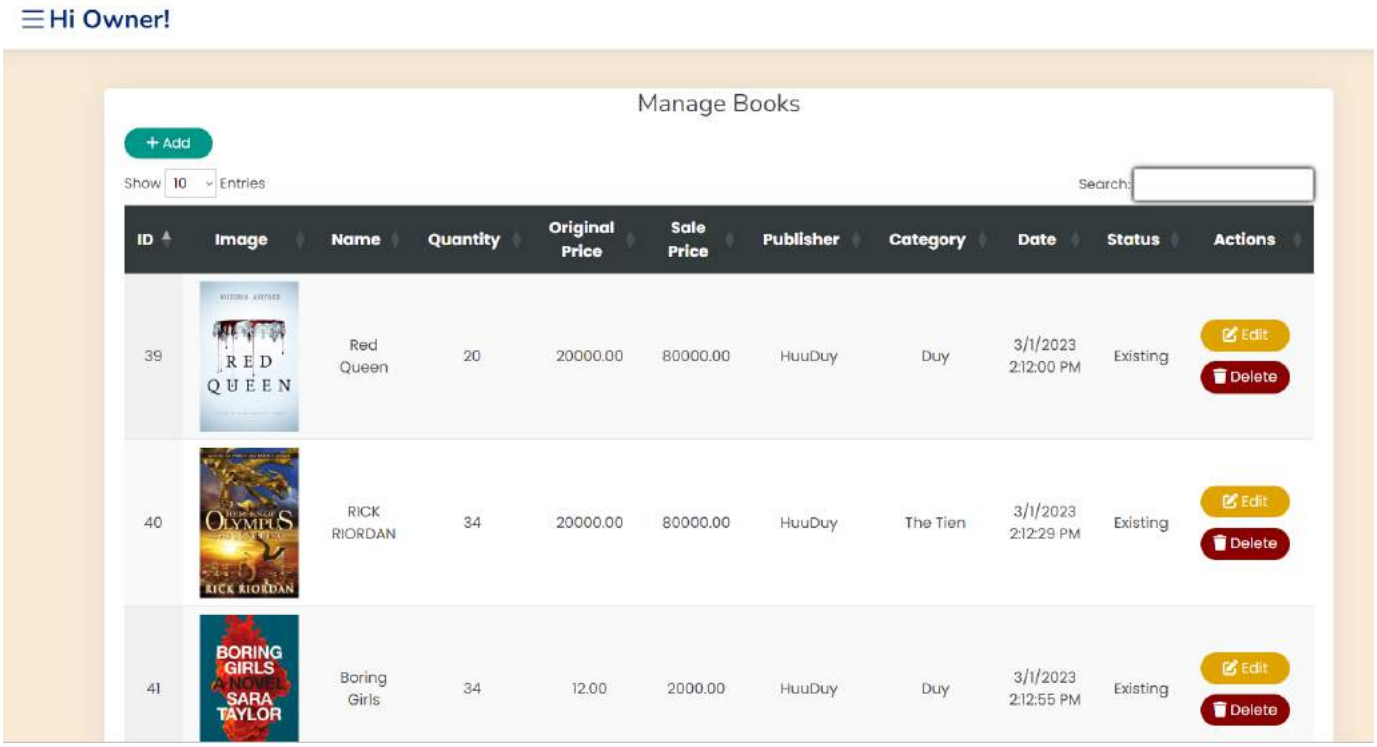


Figure 29. Manage Book interface

Now, the owner can perform adding, updating, and deleting the books. First, I will show you the add new book function. We need to click on "Add" button to perform the add new book function and when click on the Add button the form adding book will display. Then, we will fill all fields and click on Add button to add a book.

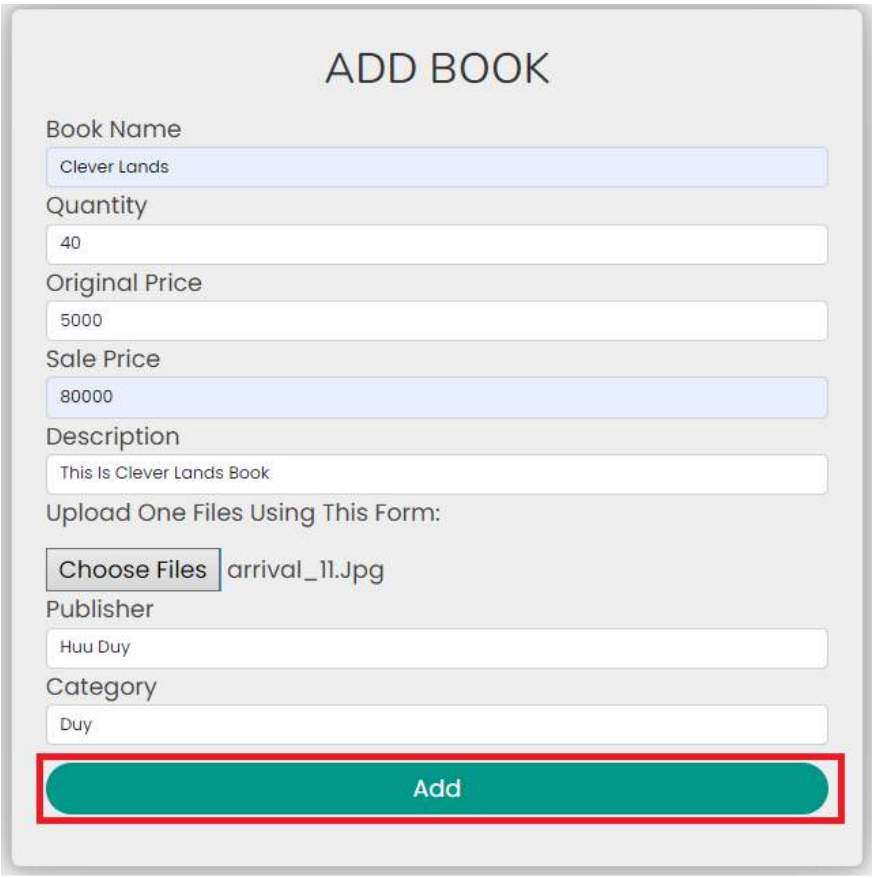


Figure 30. Form adding a new book

After adding successfully, the new book will display in the manager book page.

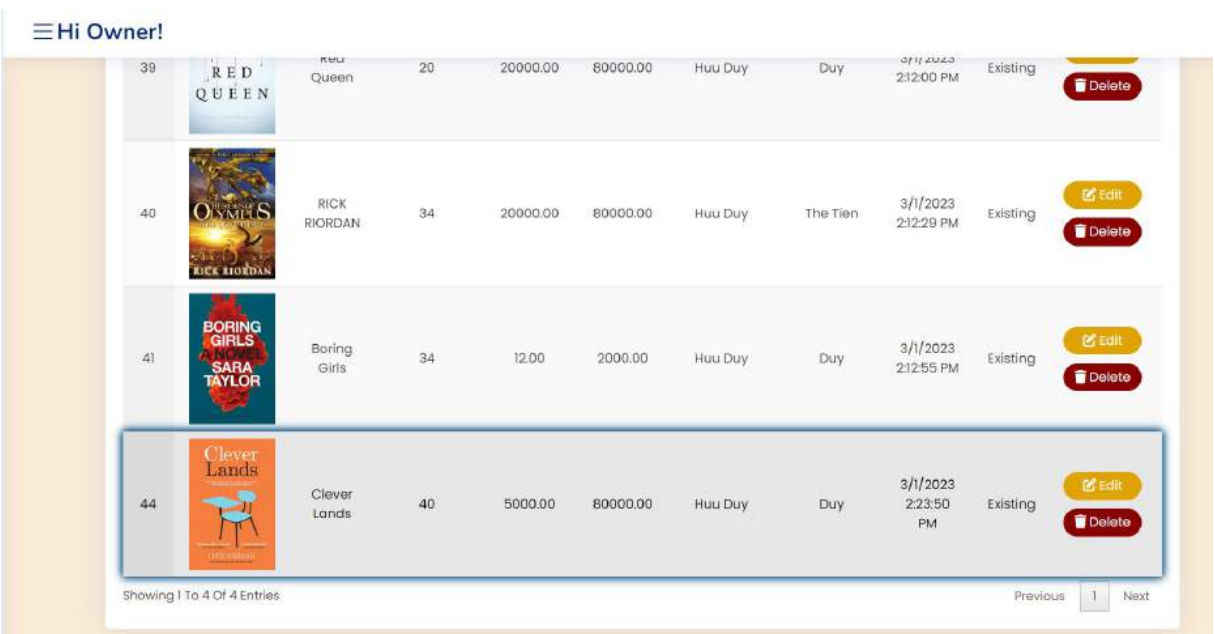


Figure 31. Manage Book interface after adding a new book

Next, I will show you about the update book function. In order to update the book's information, the owner can click on the Edit button in each books to perform updating book.

Figure 32. Manage Book interface that present for updating a book

When clicking on the Edit button, the form edit will display and the owner can update any information that they want.

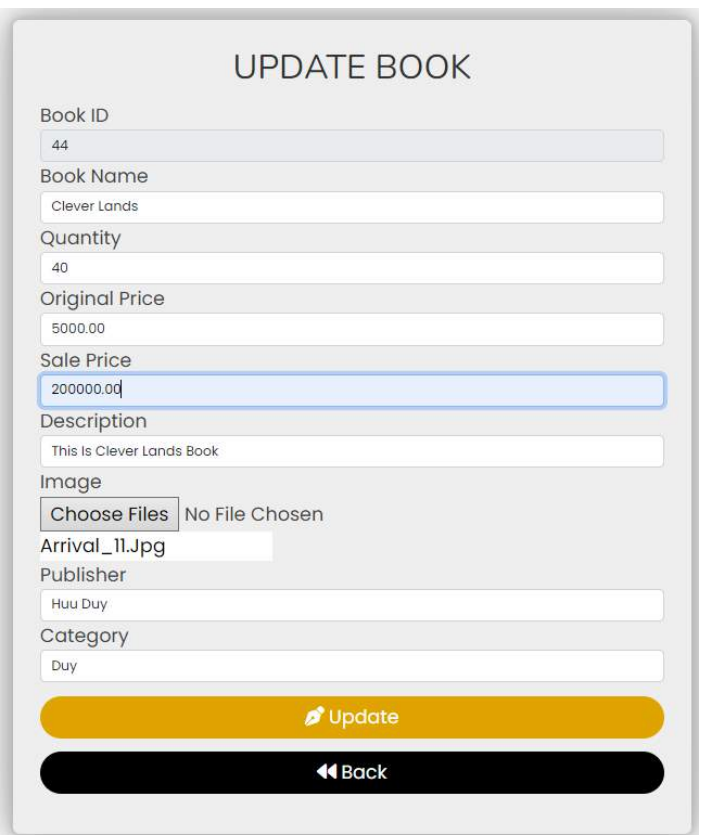


Figure 33. From updating book

In this the old information of the book and now I will update the sale price of the book from 80000 to 200000 and click on Update button.

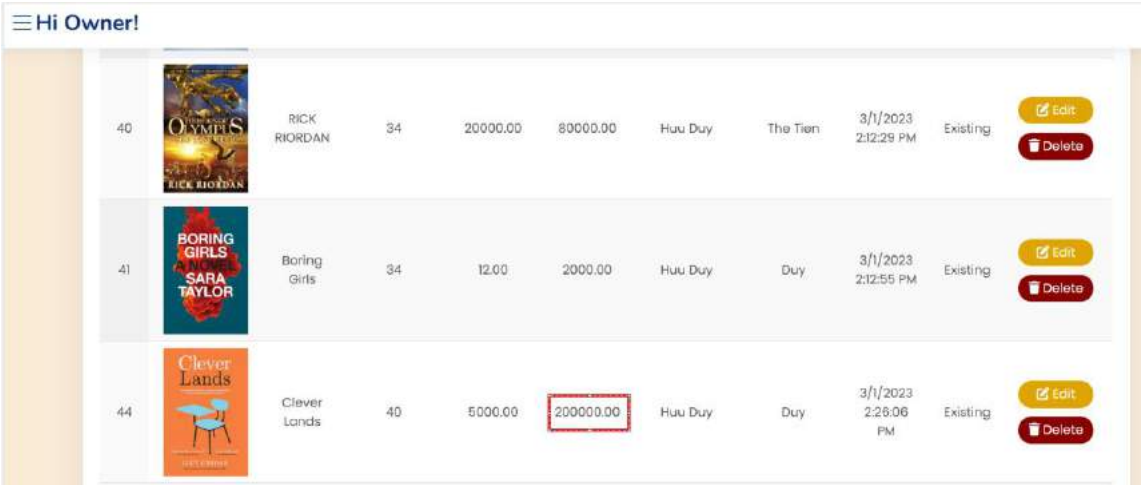


Figure 34. Manage Book interface after updating the book

Now, the book's information is updated successfully.

Last, I will show you about the delete function. In order to perform this function, the owner just needs to click on the Delete button in each book to delete.

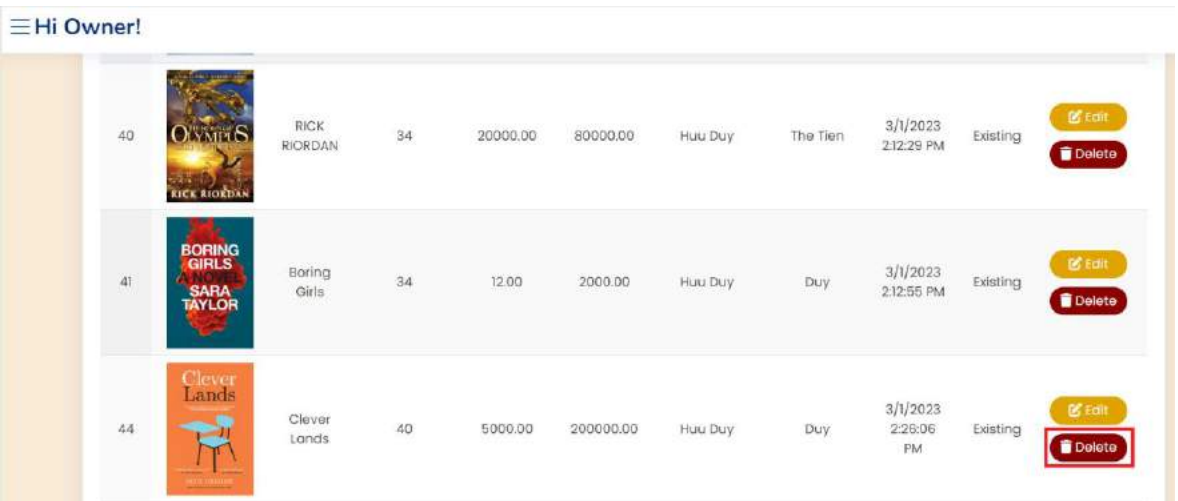


Figure 35. Manage Book interface that present for the delete book function

When deleting successfully, the book that is deleted will not show on manager book page

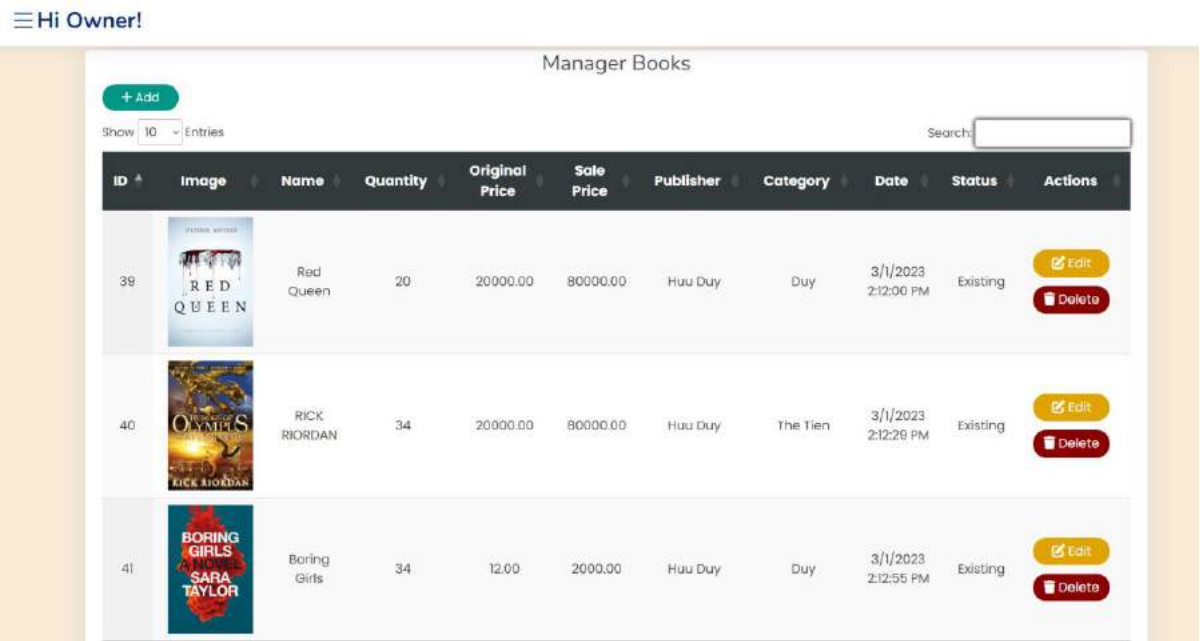


Figure 36. Manage Book interface after deleting a book successfully

Now, the book is deleted

3.1.8. Manage Category (Owner) function

In this function to perform management the category, the Owner login into their account. After they need to click on the Manager Category to go to the manager category page.

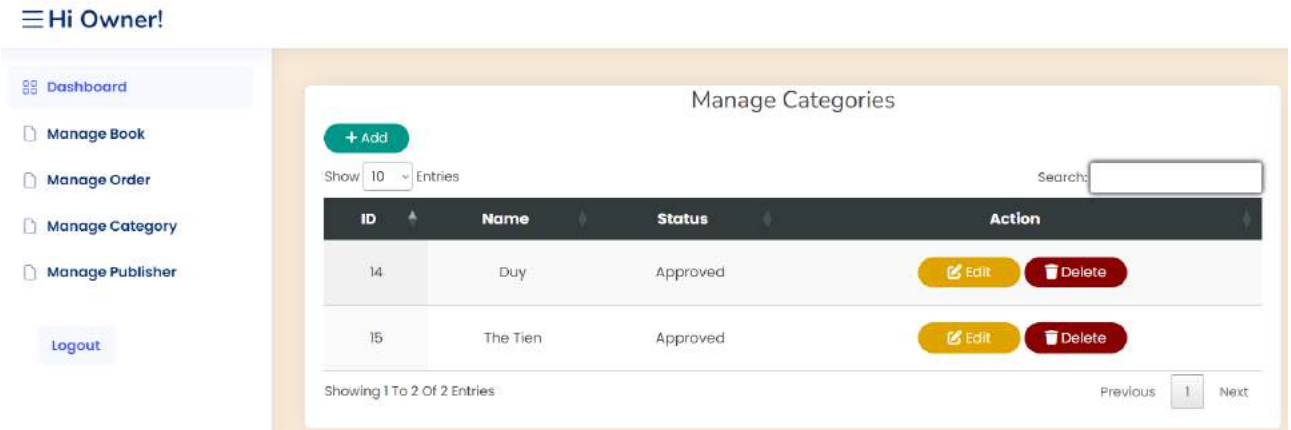


Figure 37. Manage Category interface

In this function, the owner can add, update, and delete the categories. First, when the owner wants to add a new category, they can click on the Add button to perform adding. After click on the Add button, the form add will be displayed

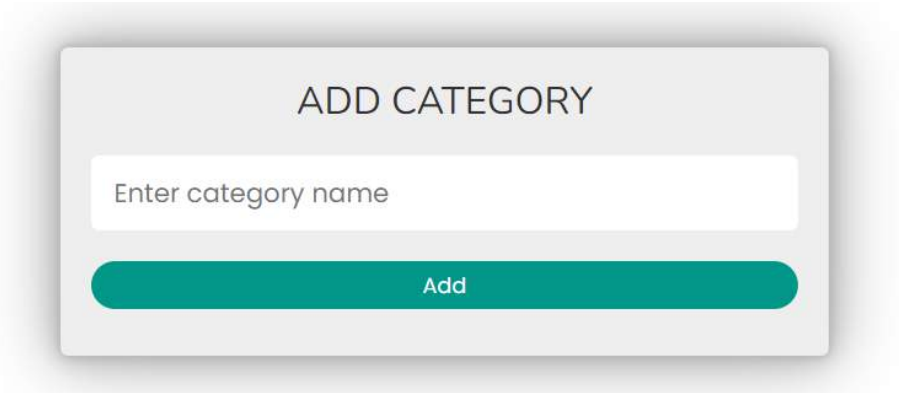


Figure 38. Form adding a new category

Now, the owner will enter the category name to add.

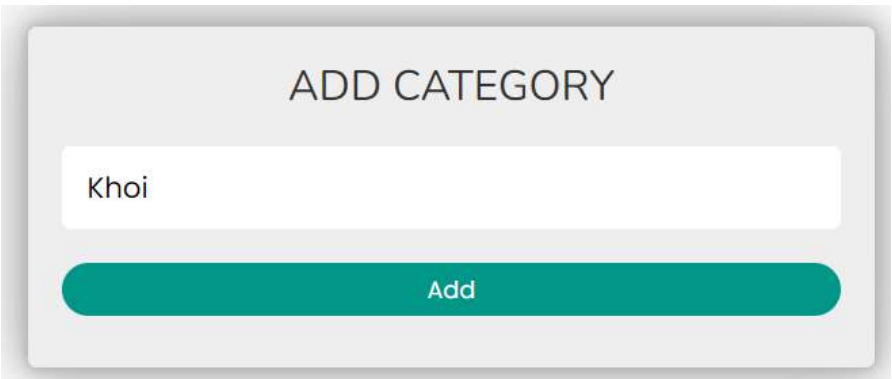


Figure 39. Form adding category that present for the adding a new category function

After entering the category name, the owner needs to click on the Add button to add the new category.

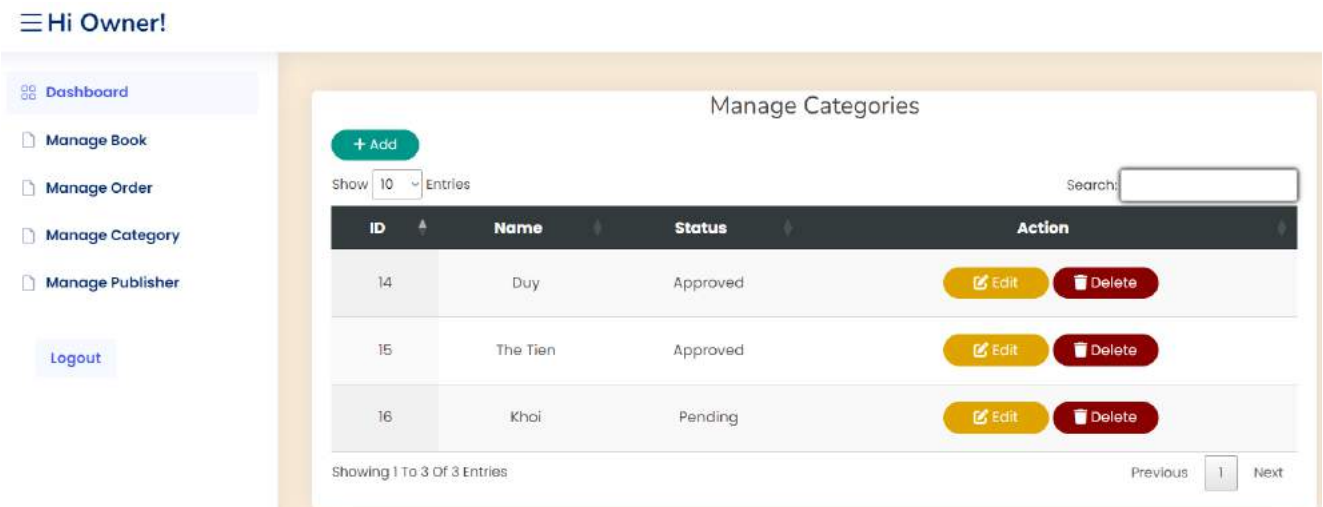


Figure 40. Manage Category interface after adding a new category

Now, the category with id is 16 is added, but the status of this category is pending because the admin haven't confirmed this category yet. When the owner adds a new category, that category will be added to the system, but it is not used and the category that added is just used when the admin confirm. Now, I will login into the admin account to perform confirming the category just added.

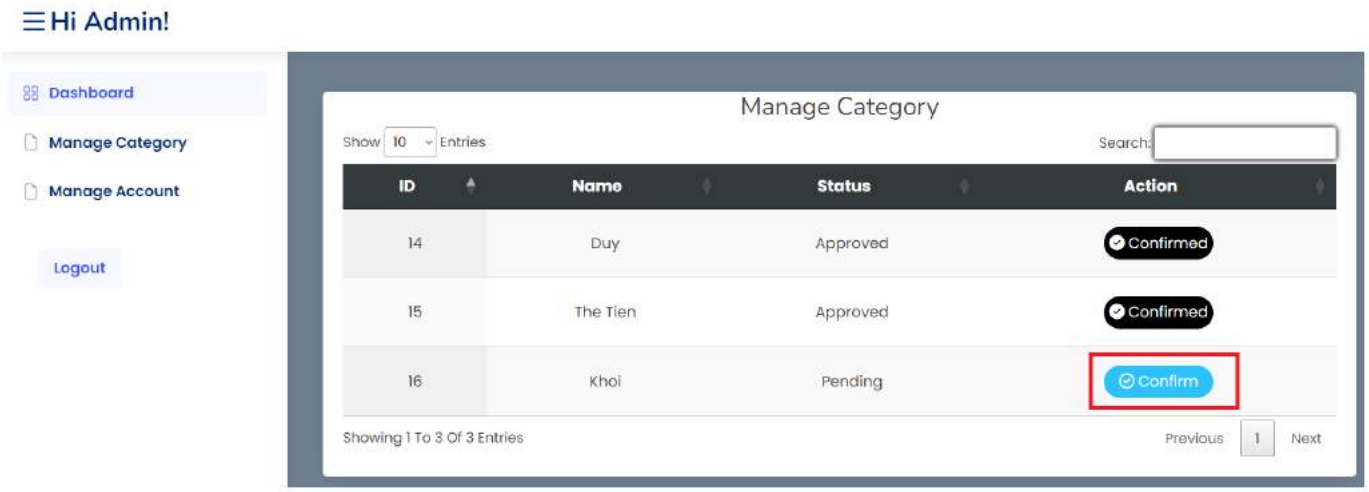


Figure 41. Manage Category interface of the Admin role

When I confirm the new category, the category will be used.

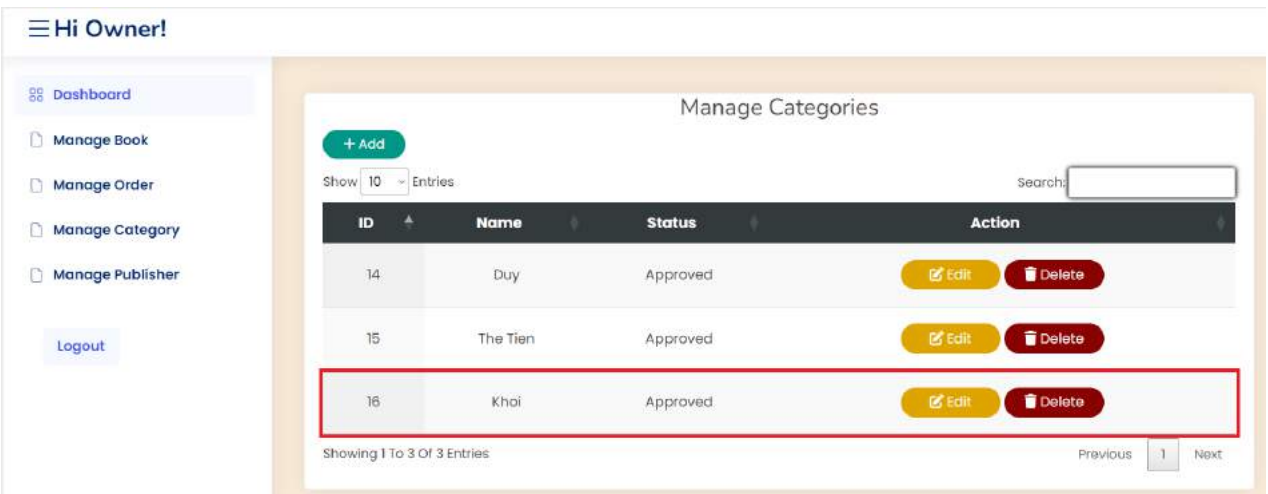


Figure 42. Manage Category interface after the category is confirmed

Now, the new category is approved.

Next, when the owner wants to update the category, the owner can click on the Edit button in each category.

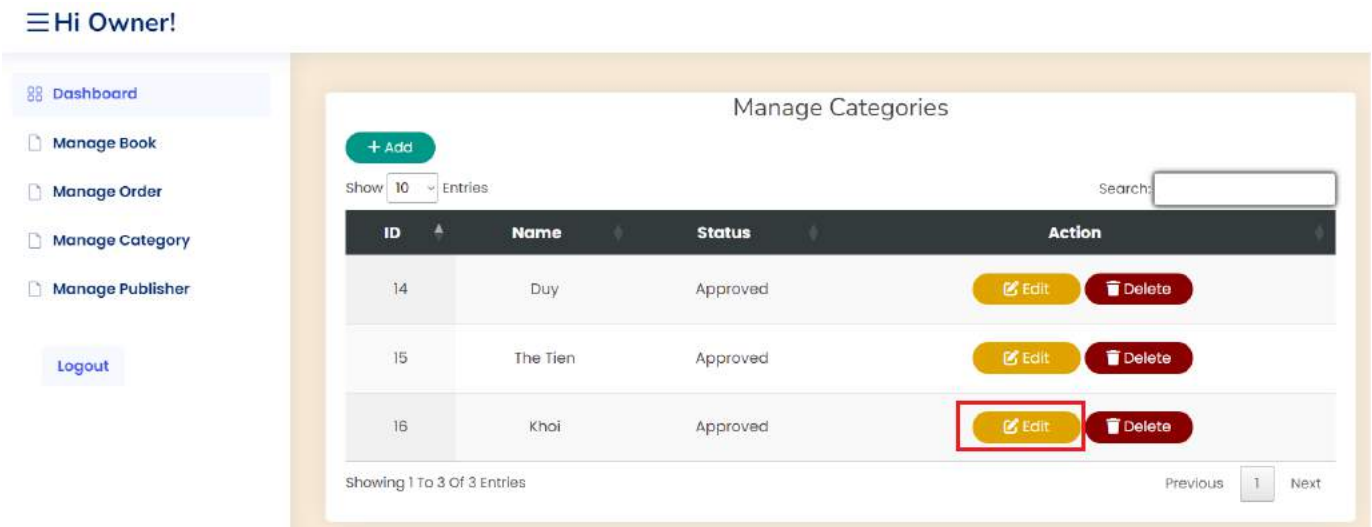


Figure 43. Manage Category interface that present for the updating category function

Now, I will perform updating the category with id is 16. I will update the name of this category from "Khoi" to "Dang Khoi".

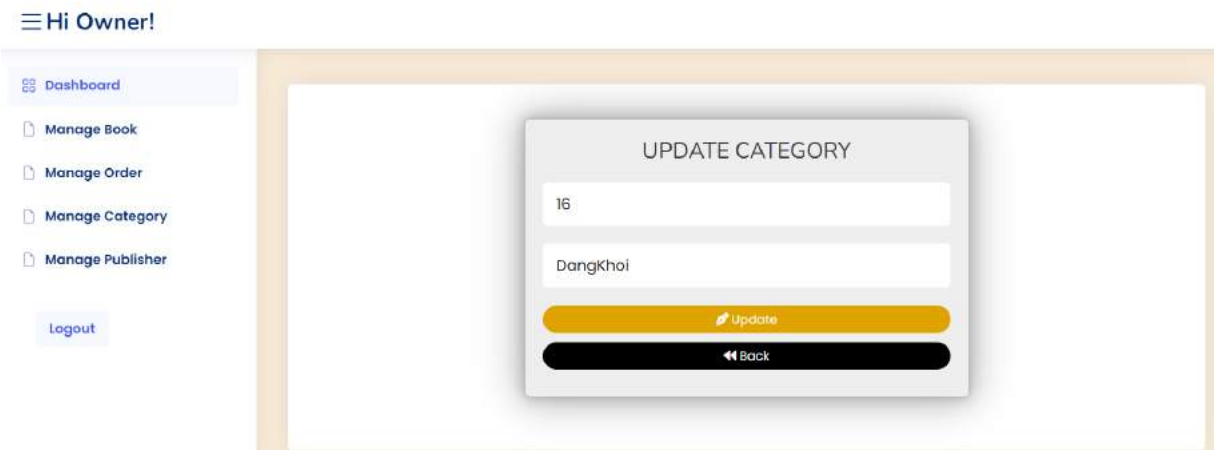


Figure 44. Form updating the category

After entering the new name, I will click on the Update to update the category.

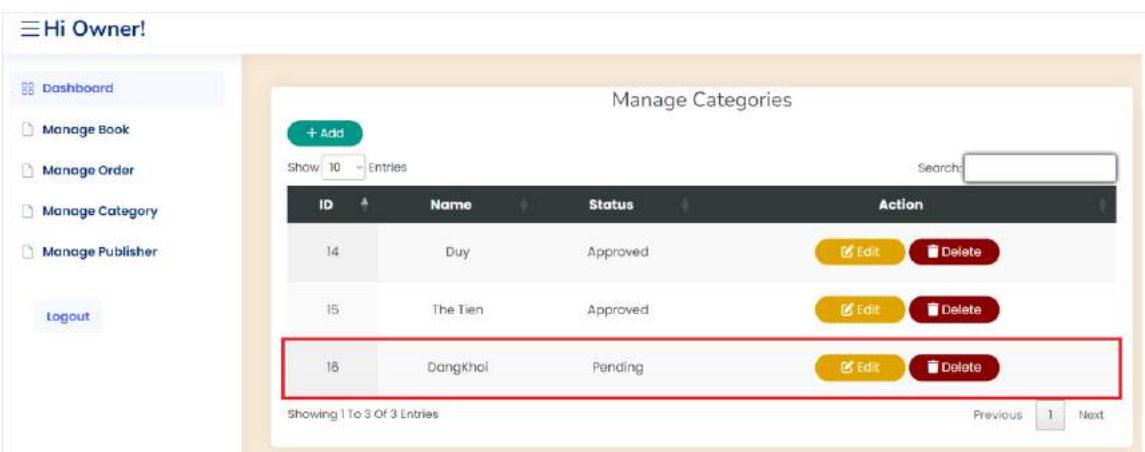


Figure 45. Manage Category interface after updating the category successfully

Now, the category with id 16 is updated, but the status of this category is pending because this category hasn't been confirmed yet by the admin. It is same with the adding function, so the admin needs to confirm this category, then this category can be used. Now, I will login into the admin account to confirm this category.

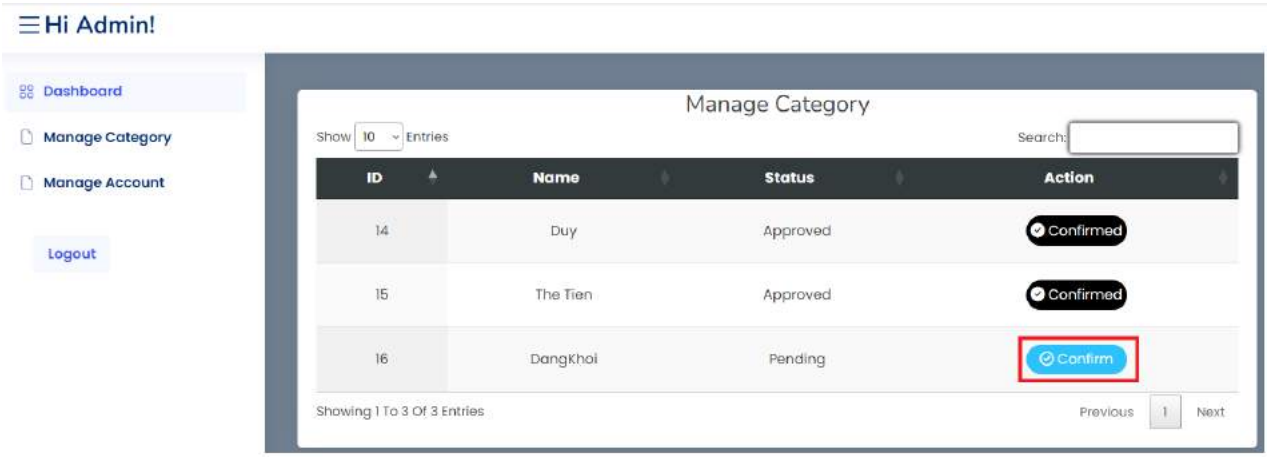


Figure 46. Manage Category of the admin role that present for the confirm category function

When I click on the confirm button, the category will be confirmed and this category's status will be changed.

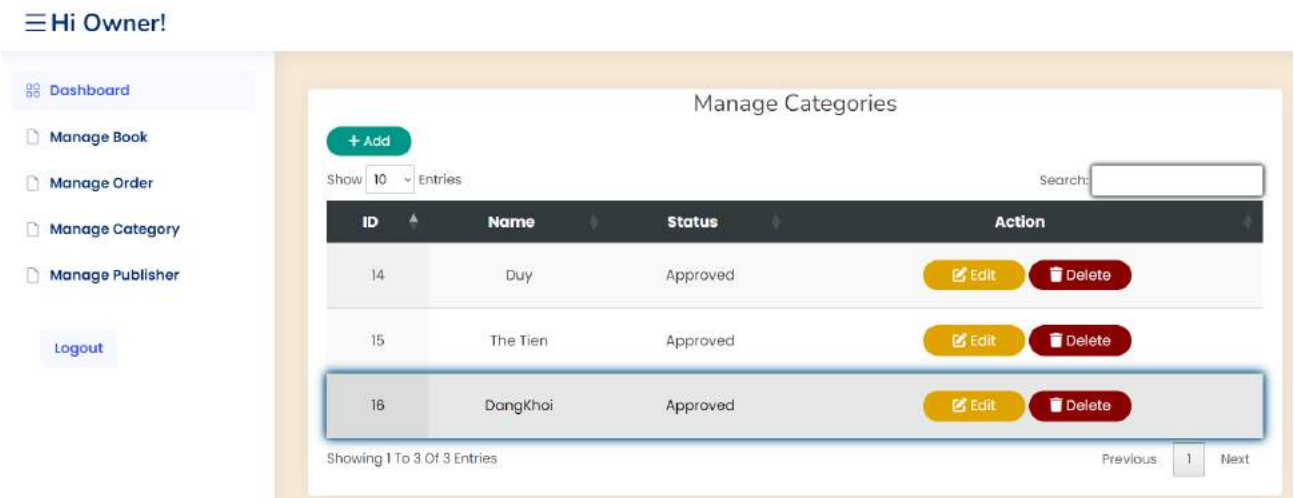


Figure 47. Manage Category interface after confirmed

Now, the category with id 16 can be used.

3.1.9. Manage Category (Admin) function

With admin, they can manage categories and perform the confirm function for each category. The admin can view all of the categories that the owner added, and the admin can confirm or not confirm the categories that owner added. If the admin confirms, the category's status will be changed to "Approved" and those categories can be used, but the admin does not confirm the category's status is "Pending" and those categories will not be used.

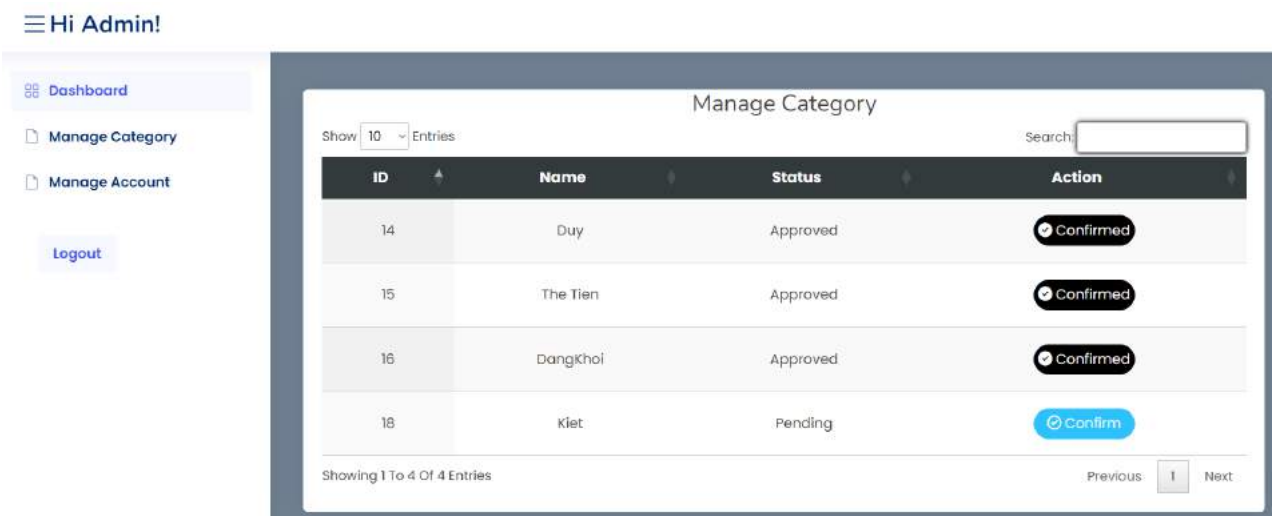


Figure 48. Manage Category interface of the Admin role

3.1.10. Manage Publisher function

With the manager publisher function, the owner can add, update, and delete the publishers. In order to perform this function, the user needs to login into the owner account and when user login into the owner account, the web will go to the owner dashboard, then user needs to click on the Manager Publisher to go to the manager publisher page.

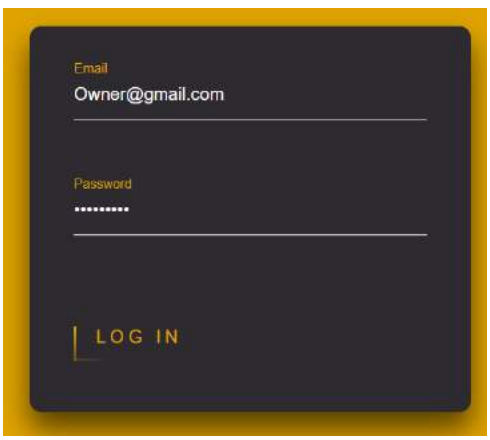


Figure 49. Login interface

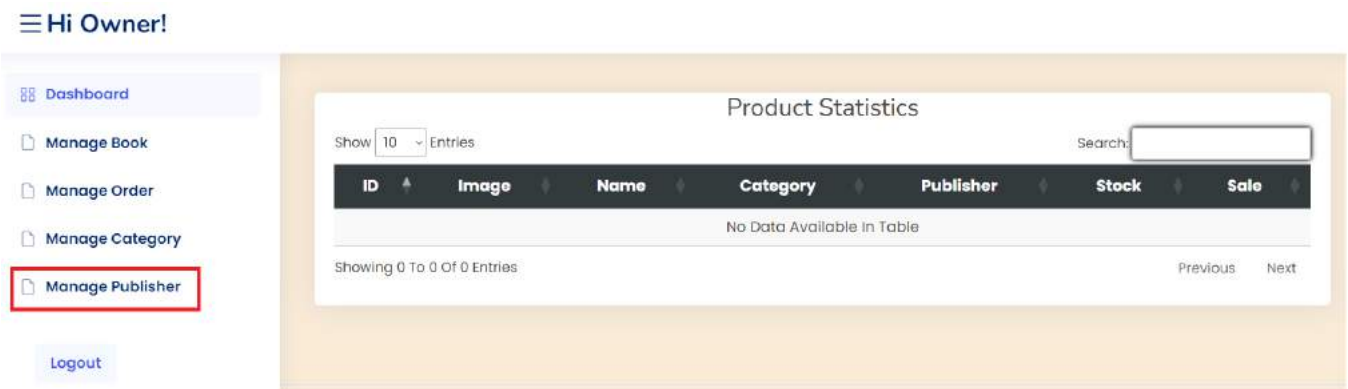


Figure 50. Dashboard of owner role that present for manage publisher function

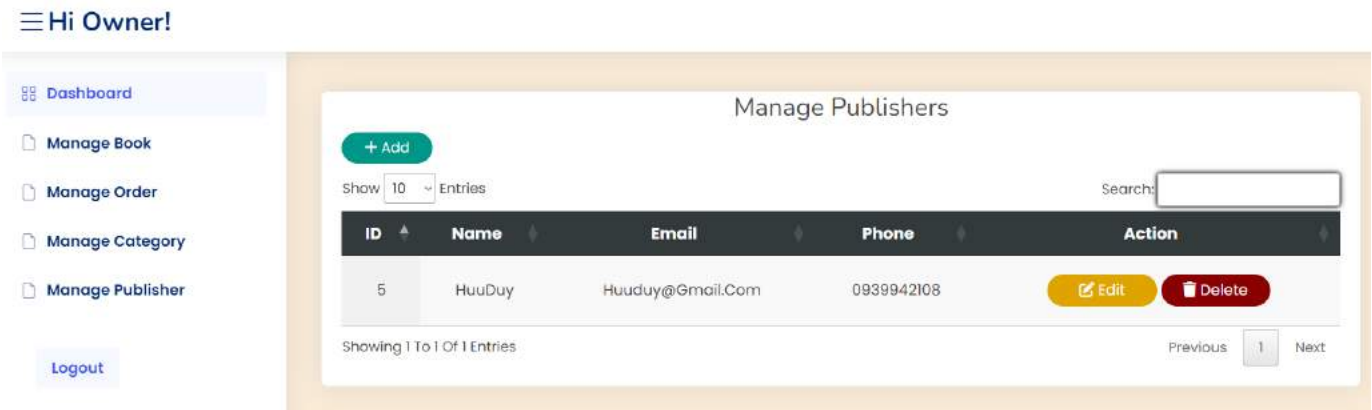


Figure 51. Manage Publisher interface

Now, I will perform adding a new publisher. I will click on the Add button to perform adding a new publisher function and when I click on the Add button, the form add will be displayed. Then, I will fill in all information fields to add and click on Add button to add a new publisher.

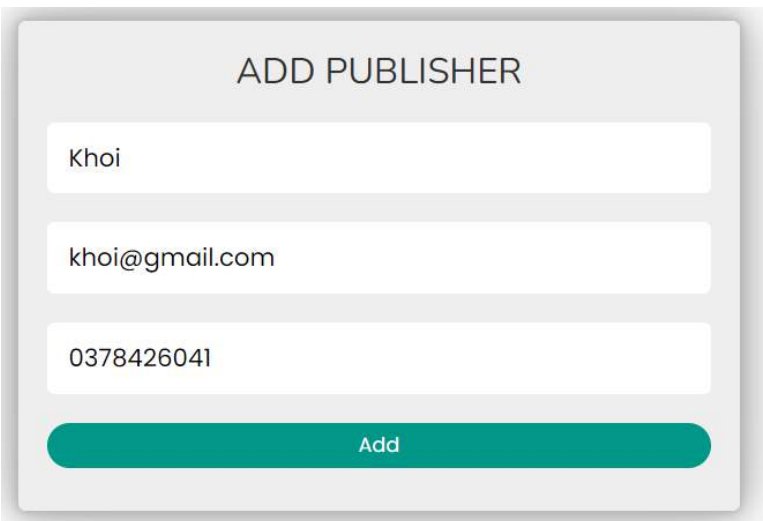


Figure 52. Form adding a new publisher

Now a new publisher has been added to the system.

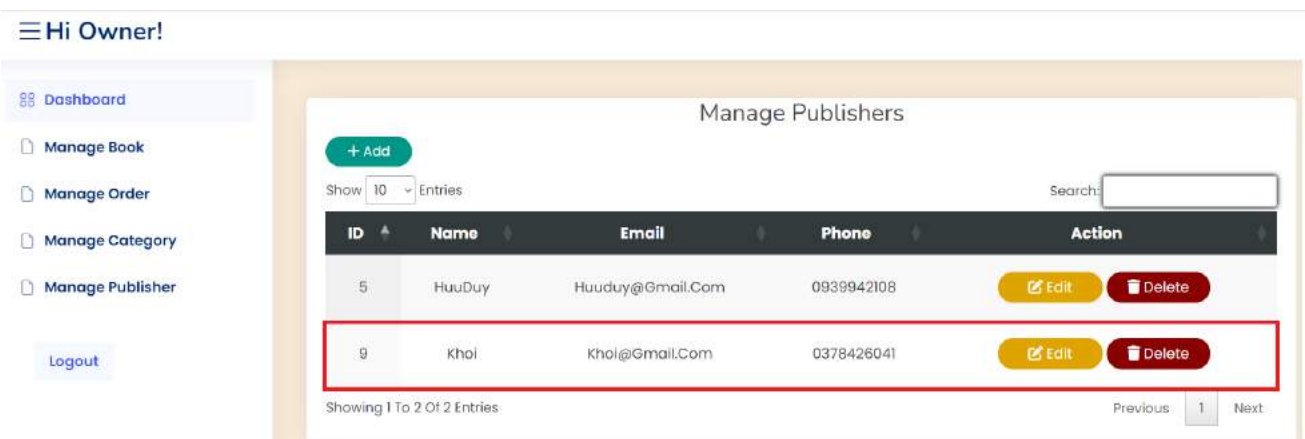


Figure 53. Manage Publisher interface after adding a new publisher successfully

Next. I will perform the update publisher function, in order to this function I will click on the Edit button in each publisher and the form add publisher will be displayed

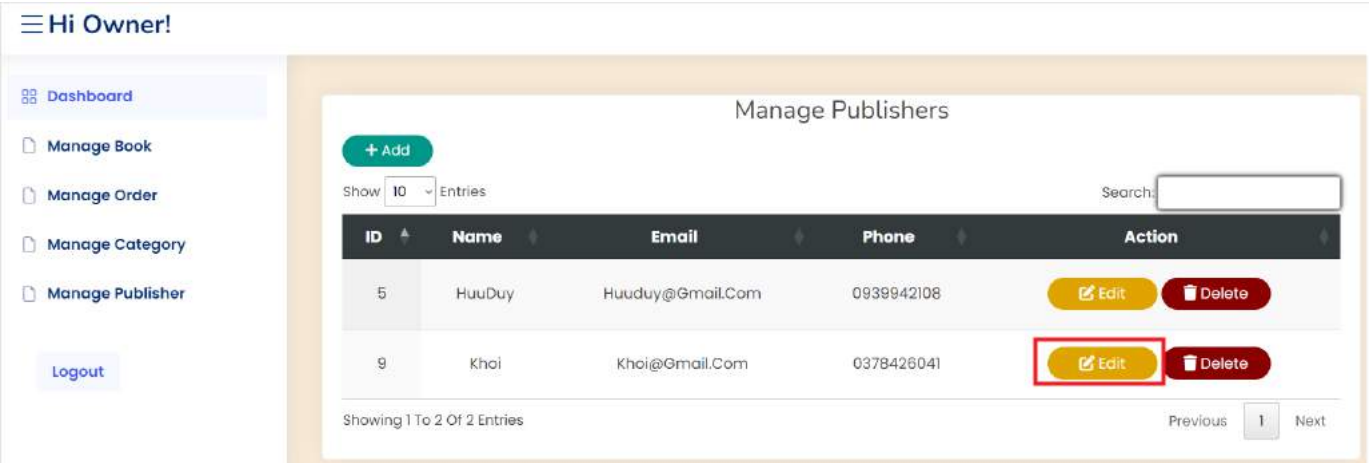


Figure 54. Manage Publisher interface that present for the updating publisher function

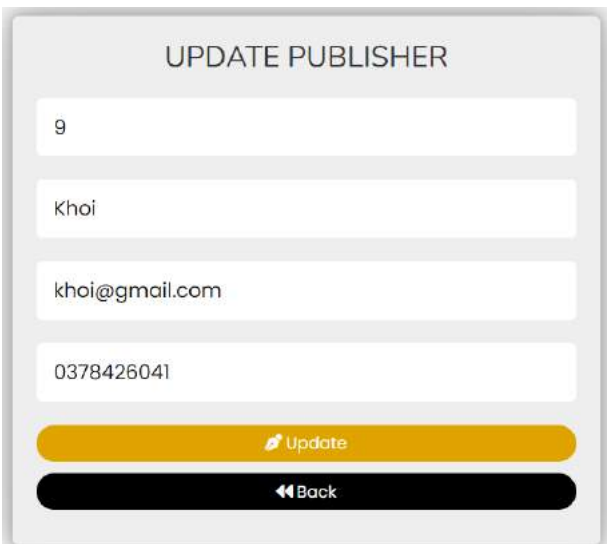


Figure 55. Form updating publisher

Now, I will update the publisher named Khoi to DangKhoi and click on the Update button.

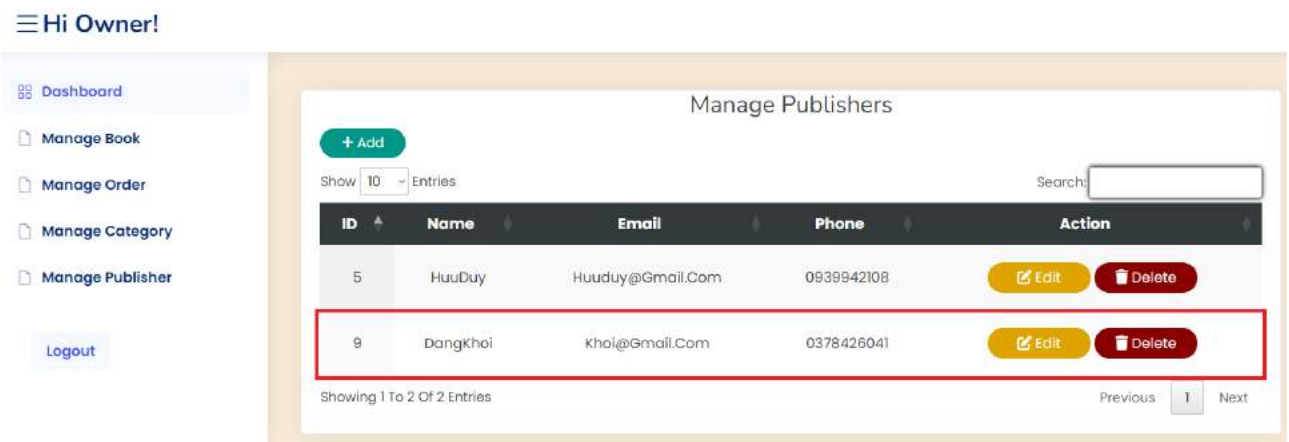


Figure 56. Manage Publisher interface after updating a publisher successfully

Now, I will perform a delete publisher function. In order to this function, I will click on "Delete" button in each Publisher.

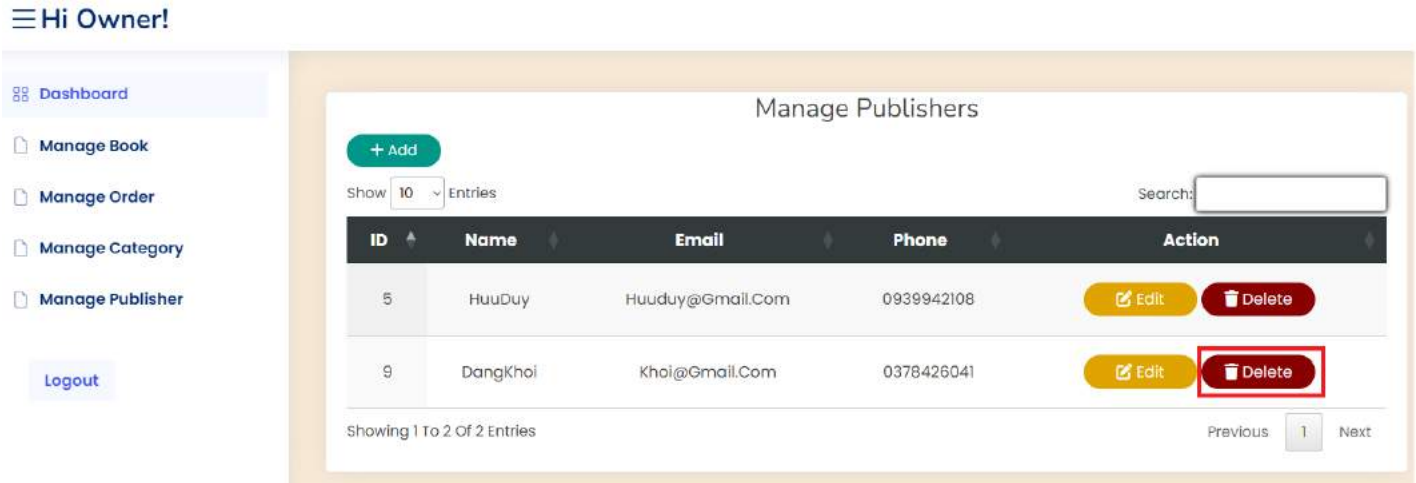


Figure 57. Manage Publisher interface that present for the delete a publisher function

Now the publisher is deleted.

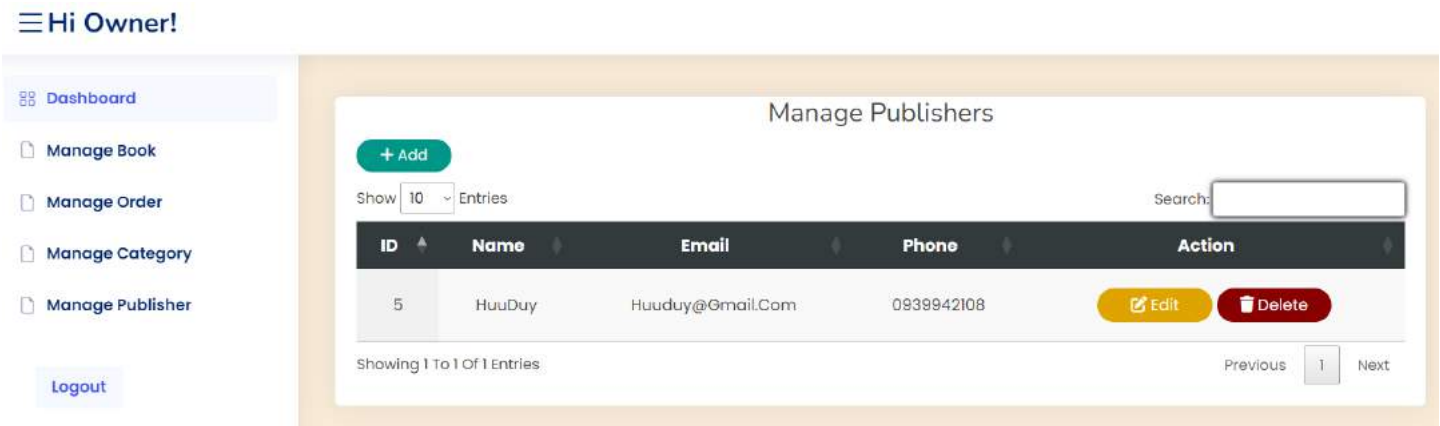


Figure 58. Manage Publisher interface after deleting the publisher

3.1.11. Manage Order function

With this function, the owner can view all of the customer's orders and the owner can confirm those orders.

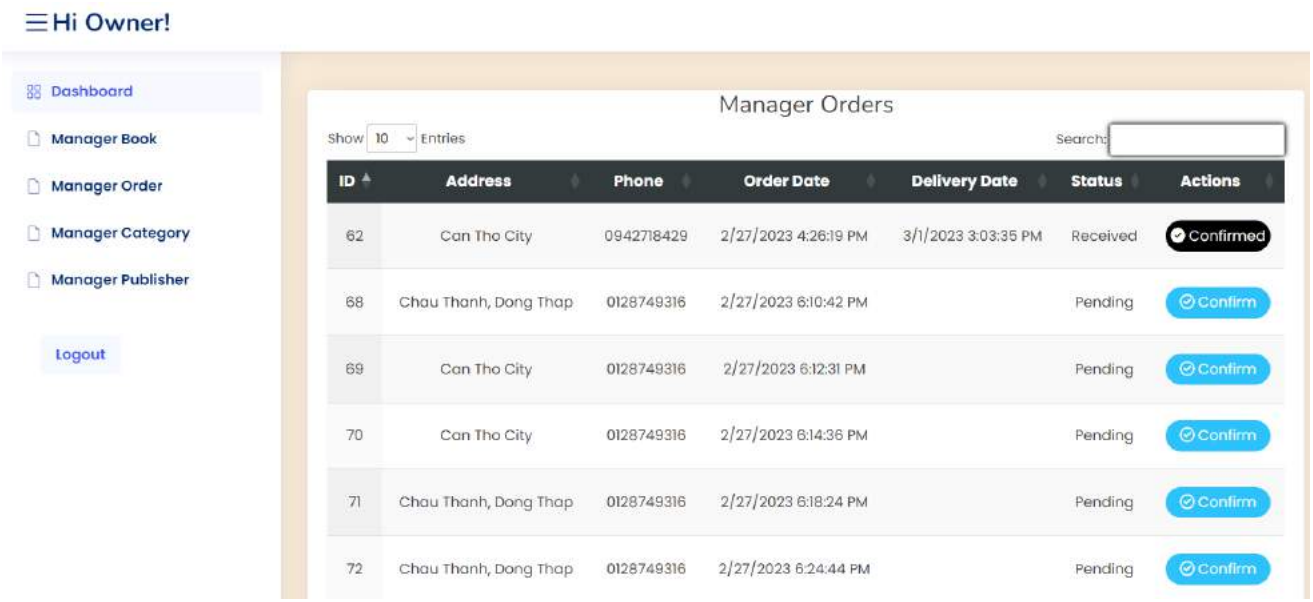


Figure 59. Manage Order interface

If the owner confirms any customer's orders, the order's delivery date and status will be changed. This means that the order will be shipped. Now, I will perform confirming the order with id is 68.

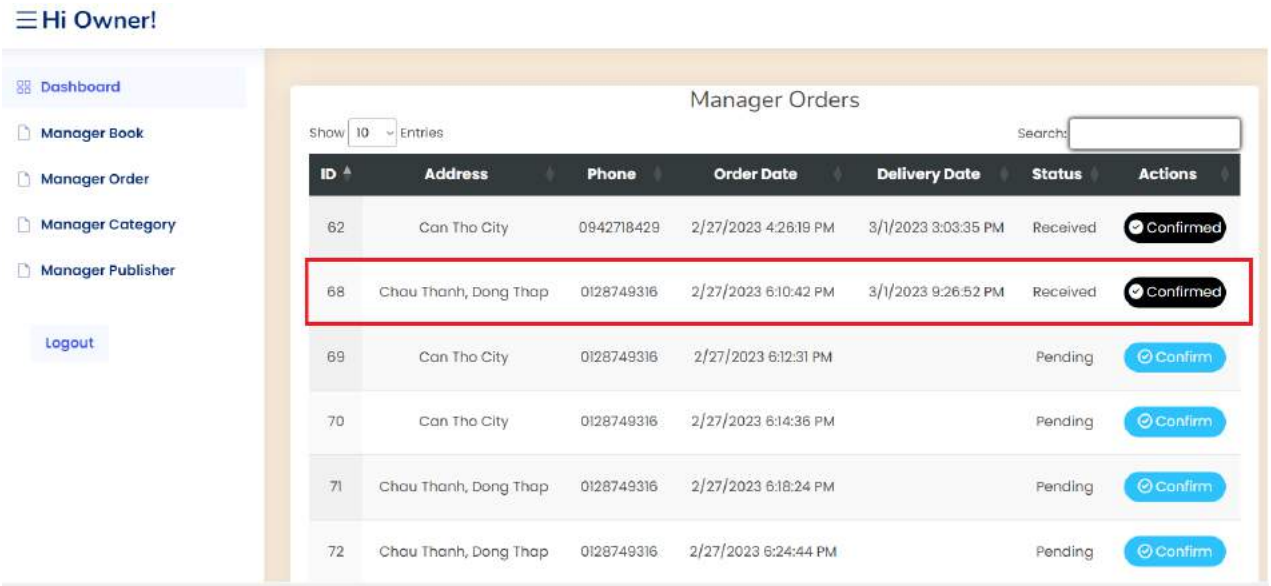


Figure 60. Manage Order interface after confirming the order

Now, the order with id 68 is shipped and the customer received their order.

3.1.12. Manage Account function

Manage Account, this is a function of Admin. In order to perform this function, the admin can login into their account, then the dashboard page of the admin will display.

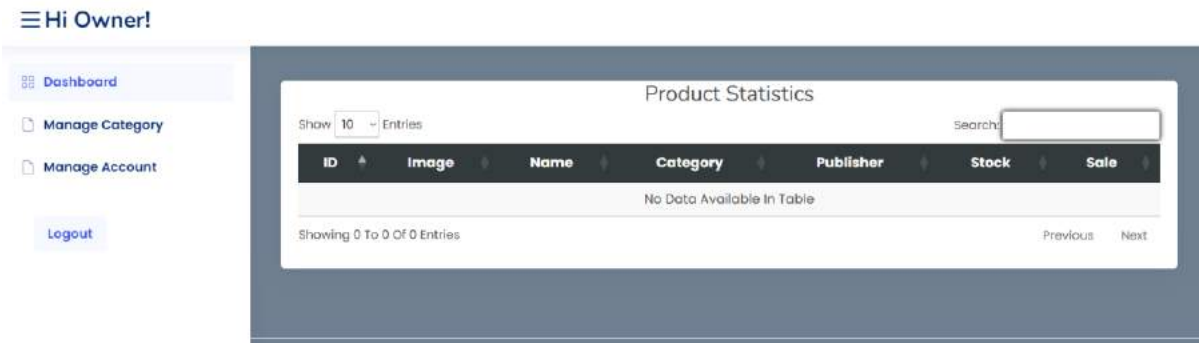


Figure 61. Dashboard interface of the admin role

Now, the admin need to click on the Manager Account to perform manage owner's account. With the role is Admin, they just can manage account of the owner and manage categories. In the manage account function, the admin can lock and activate any owner's account.

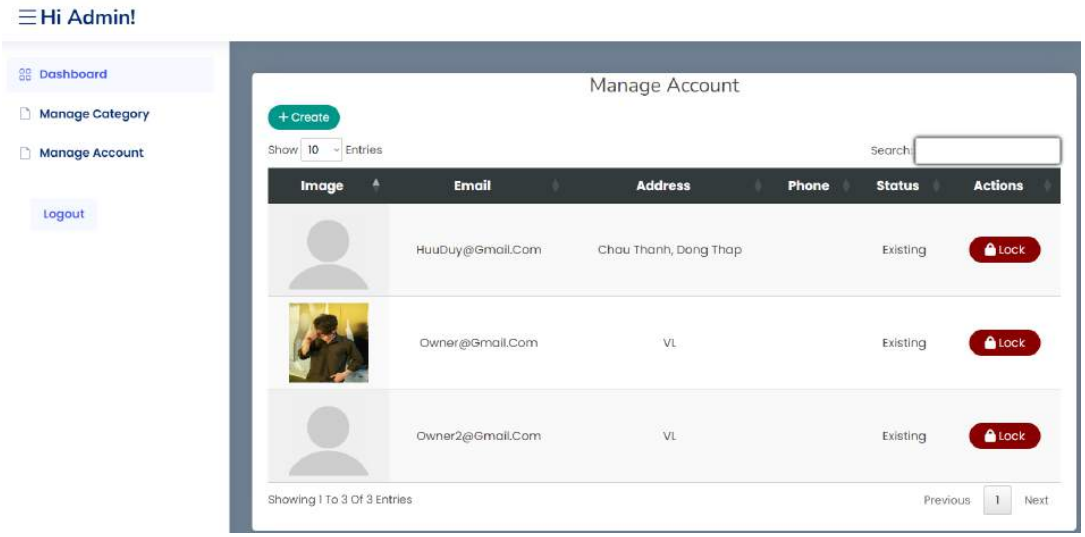


Figure 62. Manage Account interface

The admin can click on the Lock button to lock the account that they want.

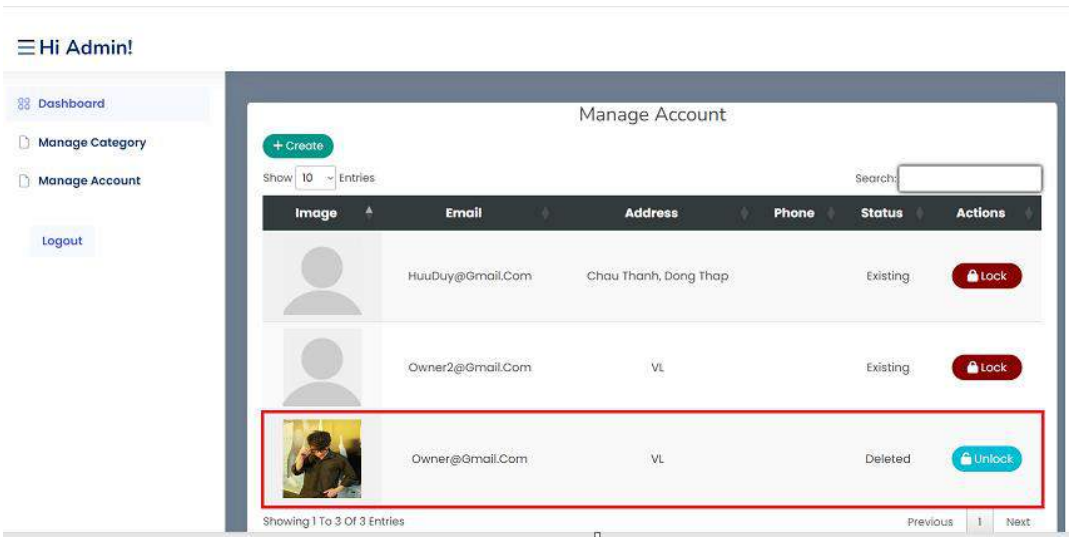


Figure 63. Manage Account interface that present for the lock account function

Now, the account Owner@gmail.com has been locked and if the admin wants active this account, they just need to click on the Unlock button to unlock the account.

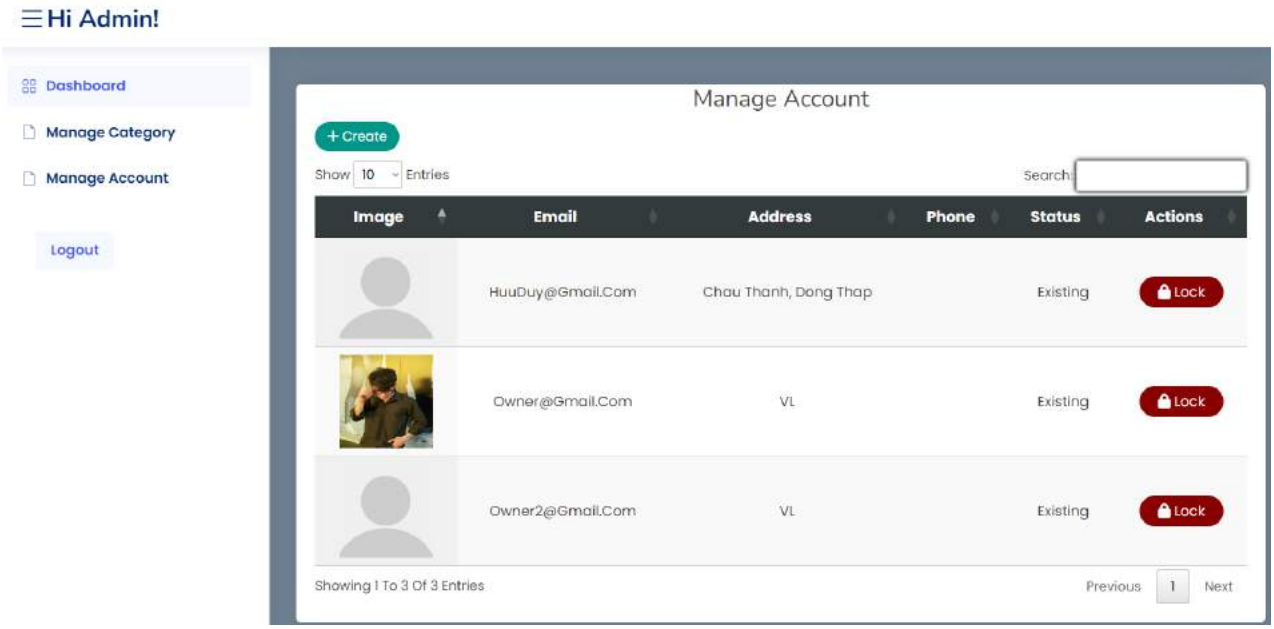


Figure 64. Manage Account interface after unlocking the account

Now, the account Owner@gmail.com has been unlocked.

In addition, the admin can create a new account for the owner, admin, or user. In order to this function, the admin will click on the Create button to perform this function.

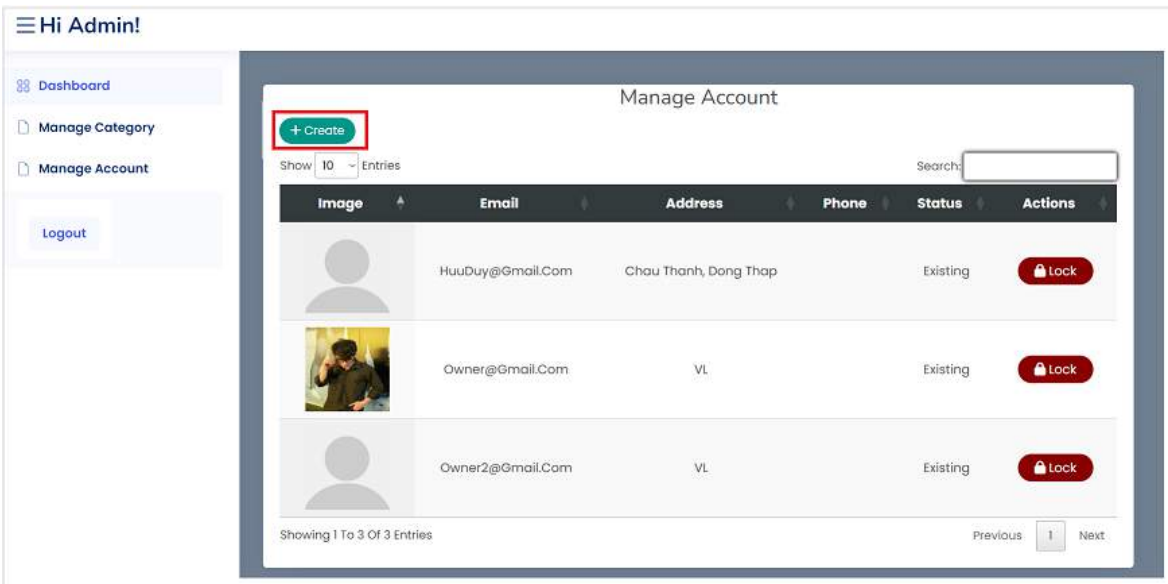


Figure 65. Manage Account interface that present for the create a new account

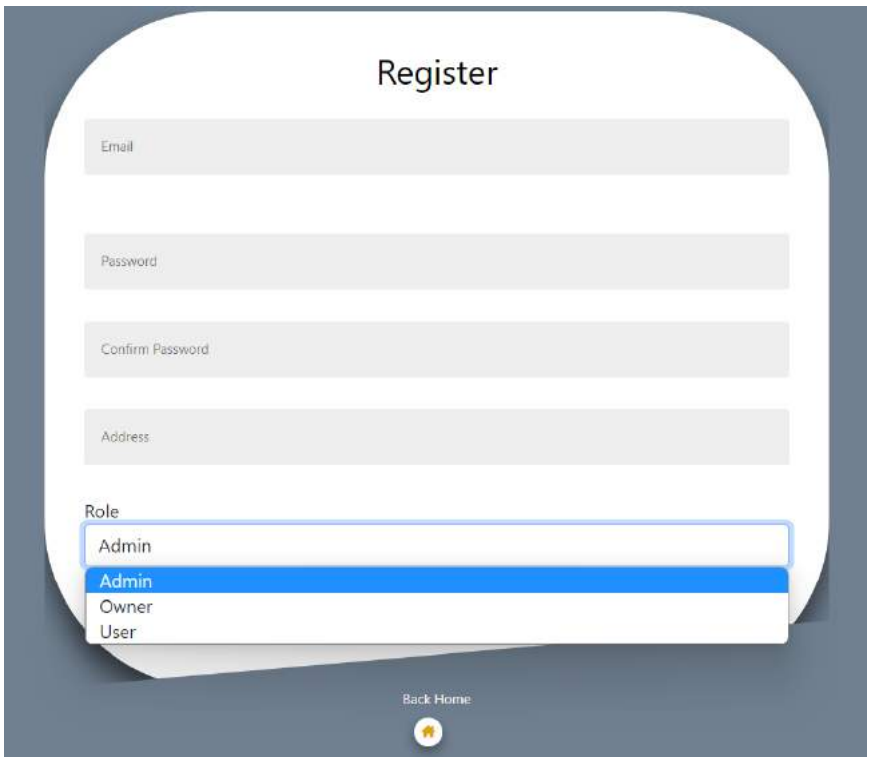


Figure 66. Form creating a new account

Now, the admin will enter information of the account and choose role for the account and click on the Sign up button to register. I will perform add a new owner's account.

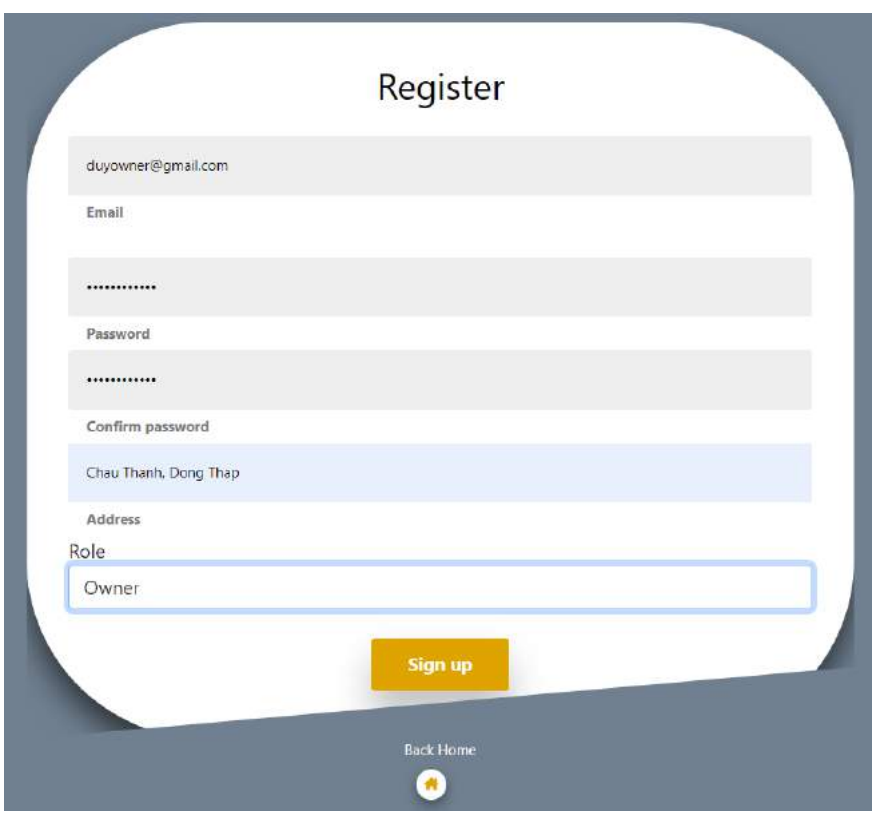


Figure 67. Form creating a new account

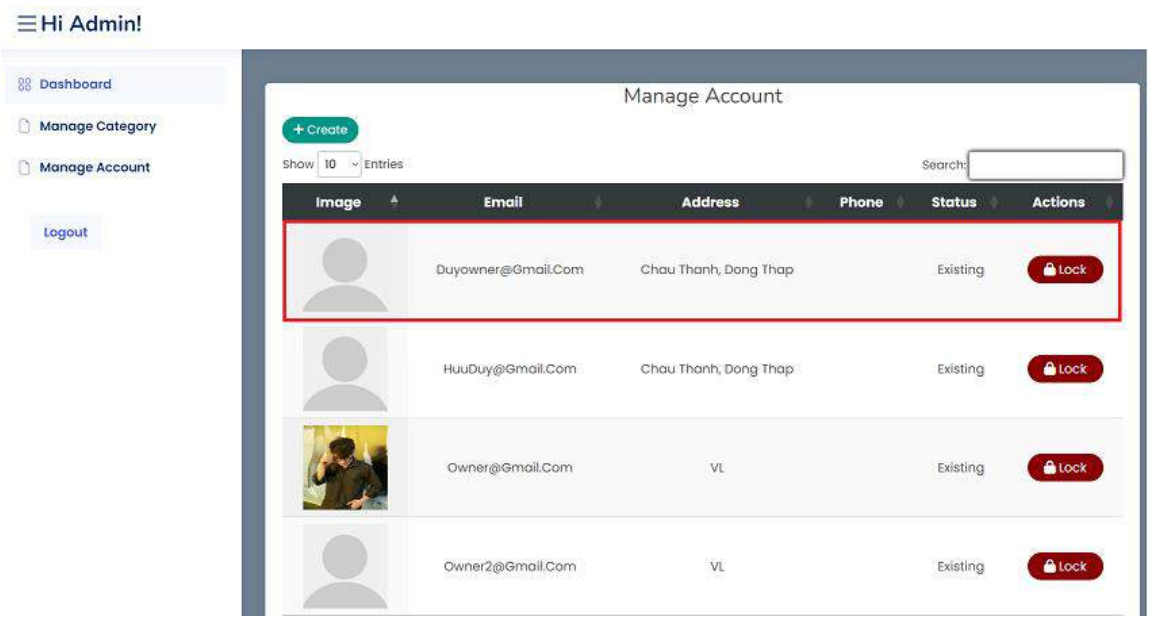


Figure 68. Manage Account interface after creating a new account successfully

Now, the owner's account "Duyowner2gmail.com" is created.

3.1.13. Manage Profile function

In the manage profile function, the user can update their information. They can change password, address, phone, and upload account's avatar. When update successfully the message "Your profile has been updated" to notify user.

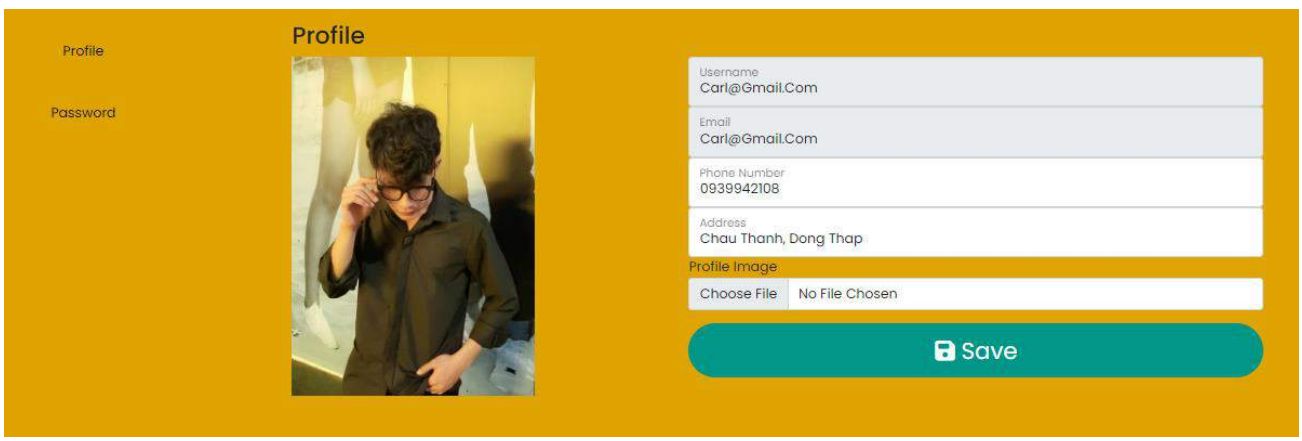


Figure 69. Manage Profile interface

When a user wants to change password they can click on the Password to go to the change password page.

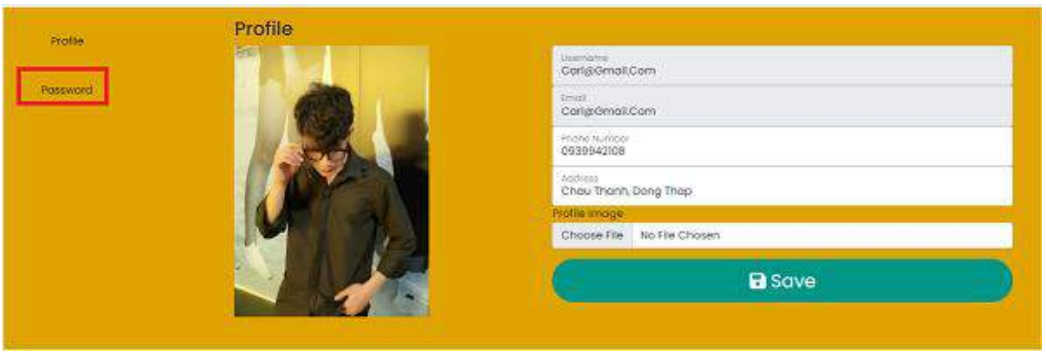


Figure 70. Manage Profile interface that present for the change password function

When go to the change password page, the user needs to fill the current password to perform the change password function, then user will enter the new password and confirm the new password and click on the Update password button to change password.

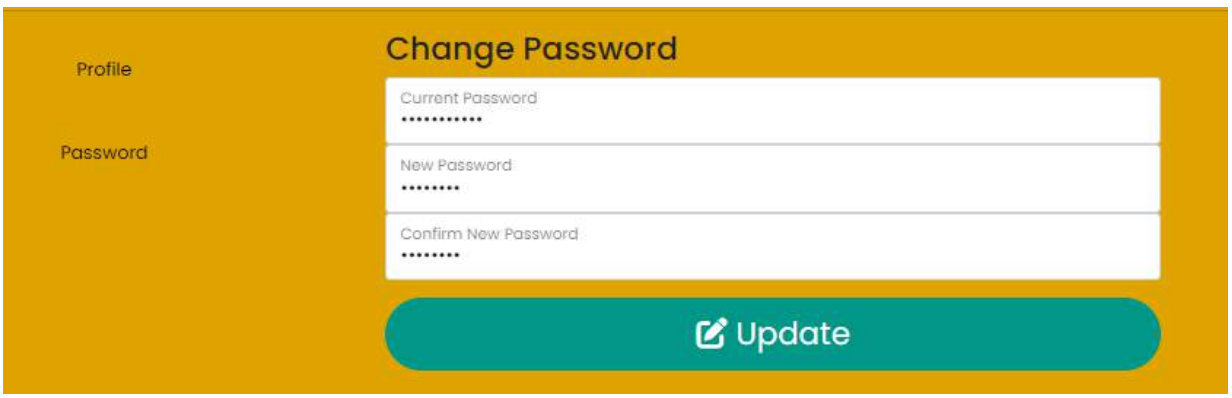


Figure 71. Form changing the password

When changing password successfully the web also will display a message to notify user.

Figure 72. Message notifies that change password successfully

3.1.14. Cancel Order

In this function, the customer can perform canceling their orders that they ordered. In order to perform this function, they need to login into their account and go to the Order page to view all of the orders that they ordered

Figure 73. Login interface

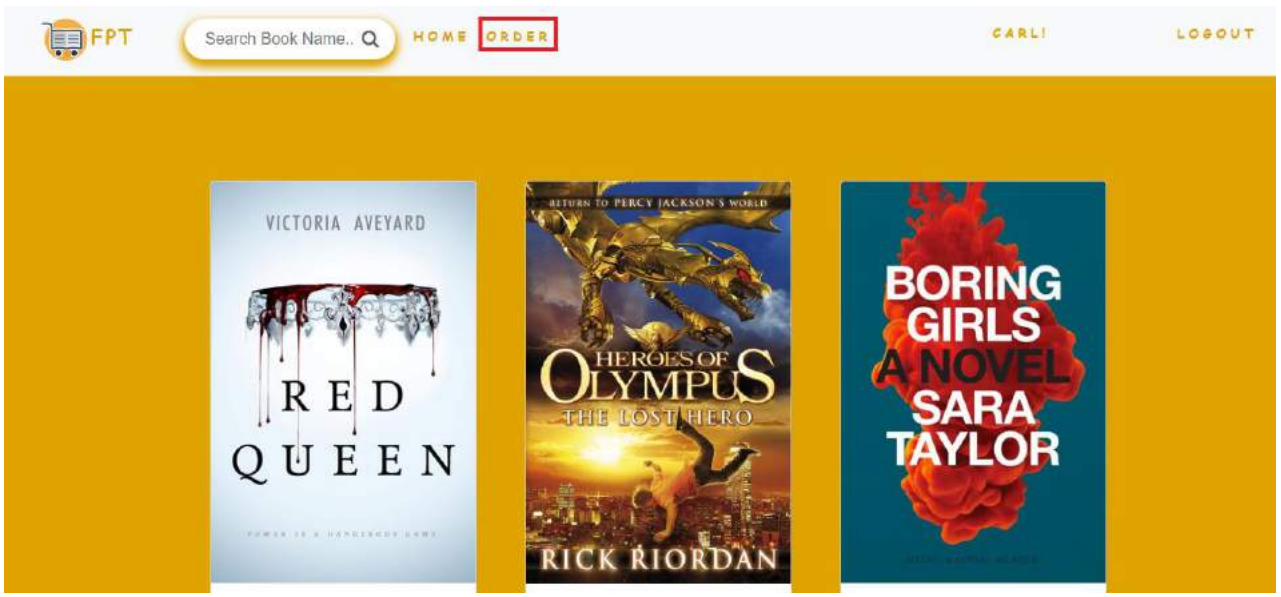


Figure 74. Home interface after login with the user account

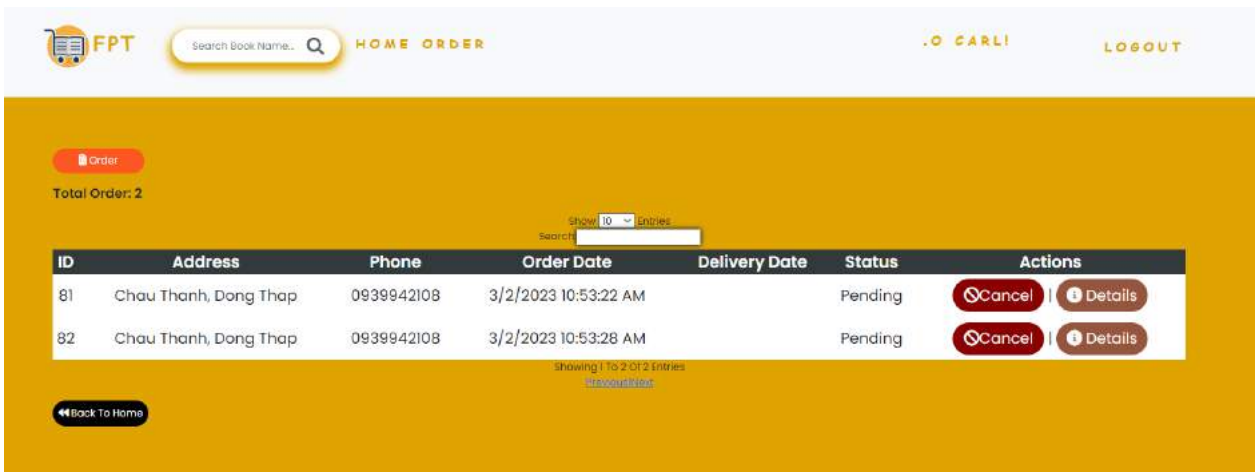


Figure 75. Manage Order interface

Now, they can cancel any order that they want by clicking on the Cancel button to perform canceling the order

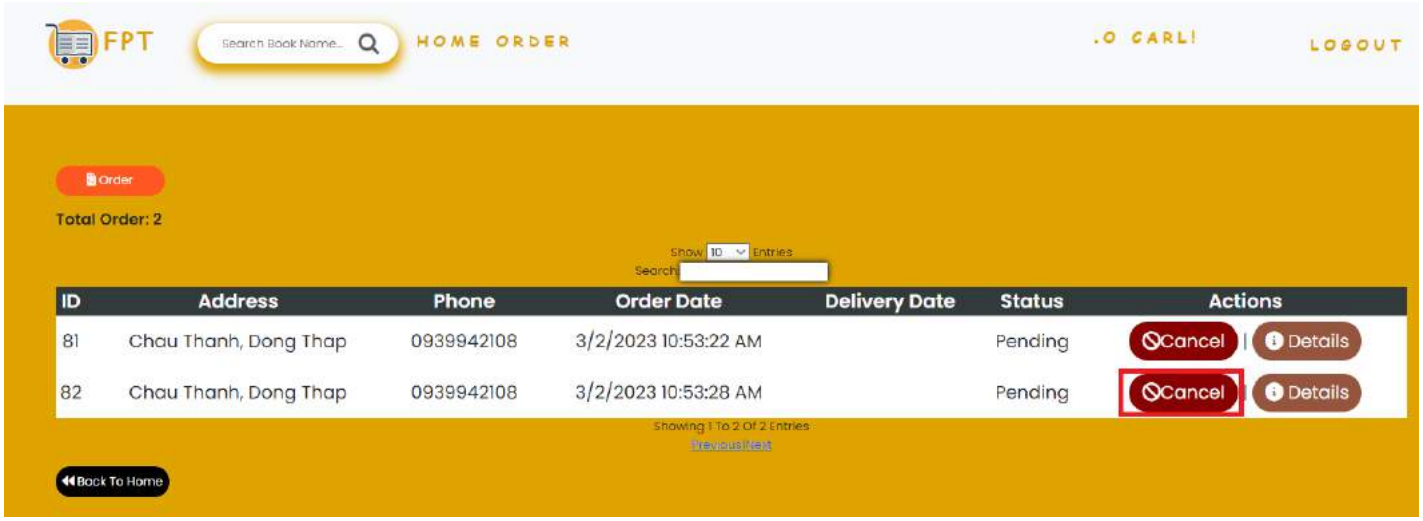


Figure 76. Manage Order that present for the cancel order function

Now, I will cancel the order with id is 82.

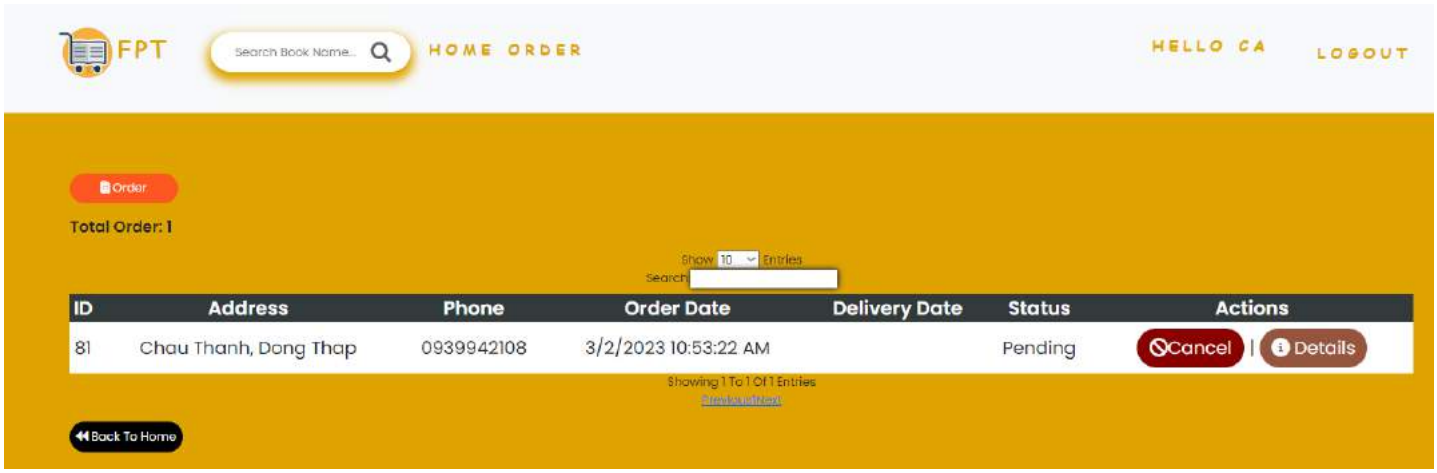


Figure 77. Manage Order interface after canceling order successfully

Now, the order with id 82 is canceled.

3.2. Folder Structure

3.2.1. The overview of folder structure

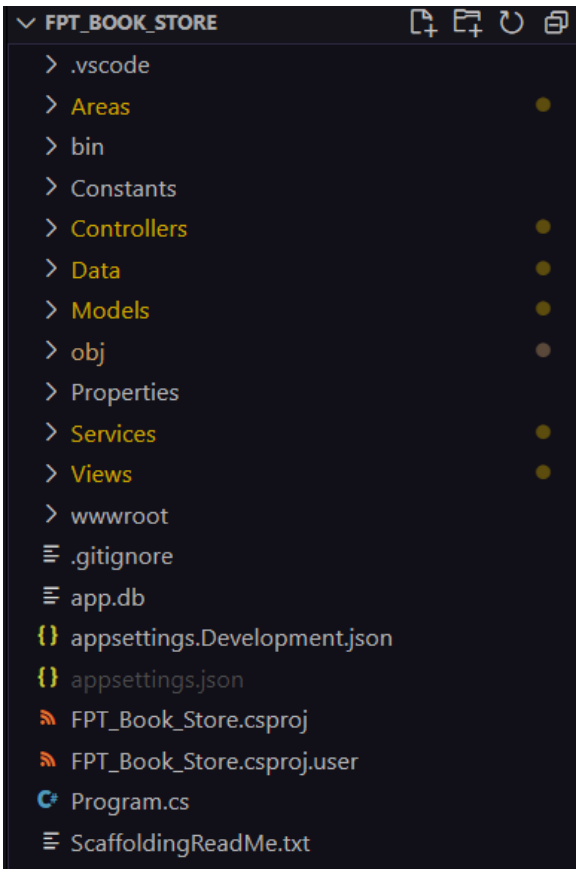


Figure 78. The overview of folder structure

3.2.2. The folder structure of Areas/Admin

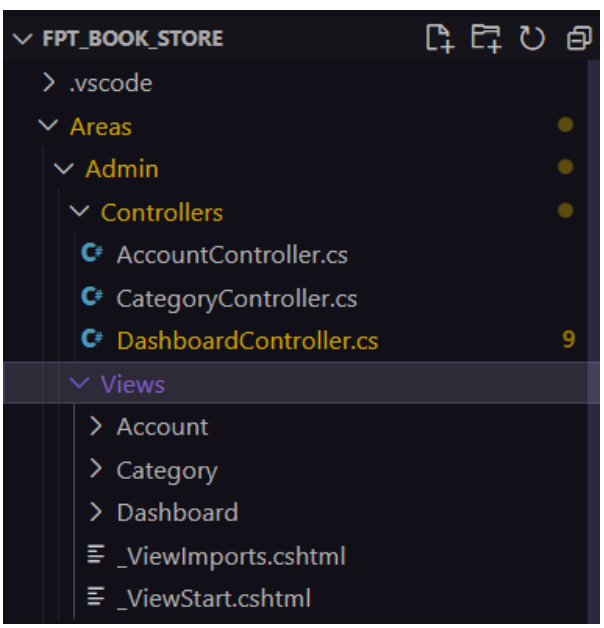


Figure 79. The folder structure of Areas/Admin

3.2.3. The folder structure of Areas/Customer

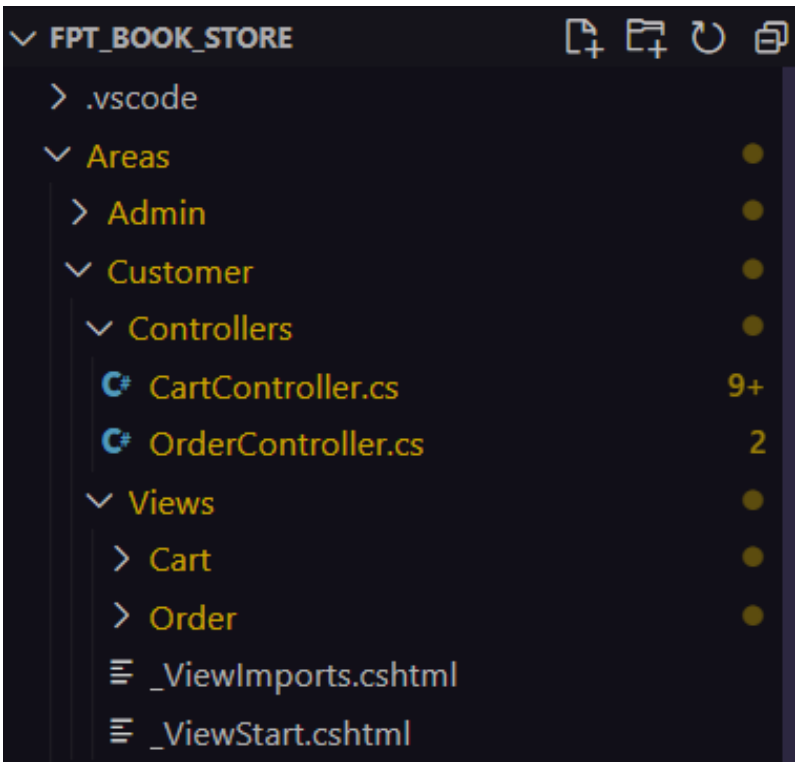


Figure 80. The folder structure of Areas/Customer

3.2.4. The folder structure of Areas/Identity

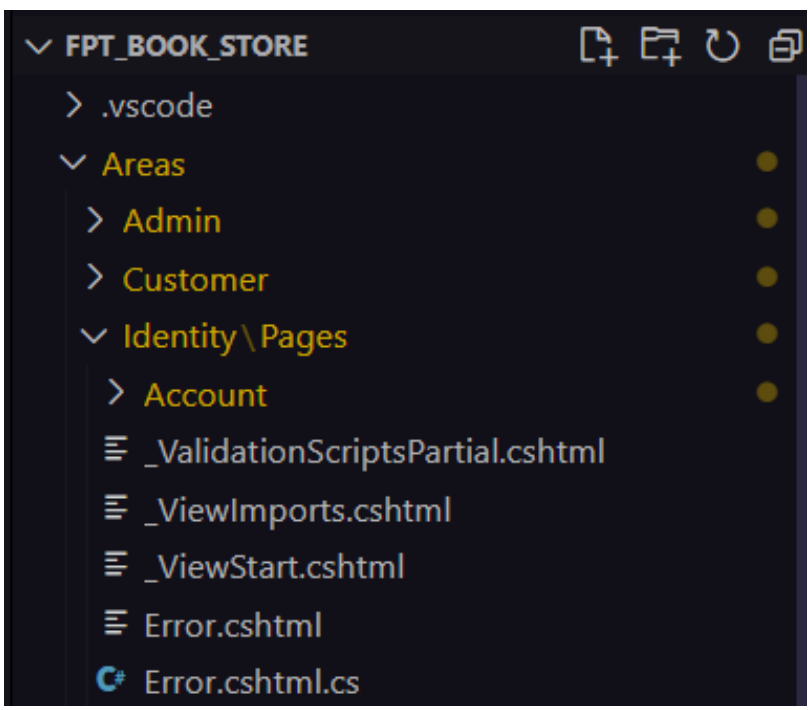


Figure 81. The folder structure of Areas/Identity

3.2.5. The folder structure of Areas/Owner

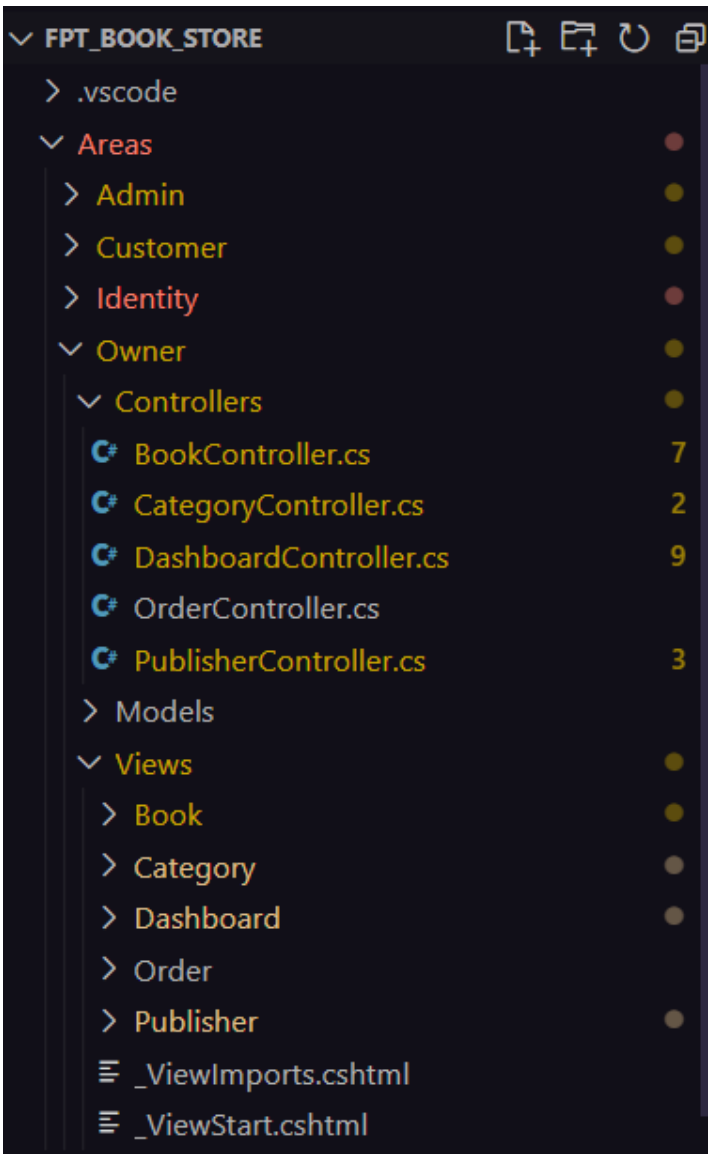


Figure 82. The folder structure of Areas/Owner

3.2.6. Models

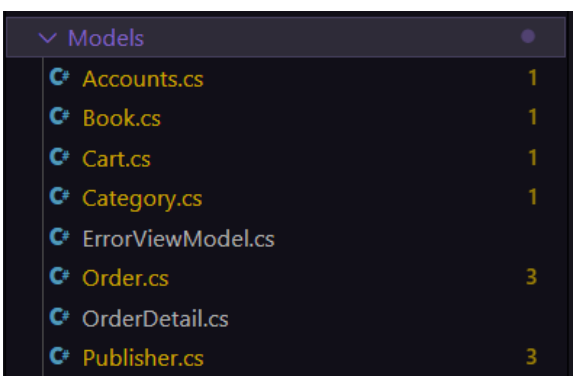


Figure 83. Models

Table 2. Models explain table

No.	File	Source Code

1	Accounts.cs	`namespace FPT_Book_Store.Models`
		`{`
		`public class Accounts : IdentityUser`
		`{`
		`[Required(ErrorMessage = "Please, enter the address!")]`
		`[StringLength(100)]`
		`public string Account_Address { get; set; }`
		
		`public string? Account_Image { get; set; }`
		
		`public string? Account_Deleted { get; set; } = Status.Existing.ToString();`
		
		`public virtual ICollection<Order>? Orders { get; set; }`
		
		`public virtual ICollection<Cart>? Carts { get; set; }`
		`}`
		`}`

2	Books.cs	`namespace FPT_Book_Store.Models`
		`{`
		`public class Book`
		`{`
		`[Key]`
		`[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`
		`public int Book_ID { get; set; }`
		
		`public int Publisher_ID { get; set; }`
		`[ForeignKey("Publisher_ID")]`
		`public virtual Publisher? Publisher { get; set; }`
		
		`public int Category_ID { get; set; }`
		`[ForeignKey("Category_ID")]`
		`public virtual Category? Category { get; set; }`
		
		`[Required(ErrorMessage = "Please, enter the book name!")]`
		`[StringLength(50, ErrorMessage = "Please, enter the book name must be between {2} and {1}.", MinimumLength = 1)]`
		`public string Book_Name { get; set; }`


```
        public string? Book_Description { get; set; }

        [Display(Name = "Quantity")]
        [Range(0, 10000)]
        public int Book_Quantity { get; set; }

        [Display(Name = "Original Price")]
        [Range(0, 1000000000)]
        public decimal Book_OriginalPrice { get; set; }

        [Display(Name = "Sale Price")]
        [Range(0, 1000000000)]
        public decimal Book_SalePrice { get; set; }

        public DateTime? Book_Date { get; set; }

        public string? Book_Image { get; set; }

            public  string?  Book_Deleted  {  get;  set;  }  =
Status.Existing.ToString();

        public virtual ICollection<Cart>? Carts { get; set; }

         public virtual ICollection<OrderDetail>? OrdersDetail {
get; set; }
    }
}
```

3	Cart.cs	

```
namespace FPT_Book_Store.Models
{
    public class Cart
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Cart_ID { get; set; }

        public string Account_ID { get; set; }
        [ForeignKey("Account_ID")]
        public virtual Accounts? Account { get; set; }

        public int Book_ID { get; set; }
```

		`[ForeignKey("Book_ID")]`
		`public virtual Book? Book { get; set;}`
		
		`public int? Cart_Quantity { get; set; }`
		
		`public string? Cart_Deleted { get; set; }`
		`}`
		`}`

4	Category.cs	`namespace FPT_Book_Store.Models`
		`{`
		`public class Category`
		`{`
		`[Key]`
		`[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`
		`public int Category_ID { get; set; }`
		
		`[Required(ErrorMessage = "Please, enter the category!")]`
		`[StringLength(30, ErrorMessage = "Please, enter the category must be between {2} and {1}.", MinimumLength = 1)]`
		`public string Category_Type { get; set; }`
		
		`public string? Category_Status { get; set; } = Status.Pending.ToString();`
		
		`public string? Category_Deleted { get; set; } = Status.Existing.ToString();`
		
		`public virtual ICollection<Book>? Books { get; set; }`
		`}`
		`}`

5	Order.cs	`namespace FPT_Book_Store.Models`
		`{`
		`public class Order`
		`{`
		`[Key]`
		`[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`
		`public int Order_ID { get; set; }`

```
        public string Account_ID { get; set; }
        [ForeignKey("Account_ID")]
        public virtual Accounts? Account { get; set; }

            [Required(ErrorMessage = "Please, enter the phone
number!")]
        [RegularExpression(@"^0[0-9]{9}",
        ErrorMessage = "Please, enter a valid phone number!")]
        public string Order_Phone { get; set; }

        [Required(ErrorMessage = "Please, enter the address!")]
        [StringLength(100)]
        public string Order_Address { get; set; }

        public DateTime? Order_OrderDate { get; set; }

        public DateTime? Order_DeliveryDate { get; set; }

        public string? Order_Status { get; set; }

         public virtual ICollection<OrderDetail>? OrdersDetail {
get; set; }
    }
}
```

6	OrderDetail.cs

```
namespace FPT_Book_Store.Models
{
    public class OrderDetail
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int OrderDetail_ID {get;set;}

        public int Order_ID {get;set;}
        [ForeignKey("Order_ID")]
        public virtual Order? Order {get;set;}

        public int Book_ID {get;set;}
        [ForeignKey("Book_ID")]
        public virtual Book? Book {get;set;}

        [Display(Name = "Quantity")]
```

		`[Range(1, 1000000000)]`
		`public int OrderDetail_Quantity {get;set;}`
		`}`
		`}`

7	Publisher.cs	`namespace FPT_Book_Store.Models`
		`{`
		`public class Publisher`
		`{`
		`[Key]`
		`[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`
		`public int Publisher_ID { get; set; }`
		
		`[Required(ErrorMessage = "Please, enter a publisher name!")]`
		`[StringLength(50, ErrorMessage = "Please, enter the publisher name length must be between {2} and {1}.", MinimumLength = 1)]`
		`[RegularExpression(@"^[A-Za-z]{1,50}$",`
		`ErrorMessage = "Please, enter a valid publisher name!")]`
		`public string Publisher_Name { get; set; }`
		
		`[Required(ErrorMessage ="Please, enter the email!")]`
		`[StringLength(100)]`
		`[RegularExpression(@"^[\w!#$%&'*+\-/=?\^_`{\|}~]+(\.[\w!#$%&'*+\-/=?\^_`{|}~]+)*"`
		`+ "@"+ @"((([\-\w]+\.)+[a-zA-Z]{2,4})|(([0-9]{1,3}\.){3}[0-9]{1,3}))$",`
		`ErrorMessage = "Please, enter a valid email address!")]`
		`public string Publisher_Email { get; set; }`
		
		`[Required(ErrorMessage ="Please, enter the phone number!")]`
		`[RegularExpression(@"^0[0-9]{9}",`
		`ErrorMessage = "Please, enter a valid phone number!")]`
		`public string Publisher_Phone { get; set; }`
		
		`public string? Publisher_Deleted { get; set; } = Status.Existing.ToString();`
		
		`public virtual ICollection<Book>? Books { get; set; }`
		`}`
		`}`

8	*ErrorViewModel.cs*	

```
namespace FPT_Book_Store.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

                        public     bool     ShowRequestId     =>
!string.IsNullOrEmpty(RequestId);
    }
}
```

3.2.7. Controller

- *BookController.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace FPT_Book_Store.Areas.Owner.Controllers
{
    [Area("Owner")]
    [Route("Owner/[controller]/[action]")]
    [Authorize(Roles = "Owner")]
    public class BookController : Controller
    {
        private readonly ApplicationDbContext _db;

        public BookController(ApplicationDbContext db)
        {
            _db = db;
        }

        public async Task<IActionResult> Index()
        {
            IEnumerable<Book> ds = _db.Books.Where(b => b.Book_Deleted ==
Status.Existing.ToString() && b.Category.Category_Status
            == Status.Approved.ToString()).ToList();
```

```
            ViewData["Publisher"] = _db.Publishers.ToList();
            ViewData["Category"] = _db.Categories.ToList();

            return View(ds);
        }

        public async Task<ActionResult> Create()
        {
            ViewData["Publisher"] = _db.Publishers.Where(p => p.Publisher_Deleted ==
Status.Existing.ToString()).ToList();
            ViewData["Category"] = _db.Categories.Where(c => c.Category_Status ==
Status.Approved.ToString()).ToList();
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Create(IFormFile Book_Image, Book books)
        {
            if(ModelState.IsValid){
                var filePaths = new List<string>();
                string file =
Path.GetExtension(Book_Image.FileName).ToLower().Trim();
                if(Book_Image.Length > 0){
                    if(file == ".jpg" || file == ".png"){
                        var filePath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot", "Uploads//Item_Image", Book_Image.FileName);

                        using (var stream = new FileStream(filePath,
FileMode.Create)){
                        await Book_Image.CopyToAsync(stream);
                        }
                    }
                    else{
                        TempData["message"] = "File Type invalid. Only accept the
file .jpg and .png!";
                        return RedirectToAction("Create");
                    }

                }

                books.Book_Date = DateTime.Now;
                books.Book_Image = Book_Image.FileName;
                _db.Books.Add(books);
                _db.SaveChanges();
                return RedirectToAction("Index");
            }

```

```
            return View(books);
        }

        public async Task<IActionResult> Edit(int id)
        {
            Book obj = _db.Books.Find(id);
            // ViewData["Publisher_ID"] = new SelectList(_db.Publishers,
"Publisher_ID", "Publisher_Name");
             ViewData["Publisher"] = _db.Publishers.Where(p => p.Publisher_Deleted ==
Status.Existing.ToString()).ToList();
            ViewData["Category"] = _db.Categories.Where(c => c.Category_Status ==
"Approved").ToList();
            if(obj == null){

                return RedirectToAction("Index");
            }
            return View(obj);
        }

        [HttpPost]
        public async Task<IActionResult> Edit(Book obj, IFormFile? Book_Images)
        {
            try
            {
                if(ModelState.IsValid){
                    var filePaths = new List<string>();
                    string file =
Path.GetExtension(Book_Images.FileName).ToLower().Trim();
                    if(Book_Images != null){
                        if(file == ".jpg" || file == ".png"){
                            var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "Uploads//Item_Image",
Book_Images.FileName);

                            using (var stream = new FileStream(filePath,
FileMode.Append)){
                                await Book_Images.CopyToAsync(stream);
                            }
                            obj.Book_Image = Book_Images.FileName;
                        }
                        else{
                            TempData["message"] = "File Type invalid. Only accept the
file .jpg and .png!";
                            return RedirectToAction("Edit", new {id = obj.Book_ID});
                        }
                    }

```

```
                    obj.Book_Date = DateTime.Now;
                    _db.Books.Update(obj);
                    _db.SaveChanges();
                }
            }
            catch (System.Exception)
            {
                return RedirectToAction("Index");
            }

            return View("Book/Index");
        }

        public async Task<IActionResult> Delete(int id)
        {
            Book obj = _db.Books.Find(id);
            if(obj != null){
                obj.Book_Deleted = Status.Deleted.ToString();
                _db.Books.Update(obj);
                _db.SaveChanges();

                IEnumerable<Cart> ds = _db.Carts.Where(c => c.Book_ID ==
obj.Book_ID).ToList();
                foreach(var i in ds){
                    _db.Carts.Remove(i);
                    _db.SaveChanges();
                }
            }
            return RedirectToAction("Index");
        }

    }
}
```

Table 3. BookController explain table

No.	Action	Describe

1	`public async Task<IActionResult> Index()`
	`{`
	`IEnumerable<Book> ds = _db.Books.Where(b => b.Book_Deleted == Status.Existing.ToString() && b.Category.Category_Status == Status.Approved.ToString()).ToList();`
	`ViewData["Publisher"] = _db.Publishers.ToList();`
	`ViewData["Category"] = _db.Categories.ToList();`
		This action is used to show all of the books in the system to the Manager Book page. This action will return the Manager Book page and the list of the books

```
            return View(ds);
        }
```

2

```
 public async Task<ActionResult> Create()
        {
            ViewData["Publisher"] =
_db.Publishers.Where(p => p.Publisher_Deleted ==
Status.Existing.ToString()).ToList();
            ViewData["Category"] = _db.Categories.Where(c
=> c.Category_Status ==
Status.Approved.ToString()).ToList();
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Create(IFormFile
Book_Image, Book books)
        {
            if(ModelState.IsValid){
                var filePaths = new List<string>();
                string file =
Path.GetExtension(Book_Image.FileName).ToLower().Trim();
                if(Book_Image.Length > 0){
                    if(file == ".jpg" || file == ".png"){
                        var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"Uploads//Item_Image", Book_Image.FileName);

                        using (var stream = new
FileStream(filePath, FileMode.Create)){
                        await
Book_Image.CopyToAsync(stream);
                        }
                    }
                    else{
                        TempData["message"] = "File Type
invalid. Only accept the file .jpg and .png!";
                        return
RedirectToAction("Create");
                    }

                }

                books.Book_Date = DateTime.Now;
                books.Book_Image = Book_Image.FileName;
                _db.Books.Add(books);
```

Two these actions are used to add a new book to the system. In the Create(), this action will return the value of the publisher and the category. In the Create(IFormFile Book_Image, Book books), this action will get the value from the form add and perform adding a new book and return the Manager Book page and an object of book

	``
	_db.SaveChanges();
	return RedirectToAction("Index");
	}
	
	return View(books);
	}
	``	

3	```
	public async Task<IActionResult> Edit(int id)
	{
	Book obj = _db.Books.Find(id);
	ViewData["Publisher_ID"] = new SelectList(_db.Publishers, "Publisher_ID", "Publisher_Name");
	ViewData["Category"] = _db.Categories.Where(c => c.Category_Status == "Approved").ToList();
	if(obj == null){
	
	return RedirectToAction("Index");
	}
	return View(obj);
	}
	``
		This action will return the update form and an object of the book with the id that get

4	```
	[HttpPost]
	public async Task<IActionResult> Edit(Book obj, IFormFile? Book_Images)
	{
	try
	{
	if(ModelState.IsValid){
	var filePaths = new List<string>();
	string file = Path.GetExtension(Book_Images.FileName).ToLower().Trim();
	if(Book_Images != null){
	if(file == ".jpg" || file == ".png"){
	var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "Uploads//Item_Image", Book_Images.FileName);
	
	using (var stream = new FileStream(filePath, FileMode.Append)){
	await Book_Images.CopyToAsync(stream);
	}
	``
		This action will get the value of the book from the form edit and perform updating function and return the Manager Book page and the object of book that update

```
                            obj.Book_Image =
Book_Images.FileName;
                        }
                        else{
                            TempData["message"] = "File
Type invalid. Only accept the file .jpg and .png!";
                            return
RedirectToAction("Edit", new {id = obj.Book_ID});
                        }
                    }

                    obj.Book_Date = DateTime.Now;
                    _db.Books.Update(obj);
                    _db.SaveChanges();
                }
            }
            catch (System.Exception)
            {
                return RedirectToAction("Index");
            }

            return View("Book/Index");
        }
```

5

```
public async Task<IActionResult> Delete(int id)
        {
            Book obj = _db.Books.Find(id);
            if(obj != null){
                                    obj.Book_Deleted    =
Status.Deleted.ToString();
                _db.Books.Update(obj);
                _db.SaveChanges();

                 IEnumerable<Cart> ds = _db.Carts.Where(c
=> c.Book_ID == obj.Book_ID).ToList();
                foreach(var i in ds){
                    _db.Carts.Remove(i);
                    _db.SaveChanges();
                }
            }
            return RedirectToAction("Index");
        }
```

This action will return the Manager Book page and this action is used to change the book's status

- *CategoryController.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FPT_Book_Store.Areas.Owner.Controllers
{
    [Area("Owner")]
    [Route("Owner/[controller]/[action]")]
    [Authorize(Roles = "Owner")]
    public class CategoryController : Controller
    {
        private readonly ApplicationDbContext _db;
        public CategoryController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult Index()
        {
            IEnumerable<Category> ds = _db.Categories.Where(c => c.Category_Deleted ==
Status.Existing.ToString()).ToList();
            return View(ds);
        }

        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Category obj)
        {
            if (ModelState.IsValid)
            {
```

```
                obj.Category_Status = Status.Pending.ToString();
                obj.Category_Deleted = Status.Existing.ToString();
                _db.Categories.Add(obj);
                _db.SaveChanges();


                return RedirectToAction("Index");
            }
            return View(obj);
        }

        public IActionResult Edit(int id)
        {
            Category obj = _db.Categories.Find(id);
            if (obj == null)
            {
                return RedirectToAction("Index");
            }
            return View(obj);
        }

        [HttpPost]
        public IActionResult Edit(Category obj)
        {
            if (ModelState.IsValid)
            {
                obj.Category_Status = Status.Pending.ToString();
                obj.Category_Deleted = Status.Existing.ToString();
                _db.Categories.Update(obj);
                _db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(obj);
        }

        public IActionResult Delete(int id)
        {
            Category obj = _db.Categories.Find(id);
            if(obj != null){
                obj.Category_Deleted = Status.Deleted.ToString();
```

```
                _db.Categories.Update(obj);
                _db.SaveChanges();
            }
            return RedirectToAction("Index");
        }
    }
}
```

Table 4. CategoryController explain table

No.	Action	Describe

1	`public IActionResult Index()`
	`{`
	`IEnumerable<Category> ds = _db.Categories.Where(c => c.Category_Deleted == Status.Existing.ToString()).ToList();`
	`return View(ds);`
	`}`
		This action will return the Manager Category page and a list of the categories

2	`public IActionResult Create()`
	`{`
	`return View();`
	`}`
	
	`[HttpPost]`
	`public IActionResult Create(Category obj)`
	`{`
	`if (ModelState.IsValid)`
	`{`
	`obj.Category_Status = Status.Pending.ToString();`
	`obj.Category_Deleted = Status.Existing.ToString();`
	`_db.Categories.Add(obj);`
	`_db.SaveChanges();`
	
	`return RedirectToAction("Index");`
	`}`
	`return View(obj);`
	`}`
		Two these actions are used to perform adding a new category function. In the first Create() , this action will return the form add of the category. In the second Create(Category obj), this action will get the value of category that send from the form add and this action will return the Manager Category page and an object of the category and if adding is failure the action will return the form add with the value that get before

3	`public IActionResult Edit(int id)`
	`{`
	`Category obj = _db.Categories.Find(id);`
	`if (obj == null)`
	`{`
	`return RedirectToAction("Index");`
	`}`
	`return View(obj);`
	`}`
		This action will return the form update of the category and an object of category with the id of that category

4	`[HttpPost]`
	`public IActionResult Edit(Category obj)`
	`{`
	`if (ModelState.IsValid)`
	`{`
	`obj.Category_Status = Status.Pending.ToString();`
	`obj.Category_Deleted = Status.Existing.ToString();`
	`_db.Categories.Update(obj);`
	`_db.SaveChanges();`
	`return RedirectToAction("Index");`
	`}`
	`return View(obj);`
	`}`
		This action will get the category's value from the form update and perform updating function and this action will return the Manager Category page and an object of the category that update. If adding is failure the action will return the form update with the value that get before.

5	`public IActionResult Delete(int id)`
	`{`
	`Category obj = _db.Categories.Find(id);`
	`if(obj != null){`
	`obj.Category_Deleted = Status.Deleted.ToString();`
	`_db.Categories.Update(obj);`
	`_db.SaveChanges();`
	`}`
	`return RedirectToAction("Index");`
	`}`
		This action is used to change the category's status and this action will return the Manger Category page

- *OrderController.cs*

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace FPT_Book_Store.Areas.Owner.Controllers
{
    [Area("Owner")]
    [Route("Owner/[controller]/[action]")]
    [Authorize(Roles = "Owner")]
    public class OrderController : Controller
    {
        private readonly ApplicationDbContext _db;

        public OrderController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult ShowOrder()
        {
                IEnumerable<Order> ds = _db.Orders.Where(o => o.Order_Status !=
Status.Canceled.ToString()).
            OrderBy(o => o.Order_Status).Include(a => a.Account).ToList();
            return View(ds);
        }


        public IActionResult ConfirmOrder(int id)
        {
            IEnumerable<Order> ds = _db.Orders.Where(o => o.Order_ID == id).ToList();
            if(ds.Count() > 0){
                foreach(var item in ds){
                    item.Order_Status = Status.Received.ToString();
                    item.Order_DeliveryDate = DateTime.Now;
                    _db.Orders.Update(item);
```

```
                    _db.SaveChanges();
                    break;
                }
            }
            return RedirectToAction("ShowOrder");
        }

    }
}
```

Table 5. OrderController explain table

No.	Action	Describe

1	`public IActionResult ShowOrder()`
	`{`
	`IEnumerable<Order> ds = _db.Orders.Where(o => o.Order_Status != Status.Canceled.ToString()).`
	`OrderBy(o => o.Order_Status).Include(a => a.Account).ToList();`
	`return View(ds);`
	`}`
		This action will show all order of the customer and this action will return the Manager Order page and a list of orders

2	`public IActionResult ConfirmOrder(int id)`
	`{`
	`IEnumerable<Order> ds = _db.Orders.Where(o => o.Order_ID == id).ToList();`
	`if(ds.Count() > 0){`
	`foreach(var item in ds){`
	`item.Order_Status = Status.Received.ToString();`
	`item.Order_DeliveryDate = DateTime.Now;`
	`_db.Orders.Update(item);`
	`_db.SaveChanges();`
	`break;`
	`}`
	`}`
	`return RedirectToAction("ShowOrder");`
	`}`
		This action is used to perform the confirmation of the customer's order function. This action will return the Manager Order page

- *PublisherController.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using FPT_Book_Store.Data;
using Microsoft.AspNetCore.Mvc;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using FPT_Book_Store.Constants;
using Microsoft.AspNetCore.Authorization;

namespace FPT_Book_Store.Areas.Owner.Controllers
{
    [Area("Owner")]
    [Route("Owner/[controller]/[action]")]
    [Authorize(Roles = "Owner")]
    public class PublisherController : Controller
    {
        private readonly ApplicationDbContext _db;

        public PublisherController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult Index()
        {
                    IEnumerable<Publisher>  publishers  =  _db.Publishers.Where(p  =>
p.Publisher_Deleted == Status.Existing.ToString());

            return View(publishers);
        }

        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
```

```
        public IActionResult Create(Publisher obj)
        {
            if(ModelState.IsValid){
                obj.Publisher_Deleted = Status.Existing.ToString();
                _db.Publishers.Add(obj);
                _db.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(obj);
        }



        public IActionResult Edit(int id)
        {
            Publisher publisher = _db.Publishers.Find(id);

            if(publisher == null){

                return RedirectToAction("Index");
            }

            return View(publisher);
        }

        [HttpPost]
        public IActionResult Edit(Publisher obj)
        {

            if(ModelState.IsValid){

                obj.Publisher_Deleted = Status.Existing.ToString();
                _db.Publishers.Update(obj);
                _db.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(obj);
        }
```

```
        public IActionResult Delete(int id)
        {
            Publisher publisher = _db.Publishers.Find(id);

            if(publisher != null){
                publisher.Publisher_Deleted = Status.Deleted.ToString();
                _db.Publishers.Update(publisher);
                _db.SaveChanges();

            }

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

Table 6. PublisherController explain table

No.	Action	Describe

1	`public IActionResult Index()`
	`{`
	`IEnumerable<Publisher> publishers = _db.Publishers.Where(p => p.Publisher_Deleted == Status.Existing.ToString());`
	`return View(publishers);`
	`}`
		This action will show all the publishers and this action will return the Manager Publisher page and a list of the Publishers

2	`public IActionResult Create()`
	`{`
	`return View();`
	`}`
	`[HttpPost]`
	`public IActionResult Create(Publisher obj)`
	`{`
	`if(ModelState.IsValid){`
	`obj.Publisher_Deleted = Status.Existing.ToString();`
	`_db.Publishers.Add(obj);`
		Two of these actions will perform the adding a new publisher. In the first, Create() action will return the form add the publisher. In the second, Create(Publisher obj) action will get the new value of publisher from the form add and perform adding and this action will return the Manager Publisher page and

	``_db.SaveChanges();
	return RedirectToAction("Index");
	}
	
	return View(obj);
	}```
		an object of the new publisher. If adding is failure the action will return the form add with the value that get before

3	```public IActionResult Edit(int id)
	{
	Publisher publisher = _db.Publishers.Find(id);
	
	if(publisher == null){
	
	return RedirectToAction("Index");
	}
	
	return View(publisher);
	}```
		This action will return the form update of the publisher and the publisher's value with the id of that publisher

4	```[HttpPost]
	public IActionResult Edit(Publisher obj)
	{
	
	if(ModelState.IsValid){
	
	obj.Publisher_Deleted = Status.Existing.ToString();
	_db.Publishers.Update(obj);
	_db.SaveChanges();
	return RedirectToAction("Index");
	}
	
	return View(obj);
	}```
		This action will get the value of the publisher from the form update and perform updating and this action will return the Manager Publisher page and an object of the publisher that update. If update is failure the action will return the form update with the object of the publisher

5	```public IActionResult Delete(int id)
	{
	Publisher publisher = _db.Publishers.Find(id);
	
	if(publisher != null){```
		This action will return the Manager Publisher page and this action perform update the status of the publisher

```
                        publisher.Publisher_Deleted  =
Status.Deleted.ToString();
                _db.Publishers.Update(publisher);
                _db.SaveChanges();

            }

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

- *DashboardController.cs*

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using FPT_Book_Store.Data;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using FPT_Book_Store.Models;
using FPT_Book_Store.Constants;
using Newtonsoft.Json;
using FPT_Book_Store.Areas.Owner.Models;

namespace FPT_Book_Store.Areas.Owner.Controllers
{
    [Area("Owner")]
    [Route("Owner/[controller]/[action]")]
    [Authorize(Roles = "Owner")]
    public class DashboardController : Controller
    {
        private readonly ApplicationDbContext _db;

        public DashboardController(ApplicationDbContext db)
        {
            _db = db;
        }
        public IActionResult Index()
```

```
        {
            IEnumerable<Statistic> list = _db.OrdersDetail.
             Where(o => o.Order.Order_Status == Status.Received.ToString()).GroupBy(o
=> o.Book.Book_ID).
            Select(t => new Statistic
            {
                Book_Image = t.First().Book.Book_Image,
                Book_ID = t.Key,
                Book_Name = t.First().Book.Book_Name,
                Category_Type = t.First().Book.Category.Category_Type,
                Publisher_Name = t.First().Book.Publisher.Publisher_Name,
                Book_Quantity = t.First().Book.Book_Quantity,
                OrdersDetail_Quantity = t.Sum(o => o.OrderDetail_Quantity)

            }).ToList();


            return View(list);
        }

    }
}
```

Table 7. DashboardController explain table

No.	Action	Describe

1	`public IActionResult Index()`
	`{`
	`IEnumerable<Statistic> list = _db.OrdersDetail.`
	`Where(o => o.Order.Order_Status == Status.Received.ToString()).GroupBy(o => o.Book.Book_ID).`
	`Select(t => new Statistic`
	`{`
	`Book_Image = t.First().Book.Book_Image,`
	`Book_ID = t.Key,`
	`Book_Name = t.First().Book.Book_Name,`
	`Category_Type = t.First().Book.Category.Category_Type,`
		This action will perform statistic the book that sold. This action will return the Dashboard page and the list of the books that sold

```
                                    Publisher_Name    =
t.First().Book.Publisher.Publisher_Name,
                                     Book_Quantity    =
t.First().Book.Book_Quantity,
                  OrdersDetail_Quantity = t.Sum(o =>
o.OrderDetail_Quantity)

            }).ToList();


            return View(list);
        }
```

- *CartController.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace FPT_Book_Store.Areas.Customer.Controllers
{
    [Area("Customer")]
    [Route("Customer/[controller]/[action]")]
    [Authorize(Roles="User")]
    public class CartController : Controller
    {
        private readonly ApplicationDbContext _db;

        public CartController(ApplicationDbContext db)
        {
            _db = db;
```

```
        }
        public IActionResult ShowCart()
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
           IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).Include(b
=> b.Book).ToList();

            decimal? total = 0;

            if (ds.Count() >= 1)
            {
                foreach (var i in ds)
                {
                   total += i.Book.Book_SalePrice * i.Cart_Quantity;
                }
            }

            TempData["total"] = total;

            return View(ds);
        }

        public IActionResult AddCart(int id)
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
            var product = _db.Books.Where(b => b.Book_ID == id).FirstOrDefault();

              IEnumerable<Cart> ls = _db.Carts.Where(c => c.Account_ID == user &&
c.Cart_Quantity >= 1
            && c.Book_ID == product.Book_ID).ToList();

            if (ls.Count() > 0)
            {
                foreach (var item in ls)
                {
                    item.Cart_Quantity = item.Cart_Quantity + 1;
                    _db.Carts.Update(item);
                    _db.SaveChanges();
                    break;
                }
```

```
            }
            else
            {
                Cart a = new Cart();
                a.Account_ID = user;
                a.Book_ID = product.Book_ID;
                a.Cart_Quantity = 1;
                a.Cart_Deleted = Status.Existing.ToString();
                _db.Carts.Add(a);
                _db.SaveChanges();
            }
            return RedirectToAction("ShowCart");
        }

        public IActionResult UpdateIncrease(int id)
        {
            IEnumerable<Cart> ls = _db.Carts.Where(c => c.Cart_ID == id).ToList();

            if (ls.Count() > 0)
            {
                foreach (var item in ls)
                {
                    item.Cart_Quantity = item.Cart_Quantity + 1;
                    _db.Carts.Update(item);
                    _db.SaveChanges();
                    break;
                }
            }

            return RedirectToAction("ShowCart");
        }

        public IActionResult UpdateDescrease(int id)
        {
            IEnumerable<Cart> ls = _db.Carts.Where(c => c.Cart_ID == id).ToList();

            if (ls.Count() > 0)
            {
                foreach (var item in ls)
                {
```

```
                    if (item.Cart_Quantity > 0)
                    {
                        item.Cart_Quantity = item.Cart_Quantity - 1;
                        _db.Carts.Update(item);
                        _db.SaveChanges();
                    }

                    if (item.Cart_Quantity <= 0)
                    {
                        Cart cart = _db.Carts.Find(id);
                        _db.Carts.Remove(cart);
                        _db.SaveChanges();
                    }

                    break;
                }
            }

            return RedirectToAction("ShowCart");
        }

        public IActionResult Delete(int id)
        {
            Cart cart = _db.Carts.Find(id);
            if (cart != null)
            {
                _db.Carts.Remove(cart);
                _db.SaveChanges();
                TempData["message"] = "The book has been deleted successfully!";
            }
            return RedirectToAction("ShowCart");
        }

        public IActionResult Order(int id, string address, string phone)
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
                Cart ds = _db.Carts.Where(c => c.Cart_ID == id).Include(a =>
a.Account).Include(c => c.Book).First();

            if (address == "" || phone == "")
```

```
            {
                TempData["error"] = "To place an order, please enter the address and phone number!";
                return RedirectToAction("ShowCart", "Cart");
            }

            Order orders = new Order();

            orders.Account_ID = user;
            orders.Order_Address = address;
            orders.Order_Phone = phone;
            orders.Order_OrderDate = DateTime.Now;
            orders.Order_Status = Status.Pending.ToString();

            _db.Orders.Add(orders);
            _db.SaveChanges();

            OrderDetail od = new OrderDetail();

            od.Order_ID = orders.Order_ID;
            od.Book_ID = ds.Book_ID;
            od.OrderDetail_Quantity = (int)ds.Cart_Quantity;

            _db.OrdersDetail.Add(od);
            _db.SaveChanges();

                IEnumerable<Book> lstBook = _db.Books.Where(b => b.Book_ID == ds.Book_ID).ToList();

            foreach(var book in lstBook){
                book.Book_Quantity = ((int)(book.Book_Quantity - ds.Cart_Quantity));
                _db.Books.Update(book);
                _db.SaveChanges();
            }

            _db.Carts.Remove(ds);
            _db.SaveChanges();

            TempData["message"] = "Order is Successfully!";

```

```
            return RedirectToAction("ShowCart", "Cart");
        }

        public IActionResult OrderAll(string address, string phone)
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
            IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).Include(a
=> a.Account).Include(c => c.Book).ToArray();

            if(address == "" || phone == ""){

                TempData["error"] = "To place an order, please enter the address and
phone number!";
                return RedirectToAction("ShowCart", "Cart");
            }

            Order order = new Order();

                if (ds.Count() > 0)
                {
                    order.Account_ID = user;
                    order.Order_Phone = phone;
                    order.Order_Address = address;
                    order.Order_OrderDate = DateTime.Now;
                    order.Order_Status = Status.Pending.ToString();

                    _db.Orders.Add(order);
                    _db.SaveChanges();
                }

            Order lst = _db.Orders.Where(o => o.Order_ID == order.Order_ID).First();
                IEnumerable<Cart>  carts  =  _db.Carts.Where(c  =>  c.Account_ID  ==
user).ToList();

                    foreach (var c in carts)
                    {
                        OrderDetail od = new OrderDetail();
                        od.Order_ID = lst.Order_ID;
                        od.Book_ID = c.Book_ID;
                        od.OrderDetail_Quantity = (int)c.Cart_Quantity;
```

```
                        _db.OrdersDetail.Add(od);
                        _db.SaveChanges();

                      Book book = _db.Books.Where(b => b.Book_ID == c.Book_ID).First();

                        if(book != null){

                                    book.Book_Quantity  =  ((int)(book.Book_Quantity  -
c.Cart_Quantity));
                            _db.Books.Update(book);
                            _db.SaveChanges();
                        }
                    }

                    foreach (var cart in carts)
                    {
                        _db.Carts.Remove(cart);
                        _db.SaveChanges();
                    }

                TempData["message"] = "Order is Successfully!";

            return RedirectToAction("ShowCart", "Cart");
        }

        public IActionResult ConfirmFormOrder(int id)
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
            Cart ds = _db.Carts.Where(c => c.Cart_ID == id && c.Account_ID == user).
            Include(a => a.Book).Include(c => c.Book.Category).First();

            ViewData["user"] = _db.Accounts.Where(a => a.Id == user).ToList();

            if (user == "")
            {
                TempData["error"] = "User ID Error!";
                return RedirectToAction("ShowCart", "Cart");
            }

```

```
            if (ds.Cart_Quantity > ds.Book.Book_Quantity)
            {
                TempData["message"] = "The " + "\"" + ds.Book.Book_Name + "\"" + "
book in stock is only " + ds.Book.Book_Quantity;
                return RedirectToAction("ShowCart", "Cart");
            }

            return View(ds);
        }

        public IActionResult ConfirmFormOrderAll()
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
            IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).
                    Include(a  =>  a.Account).Include(c  =>  c.Book).Include(c  =>
c.Book.Category).ToList();
            ViewData["user"] = _db.Accounts.Where(a => a.Id == user).ToList();

            if (user == "")
            {
                TempData["error"] = "User ID Error!";
                return RedirectToAction("ShowCart", "Cart");
            }

            if(ds.Count() <= 0){

                TempData["error"] = "Please add the book to the cart!";
                return RedirectToAction("ShowCart", "Cart");
            }

            foreach (var ls in ds)
            {
                if (ls.Cart_Quantity > ls.Book.Book_Quantity)
                {
                    TempData["error"] = "The " + ls.Book.Book_Name + " book in stock
is only " + ls.Book.Book_Quantity;
                    return RedirectToAction("ShowCart", "Cart");
                }
            }

```

```
            return View(ds);
        }

    }
}
```

Table 8. CartController explain table

No.	Action	Describe

1	`public IActionResult ShowCart()`
	`{`
	`var user = User.FindFirstValue(ClaimTypes.NameIdentifier);`
	`IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).Include(b => b.Book).ToList();`
	
	`decimal? total = 0;`
	
	`if (ds.Count() >= 1)`
	`{`
	`foreach (var i in ds)`
	`{`
	`total += i.Book.Book_SalePrice * i.Cart_Quantity;`
	`}`
	`}`
	
	`TempData["total"] = total;`
	
	`return View(ds);`
	`}`
		This action is used to show the cart of a customer. This action will return the Cart page and the list of cart of a customer

2	`public IActionResult AddCart(int id)`
	`{`
	`var user = User.FindFirstValue(ClaimTypes.NameIdentifier);`
	`var product = _db.Books.Where(b => b.Book_ID == id).FirstOrDefault();`
	
	`IEnumerable<Cart> ls = _db.Carts.Where(c => c.Account_ID == user && c.Cart_Quantity >= 1`
	`&& c.Book_ID == product.Book_ID).ToList();`
		This action is used to perform the add to cart function. This action will return the Cart page

	``if (ls.Count() > 0)
	{
	foreach (var item in ls)
	{
	item.Cart_Quantity = item.Cart_Quantity + 1;
	_db.Carts.Update(item);
	_db.SaveChanges();
	break;
	}
	}
	else
	{
	Cart a = new Cart();
	a.Account_ID = user;
	a.Book_ID = product.Book_ID;
	a.Cart_Quantity = 1;
	a.Cart_Deleted = Status.Existing.ToString();
	_db.Carts.Add(a);
	_db.SaveChanges();
	}
	return RedirectToAction("ShowCart");
	}```	

3	```public IActionResult UpdateIncrease(int id)
	{
	IEnumerable<Cart> ls = _db.Carts.Where(c => c.Cart_ID == id).ToList();
	
	if (ls.Count() > 0)
	{
	foreach (var item in ls)
	{
	item.Cart_Quantity = item.Cart_Quantity + 1;
	_db.Carts.Update(item);
	_db.SaveChanges();
	break;
	}
	}```
		This action performs update the number of products in customer's cart. This action will update increasing the number of products in cart of a customer and this action will return the Cart page

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

4	`public IActionResult UpdateDescrease(int id)`
	`{`
	`IEnumerable<Cart> ls = _db.Carts.Where(c => c.Cart_ID == id).ToList();`
	
	`if (ls.Count() > 0)`
	`{`
	`foreach (var item in ls)`
	`{`
	`if (item.Cart_Quantity > 0)`
	`{`
	`item.Cart_Quantity = item.Cart_Quantity - 1;`
	`_db.Carts.Update(item);`
	`_db.SaveChanges();`
	`}`
	
	`if (item.Cart_Quantity <= 0)`
	`{`
	`Cart cart = _db.Carts.Find(id);`
	`_db.Carts.Remove(cart);`
	`_db.SaveChanges();`
	`}`
	
	`break;`
	`}`
	`}`
	
	`return RedirectToAction("ShowCart");`
	`}`
		This action performs update the number of products in customer's cart. This action will update decreasing the number of products in cart of a customer and this action will return the Cart page

5	`public IActionResult Delete(int id)`
	`{`
	`Cart cart = _db.Carts.Find(id);`
	`if (cart != null)`
	`{`
	`_db.Carts.Remove(cart);`
	`_db.SaveChanges();`
		This action performs deleting the product in customer's cart out of their cart. This action will return the Cart page

	``TempData["message"] = "The book has been deleted successfully!";
	}
	return RedirectToAction("ShowCart");
	}```	

6	```public IActionResult Order(int id, string address, string phone)
	{
	var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
	Cart ds = _db.Carts.Where(c => c.Cart_ID == id).Include(a => a.Account).Include(c => c.Book).First();
	
	if (address == "" || phone == "")
	{
	TempData["error"] = "To place an order, please enter the address and phone number!";
	return RedirectToAction("ShowCart", "Cart");
	}
	
	Order orders = new Order();
	
	orders.Account_ID = user;
	orders.Order_Address = address;
	orders.Order_Phone = phone;
	orders.Order_OrderDate = DateTime.Now;
	orders.Order_Status = Status.Pending.ToString();
	
	_db.Orders.Add(orders);
	_db.SaveChanges();
	
	OrderDetail od = new OrderDetail();
	
	od.Order_ID = orders.Order_ID;
	od.Book_ID = ds.Book_ID;
	od.OrderDetail_Quantity = (int)ds.Cart_Quantity;```
		This action will perform the order function. This action will perform order each product that have in the cart. This action gets three parameters and return the Cart page

| | ```
_db.OrdersDetail.Add(od);
_db.SaveChanges();

IEnumerable<Book> lstBook = _db.Books.Where(b => b.Book_ID == ds.Book_ID).ToList();

foreach(var book in lstBook){
book.Book_Quantity = ((int)(book.Book_Quantity - ds.Cart_Quantity));
_db.Books.Update(book);
_db.SaveChanges();
}

_db.Carts.Remove(ds);
_db.SaveChanges();

TempData["message"] = "Order is Successfully!";

return RedirectToAction("ShowCart", "Cart");
}
```| |
| 7 | ```
public IActionResult OrderAll(string address, string phone)
{
var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).Include(a => a.Account).Include(c => c.Book).ToArray();

if(address == "" || phone == ""){

TempData["error"] = "To place an order, please enter the address and phone number!";
return RedirectToAction("ShowCart", "Cart");
}

Order order = new Order();

if (ds.Count() > 0)
```| *This action will perform the order all function. This action help perform order all of product that have in the cart of customer and this action gets two parameters and return the Cart page* |

```
                {
                    order.Account_ID = user;
                    order.Order_Phone = phone;
                    order.Order_Address = address;
                   order.Order_OrderDate = DateTime.Now;
                                    order.Order_Status   =
Status.Pending.ToString();

                    _db.Orders.Add(order);
                    _db.SaveChanges();
                }

            Order lst = _db.Orders.Where(o => o.Order_ID
== order.Order_ID).First();
            IEnumerable<Cart> carts = _db.Carts.Where(c
=> c.Account_ID == user).ToList();

                    foreach (var c in carts)
                    {
                     OrderDetail od = new OrderDetail();
                        od.Order_ID = lst.Order_ID;
                        od.Book_ID = c.Book_ID;
                              od.OrderDetail_Quantity  =
(int)c.Cart_Quantity;

                        _db.OrdersDetail.Add(od);
                        _db.SaveChanges();

                       Book book = _db.Books.Where(b =>
b.Book_ID == c.Book_ID).First();

                        if(book != null){

                                        book.Book_Quantity  =
((int)(book.Book_Quantity - c.Cart_Quantity));
                            _db.Books.Update(book);
                            _db.SaveChanges();
                        }
                    }

                    foreach (var cart in carts)
                    {
```

	``_db.Carts.Remove(cart);
	_db.SaveChanges();
	}
	
	TempData["message"] = "Order is Successfully!";
	
	return RedirectToAction("ShowCart", "Cart");
	}```	

8	```public IActionResult ConfirmFormOrder(int id)
	{
	var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
	Cart ds = _db.Carts.Where(c => c.Cart_ID == id && c.Account_ID == user).
	Include(a => a.Book).Include(c => c.Book.Category).First();
	
	ViewData["user"] = _db.Accounts.Where(a => a.Id == user).ToList();
	
	if (user == "")
	{
	TempData["error"] = "User ID Error!";
	return RedirectToAction("ShowCart", "Cart");
	}
	
	if (ds.Cart_Quantity > ds.Book.Book_Quantity)
	{
	TempData["message"] = "The " + "\"" + ds.Book.Book_Name + "\"" + " book in stock is only " + ds.Book.Book_Quantity;
	return RedirectToAction("ShowCart", "Cart");
	}
	
	return View(ds);
	}```
		This action supports the order function. This action will show the form for the customer enter the information such address and phone number when order and this action will return the list of carts

9	`public IActionResult ConfirmFormOrderAll()`
	`{`
	`var user = User.FindFirstValue(ClaimTypes.NameIdentifier);`
	`IEnumerable<Cart> ds = _db.Carts.Where(c => c.Account_ID == user).`
	`Include(a => a.Account).Include(c => c.Book).Include(c => c.Book.Category).ToList();`
	`ViewData["user"] = _db.Accounts.Where(a => a.Id == user).ToList();`
	
	`if (user == "")`
	`{`
	`TempData["error"] = "User ID Error!";`
	`return RedirectToAction("ShowCart", "Cart");`
	`}`
	
	`if(ds.Count() <= 0){`
	
	`TempData["error"] = "Please add the book to the cart!";`
	`return RedirectToAction("ShowCart", "Cart");`
	`}`
	
	`foreach (var ls in ds)`
	`{`
	`if (ls.Cart_Quantity > ls.Book.Book_Quantity)`
	`{`
	`TempData["error"] = "The " + ls.Book.Book_Name + " book in stock is only " + ls.Book.Book_Quantity;`
	`return RedirectToAction("ShowCart", "Cart");`
	`}`
	`}`
	
	`return View(ds);`
	`}`
		This action supports the order all function. This action will show the form for the customer enter the information such address and phone number when order and this action will return the list of carts

- *OrderController.cs of Areas/Customer*

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace FPT_Book_Store.Areas.Customer.Controllers
{
    [Area("Customer")]
    [Route("Customer/[controller]/[action]")]
    [Authorize(Roles = "User")]
    public class OrderController : Controller
    {
        private readonly ApplicationDbContext _db;

        public OrderController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult ViewOrder()
        {
            var user = User.FindFirstValue(ClaimTypes.NameIdentifier);
            IEnumerable<Order> ds = _db.Orders.
                    Where(o => o.Account_ID == user && o.Order_Status !=
Status.Canceled.ToString()).ToList();

            TempData["totalOrder"] = ds.Count();

            return View(ds);
```

```
        }


        public IActionResult CancelOrder(int id)
        {
            Order order = _db.Orders.Find(id);
            if(order != null){
                order.Order_Status = Status.Canceled.ToString();
                _db.Orders.Update(order);
                _db.SaveChanges();

                IEnumerable<OrderDetail> ds = _db.OrdersDetail.Where(od => od.Order_ID
== order.Order_ID).Include(od => od.Book).ToList();
                foreach(var i in ds){
                                    i.Book.Book_Quantity  =  (i.Book.Book_Quantity  +
i.OrderDetail_Quantity);
                    _db.Books.Update(i.Book);
                    _db.SaveChanges();
                }
            }
            return RedirectToAction("ViewOrder");
        }

        public IActionResult Detail(int id){
             IEnumerable<OrderDetail> ds = _db.OrdersDetail.Where(o => o.Order_ID ==
id).Include(a => a.Book).ToList();

            return View(ds);
        }

    }
}
```

Table 9. OrderController of Areas/Customer explain table

No.	Action	Describe

1	`public IActionResult ViewOrder()`
	`{`
	`var user = User.FindFirstValue(ClaimTypes.NameIdentifier);`
	`IEnumerable<Order> ds = _db.Orders.`
		This action will show all the customer's order and this action will return the ViewOrder page and a list of orders

	``Where(o => o.Account_ID == user && o.Order_Status != Status.Canceled.ToString()).ToList();
	
	TempData["totalOrder"] = ds.Count();
	
	return View(ds);
	}```	

2	```public IActionResult CancelOrder(int id)
	{
	Order order = _db.Orders.Find(id);
	if(order != null){
	order.Order_Status = Status.Canceled.ToString();
	_db.Orders.Update(order);
	_db.SaveChanges();
	
	IEnumerable<OrderDetail> ds = _db.OrdersDetail.Where(od => od.Order_ID == order.Order_ID).Include(od => od.Book).ToList();
	foreach(var i in ds){
	i.Book.Book_Quantity = (i.Book.Book_Quantity + i.OrderDetail_Quantity);
	_db.Books.Update(i.Book);
	_db.SaveChanges();
	}
	}
	return RedirectToAction("ViewOrder");
	}```
		This action is used to perform the cancel order function. This action will get one parameter and return the ViewOrder page

3	```public IActionResult Detail(int id){
	IEnumerable<OrderDetail> ds = _db.OrdersDetail.Where(o => o.Order_ID == id).Include(a => a.Book).ToList();
	
	return View(ds);
	}```
		This function will show all the order detail of each customer and this action will get one parameter and return the list of order detail

- *HomeController.cs*

```
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using System.Security.Claims;

namespace FPT_Book_Store.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly ApplicationDbContext _db;

        public HomeController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult Index()
        {
            if(User.IsInRole("Admin")){
                return Redirect("Admin/Dashboard/Index");
            }else if(User.IsInRole("Owner")){
                return Redirect("Owner/Dashboard/Index");
            }

                  IEnumerable<Book>  ds  =  _db.Books.Where(b  =>  b.Book_Deleted  ==
Status.Existing.ToString()
            && b.Category.Category_Status == Status.Approved.ToString()).ToList();
            return View(ds);
        }

        public IActionResult ShowDetail(int id)
        {
            Book obj = _db.Books.Find(id);
            ViewData["Publisher"] = _db.Publishers.ToList();

            if (obj == null)
```

```
            {
                return RedirectToAction("Index");
            }
            return View(obj);
        }

        [HttpPost]
        public IActionResult SearchBookName(Book name)
        {
            List<Book> books = _db.Books.ToList();

            if(name.Book_Name != null){
                List<Book> filteredBook = books.Where(m => m.Book_Name.ToLower().
                                    Contains(name.Book_Name.ToLower())).OrderBy(quantity
=>quantity.Book_Quantity).ToList();

                if(filteredBook.Count() == 0){

                    TempData["message"] = "Couldn't find any book titles matching your
keywords!";

                }else if((filteredBook.Count() == 1)){

                      TempData["message"] = "Found " + filteredBook.Count() + " book
matching the keyword";
                }else{

                     TempData["message"] = "Found " + filteredBook.Count() + " books
matching the keyword";
                }
                return View("Index", filteredBook);
            }
            else{
                return View("Index", books);
            }

        }
    }
}
```

Table 10. HomeController explain table

No.	Action	Describe

1	`public IActionResult Index()`
	`{`
	`  if(User.IsInRole("Admin")){`
	`  return Redirect("Admin/Dashboard/Index");`
	`  }else if(User.IsInRole("Owner")){`
	`  return Redirect("Owner/Dashboard/Index");`
	`  }`
	`  // IEnumerable<Book> ds = _db.Books.ToList();`
	`  IEnumerable<Book> ds = _db.Books.Where(b => b.Book_Deleted == Status.Existing.ToString()`
	`  && b.Category.Category_Status == Status.Approved.ToString()).ToList();`
	`  return View(ds);`
	`}`
		This code determines if the user is a "Admin" or "Owner," and if so, directs them to the appropriate dashboard. If not, it will then retrieve all books with a "Existing" book status and a "Approved" category status from the _db database. The ds variable will then be assigned these records, and return the View function with the data set.

2	`public IActionResult ShowDetail(int id)`
	`{`
	`  Book obj = _db.Books.Find(id);`
	`  ViewData["Publisher"] = _db.Publishers.ToList();`
	
	`  if (obj == null)`
	`  {`
	`  return RedirectToAction("Index");`
	`  }`
	`  return View(obj);`
	`}`
		Based on the provided 'id' input, this method pulls a book from the _db database. The ViewData variable is then given a list of every Publisher in the _db database. After that, the code will route the user to the "Index" action if no book object matching the provided id is discovered; otherwise, it will return the View function with the specified Book object.

3	`public IActionResult SearchBookName(Book name)`
	`{`
	
	`List<Book> books = _db.Books.ToList();`
	
	`if(name.Book_Name != null){`
	`List<Book> filteredBook = books.Where(m => m.Book_Name.ToLower().`
	`Contains(name.Book_Name.ToLower())).OrderBy(quantity =>quantity.Book_Quantity).ToList();`
	
	`if(filteredBook.Count() == 0){`
	
	`TempData["message"] = "Couldn't find any book titles matching your keywords!";`
	
	`}else if((filteredBook.Count() == 1)){`
	
	`TempData["message"] = "Found " + filteredBook.Count() + " book matching the keyword";`
	`}else{`
	
	`TempData["message"] = "Found " + filteredBook.Count() + " books matching the keyword";`
	`}`
	`return View("Index", filteredBook);`
	`}`
	`else{`
	`return View("Index", books);`
	`}`
	
	`}`
		Based on the specified book name parameter, this code will run a search on the list of books in the _db database. If the Book Name field contains text, it will look for the specified book and return a list of books that use the specified name as a keyword. Book Quantity will also be used to sort the list. A notification telling the user that no books have been located is displayed if no books match the keyword. If only one book is discovered, a notice alerting the user of the number of books discovered is presented. Finally, the View method receives the list of books.

- *Register.cshtml.cs*

```
using System.ComponentModel.DataAnnotations;
using System.Text;
using System.Text.Encodings.Web;
using Microsoft.AspNetCore.Authentication;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.WebUtilities;
using FPT_Book_Store.Constants;

namespace FPT_Book_Store.Areas.Identity.Pages.Account
{
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<Accounts> _signInManager;
        private readonly UserManager<Accounts> _userManager;
        private readonly IUserStore<Accounts> _userStore;
        private readonly IUserEmailStore<Accounts> _emailStore;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;
        private readonly RoleManager<IdentityRole> _roleManager;

        public RegisterModel(
            UserManager<Accounts> userManager,
            IUserStore<Accounts> userStore,
            SignInManager<Accounts> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender,
            RoleManager<IdentityRole> roleManager)
        {
            _userManager = userManager;
            _userStore = userStore;
            _emailStore = GetEmailStore();
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
            this._roleManager = roleManager;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
```

```
        /// </summary>
        public string ReturnUrl { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public class InputModel
        {
            [Display(Name = "Address")]
            [Required(ErrorMessage = "Please, enter the address!")]
            [StringLength(100)]
            public string Account_Address { get; set; }

            public string Role { get; set; }

            public string Account_Image { get; set; }

            public string Account_Deleted { get; set; }
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Required]
              [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
```

```
            [Display(Name = "Password")]
            public string Password { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [DataType(DataType.Password)]
            [Display(Name = "Confirm password")]
             [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
            public string ConfirmPassword { get; set; }
        }

        public List<IdentityRole> GetListRole()
        {

            return _roleManager.Roles.ToList();
        }

        public async Task OnGetAsync(string returnUrl = null)
        {
            ReturnUrl = returnUrl;
                                                        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
        }

        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");
                                                        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = CreateUser();

                user.Account_Address = Input.Account_Address;
                user.Account_Deleted = Status.Existing.ToString();

                await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
                await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
                var result = await _userManager.CreateAsync(user, Input.Password);

                if (result.Succeeded)
                {
```

```
                        if (Input.Role == "Admin")
                        {

                            await _userManager.AddToRoleAsync(user, Roles.Admin.ToString());

                        }
                        else if (Input.Role == "Owner")
                        {

                            await _userManager.AddToRoleAsync(user, Roles.Owner.ToString());

                        }
                        else
                        {
                            await _userManager.AddToRoleAsync(user, Roles.User.ToString());
                        }
                        _logger.LogInformation("User created a new account with password.");

                        var userId = await _userManager.GetUserIdAsync(user);
                        var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
                        code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                        var callbackUrl = Url.Page(
                            "/Account/ConfirmEmail",
                            pageHandler: null,
                            values: new { area = "Identity", userId = userId, code = code, returnUrl
= returnUrl },
                            protocol: Request.Scheme);

                        await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                                                $"Please confirm your account by <a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

                        if (_userManager.Options.SignIn.RequireConfirmedAccount)
                        {
                            return RedirectToPage("RegisterConfirmation", new { email = Input.Email,
returnUrl = returnUrl });
                        }
                        else
                        {
                            await _signInManager.SignInAsync(user, isPersistent: false);
                            return LocalRedirect(returnUrl);
                        }
                    }
                    foreach (var error in result.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
```

```
                }
            }

            // If we got this far, something failed, redisplay form
            return Page();
    }

        private Accounts CreateUser()
        {
            try
            {
                return Activator.CreateInstance<Accounts>();
            }
            catch
            {
                throw new InvalidOperationException($"Can't create an instance of '{nameof(Accounts)}'. " +
                $"Ensure that '{nameof(Accounts)}' is not an abstract class and has a parameterless constructor, or alternatively " +
                $"override the register page in /Areas/Identity/Pages/Account/Register.cshtml");
            }
        }
        private IUserEmailStore<Accounts> GetEmailStore()
        {
            if (!_userManager.SupportsUserEmail)
            {
                throw new NotSupportedException("The default UI requires a user store with email support.");
            }
            return (IUserEmailStore<Accounts>)_userStore;
        }
```

Table 11. Register explain table

No.	Action	Describe

1	`public List<IdentityRole> GetListRole()`
	`{`
	`return _roleManager.Roles.ToList();`
	`}`
		This code retrieves a list of roles from the `_roleManager` using the `ToList()` method. It converts the `_roleManager.Roles` enumerable into a List object and returns it as the result of the method.

2	`public async Task OnGetAsync(string returnUrl = null)`
	`{`
	`ReturnUrl = returnUrl;`
		This code invokes the '_signInManager' function

	```ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList(); }``` 'GetExternalAuthenticationSchemesAsync()' to get a list of external authentication schemes. The "ExternalLogins" variable then contains the list after it has been transformed into a "List" object. Moreover, the value is set for the'returnUrl' argument, which is used to redirect the user after authentication.	

3	```public async Task<IActionResult> OnPostAsync(string returnUrl = null)
	{
	returnUrl ??= Url.Content("~/");
	ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
	if (ModelState.IsValid)
	{
	var user = CreateUser();
	
	user.Account_Address = Input.Account_Address;
	user.Account_Deleted = Status.Existing.ToString();
	
	await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
	await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
	var result = await _userManager.CreateAsync(user, Input.Password);
	
	if (result.Succeeded)
	{
	if (Input.Role == "Admin")
	{
	
	await _userManager.AddToRoleAsync(user, Roles.Admin.ToString());
	
	}```
	
	This code first sets the `returnUrl` value and then calls the `GetExternalAuthenticationSchemesAsync()` method to get a list of available external authentication schemes and save it in the `ExternalLogins` variable. Next, it checks if the `ModelState` is valid and then it creates the new user object. It then sets the `Account_Address` and `Account_Deleted` properties with the provided input values and then calls the `SetUserNameAsync()`, `SetEmailAsync()`, and `CreateAsync()` methods. Afterwards it checks if the `Role` parameter is set to Admin, Owner, or User and if it is, it adds them to an appropriate Role. After that, it logs the action and then it initiates the email confirmation process. At the very end, it checks if email address confirmation is required and if it is, the user is redirected to the `RegisterConfirmation` page,	

```
                    else if (Input.Role == "Owner")
                    {

                                                        await
_userManager.AddToRoleAsync(user,
Roles.Owner.ToString());

                    }
                    else
                    {
                                                        await
_userManager.AddToRoleAsync(user, Roles.User.ToString());
                    }
                     _logger.LogInformation("User created
a new account with password.");

                                    var   userId   =   await
_userManager.GetUserIdAsync(user);
                                     var   code   =   await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                                                        code    =
WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code))
;
                    var callbackUrl = Url.Page(
                        "/Account/ConfirmEmail",
                        pageHandler: null,
                         values: new { area = "Identity",
userId = userId, code = code, returnUrl = returnUrl },
                        protocol: Request.Scheme);

                                                        await
_emailSender.SendEmailAsync(Input.Email,   "Confirm   your
email",
                         $"Please confirm your account by
<a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking
here</a>.");

                                                          if
(_userManager.Options.SignIn.RequireConfirmedAccount)
                    {
                                                       return
RedirectToPage("RegisterConfirmation",   new   {   email   =
Input.Email, returnUrl = returnUrl });
                    }
                    else
```

otherwise the user is logged in and redirected to the `returnUrl

```
                    {
                                                        await
_signInManager.SignInAsync(user, isPersistent: false);
                        return LocalRedirect(returnUrl);
                    }
                }
                foreach (var error in result.Errors)
                {
                   ModelState.AddModelError(string.Empty,
error.Description);
                }
            }

              // If we got this far, something failed,
redisplay form
   return Page();
}
```

4

```
private Accounts CreateUser()
{
            try
            {
                                                       return
Activator.CreateInstance<Accounts>();
            }
            catch
            {
              throw new InvalidOperationException($"Can't
create an instance of '{nameof(Accounts)}'. " +
                    $"Ensure that '{nameof(Accounts)}' is
not an abstract class and has a parameterless constructor,
or alternatively " +
                        $"override the register page in
/Areas/Identity/Pages/Account/Register.cshtml");
            }
}
```

The "Accounts" object, which is used for user authentication, is created by this code in a new instance. The 'Activator.CreateInstanceT>' method, which instantiates the type supplied as a generic argument, is used to accomplish this. A "InvalidOperationException" is therefore issued if the instantiated type is not abstract and does not have a constructor with no parameters.

5

```
private IUserEmailStore<Accounts> GetEmailStore()
{
            if (!_userManager.SupportsUserEmail)
            {
                    throw new NotSupportedException("The
default UI requires a user store with email support.");
            }
            return (IUserEmailStore<Accounts>)_userStore;
}
```

This code gets the `IUserEmailStore<Accounts>` which is used to store the user emails. It first checks if the `_userManager` supports user emails by calling the `SupportsUserEmail` and if not it throws a `NotSupportedException`. It then casts the `_userStore` variable as an

		`IUserEmailStore<Accounts>` and returns it for use.

- *Login.cshtml.cs*

```
using System.ComponentModel.DataAnnotations;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using FPT_Book_Store.Constants;

namespace FPT_Book_Store.Areas.Identity.Pages.Account
{
    public class LoginModel : PageModel
    {
        private readonly SignInManager<Accounts> _signInManager;
        private readonly ILogger<LoginModel> _logger;
        private readonly UserManager<Accounts> _userManager;

        public LoginModel(SignInManager<Accounts> signInManager, ILogger<LoginModel> logger,
        UserManager<Accounts> userManager)
        {
            _signInManager = signInManager;
            _logger = logger;
            _userManager = userManager;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        /// <summary>
```

```
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public string ReturnUrl { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [TempData]
        public string ErrorMessage { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public class InputModel
        {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            public string Email { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Required]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Display(Name = "Remember me?")]
            public bool RememberMe { get; set; }
```

```
        }

        public async Task OnGetAsync(string returnUrl = null)
        {
            if (!string.IsNullOrEmpty(ErrorMessage))
            {
                ModelState.AddModelError(string.Empty, ErrorMessage);
            }

            returnUrl ??= Url.Content("~/");

            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

                                                        ExternalLogins        =        (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            ReturnUrl = returnUrl;
        }

        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");

                                                        ExternalLogins        =        (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            if (ModelState.IsValid)
            {

                var userEmail = await _userManager.FindByEmailAsync(Input.Email);

                if(userEmail != null){

                    if(userEmail.Account_Deleted != Status.Deleted.ToString()){

                            var result = await _signInManager.PasswordSignInAsync(Input.Email,
Input.Password, Input.RememberMe, lockoutOnFailure: false);

                        if (result.Succeeded)
                        {
                            _logger.LogInformation("User logged in.");
                            return LocalRedirect(returnUrl);
                        }
                        if (result.RequiresTwoFactor)
                        {
```

```
                                return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl,
RememberMe = Input.RememberMe });
                            }
                            if (result.IsLockedOut)
                            {
                                _logger.LogWarning("User account locked out.");
                                return RedirectToPage("./Lockout");
                            }
                            else
                            {
                                ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                                return Page();
                            }
                        }else{

                            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                            return Page();
                        }

                    }
            }
                    // This doesn't count login failures towards account lockout
                    // To enable password failures to trigger account lockout, set lockoutOnFailure:
true

                // If we got this far, something failed, redisplay form
                return Page();
            }
        }
    }
```

Table 12. Login explain table

No.	Action	Describe

1	`public async Task OnGetAsync(string returnUrl = null)`
	`{`
	`if (!string.IsNullOrEmpty(ErrorMessage))`
	`{`
	`ModelState.AddModelError(string.Empty, ErrorMessage);`
	`}`
	
	`returnUrl ??= Url.Content("~/");`
	
	`// Clear the existing external cookie to ensure a clean login process`
	`await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);`
		When a user is on the "register" page, this code manages their Get request. It first determines whether a "ErrorMessage" is present and, if so, adds it to the "ModelState". After that, it either sets the "returnUrl" to the URL specified by the method's parameter or, by

	``ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
	
	ReturnUrl = returnUrl;
	}
	}```
		default, to the URL "/". The user is then logged out of all external authentication methods after which a list of available methods is fetched and displayed in a drop-down menu. The 'ReturnUrl' is then set to the'returnUrl' that was previously supplied.

2	```public async Task<IActionResult> OnPostAsync(string returnUrl = null)
	{
	returnUrl ??= Url.Content("~/");
	
	ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
	
	if (ModelState.IsValid)
	{
	
	var userEmail = await _userManager.FindByEmailAsync(Input.Email);
	
	if(userEmail != null){
	
	if(userEmail.Account_Deleted != Status.Deleted.ToString()){
	
	var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);
	
	if (result.Succeeded)
	{
	_logger.LogInformation("User logged in.");
	return LocalRedirect(returnUrl);
	}
	if (result.RequiresTwoFactor)
	{
	return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });
	}```
		When a user logs in, this code is called in response to their Post request. A list of external authentication techniques is initially obtained. The validity of the model state is then verified. If so, it determines whether the user is already in the system. If they do, it attempts to log them in using the "PasswordSignInAsync" method; if successful, it directs them to the "ReturnUrl," else it returns the current page. The same page and an error are returned if the user is nonexistent. It redisplays the form if there is any other failure.

```
                        if (result.IsLockedOut)
                        {
                            _logger.LogWarning("User account locked
out.");
                            return RedirectToPage("./Lockout");
                        }
                        else
                        {
                            ModelState.AddModelError(string.Empty,
"Invalid login attempt.");
                            return Page();
                        }
                    }else{

                            ModelState.AddModelError(string.Empty,
"Invalid login attempt.");
                        return Page();
                    }

                }
        }
               // This doesn't count login failures towards account
lockout
                // To enable password failures to trigger account
lockout, set lockoutOnFailure: true

            // If we got this far, something failed, redisplay form
            return Page();
}
```

- ChangePassword.cshtml.cs

```
using System.ComponentModel.DataAnnotations;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace FPT_Book_Store.Areas.Identity.Pages.Account.Manage
{
    public class ChangePasswordModel : PageModel
    {
        private readonly UserManager<Accounts> _userManager;
        private readonly SignInManager<Accounts> _signInManager;
        private readonly ILogger<ChangePasswordModel> _logger;

        public ChangePasswordModel(
```

```
            UserManager<Accounts> userManager,
            SignInManager<Accounts> signInManager,
            ILogger<ChangePasswordModel> logger)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [TempData]
        public string StatusMessage { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public class InputModel
        {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Required]
            [DataType(DataType.Password)]
            [Display(Name = "Current password")]
            public string OldPassword { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
```

```
            /// </summary>
            [Required]
              [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1}
characters long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
            [Display(Name = "New password")]
            public string NewPassword { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is
not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [DataType(DataType.Password)]
            [Display(Name = "Confirm new password")]
            [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do
not match.")]
            public string ConfirmPassword { get; set; }
        }

        public async Task<IActionResult> OnGetAsync()
        {
            var user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
               return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
            }

            var hasPassword = await _userManager.HasPasswordAsync(user);
            if (!hasPassword)
            {
                return RedirectToPage("./SetPassword");
            }

            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            var user = await _userManager.GetUserAsync(User);
            if (user == null)
```

```
                {
                    return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
                }

                    var changePasswordResult = await _userManager.ChangePasswordAsync(user,
Input.OldPassword, Input.NewPassword);
                if (!changePasswordResult.Succeeded)
                {
                    foreach (var error in changePasswordResult.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                    return Page();
                }

                await _signInManager.RefreshSignInAsync(user);
                _logger.LogInformation("User changed their password successfully.");
                StatusMessage = "Your password has been changed.";
                return RedirectToPage();
            }
        }
    }
```

Table 13. ChangePassword explain table

No.	Action	Describe

1	`public async Task<IActionResult> OnGetAsync()`
	`{`
	`var user = await _userManager.GetUserAsync(User);`
	`if (user == null)`
	`{`
	`return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");`
	`}`
	`var hasPassword = await _userManager.HasPasswordAsync(user);`
	`if (!hasPassword)`
	`{`
	`return RedirectToPage("./SetPassword");`
	`}`
	`return Page();`
	`}`
		When a user tries to access a page that needs a password configured, this code is executed on their Get request. It begins by obtaining the currently logged-in user; if the user is not found, a "NotFound" error is returned. Next, if the user doesn't already have a password set, it takes them to the "SetPassword" page; otherwise, it displays the same page.

2	`public async Task<IActionResult> OnPostAsync()`
	`{`
	`if (!ModelState.IsValid)`
	`{`
		When a user posts a request to update their password, this code is run in response. Before

```
                return Page();
            }

            var user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
                return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
            }

                var changePasswordResult = await _userManager.ChangePasswordAsync(user, Input.OldPassword, Input.NewPassword);
            if (!changePasswordResult.Succeeded)
            {
                foreach (var error in changePasswordResult.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
                return Page();
            }

            await _signInManager.RefreshSignInAsync(user);
                _logger.LogInformation("User changed their password successfully.");
            StatusMessage = "Your password has been changed.";

    return RedirectToPage();
}
```

attempting to retrieve the currently logged-in user, it first verifies that the values provided in the request are legitimate. If the user is not found, a "NotFound" error is returned. The user's password is then changed by calling the "ChangePasswordAsync" method; if it is successful, a "Your password has been changed." status message is set; otherwise, the error messages are added to the model state. The user is then redirected back to the original page after refreshing the sign-in process.

- Index.cshtml.cs (Profile updating)

```
using System.ComponentModel.DataAnnotations;
using FPT_Book_Store.Models;
using FPT_Book_Store.Services;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace FPT_Book_Store.Areas.Identity.Pages.Account.Manage
{
    public class IndexModel : PageModel
    {
        private readonly UserManager<Accounts> _userManager;
        private readonly SignInManager<Accounts> _signInManager;
```

```
        private readonly IFileService _fileService;

        public IndexModel(
            UserManager<Accounts> userManager,
            SignInManager<Accounts> signInManager,
            IFileService fileService)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            this._fileService = fileService;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public string Username { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [TempData]
        public string StatusMessage { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not
intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        public class InputModel
        {
            [Display(Name = "Email")]
            public string Account_Email {get; set; }

            [Display(Name = "Address")]
```

```
            [Required(ErrorMessage = "Please, enter the address!")]
            [StringLength(100)]
            public string Account_Address { get; set; }

            public string Account_Image { get; set; }
            public IFormFile Account_Picture { get; set; }
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI infrastructure and is
not intended to be used
            ///     directly from your code. This API may change or be removed in future releases.
            /// </summary>
            [Phone]
            [Display(Name = "Phone number")]
            public string PhoneNumber { get; set; }
        }

        private async Task LoadAsync(Accounts user)
        {
            var userName = await _userManager.GetUserNameAsync(user);
            var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
            var email = await _userManager.GetEmailAsync(user);

            Username = userName;

            Input = new InputModel
            {
                Account_Email = email,
                PhoneNumber = phoneNumber,
                Account_Address = user.Account_Address,
                Account_Image = user.Account_Image
            };
        }

        public async Task<IActionResult> OnGetAsync()
        {
            var user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
               return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
            }

            await LoadAsync(user);
            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
```

```
        {
            var user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
               return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");
            }

            if (!ModelState.IsValid)
            {
                await LoadAsync(user);
                return Page();
            }

            var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
            if (Input.PhoneNumber != phoneNumber)
            {
                        var  setPhoneResult  =  await  _userManager.SetPhoneNumberAsync(user,
Input.PhoneNumber);
                if (!setPhoneResult.Succeeded)
                {
                    StatusMessage = "Unexpected error when trying to set phone number.";
                    return RedirectToPage();
                }
            }

            if (Input.Account_Address != user.Account_Address)
            {
                user.Account_Address = Input.Account_Address;
                await _userManager.UpdateAsync(user);
            }

            if (Input.Account_Picture != null)
            {
                var result = _fileService.SaveImage(Input.Account_Picture);
                if (result.Item1 == 1)
                {
                    var oldImage = user.Account_Image;
                    user.Account_Image = result.Item2;
                    await _userManager.UpdateAsync(user);
                    var deleteResult = _fileService.DeleteImage(oldImage);
                }
            }

            await _signInManager.RefreshSignInAsync(user);
            StatusMessage = "Your profile has been updated";
            return RedirectToPage();
```

```
            }
        }
    }
```

Table 14. Index of Update Profile explain table

No.	Action	Describe

1	`private async Task LoadAsync(Accounts user)`
	`{`
	`var userName = await _userManager.GetUserNameAsync(user);`
	`var phoneNumber = await _userManager.GetPhoneNumberAsync(user);`
	`var email = await _userManager.GetEmailAsync(user);`
	
	`Username = userName;`
	
	`Input = new InputModel`
	`{`
	`Account_Email = email,`
	`PhoneNumber = phoneNumber,`
	`Account_Address = user.Account_Address,`
	`Account_Image = user.Account_Image`
	`};`
	`}`
		This code is used to load the user's profile information from the database into the view. It starts by using the `GetUserNameAsync` method to get the user's username, then `GetPhoneNumberAsync` and `GetEmailAsync` to get the user's phone number and email. The information from the database is then set to the `Input` model, which includes the user's email, phone number, address and image. Finally, the `Username` is set to the user's username.

2	`public async Task<IActionResult> OnGetAsync()`
	`{`
	`var user = await _userManager.GetUserAsync(User);`
	`if (user == null)`
	`{`
	`return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");`
	`}`
	
	`await LoadAsync(user);`
	`return Page();`
	`}`
		The user's profile in the database is checked using this code. Initially, it retrieves the user from the database using the 'GetUserAsync' method. Unable to load user with ID '_userManager.GetUserId(User)' is returned if the user cannot be located. The 'LoadAsync' method is invoked to load the user's profile data, though, if the user is discovered. The page is then returned at the end.

3	`public async Task<IActionResult> OnPostAsync()`
	`{`
	`var user = await _userManager.GetUserAsync(User);`
	`if (user == null)`
	`{`
	`return NotFound($"Unable to load user with ID '{_userManager.GetUserId(User)}'.");`
		This code is used to update the user profile in the database. It first checks if the values provided in the request are valid, then it attempts to get the current logged-in user, and if the user doesn't exist, it

```
            }

            if (!ModelState.IsValid)
            {
                await LoadAsync(user);
                return Page();
            }

                            var    phoneNumber    =    await
_userManager.GetPhoneNumberAsync(user);
            if (Input.PhoneNumber != phoneNumber)
            {
                            var   setPhoneResult   =   await
_userManager.SetPhoneNumberAsync(user, Input.PhoneNumber);
                if (!setPhoneResult.Succeeded)
                {
                      StatusMessage = "Unexpected error when
trying to set phone number.";
                    return RedirectToPage();
                }
            }

                        if    (Input.Account_Address    !=
user.Account_Address)
            {
               user.Account_Address = Input.Account_Address;
                await _userManager.UpdateAsync(user);
            }

            if (Input.Account_Picture != null)
            {
                                    var    result    =
_fileService.SaveImage(Input.Account_Picture);
                if (result.Item1 == 1)
                {
                    var oldImage = user.Account_Image;
                    user.Account_Image = result.Item2;
                    await _userManager.UpdateAsync(user);
                                    var   deleteResult   =
_fileService.DeleteImage(oldImage);
                }
            }

            await _signInManager.RefreshSignInAsync(user);
            StatusMessage = "Your profile has been updated";
            return RedirectToPage();
```

returns a "NotFound" error. If the user is found, it checks if the new phone number is different from the initial one, then it sets the new one. Then, it checks if the new address is different from the current one, and if so, it updates the user's address information. It then checks if the image provided by the user is different than the current, and if it is, it saves the new image, changes it in the database, and deletes the old one. Finally, it refreshes the sign in and redirects the user back to the page with a status message of "Your profile has been updated".

```
}
```

- Logout.cshtml.cs

```
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace FPT_Book_Store.Areas.Identity.Pages.Account
{
    public class LogoutModel : PageModel
    {
        private readonly SignInManager<Accounts> _signInManager;
        private readonly ILogger<LogoutModel> _logger;

        public LogoutModel(SignInManager<Accounts> signInManager, ILogger<LogoutModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        public async Task<IActionResult> OnPost(string returnUrl = null)
        {
            await _signInManager.SignOutAsync();
            _logger.LogInformation("User logged out.");
            if (returnUrl != null)
            {
                return LocalRedirect(returnUrl);
            }
            else
            {
                // This needs to be a redirect so that the browser performs a new
                // request and the identity for the user gets updated.
                return RedirectToPage();
            }
        }
    }
}
```

Table 15. Logout explain table

No.	Action	Describe

1	`public LogoutModel(SignInManager<Accounts> signInManager, ILogger<LogoutModel> logger)`
	`{`
	`_signInManager = signInManager;`
	`_logger = logger;`
	`}`
		The LogoutModel class' constructor, which aids in logging users out, is written in this code. SignInManager and

		ILogger are the two parameters that it accepts. ILogger is used for logging and tracking failures, whereas SignInManager is used to sign in and out of users. The local variables are then set to the parameters supplied by the constructor.
2		Users' logouts are handled by this code. The returnUrl parameter, which is optional, tells the "OnPost" function where to send the user when they log out. Initially, the '_signInManager' variable's 'SignOutAsync' method is invoked. Using this technique invalidates the authentication mechanism and logs the user out of their account. After that, a log message indicating the user's logout is posted to the "_logger" log file. The function then determines whether'returnUrl' has been supplied. If so, it redirects the user to the specified URL using the 'LocalRedirect' method; else, it uses the 'RedirectToPage' method to send the user back to the original page.

```
public async Task<IActionResult> OnPost(string
returnUrl = null)
{
            await _signInManager.SignOutAsync();
            _logger.LogInformation("User logged out.");
            if (returnUrl != null)
            {
                return LocalRedirect(returnUrl);
            }
            else
            {
                // This needs to be a redirect so that the
browser performs a new
               // request and the identity for the user gets
updated.
                return RedirectToPage();
            }
}
```

- AccountController.cs (Areas/Admin)

```
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace FPT_Book_Store.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Route("Admin/[controller]/[action]")]
    [Authorize(Roles = "Admin")]
    public class AccountController : Controller
    {
        private readonly ApplicationDbContext _db;

        private readonly UserManager<Accounts> _userManager;

        public AccountController(ApplicationDbContext db,  UserManager<Accounts> userManager)
        {
            _db = db;
            _userManager = userManager;
        }

        public async Task<IActionResult> Index()
        {

                                    List<Accounts>     account     =     (List<Accounts>)     await
_userManager.GetUsersInRoleAsync("Owner");

                          List<Accounts>   existingAccount   =   account.OrderByDescending(a   =>
a.Account_Deleted).ThenBy(a => a.UserName).ToList();

            return View(existingAccount);
        }

        public async Task<IActionResult> Delete(string id)
        {

            Accounts user = await _userManager.FindByIdAsync(id);

            if (user == null)
            {

                TempData["account-error-message"] = "Error! Account cannot be deleted!";

                return RedirectToAction("Index");
            }

```

```
                user.Account_Deleted = Status.Deleted.ToString();

                _db.Accounts.Update(user);

                await _db.SaveChangesAsync();

                 TempData["account-message"] = "Account " + user.UserName + " has been successfully
    deleted!";

                return RedirectToAction("Index");
            }

            public async Task<IActionResult> Activate(string id)
            {

                Accounts user = await _userManager.FindByIdAsync(id);

                if (user == null)
                {

                    TempData["account-error-message"] = "Error! Account cannot be activated!";

                    return RedirectToAction("Index");
                }

                user.Account_Deleted = Status.Existing.ToString();

                _db.Accounts.Update(user);

                await _db.SaveChangesAsync();

                 TempData["account-message"] = "Account " + user.UserName + " has been successfully
    activated!";

                return RedirectToAction("Index");
            }


        }
    }
```

Table 16. AccountController (Areas/Admin) explain table

No.	Action	Describe

1	`public async Task<IActionResult> Index()`
	`{`
	`List<Accounts> account = (List<Accounts>) await _userManager.GetUsersInRoleAsync("Owner");`
	`List<Accounts> existingAccount = account.OrderByDescending(a => a.Account_Deleted).ThenBy(a => a.UserName).ToList();`
	`return View(existingAccount);`
	`}`
		A list of user accounts with the "Owner" role is generated by this code. A list of users who are asynchronously assigned to the "Owner" role is returned by the line '_userManager.GetUsersInRoleAsync("Owner")' . The list is sorted using the "OrderByDescending" and "ThenBy" methods, first in ascending order by the "UserName" property and then in descending order by the "Account Deleted" property. The sorted list is transformed into a "List" object using the "ToList()" function so that it can be supplied to the "View" method. The list of user accounts is then shown in the specified order.

2	`public async Task<IActionResult> Delete(string id)`
	`{`
	`Accounts user = await _userManager.FindByIdAsync(id);`
	`if (user == null)`
	`{`
	`TempData["account-error-message"] = "Error! Account cannot be deleted!";`
	`return RedirectToAction("Index");`
	`}`
	`user.Account_Deleted = Status.Deleted.ToString();`
	`_db.Accounts.Update(user);`
	`await _db.SaveChangesAsync();`
	`TempData["account-message"] = "Account " + user.UserName + " has been successfully deleted!";`
	`return RedirectToAction("Index");`
		This code deletes the account with the specified ID. First, the `FindByIdAsync` method is used to find the user with the given ID. If the account is not found, an error message is stored in `TempData` and the user is redirected back to the `Index` action. Otherwise, the `Account_Deleted` property is set to the `Status.Deleted` value, and then the account is updated in the database. Finally, the changes are saved to the database, and a success message is stored in 'TempData'. The user is then redirected back to the `Index` action.

	`}`	

3	`public async Task<IActionResult> Activate(string id)`
	`{`
	
	`Accounts user = await _userManager.FindByIdAsync(id);`
	
	`if (user == null)`
	`{`
	
	`TempData["account-error-message"] = "Error! Account cannot be activated!";`
	
	`return RedirectToAction("Index");`
	`}`
	
	`user.Account_Deleted = Status.Existing.ToString();`
	
	`_db.Accounts.Update(user);`
	
	`await _db.SaveChangesAsync();`
	
	`TempData["account-message"] = "Account " + user.UserName + " has been successfully activated!";`
	
	`return RedirectToAction("Index");`
	`}`
		This code activates the account with the specified ID. First, the "FindByIdAsync" method is used to find the user with the given ID. If the account is not found, an error message is stored in `TempData` and the user is redirected back to the `Index` action. Otherwise, the `Account_Deleted` property is set to the `Status.Existing` value, and the account is updated in the database. Finally, the changes are saved to the database, and a success message is stored in "TempData." The user is then redirected back to the `Index` action.

- CategoryController.cs (Areas/Admin)

```
using FPT_Book_Store.Data;
using FPT_Book_Store.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FPT_Book_Store.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Route("Admin/[controller]/[action]")]
    [Authorize(Roles = "Admin")]
    public class CategoryController : Controller
    {
        private readonly ApplicationDbContext _db;

        public CategoryController(ApplicationDbContext db)
```

```
        {
            _db = db;
        }

        public IActionResult ShowCategory()
        {
            IEnumerable<Category> ds = _db.Categories.ToList();
            return View(ds);
        }

        public IActionResult Confirm(int id)
        {
            IEnumerable<Category> ds = _db.Categories.Where(c => c.Category_ID == id).ToList();

            if(ds.Count() > 0){
                foreach(var item in ds){
                    item.Category_Status = Constants.Status.Approved.ToString();
                    _db.Categories.Update(item);
                    _db.SaveChanges();
                    break;
                }
            }
            return RedirectToAction("ShowCategory");
        }


    }
}
```

Table 17. CategoryController (Areas/Admin) explain table

No.	Action	Describe

1	`public IActionResult ShowCategory()`
	`{`
	`IEnumerable<Category> ds = _db.Categories.ToList();`
	`return View(ds);`
	`}`
		This code accesses the database to get a list of all Category elements, which is then shown in a view. The first step is to retrieve every item from the database and place it in an IEnumerable object using the "ToList()" method. The user is subsequently presented with an HTML-marked version of the list of Category components that was supplied to the View.

2	`public IActionResult Confirm(int id)`
	`{`
	`IEnumerable<Category> ds = _db.Categories.Where(c => c.Category_ID == id).ToList();`
	
	`if(ds.Count() > 0){`
	`foreach(var item in ds){`
	`item.Category_Status = Constants.Status.Approved.ToString();`
	`_db.Categories.Update(item);`
	`_db.SaveChanges();`
	`break;`
	`}`
	`}`
	`return RedirectToAction("ShowCategory");`
	`}`
		This code validates a Category for a particular ID. The Categories table is first searched using the "Where" technique to locate a Category matching the given ID. The "Category Status" field is set to "Approved" and the modifications are stored to the database if a matching Category is discovered. The user is finally forwarded to the "ShowCategory" action.

- DashboardController.cs (Areas/Admin)

```
using FPT_Book_Store.Areas.Owner.Models;
using FPT_Book_Store.Constants;
using FPT_Book_Store.Data;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FPT_Book_Store.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Route("Admin/[controller]/[action]")]
    [Authorize(Roles = "Admin")]
    public class DashboardController : Controller
    {
        private readonly ApplicationDbContext _db;

        public DashboardController(ApplicationDbContext db)
        {
            _db = db;
        }
        public IActionResult Index()
        {
            IEnumerable<Statistic> list = _db.OrdersDetail.
                Where(o => o.Order.Order_Status == Status.Received.ToString()).GroupBy(o => o.Book.Book_ID).
            Select(t => new Statistic
            {
                Book_Image = t.First().Book.Book_Image,
                Book_ID = t.Key,
                Book_Name = t.First().Book.Book_Name,
```

```
                    Category_Type = t.First().Book.Category.Category_Type,
                    Publisher_Name = t.First().Book.Publisher.Publisher_Name,
                    Book_Quantity = t.First().Book.Book_Quantity,
                    OrdersDetail_Quantity = t.Sum(o => o.OrderDetail_Quantity)

                }).ToList();

                return View(list);
            }
        }
    }
```

Table 18. DashboardController (Areas/Admin) explain table

No.	Action	Describe

1	`public IActionResult Index()`
	`{`
	`IEnumerable<Statistic> list = _db.OrdersDetail.Where(o => o.Order.Order_Status == Status.Received.ToString()).GroupBy(o => o.Book.Book_ID).Select(t => new Statistic`
	`{`
	`Book_Image = t.First().Book.Book_Image,`
	`Book_ID = t.Key,`
	`Book_Name = t.First().Book.Book_Name,`
	`Category_Type = t.First().Book.Category.Category_Type,`
	`Publisher_Name = t.First().Book.Publisher.Publisher_Name,`
	`Book_Quantity = t.First().Book.Book_Quantity,`
	`OrdersDetail_Quantity = t.Sum(o => o.OrderDetail_Quantity)`
	`}).ToList();`
	`return View(list);`
	`}`
		This code pulls a collection of statistics for the Index view from the OrdersDetail table. Then, the OrdersDetail database is filtered using the "Where" method to only return orders having a "Order Status" of "Received." In order to construct data with a "Book Image," "Book ID," "Book Name," "Category Type," "Publisher Name," "Book Quantity," and "OrderDetail Quantity" attribute, relevant tables are then joined using the "GroupBy" and "Select" methods. A single "list" made up of all the data is created, which is then returned and sent to the View for display.

3.3. GitHub repository

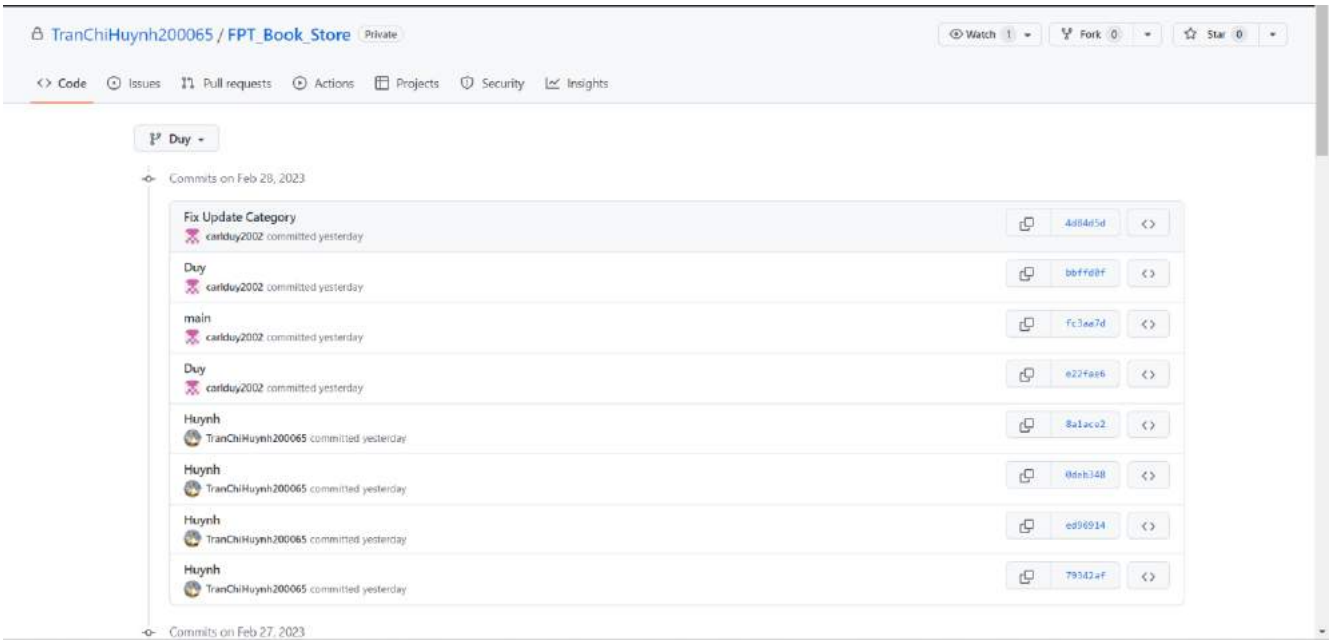


Figure 84. GitHub repository

GitHub Link: https://github.com/TranChiHuynh200065/FPT_Book_Store/commits/Duy

3.4. Deployment Result

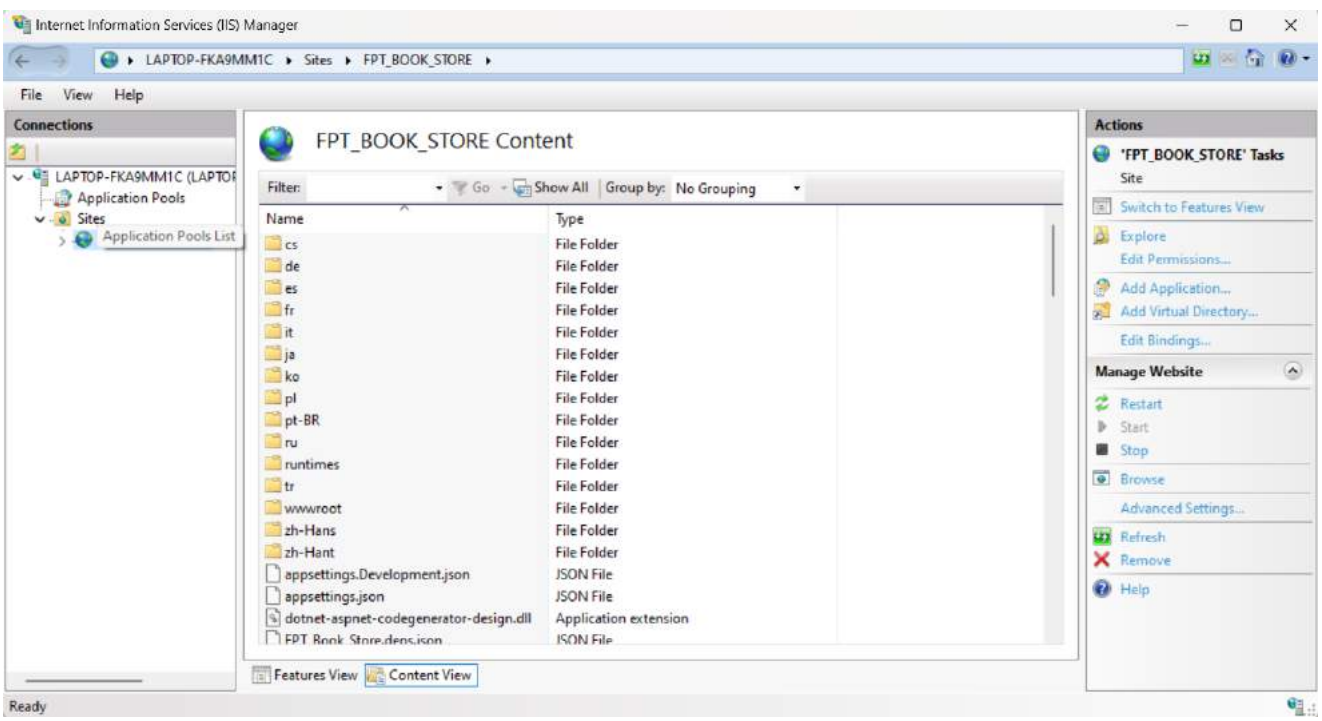
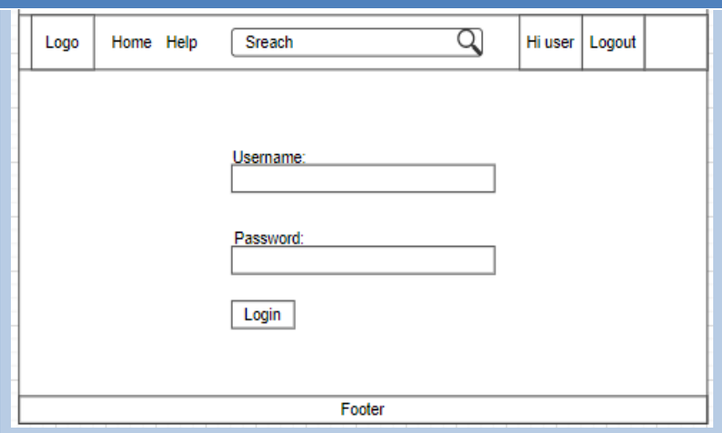
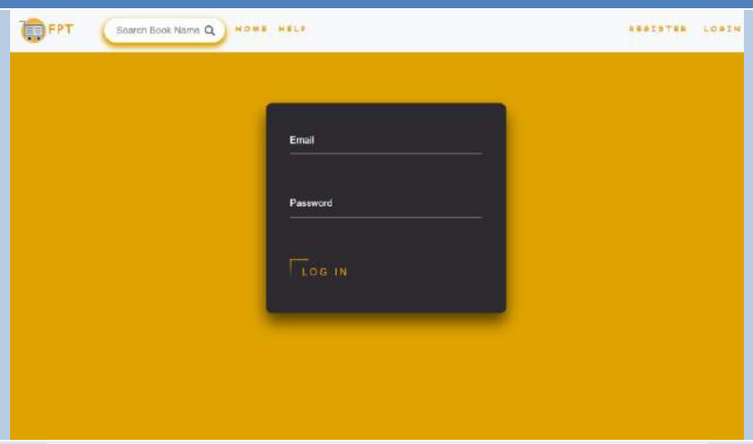
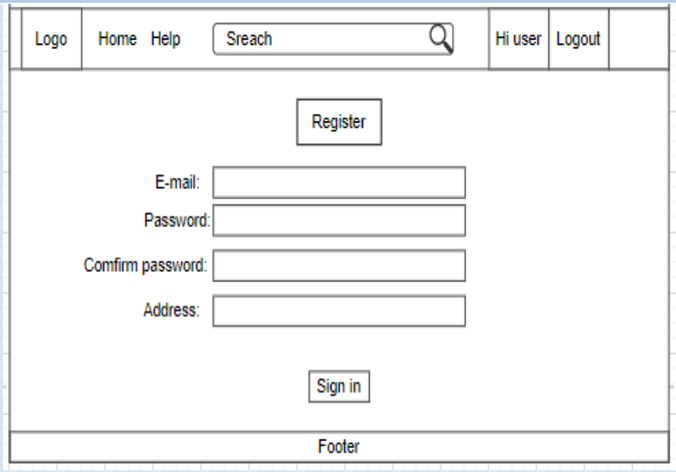


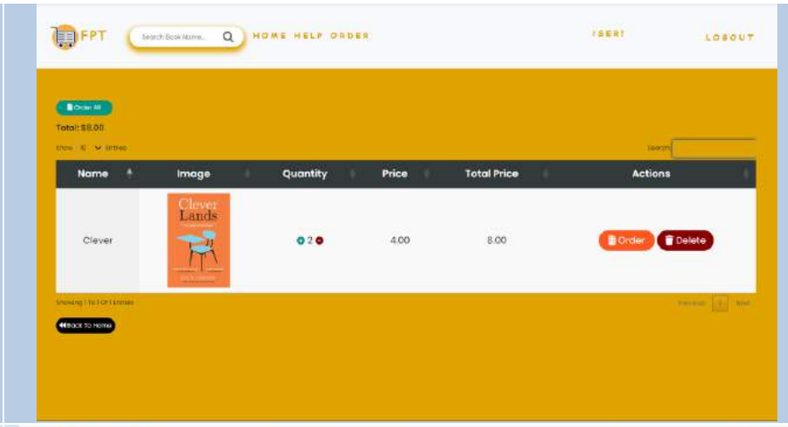
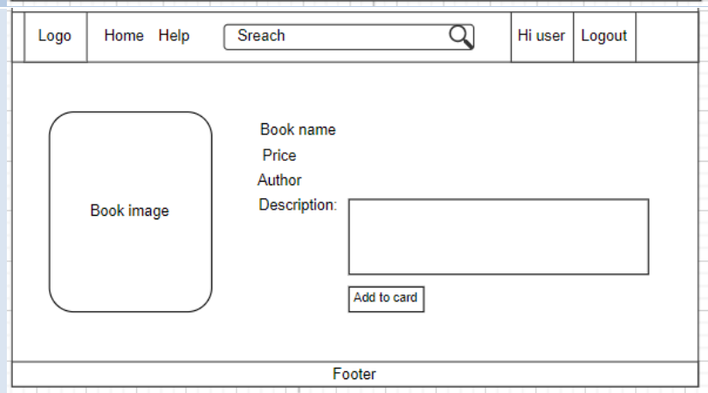
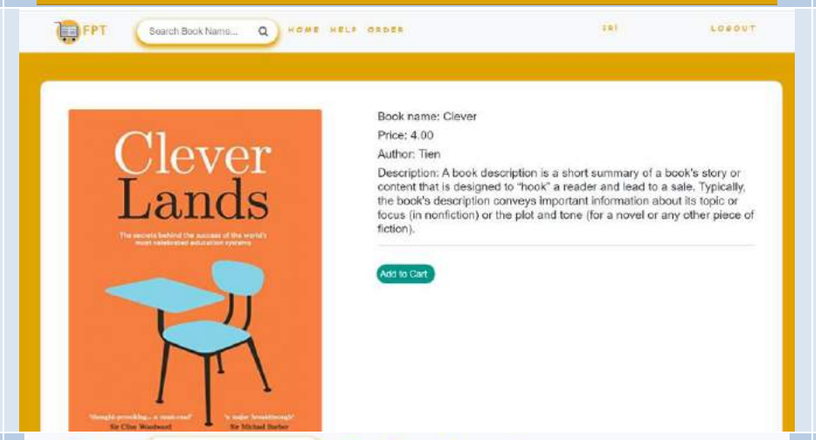
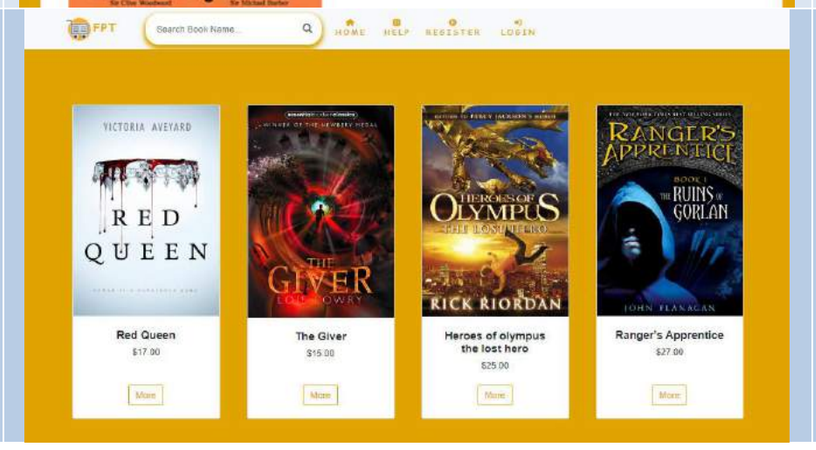
Figure 85. Deployment Result

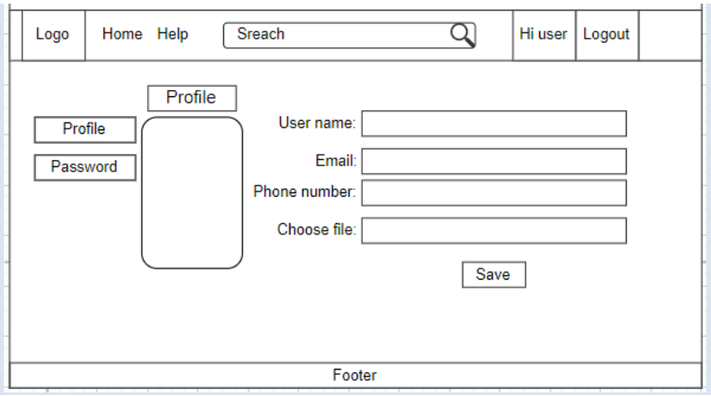
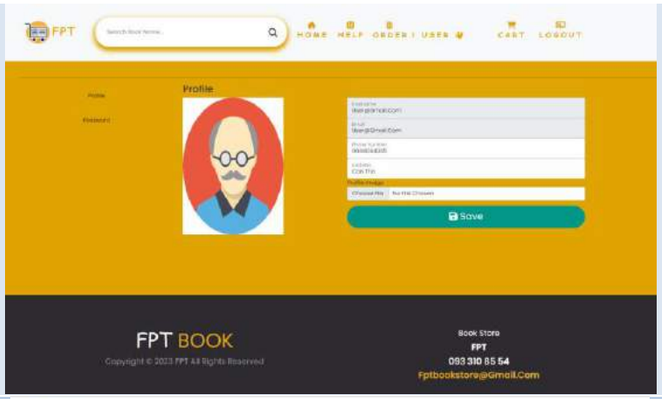
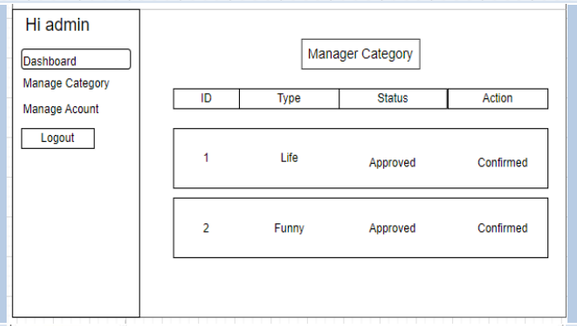
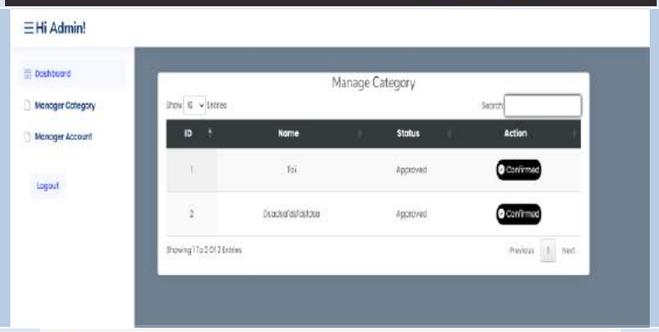
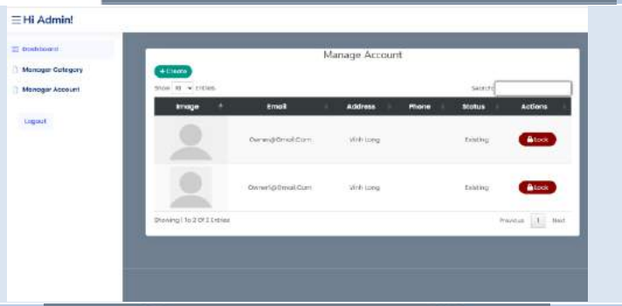
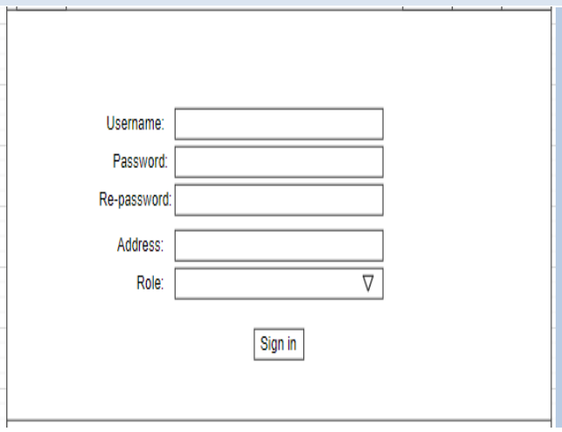
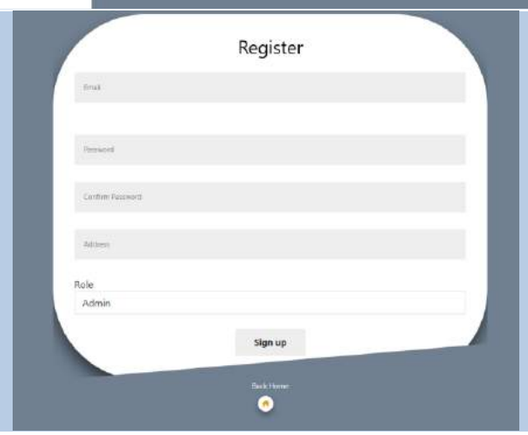
III. Application Evaluation

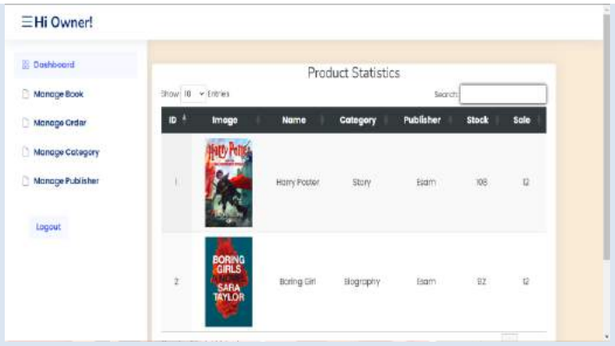
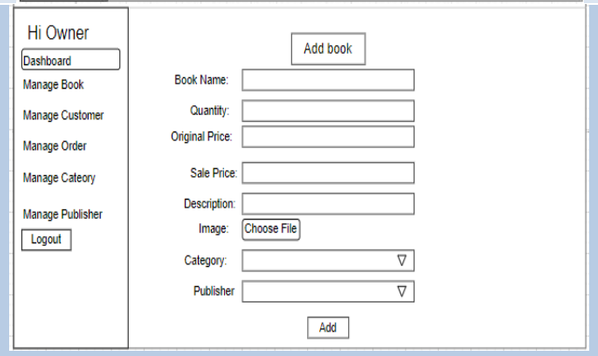
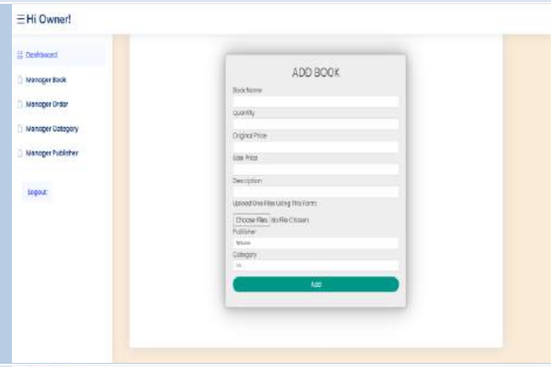
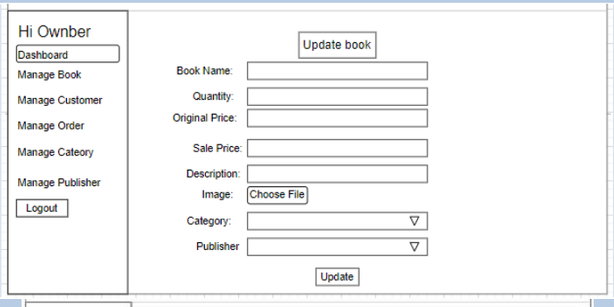
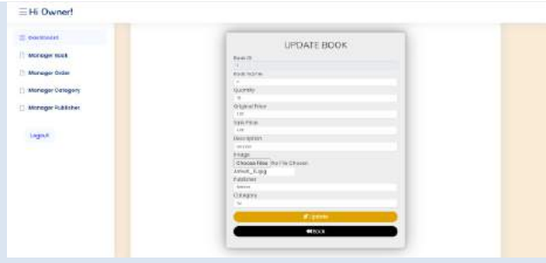
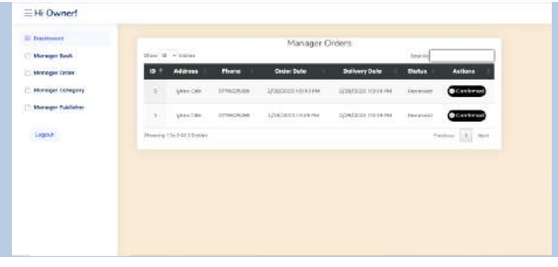
1. Mock-up

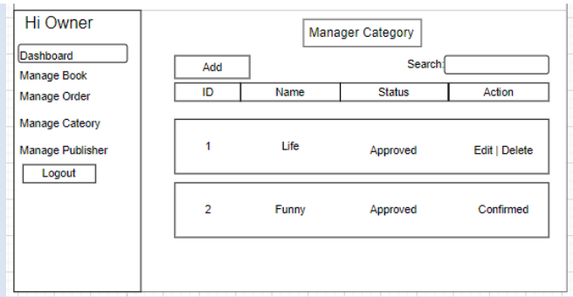
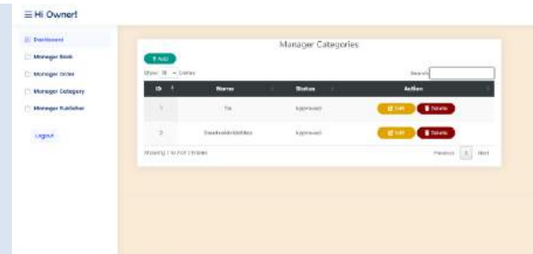
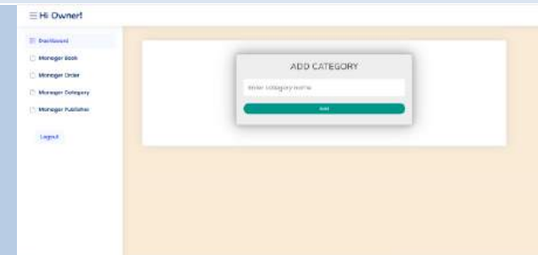
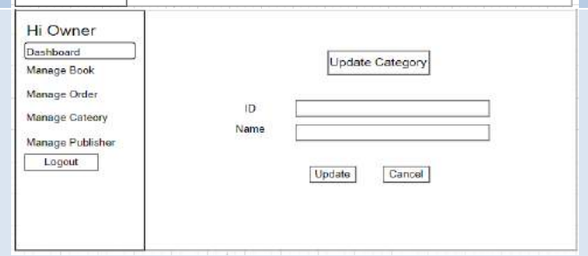
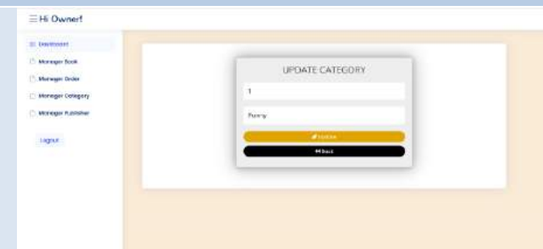
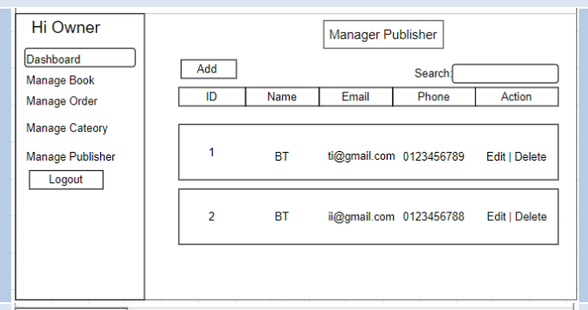
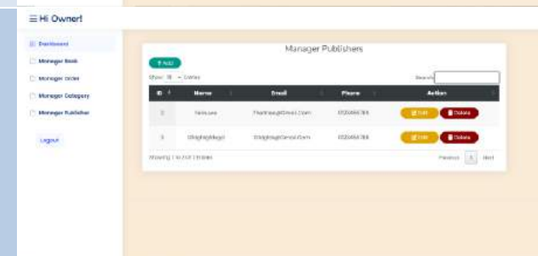
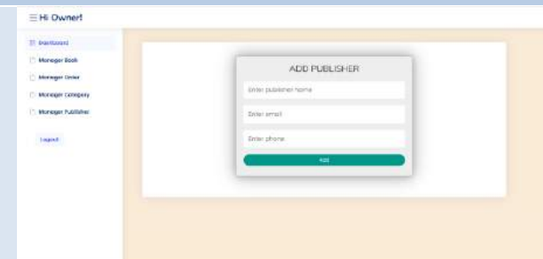
Table 19. Mock-up table

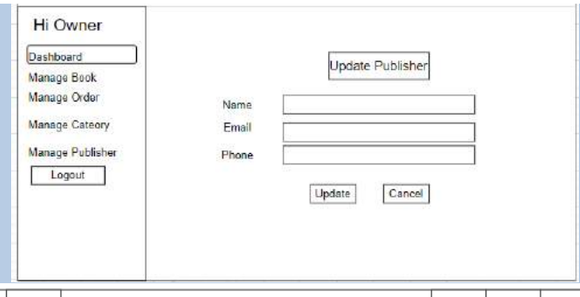
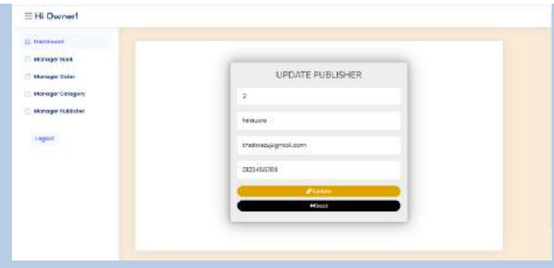
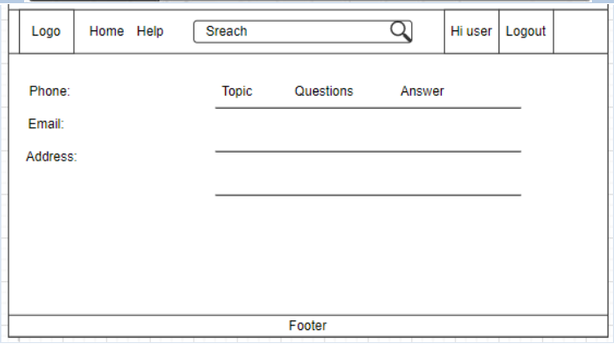
No.	Use case	Describe	Mock-up	Result	Complete (%)	Status
1	Log in	The website's login page.			100%	Finish
2	Register of user	The website's register of user page.			100%	Finish

3	View Cart	The website's view cart page			100%	Finish
4	View Product Details	The website's product details page			100%	Finish
5	View home page	The website's home page			100%	Finish

6	View Profile	The website's view profile page			100%	Finish
7	Viewing manage category	The website's manage category page			100%	Finish
8	Viewing manage account	The website's manage account page			100%	Finish
9	Register of admin	The website's register of admin page.			100%	Finish

10	Viewing manage book	The website's manage book page				100%	Finish
11	Viewing add book	The website's add book page				100%	Finish
12	Viewing edit book	The website's edit book page				100%	Finish
13	Viewing manage order	The website's manage order page				100%	Finish

14	Viewing manage category	The website's manage category page						100%	Finish
15	Viewing add category	The website's add category page						100%	Finish
16	Viewing edit category	The website's edit category page						100%	Finish
17	Viewing manage publisher	The website's manage publisher page						100%	Finish
18	Viewing add publisher	The website's add publisher page						100%	Finish

19	Viewing edit publisher	The website's edit publisher page				100%	Finish
20	Viewing help	The website's help page				100%	Finish

2. Analyse the factors that influence the performance of a business application

2.1. Register

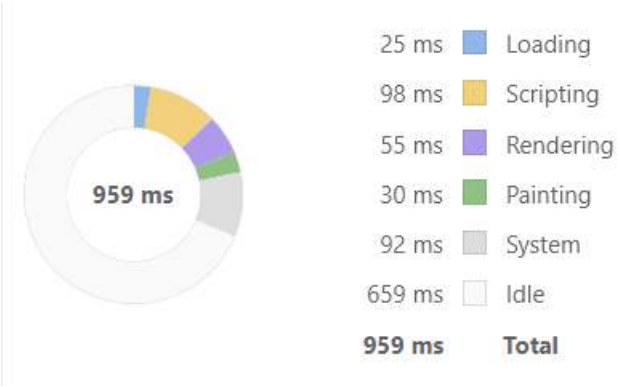


Figure 86. Register performance

In the register function the performance of the application takes all time is 959ms with the Loading is 25ms, Scripting 98ms, Rendering is 55ms, Painting is 30ms, System is 92ms, and Idle is 659ms.

2.2. Login

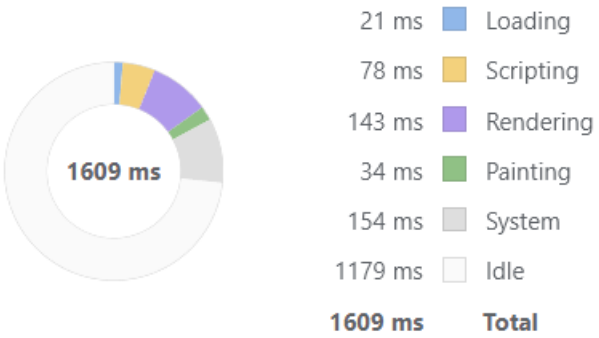


Figure 87. Login performance

In the login function the performance of the application takes all time is 1609ms with the Loading is 21ms, Scripting 78ms, Rendering is 143ms, Painting is 34ms, System is 154ms, and Idle is 1179ms.

2.3. Add New Book

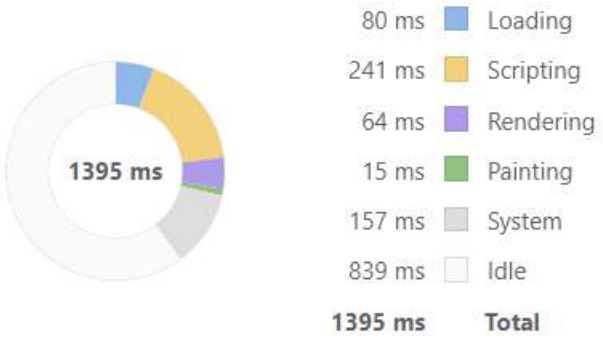


Figure 88. Add new Book performance

In the adding a new book function the performance of the application takes all time is 1395ms with the Loading is 80ms, Scripting 241ms, Rendering is 64ms, Painting is 15ms, System is 157ms, and Idle is 839ms.

2.4. Update Profile

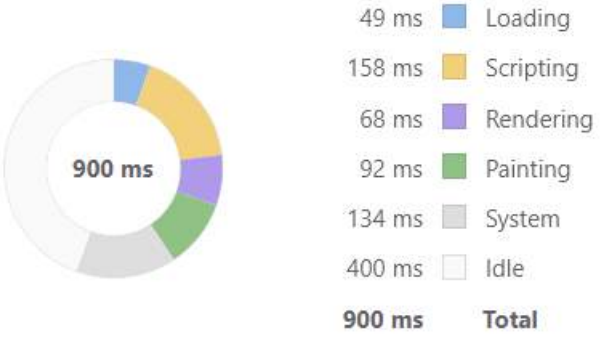


Figure 89. Update Profile performance

In the updating profile function the performance of the application takes all time is 900ms with the Loading is 49ms, Scripting 158ms, Rendering is 68ms, Painting is 92ms, System is 134ms, and Idle is 400ms.

2.5. Conclusion

The non-function requirements are required in the assignment 1:

- **Operational:** The system should be able to work on any Web browser
- **Security:** Just admin can access and manage account, the system must have measures to protect against viruses, and functions must be allocated to each appropriate role
- **Performance:** The interaction between the user and the system should not exceed 3 seconds and the system supports concurrent users

After deploying the application to the IIS server, I measure the application's performance through four basic functions such as Register, Login, Add new book, and Update profile. Through the measure processing I can calculate the time that the application took to perform a function from some main operations such as Loading, Painting, and Rendering. From these measurements data, I can compare with the requirement non-function in assignment 1. The application met requirements about the time to perform a feature does not exceed more than 3 seconds. Besides, the application can run in some browsers such as Opera, Chrome, and Microsoft Edge. In addition, about the authorization for each account like admin, owner, and user is also authorize clearly for each account. To sum up, our application met the non-function requirements through measure processing.

3. Critically evaluate the strengths and weaknesses of the business application

In this project, we start in the 3/1/2023 and finished in 3/3/2023. After building the application for the FPT company about the FPT BOOK STORE website, we tested some business function of the application. Besides, we also tested the performance of the application whether meet the given requirements in the initiation or not? From that we evaluated the application that we built and developed. With this application we saw some of the following strengths and weaknesses.

3.1. Strengths

After building the website, I saw the website has some of the following strengths:

- A wide range of functions for customers, including adding items to their cart, viewing orders, and canceling orders.
- Ability for owners to add, edit, delete, and search for books, publishers, and categories.
- Admin has the right to add, lock, and unlock the owner account and confirm categories.
- The website's interface is friendly with the user.
- Processing operations, not more than 3 seconds

3.2. Weaknesses

After building the website, I saw the website has some of the following weaknesses:

- No options for users to make recommendations or give ratings on purchased items.
- Customer reviews are not visible to potential customers browsing the website.
- No options for customers to save their payment information to speed up future orders.

References

GeeksforGeeks, 2019. *www.geeksforgeeks.org.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-c-sharp/
[Accessed 5 Februay 2023].

Goyal, Y., 2022. *www.educba.com.* [Online]
Available at: https://www.educba.com/advantages-of-html/
[Accessed 4 Februay 2022].

Kinsta, 2022. *kinsta.com.* [Online]
Available at: https://kinsta.com/knowledgebase/what-is-github/
[Accessed 5 Februay 2023].

Morris, W., 2022. *elegantthemes.* [Online]
Available at: https://www.elegantthemes.com/blog/wordpress/microsoft-iis?utm_source=Blog&utm_medium=Manual%20Divi%20Targets&utm_campaign=Google%20Search&retargeting=off&gclid=Cj0KCQiA54KfBhCKARIsAJzSrdpcsrMDOqaVMgckOo_tRgyAbqvdBRdq3UNps83RMqTnVYmEmgHqXXUaAtOoEALw_wcB

Reddy, A., 2019. *www.tutorialspoint.com.* [Online]
Available at: https://www.tutorialspoint.com/What-are-the-advantages-of-CSS
[Accessed 5 Februay 2023].

Shanmugam, B., 2022. *www.bdrsuite.com.* [Online]
Available at: https://www.bdrsuite.com/blog/system-databases-sql-server/
[Accessed 5 Februay 2023].

tutorialsteacher, 2023. *tutorialsteacher.com.* [Online]
Available at: https://www.tutorialsteacher.com/core/dotnet-core

Uzayr, S. b., 2021. *link.springer.com.* [Online]
Available at: https://link.springer.com/chapter/10.1007/978-1-4842-7344-9_1