WOLKITE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

TITLE: ANDROID-BASED BOOKSTORE AND BOOK EXCHANGE PLATFORM

Prepared by:

Name	ID
1. Yared Tekileselassie	CIR/292/11
2. Metasebiya Gemechu	CIR/070/11
3. Tolawak Yonas	CIR/093/11

Under the guidance of Mr. Tesfahun Nurre (MSc.)

Wolkite University, Wolkite, Ethiopia

July 15, 2022

COLLEGE OF COMPUTING AND INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

PROJECT TITLE: ANDROID-BASED BOOKSTORE AND PLATFORM EXCHANGE

IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF COMPUTER SCIENCE

BY

1. YARED TEKILESELASSIE ……. CIR/292/11
2. METASEBIYA GEMECHU ……. CIR/070/11
3. TOLAWAK YONAS…….……. CIR/093/11

DECLARATION

This is to declare that this project work which is done under the supervision of Mr. Tesfahun Nuree and having the title Android-Based Bookstore And Exchange Platform the sole contribution of Yared Tekileselassie, Metasebiya Gemechu and Tolawak Yonas. No part of the project work has been reproduced illegally (copy and paste) which can be considered as Plagiarism. All referenced parts have been used to argue the idea and have been cited properly. We will be responsible and liable for any consequence if violation of this declaration is proven.

Date: _____

Group Members:

Full Name	Signature
1. Yared Tekileselassie	_____
2. Metasebiya Gemechu	_____
3. Tolawak Yonas	_____

APPROVAL FORM

This is to confirm that the project report entitled Android-Based Bookstore And Exchange Platform submitted to Wolkite University, College of Computing and Informatics Department of computer science by Yared Tekileselassie, Metasebiya Gemechu and Tolawak Yonas is approved for submission.

Student Team Approval Form

Student Name	Signature	Date
---	---	---
---	---	---
---	---	---
---	---	---

Advisor and Department Head Approval Form

Advisor Name	Signature	Date
---	---	---
Department Head Name	Signature	Date
---	---	---

Examiner Approval Form

Examiner 1 Name	Signature	Date
---	---	---
Examiner 2 Name	Signature	Date
---	---	---
Examiner 3 Name	Signature	Date
---	---	---

Contents

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

We would like to thank GOD for giving us strength and health to start this project. We are also grateful to our advisor Mr. Tesfahun Nurre for his constructive guidance from the beginning of the project up to now, and then we would like to thanks our college for giving us this chance to do this final project. Finally, we want to extend our thanks to our parents for their encouragement, motivation and support throughout our study. And also, we would like to thank Ethio bookstore and Jafar bookstore.

ABBREVIATION

API	Application Program Interface
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
UML	Unified Modeling Language
HTML	Hyper Text Markup Language
CSS	Cascade Style Sheet
JS	JavaScript
OOSAD	Object-Oriented Software Analysis and Design
UI	User Interface
UX	User Experience
JSON	JavaScript Object Notation
JWT	JSON Web Token

ABSTRACT

Android-based bookstore and exchange platform is a technology-based online book shopping system that attempts to solve a number of problems in the purchase of a desired book. It facilitates communication between a bookshop owner and any client interested in purchasing a book, as well as the exchange of used books with other users of our system. Android-based bookstore and trading platform is the name of our suggested system. The goal of this system is to make life easier for bookstore owners and book buyers by creating an android-based bookshop and exchange platform that utilizes current technologies. According to this problem statement, there is no effective online bookshop and exchange platform that allows book readers to easily access their favorite books, pay for them online, and trade books with other book readers. This project seeks to meet the basic requirements of an online shopping app, such as user registration, adding books for sale or exchange, shopping cart, book ordering, and exchange functionality allowing consumers to trade books with other customers.

Finally, the Android-based bookstore and exchange platform is an android-based system with an integrated mobile app software application that will improve the experience most people have while purchasing or exchanging books.

Keywords Android bookstore, book exchange, book ordering

CHAPTER ONE: INTRODUCTION

In today's world, the internet plays a critical part in people's lives. As people's living levels rise, their expectations for life's quality and efficiency rise as well. As a result, the traditional bookstore's inconvenient nature progressively emerges, and the online bookshop has steadily been more widely used. The online bookstore is a book industry revolution. Because the time, location, and space available to traditional bookstores are limited, the types of books and books available are limited. However, the internet bookstore disrupted the traditional bookstore's management model; as long as you have a smartphone, you can buy a book from anywhere, saving time and effort and effectively shortening the book selection link time. The online bookshop system is built on the premise of providing consumers with convenience and service. The fundamental concept of the online bookstore application is to provide a virtual shop that allows users to quickly access and purchase any type of book over the internet, and the book exchange concept is to provide a mechanism for users to easily swap books with other individuals who use the application. The most important issue to examine is the annoyance that occurs throughout the book buying and book exchange experience, such as not being able to find the book that a user wants to buy or not being able to find other book readers who want to swap books. The existing methods are as follows: book owners sell their books in their store and collect payment in cash; they organize their books on physical shelves; if they want a book, they must search for it on the shelf; and anyone who wants to buy a book must go to the store in person and search for the preferred book on the bookshelf; and anyone who wants to exchange a book with other people must meet in other ways, such as when they buy books or in a coffee shop. The disadvantages of the existing system for book store owners include: they cannot provide the books that are available in their store to many book readers; manual organization of books that are available in the store is time consuming and exhausting; the bookstore owner cannot know about his customers and thus cannot easily distribute his books to them; when we see book readers, they must physically go to the book store to find the book they prefer; and they must physically go to the book store to find the book they prefer. The things that need to be improved are the shopping procedure and the experience that book store owners and book readers go through, the bookstore organization will also need to be improved. The procedures can be automated using the available technologies, our proposed system will contribute a solution to this problem by letting bookstore owners organize their bookstore shelves,

customers and book distribution, with the application we provide this can be easily happen, and book readers will buy books easily without leaving their doorsteps, because book distribution, online payment and delivery will happen without leaving the application, and the book swapping feature will let the users upload books for exchange and request if they get the book they want that are nearby their location. The purchasing process and the experience that book store owners and readers have should be improved, as well as the bookstore organization. The procedures can be automated using available technologies, and our proposed system will contribute to a solution to this problem by allowing bookstore owners to organize their bookstore shelves, customers, and book distribution. With the application we provide, this can be done easily, and book readers will be able to buy books without leaving their homes, because book distribution, online payment, and delivery will all take place without leaving the application, and book swapping will take place without leaving the application. Finally, an online book store and exchange platform allows users to search, exchange, and buy books based on title, author, and subject matter. The books that the user has chosen are shown in a user-friendly way, and the user can order their book online via online payment. Instead of going out to a book store and wasting time, the user can utilize this platform to buy books from book stores online and swap books with other users based on their geographical area.

1.1. Statement of the problem

People in our country buy books by going to a bookstore in person, which wastes the user's valuable time. Additionally, the user may not be able to find the book they want, or the bookstore may be packed, requiring the user to wait for a chance to buy the book. Even if there are a few online book store platforms, this is the conventional method of ordering books. And there is no functional platform that allows a person to exchange books with a nearby user. Those platforms are only available in a few cities across our country, and they only serve a small number of people. They don't solve the above problem, and they also lack some features that meet the user's needs.

1.2. Objectives of the project

1.2.1. General Objective

The general objective of this project is to develop Android Based Book Store and Book Exchange Platform.

1.2.2. Specific objective

The specific objective of this project is the following: -

- To study the existing system
- To collect requirement
- To analysis the requirement
- To design user interface and database
- To implement the system
- To test the system
- To deploy the system

1.3. Feasibility Analysis

1.3.1. Technical Feasibility

This Mobile application is going to use powerful programming languages and best frameworks in order to ensure a good user experience for the end user, as well as adopting good coding practices for the developer. The structure of the application will consist of a backend and a frontend.

1.3.2. Operational Feasibility

The main needs of users from the proposed system are searching nearby bookstores, ordering books online from their home any place, paying the payment through online payment gateways, viewing the book, get the order from their desired location.

1.3.3. Economic Feasibility

Our proposed system is economically feasible because when we compare the cost that we need to develop and implement the proposed system less expense than the existing manual system or not require much more cost and material to implement the system. Here we have stated the costs related to the project and the benefits that are going to be gained after the completion of the project by performing a cost-benefit analysis.

1.3.3.1. Tangible Benefits

Cost reduction and avoidance, increase the income of the organization, improving response time, producing error free out put such as redundancy, increased management planning and control.

1.3.3.2. Intangible Benefits

Increase customer satisfaction, faster decision making, increase accuracy, increase speed activity, flexibility, and authentication.

1.4. Scope and Limitation of the Project

1.4.1. Scope of the project

The Scope of Android book store and exchange platform is to design

- Book ordering
- Book exchange with nearby users
- Delivery
- Payment
- Simulation of order tracking

1.5. Limitation of the project

Due to financial constraints, the system can't cover order live tracking because the Google Real-time Map Geolocation API takes high cost, and also direct payment gateway with payment platforms because the process to get local payment APIs is much complex.

1.6. Significance of the Project

In view of the rapid development of technology in almost all the fields of operation and its use in relation to information management, it has become important to look into the development of online book store and exchange platform to meet the customers' demands. The main significance of the project is to help bookstore owners to sell their books easily with statistical data about their store and users to get books easily and exchange used books with nearby users.

Beneficiary of the Project

- **For customers**
  - Customers save their valuable time by avoiding the traditional way of ordering book.

- Reduction of unnecessary cost for transportation
- Customers only focus on their business and their order delivered to their home
- User can exchange books with other users based on their location (Geopositioning)
- They can save their energy by avoiding physically go to the bookstore
- Based on the current situation avoiding physical contacts with other peoples, so it's important to prevent COVID-19

➢ **For Bookstore owners**

- They easily offer their services
- They can advertise their services on the system
- They can serve many customers as compared to the traditional way of ordering
- They can get statistical data about their business

➢ **Benefits for the owners of the system**

- By building our system we will gain more knowledge and experience.
- We will do our parts on creating a generation that is interested in reading book, and it'll benefit us fulfil our obligation.

1.7. Methodology and Tools

1.7.1. Data Collection Tools/Techniques

To do this project, the project team will use data collection methodology to gather the necessary information that is needed to develop the project.

- **Document Analysis: -** In developing Android Based Book store & Book Exchange Platform we analyze forms and paper works of the existing system and related works from internet, and this will enable us to understand the problem domain better.
- **Interview: -** In developing Android Based Book store & Book Exchange Platform we would like to have conversation with book reader worker to understand their problem area and their interest.
- **Questionnaires: -** we distribute questioner paper for the registrar to know more about the Android Based Book store & Book Exchange Platform.

1.7.2. System Analysis and Design

In the system analysis and design phase of a project we will use the object-oriented approach. Because it is a better way to construct, manage and assemble objects that are implemented in our system. In this project, the team will use object-oriented system development methodology for the design.

This technique has several phases, some of them are: -

1.7.2.1. Object Oriented Analysis

During this phase the team uses to model the function of the system (use case modelling), find and identify the business objects, organize the objects and identify the relationship between them and finally model the behavior of the objects in detail.

1.7.2.2. Object Oriented Design

During this phase our team uses E-draw and Visio software to refine the use case model and rational rose for designing the sequence, activity diagrams and to model object interactions and behavior that support the use case scenario.

The reason why we have selected object-oriented system analysis and design method specifically UML model is because of the following advantages: -

- To enable a high degree of reusability of designs.
- To decrease the cost of software maintenance.
- Reduce maintenance burden.
- Increased consistency among analysis, design, and programming activities.
- Improved communication among users, analysis, design and programming.

1.7.3. System Development Model

System development model is a technique that is used to show how the proposed system will be developed. In our project we will use Iterative Model because of the following reasons:

- Iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

- Life cycle model does not attempt to start with a full specification of requirements.

- This process is then repeated, producing a new version of the software at the end of each iteration of the model.

- The progress is easy measurable
- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.

1.7.4. System Testing Methodology

- **Unit testing**

  Since the designed system is in object-oriented method the team firstly tested the system in individual class level.

  To check each modules API we will test the modules using postman

- **Integration Tests**

To check whether the unit test working together correctly we will do Integration testing using postman

- **System testing**

After all the testing is performed, the system will be tested by a user who will be invited by the team. Especially at this level of testing the team seriously tested the system.

1.7.5. Development Tools and Technologies

1.7.5.1. Frontend Technologies

- Dart Main Programming language for Mobile application [1]
- Flutter (It's a cross platform mobile app framework) [2]
- ReactJs (It's JavaScript library with Redux for state management) used for Admin Dashboard [3]
- CSS and Other UI libraries: is the language we use to style an HTML document.

1.7.5.2. Backend Technologies

- NodeJS (Server side with Express framework) [4]

- MongoDB (Document-oriented database) [5]
- Firebase for (additional authentication and push notification) [6]
- JavaScript as our main backend language [7]

1.7.5.3. Documentation and Modeling Tools

Documentation tools

- MS word 2021: For the purpose of writing documentation.
- Microsoft PowerPoint 2021: Is software that we use for presentation purpose.

Modeling Tools

- E- Draw max: It is used to draw use case diagrams, pie charts, etc.
- Gantchart: A type of bar chart that illustrates a project schedule,

For Testing API

- Postman

For Version Controller

- Git [8]

For Collaboration

- Github [9], Google Docs

1.8. Document Organization

This project document deals all about Android bookstore and exchange platform. It has five chapters: introduction, existing system, proposed, analysis and design phase.

The first chapter is introduction part deals with the background for the project area and statement of the problem, scope and limitation of the project, methodology in terms of data collection, system analysis and design, system development, testing as well as development tools and feasibility study of the project in terms of operational, technical and economic feasibility.

The second chapter shows the user of existing system, major function, drawback in terms of performance, input/output, security as well as efficiency and business rule of the existing system.

The third chapter shows discussions of the new proposed system detailed description of system functionalities in terms of functional and non-functional requirement in the case of non-functional requirement we included here user interface, hardware consideration, security issues, performance, error handling, quality and resource issue.

The fourth chapter deeply deals system design with using a UML (Unified Modelling Language) diagrams like, use case diagram, class diagram, sequence diagram, activity diagrams, and state chart diagram thus diagram generally shows three system model mean functional, object and dynamic.

The fifth chapter it shows the design goal in terms of performance, dependability, maintenance, end user and priority of design goal, and proposed architecture of the solution including subsystem decomposition, deployment diagram, detail class diagram, persistence model for tables mapping followed with the access control as well as package and algorithm design are included in this chapter.

CHAPTER TWO: LITERATURE REVIEW

2.1. Literature review on Existing system

2.1.1. Ethio Bookstore

It's an online portal that ships throughout Ethiopia. It's an online platform for creating a store that combines the advantages of online purchase with book discovery. The disadvantage of Ethio Bookstore is that they do not have an online payment system that allows clients to pay using online payment platforms; instead, they only accept money upon delivery. The second disadvantage is that their application's user interface is web-based, making it difficult to utilize.This system is a web-based e-commerce site that allows users to purchase a variety of books. They attempted to address a problem in the online shopping sector, which is a significant role in day-to-day book shopping. The difference between the current system and our proposed system is that ours will be android-based and include a payment system, allowing book readers to effortlessly shop.

Figure 2- 1: Ethio bookstore cart page

Figure 2- 2: Ethio bookstore home page

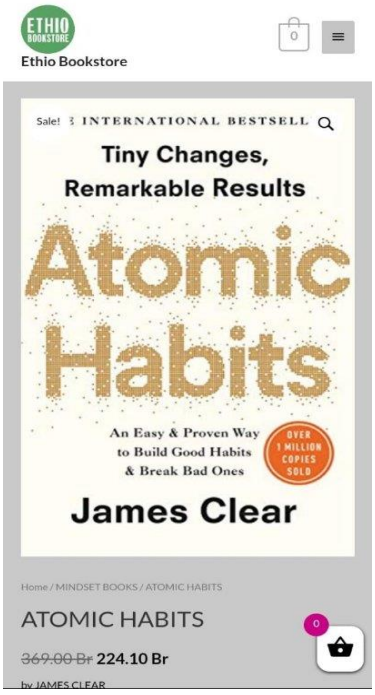

Figure 2- 3: Ethio bookstore books list page **Figure 2- 4: Ethio bookstore book detail page**

2.1.2. Jafar Bookstore

It provider of an online book store service intended to simplify book store and delivery for consumer. Their limitation is same as Ethio Bookstore but they don't have online payment system, and also they don't have mobile application for users to buy books easily using their mobile device, the book exchange feature is absent on all existing systems.

2.2. Users of Existing System

The online book-store's main users are divided into three categories, one is the front user, and one is the background user. Front-end users are mainly customers and who consume online book-stores. Front-end users can register, login, query, join shopping cart, place orders, submit orders, modify personal information, confirm receipt and add comments on the website. And the background users add, modify or delete the book classification; add, modify or delete the book information, manage the order information and reply to the user's comments. The user is the administrator, the administrator to play the role of overall planning, master and control the front information and so on.

2.3. Major Functions of the Existing System

The major functions of the existing system is that using application or website customers can query

books using their mobile device, add books to the cart and finally order it and the traditional way of function is customers go to the front of the shop and find their desired book from shelf and finally buy it. The existing system also include manual payment features, if its online store customers pay when their book is delivered.

2.4. Drawbacks of the Existing System

- It's not integrated with online payment platforms, like banks and other middle ware platforms.
- The user interface of their system is not user friendly.
- This process is so much time-consuming and does not provide trust in retailers.
- There is the threat of forgery if you are new to marketing and do not know how to bargain with these street sellers.
- There is no proper way of getting to know about the actual price of the books sometimes shopkeepers sell it more than the market price by saying that the book is not available in the market.
- It might be the case that shopkeepers do not refund money if you purchase some wrong books.

2.5. Business Rules of the Existing System

- **BR01** – The customer must be registered
- **BR02 -** The customer must have a valid email and password
- **BR03** – The admin must have a valid email and password
- **BR04** – The admin must fill the book detail properly
- **BR05 -** The customer must pay to get his books
- **BR06 -** The customer must get a receipt

CHAPTER THREE: PROPOSED SYSTEM

3.1. Background Overview

The proposed system provides:

- **Efficient way of applicant's data management: -** Since the proposed system uses a database system there is no loss of data .The applicant information will highly secured, the search and update of applicant will be simple.
- **Dashboard:** - Android-Based Book Store and Book Exchange Platform will offer web based dashboard that lets book-store owners add books for sell, get statistical data about their store and get data about their customers.
- **Exchange Platform:** - Our system will provide geolocation based book exchange feature that let users exchange their books with other user by getting their location and listing nearby users.

3.2. Functional Requirement

In software engineering and systems engineering, a functional requirement defines a function of a system or its component, where a function is described as a specification of behaviour between outputs and inputs. Functional requirements define the fundamental actions that the system must perform. This android-based system connects book-stores with customers and users remotely. While connecting book-stores with customers the system will provide different books based on the user's desire and they can easily order books. The system classifies book-stores based on their location and also the users based on their geolocation which we use to get nearby books for exchange. If a user wants to buy a book from the book-store the app will provide a list of books based on their popularity, genre, and price, then the user can order the book by finishing the payment process, finally, the book will be delivered to the address the user want. If a user wants to exchange a book, he/she can post the book, and other nearby users can see their information.

3.2.1. System Actors module Functional requirement

3.2.1.1. Customer

The user module in the system contains the functions of users.

- CM00 – Search online book-stores

  The system shall enable the customer to search online book-stores by their name and by their location.
- CM02 – Select book-store

  The system shall enable the customer to select book-store that is found in the system.

- CM03 – Search book

  The system shall enable the customer to search book by their name, genre and price range.
- CM04 – Select book

  The system shall enable the customer to select book that is found in the system.
- CM05 – Add a book to the cart

  The system shall enable the customer to add book to cart that is found in the system.
- CM06 – Order book

  The system shall enable the customer to order book that is in the cart.
- CM07 – View information

  The system shall enable the customer to view their profile, order record, favorite books, and payment record.
- CM08 – Add book for exchange

  The system shall enable the customer to upload books for exchange.
- CM09 – Get exchange books list

  The system shall enable the customer to get list of books that are for exchange.
- CM10 – Request books that are up for exchange

  The system shall enable the customer to send requests for users to get the book that is for exchange.
- CM11 – Notification

  The system shall enable the customer to get notification when order is approved.
- CM12 – Notification

  The system shall enable the customer to get notification when a user-approved book for exchange.

3.2.1.2. Book-store Owner

The book-store owner module helps book-store owner

- BOM00 – Add books

  The system shall enable the bookstore owner to add books to the online store.
- BOM01 – Get added book lists

  The system shall enable the bookstore owner to get added book lists from the online store.
- BOM02 – Update books

The system shall enable the bookstore owner to update book information.

- BOM03 – Delete books

  The system shall enable the bookstore owner to delete books from the online store.
- BOM04 – Get orders list

  The system shall enable the bookstore owner to get customers order list.
- BOM05 – Search book

  The system shall enable the bookstore owner to search books by name, genre, price, stock, and added date.
- BOM06 – Get statistical data about the store

  The system shall enable the bookstore owner to get statistical data such as sold books list, popular genres, customers count and list, and profit.
- BOM07 – Notification

  The system shall enable the bookstore owner to get notification when a book is ordered

3.2.1.3. Administrator

The administrator module gives authority to administrators

- AM00 – Authorize

  The system shall enable the administrator to authorize and approve book-store owners.
- AM01 – Get statistical data about the system

  The system shall enable the administrator to get statistical data such as number of users, number of book-stores.
- AM02 – Dispatch delivery

  The system shall enable the administrator to approve order and dispatch delivery.
- AM00 – Notification

  The system shall enable the administrator to get notification when book-store has delivery.
- AM00 – Control users

  The system shall enable the administrator to control user’s suspicious activities.

3.2.1.4. Common functionality to all modules

- CM00 - Sign-in

  The system shall enable all actors to sign-in to the system using their email, phone number or username and password that is stored in the database.

- CM01- Sign-up

  The system shall enable all actors to sign-up to the system by providing legal information.
- CM02- Sign-out

  The system shall enable all actors to sign-out from the system.
- CM03-Give review

  The system shall enable all actors to give feedback or review to a user, book-store and book.
- CM04- View feedback

  The system shall enable all actors to view feedback or review which is given from users.
- CM05-Reset password

  The system shall enable all the actors to set a new password if the user forgets the current password.
- CM06-View profile

  The system shall enable all the actors to view his/her profile which has information about him/her.
- CM07-Edit profile

  The system shall enable all the actors to edit the current profile.

3.3. Non-functional Requirement

Non-functional requirements define how a system is supposed to be and also describes constraints for implementing the project. As the system is going to be implemented in the online shopping sector which includes online payment the major constraint is a security issue. So the system should have a high authentication function. while login into the system should generate a dynamic web token that is encrypted with users' login information and encryption key which is stored securely and all user's API requests will be authorized according to that web token. We will use HTTP header protection because HTTP header leaks sensitive information about the application. As we are planning to implement the system with MongoDB our system will be aware of NoSQL injection. Several modern applications use NoSQL databases. Attackers will try to invade our application with this NoSQL injection. It is very important to take this problem into account. Our system also can prevent Cross-Site Scripting (XSS) attacks. Cross-Site Scripting (XSS) attacks are

a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites generally, the non-functional requirements of our system are as follows.

3.3.1. User Interface and Human Factors

Our mobile application and web dashboard use edge cutting design techniques and tools to present a satisfying user interface and user experience, so it can easily be used and operated by any non-technical or technical user as the system uses a GUI user interface.

3.3.2. Hardware Consideration

The system contains an android-based app and a web-based dashboard. So the end-user part is portable with any android device and the dashboard is fully portable with any system using any web browser should be able to use the features of the system. The mobile app shall work properly on Android and the dashboard on any operating systems like Microsoft Windows, Apple macOS, and Linux.

3.3.3. Security Issues

Our system will use user validation during login to ensure that the user is valid or not. Access to the software is given only to valid operators. We need a username, email or phone number and password to get access to the software. Execution qualities such as testability, maintainability, and scalability are embedded in the architecture of the software system. And the system will have modulus that prevents the following attacks.

3.3.3.1. NoSQL injection attack

Attackers can inject code into commands for databases that don't employ SQL queries, such as MongoDB, using NoSQL injection flaws. Let's look at the differences between NoSQL injection and standard SQL injection, as well as what we can do to prevent it.

The attacker can use SQL injection to run commands in the database. NoSQL databases, unlike relational databases, do not have a standard query language. The query syntax for NoSQL is product-specific, and queries are written in the application's programming language: PHP, JavaScript, Python, Java, and so on. This means that if the injection is successful, the attacker will be able to run commands not just in the database, but also in the application itself, which is significantly more harmful.

The following techniques will be used to prevent NoSQL attacks:

1. **Secure Coding Practices**

   We select the most recent versions of these continually evolving and new datasets. For example, the popular MongoDB was designed to be vulnerable on numerous levels, resulting in fatal injection attacks, but the most recent versions have been enhanced on the security front.

2. **Input Validation**

   Unsanitized user inputs must be avoided in application code when creating database queries, just as they must be avoided in SQL Injections. To ensure that harmful values are not used, all user inputs must be verified.

3. **Least Privilege Policy**

   As a rule, in web application security, the principle of least privilege must be followed. This way, the reach of attackers, in case of successful attacks, can be limited.

3.3.3.2. HTTP header attack

The HTTP header injection vulnerability is a web application security phrase that describes when an attacker uses a web application to deceive it into injecting extra HTTP headers into legitimate HTTP responses. HTTP header injection is a technique that can be used to help malicious assaults like cross-site scripting and web cache poisoning. [10]

Helmet.js will be used to prevent HTTP header attacks. Helmet.js is a Node.js module that aids in the secure transmission of HTTP headers. It's done in the context of express apps. Helmet.js can also be said to aid in the security of express applications. Sets numerous HTTP headers to protect against cross-site scripting (XSS), clickjacking, and other threats.

3.3.3.3. XXS attack prevention

Cross-site scripting (commonly known as XSS) is a web security flaw that allows an attacker to manipulate how users interact with a susceptible application. It allows an attacker to get around the same origin policy, which is meant to keep websites separate from one another. [11]

3.3.4. Performance Consideration

Our system will carry out and complete its duty with the quickest possible response time to the user. The system as a whole will be quick and error-free. It will come with built-in mistake detection and correcting features. Because we're using Node.js as a runtime environment, the system should be able to easily process a big volume of data. Node.js is a single-threaded and

asynchronous. This means that all I/O activities do not obstruct other processes. You can also send emails, read files, query the database, and so on all at the same time.

There will be no distinct Node.js process for each request to the web server. However, one Node.js process would be running at all times, listening for connections. All JavaScript code is run in the main thread of the process, while all other I/O activities are run in other threads, resulting in nearly no delays.

3.3.5. Error Handling and Validation

When users make some error: The system has the capacity for error handling. Before submitting any API queries, our system validates user input on the client side, which speeds up the process. If all of the inputs are valid, we send an API request and check for required validations on the server side, for example, we will check if the user email or phone number has already been used, or if the password is incorrect, and we will send a response with the error object, and finally we will render the server errors in the client side. With minimum database redundancy. If there is an error in any component or module, then it should not affect the remaining part of the software.

3.3.6. Quality Issues

The system should be reliable as it handles the data of the bookstore. To ensure reliability, this system is being designed using software that is established to be stable and easy to use and should easily be available at any desired time because the project is a web-based application.

3.3.7. Backup and Recovery

The system should be able to fault tolerate (hardware and software failure) by assisting the user with brief guidance, if any error happens that can damage the system a backup database is available in a MongoDB cluster so all the data are safe.

3.3.8. Physical Environment

The system database, back-end server, and front-end web app will be hosted on cloud platforms such as AWS, Google Cloud, Mongo Atlas, Herokuapp, Netlify, and Vercel, while the mobile app will be available on Google Play.

3.3.9. Resource Issues

Because of the non-blocking behavior of our back-end, our system can manage more than 10000 requests or simultaneous clients per second, and our system and our test server capacity should be able to accommodate several users at the same time.

3.3.10. Documentation

In-app guidelines for users and documentation for book-store owners are available, the in-app guidelines help the user navigate through the app, and the documentation for the book-store owner helps access the dashboard and get information easily. The admin will also have a documentation for proper understanding of the system.

CHAPTER FOUR: SYSTEM ANALYSIS

4.1. System Model

System design is the transformation of the analysis model into a system design model. Up to now we were in the problem domain. System design is the first part to get into the solution domain in a software development. This chapter focuses on transforming the analysis model into the design model that takes into account the non-functional requirements and constraints described in the problem statement and requirement analysis sections discussed earlier. The purpose of designing is to show the direction how the system is built and to obtain clear and enough information needed to drive the actual implementation of the system. It is based on understanding of the model the software built on. The objectives of design are to model the system with high quality. Implementing of high quality system depend on the nature of design created by the designer. If one wants to change to the system after it has been put in to operation depends on the quality of the system design. So if the system is design effetely, it will be easy to make changes to it.

4.1.1 Use Case Model

A use case defines a goal-oriented set of interactions between external users and the system under consideration or development. Thus a Use Case Scenario is a description that illustrates, step by step, how a user is intending to use a system, essentially capturing the system behavior from the user's point of view. In order to create relevant use cases for the system, the following actors for the system have been identified:

- Customer
- Book-store Owner
- Admin

Use case identification

- Register/Sign up
- Login
- Logout
- Forgot password/Reset Password
- Get books list
- Get exchange books list

- Get books by category
- Search book
- Add books for exchange
- Add books for sell
- Add book to cart
- Request book exchange
- Order book
- Make payment
- Approve book-store
- Edit profile
- Update book information
- Delete book
- Get statistical data
- Get orders list
- View book detail
- Dispatch delivery
- Control users

4.1.1.1. Use Case Diagram

A use case diagram is a dynamic or behavior diagram in UML. We use case diagrams to model the functionality of our system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, in our case the system is mobile application and web app. The "actors" are people or entities operating under defined roles within the system.

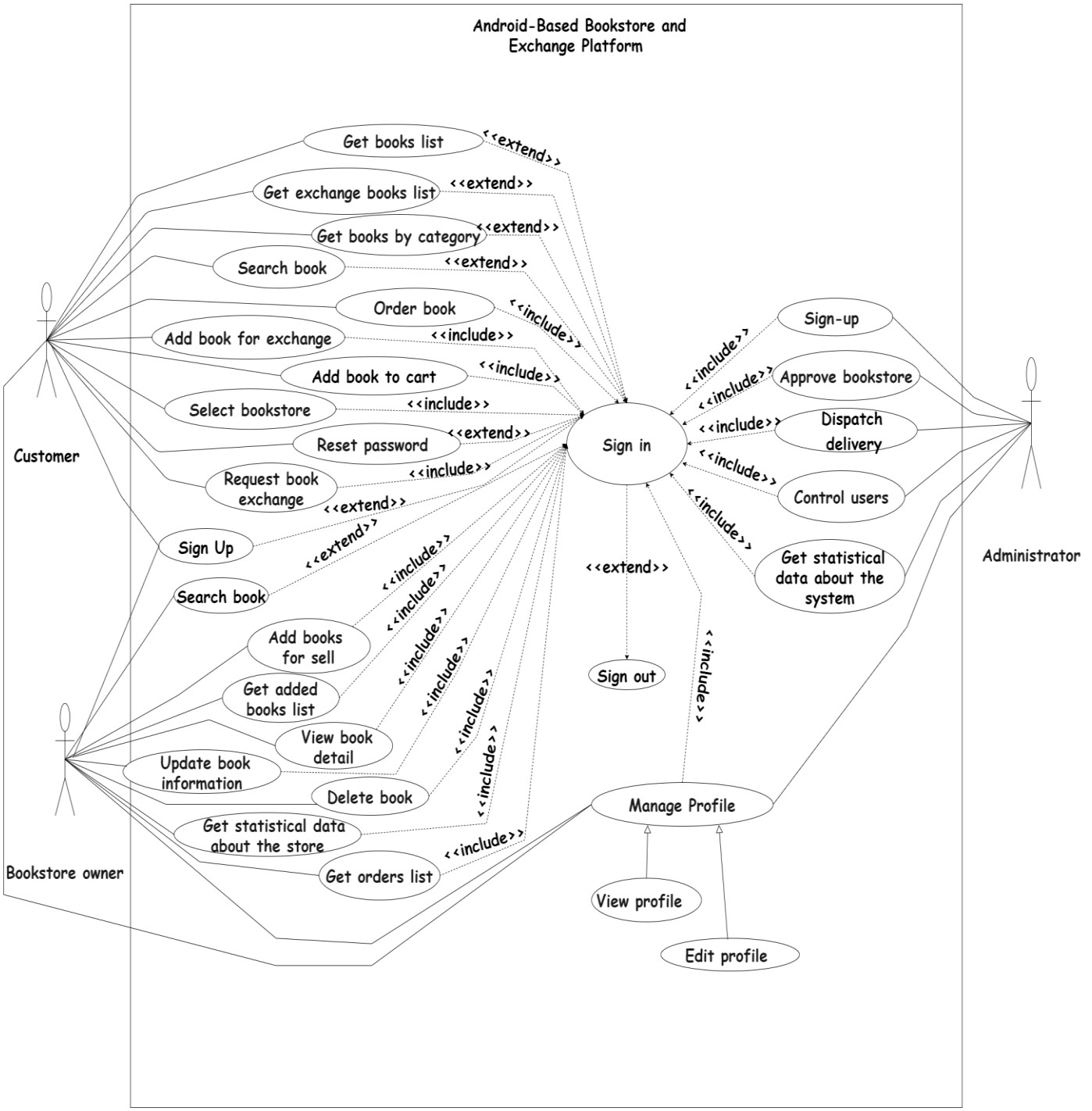

Figure 4- 1: Use case diagram

4.1.1.2. Use Case Description

Table 4-1: Sign up use case description

Use case name	Sign up	
Use case ID	UC-01	
Brief description	This use case allows the actors to create an account.	
Participant Actor	Book-store owner, Customer, Administrator	
Post condition	1. A new user will be created and a new id will be generated 2. All user information will be stored in the database 3. The actor will have access to the system based on their role after verification.	
Basic course of action	**User actions:** 1. User click the sign-up button on a website or mobile app 3. The user fills the provided form according to its category and clicks the submit button 6. The user inserts the confirmation code	**System response:** 2. The system displays the registration form 4. The system checks the input 5. The system displays a confirmation text field and sends a confirmation code via SMS 7. The system checks the confirmation code 8. The system activates the user account after confirmation

The alternate course of action	**Case:** 1. If the actor enters invalid information or did not fill the form 1.1. registration will be denied 1.2.the actor will be notified about the situation 1.3.the actor will be asked to retry 1.2. If the actor is already registered 1.2.1. The actor will be notified about the situation 1.2.2. Redirected to the login page.

Table 4-2: Sign in use case description

Use case name	Sign in
Use case ID	UC-02
Brief description	This use case allows a user to sign in and access the system by getting the user's email address and password.
Participant Actor	Bookstore owner, Customer, Administrator
Precondition	1. The actor should have a valid account.
Post condition	The user logged into the system to execute the privileged tasks and generate a dynamic token for API request authorization.

Basic course of action	**User actions:** 1.User open dashboard website or mobile app and click the sign-in button 3.The user inputs email and password	**System response:** 2.The system displays a sign in form 4.The system checks the input validity 5.The system allows users to perform their privileged tasks and generate dynamic token
The alternate course of action	**Case:** 2.1.When the user enters the wrong username and password. 2.1.1 The system displays "incorrect username and password" and goes back to step 3	

Table 4-3: Sign out use case description

Use case name	Sign out
Use case ID	UC-03
Brief description	Signing out the current user from the system
Participant Actor	Book-store owner, Customer, Administrator
Precondition	The user must be signed in
Post condition	Sign-out from the system

Basic course of action	**User actions:** 1. User click on the account button 3. Click on the sign out button	**System response:** 2. The system shows the sign-out button
The alternate course of action	**Case:** 3.1.The user is not signed in 3.1.1. The system will redirect to the sign-in page	

Table 4-4: Authentication use case description

Use case name	Authentication
Use case ID	UC-04
Brief description	The actor requests the system to sign in after opening the website or mobile application and going into the sign-in page.
Participant Actor	Book-store owner, Customer, Administrator
Precondition	1. The actor must have been registered on the system. 2. The actor must have his/her role.
Post condition	1. The actor will be authorized to use the system based on his/her role. 2. Dynamic token will be generated 3. The actor will be redirected to the actor's homepage depending on their access level.

Basic course of action	**User actions:** 1. The user sends an email and password.	**System response:** 2. The system verifies Email and password.
The alternate course of action	**Case:** 4.1. The actor enters an invalid email or password 4.1.1. The actor will be notified about the situation. 4.1.2. The actor will be prompted to enter email and password. 4.1.3. The actor will be prompted if he/she has forgotten their password.	

Table 4-5: API Request Authorization use case description

Use case name	API Request Authorization
Use case ID	UC-05
Brief description	Authorize all API request to the server
Participant Actor	Book-store owner, Customer, Administrator
Precondition	1. The Actor should have an account 2. The Actor should be logged in 3. The actor should have the correct token that is generated when the actor logged in
Post condition	Get access to the API request

Basic course of action	**User actions:** 1. Request API	**System response:** 2. Check if the web token that is generated when the user logged in with the current web token.
The alternate course of action	**Case:** 5.1. If the web token is invalid 5.1.1. The API request will be denied	

Table 4-6: View profile use case description

Use case name	View profile
Use case ID	UC-06
Brief description	View his/her profile (information)
Participant Actor	Book-store owner, Customer, Administrator
Precondition	1. The user account should exist 2. The user should sign in 3. The user should have the correct web token
Post condition	View his /her current profile information.

Basic course of action	**User actions:** 1. User click on the account button 3. User Click on the profile button the website or mobile application	**System response:** 2. The system shows the profile button 4. The system will navigate to the user profile page
The alternate course of action	**Case:** 6.1. The user is not signed in 6.1.1. The system will redirect to the sign-in page 6.2. The user has an incorrect token 6.2.2. The API request of getting to view the profile will be denied	

Table 4-7: Edit profile use case description

Use case name	Edit profile
Use case ID	UC-07
Brief description	Edit users profile information
Participant Actor	Book-store owner, Customer, Administrator
Precondition	1. The user account should exist 2. The user should sign in 3. The user should have the correct token

Post condition	The user profile information will be changed	
Basic course of action	**User actions:** 1. User click the account button 3. The user will click the profile button 6. The user will click the edit profile button 8. The user will edit his/ her profile	**System response:** 2. The system shows the profile button 4. The system will navigate to the profile page 5. The system will show the edit profile button 7. The system will navigate to the edit profile page and show the editing form
The alternate course of action	**Case:** 7.1. The user is not signed in or the user 7.1.1. The system will redirect to the sign-in page 7.2. The user has an incorrect web token 7.2.2. The API request put to edit the profile will be denied	

Table 4-8: Reset password use case description

Use case name	Reset password
Use case ID	UC-08

Brief description	Reset password if the user forgets his/her password	
Participant actor	Book-store owners, Customer, Administrator	
Precondition	1. The user must have an account	
Post condition	Set a new password	
Basic course of action	**User action:** 1. The user opens the login page/screen 3. The user clicks the forget password button 4. The user inserts his/her email address or phone number and clicks submit button 6. The user inserts the code 8. The user fills the form field	**System action:** 2. The system provides login pages and show the forget password button 4. The system asks for the email address or phone number that is used when the user signed in 5. The system will send a code to the email address or phone number and ask for that code in the system. 7. The system provides a form that has a new password and confirms the password field
An alternate course of action	**Case:** 8.1. If the user inserts an unregistered email or phone number 8.1.1. The system will tell the user that the inserted email or phone number is not registered. And asks again for an email or phone number 8.2. if the sent code and inserted code is not the same 8.2.1 The system asks for the correct code	

	8.3. If the code is correct and the time is up 8.3.1. The system will not proceed to the resetting page

Table 4-9: View book detail use case description

Use case name	View book detailed	
Use case ID	UC-19	
Brief description	View detailed information of the book	
Participant Actor	Customer, Book-store owner	
Precondition	The actor chooses a book and select to view the detailed information	
Postcondition	Get detailed information	
Basic course of action	**User action:** For the user 2 The user selects a book from the given list	**System action:** 1 The system shows detailed information about the book that is selected

Table 4-10: View book store detail use case description

Use case name	View book store detail
Use case ID	UC-10
Brief description	View detailed information of the book store
Participant Actor	Customer, Book-store owner

Precondition	The user must select the book store to view the book	
Postcondition	Get detailed information	
Basic course of action	**User action:** For the customer 2 The user selects a book from the given list	**System action:** 1 The system shows detailed information about the book that is found in the selected user

Table 4-11: Search book-stores use case description

Use case name	Search book-stores	
Use case ID	UC-11	
Brief description	The actor can search the book store	
Participant Actor	Customer, Book-store owner	
Precondition	the customer must be signed in to the system	
Postcondition	The customer can search book store	
Basic course of action	**User action:** 1. The actor selects the search for the book store button. 3. The actor enters the book store name and presses enter	**System action:** 2. The system displays a text field

Table 4-12: Search book use case description

Use case name	Search book	
Use case ID	UC-12	
Brief description	The actor can search the book	
Participant Actor	User, Book-store owner	
Precondition	The actor must be signed in to the system	
Postcondition	The actor can search book store	
Basic course of action	**User action:** 1. The actor selects seethe arch the book store button. 3. The actor enters the book name and presses enter	**System action:** 2. The system displays a text field

Table 4-13: Select book use case description

Use case name	Select book	
Use case ID	UC-13	
Brief description	The actor select the books from the listed books or by searching book that are found in the system	
Participant Actor	Customer	
Postcondition	The user gets the book detailed information	
Basic course of action	**User action:**	**System response:**

	For book store 1 The actor selects the book from the list provided	**For the actor** 2 The system provide the book detail to the actor
The alternate flow of Action	**Case :** **For customer** If the book is not found in the book store registered in the system 13.1. The system will show the book is not found under the registered bookstore. And give an alternative way which allows the user to find the book out of the system by giving file about the to export in a pdf format	

Table 4-14: Select bookstore detailed use case description

Use case name	Select book store	
Use case ID	UC-14	
Brief description	The actor selects the book-store from the listed in the system or by searching book store that is found in the system	
Participant Actor	Customer	
Precondition	The actor must be logged in	
Postcondition	The actor get information about the selected book-store	
Basic course of action	**User action:** **For book store** 1.The user selects the bookstore from the list provided	**System response:** **For the customers** 2 The system gives detailed information about the book-store

The alternate flow of Action	**Case :** If the actor is not logged in 14.1. The system redirects to the login page

Table 4-15: Add a book to the cart use case description

Use case name	Add a book to the cart	
Use case ID	UC-15	
Brief description	The user adds the desired book to the cart	
Participant Actor	Customer	
Precondition	The user must log in	
Postcondition	Book is added to the cart	
Basic course of action	**User action:** 1 The user clicks the book 3. The user clicked add to cart button	**System response:** 2. The system shows detailed information about the book 4. The system adds the selected book to cart cart

Table 4-16: Update book detail use case description

Use case name	Update book detail
Use case ID	UC-16
Brief description	The actor updates the book information

Participant Actor	Book-store owner	
Precondition	1. All updating information must be found first	
Postcondition	1. All updating information will be updated	
Basic course of action	**User action:** 1 The actor clicks the book. 3 The user clicks the edit button	**System response:** 2 The system navigates to the book detail page and show book information with the edit button 4 The system navigate to the edit page with book information

Table 4-17: Update book store detail use case description

Use case name	Update book-store detail	
Use case ID	UC-17	
Brief description	The book-store owner updates information of the book store	
Participant Actor	Book-store owner	
Precondition	1. All updating information must be found first	
Postcondition	1. All updating information will be updated	
Basic course of action	**User action:** 1 The user clicks the Bookstore button from the	**System response:** 2 The system navigates to the Bookstore detail page and show

	Sidebar. 3. The user clicks the edit button	bookstore information with the edit button 4 The system navigate to the edit form page with bookstore information

4.2. Object Model

4.2.1. Class Diagram

We used it used for general conceptual modeling of the structure of the application, and detailed modelling, translating the models into programming code, we also used it for data modeling.

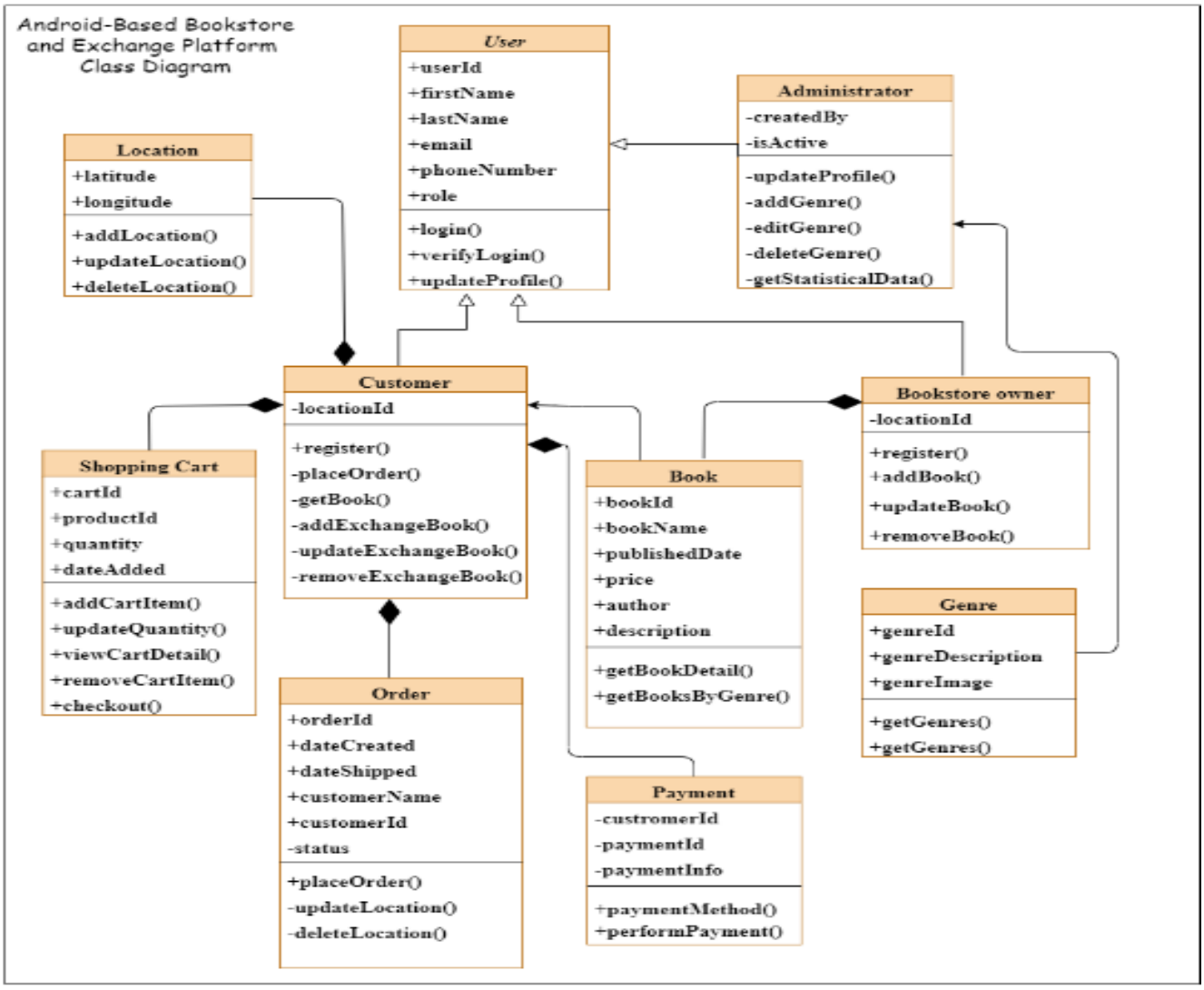

Figure 4- 2: Class diagram

4.2.2. Data Dictionary

A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them. A first step in analyzing a system of objects with which users interact is to identify each object and its relationship to other objects. After each data object or item is given a descriptive name, data type and its relationship are described.

Table 4-18: Data dictionary for user table

Descriptive Name	Data type	Description
userId	string	Holds Id of the user
firstName	string	Holds the first name of the user
lastName	string	Holds the last name of the user
email	string	Hold the email address of the user
phoneNumber	int	Hold the phone number of the user
role	string	Hold role of the user

Table 4-19: Data dictionary for location table

Descriptive Name	Data type	Description
latitude	float	Holds latitude of the customer
longitude	float	Holds longitude of the customer

Table 4-20: Data dictionary for Shopping cart table

Descriptive Name	Data type	Description
cartId	string	Holds Id of the shopping cart
productId	string	Holds Id of the product
quantity	float	Holds quantity of the product
dateAdded	date	Holds date added of the product

Table 4-21: Data dictionary for Order table

Descriptive Name	Data type	Description
ordered	string	Holds Id of the order
dateCreated	date	Holds date of the order created
dateShipped	date	Holds date of the shipped
customerName	string	Holds the name of the Customer
customerId	string	Holds Id of the Customer
status	string	Holds status of the Order

Table 4-22: Data dictionary for Book table

Descriptive Name	Data type	Description
bookId	string	Holds Id of the book
bookName	string	Holds the name of the book

publishedDate	date	Holds date of the published
price	float	Holds price of the Book
author	string	Holds author of the book
description	string	Holds description of the book

Table-4 23: Data dictionary for Payment table

Descriptive Name	Data type	Description
customerId	string	Holds Id of the customer
payment	string	Holds Id of the payment
paymentInfo	string	Holds Information about payment

Table 4-24: Data dictionary for Administrator table

Descriptive Name	Data type	Description
Created by	string	Holds the Id of the admin that created the account
isActive	boolean	Holds the user is active or not

Table 4-25: Data dictionary for Bookstore owner table

Descriptive Name	Data type	Description
location	string	Holds location of the bookstore owner

Table 4-26: Data dictionary for Genre table

Descriptive Name	Data type	Description
genre	string	Holds Id of the genre
Gene description	string	Holds description of the genre
genreImage	string	Holds image of the genre

4.3. Dynamic Model

4.3.1. Sequence Diagram

A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. [12]

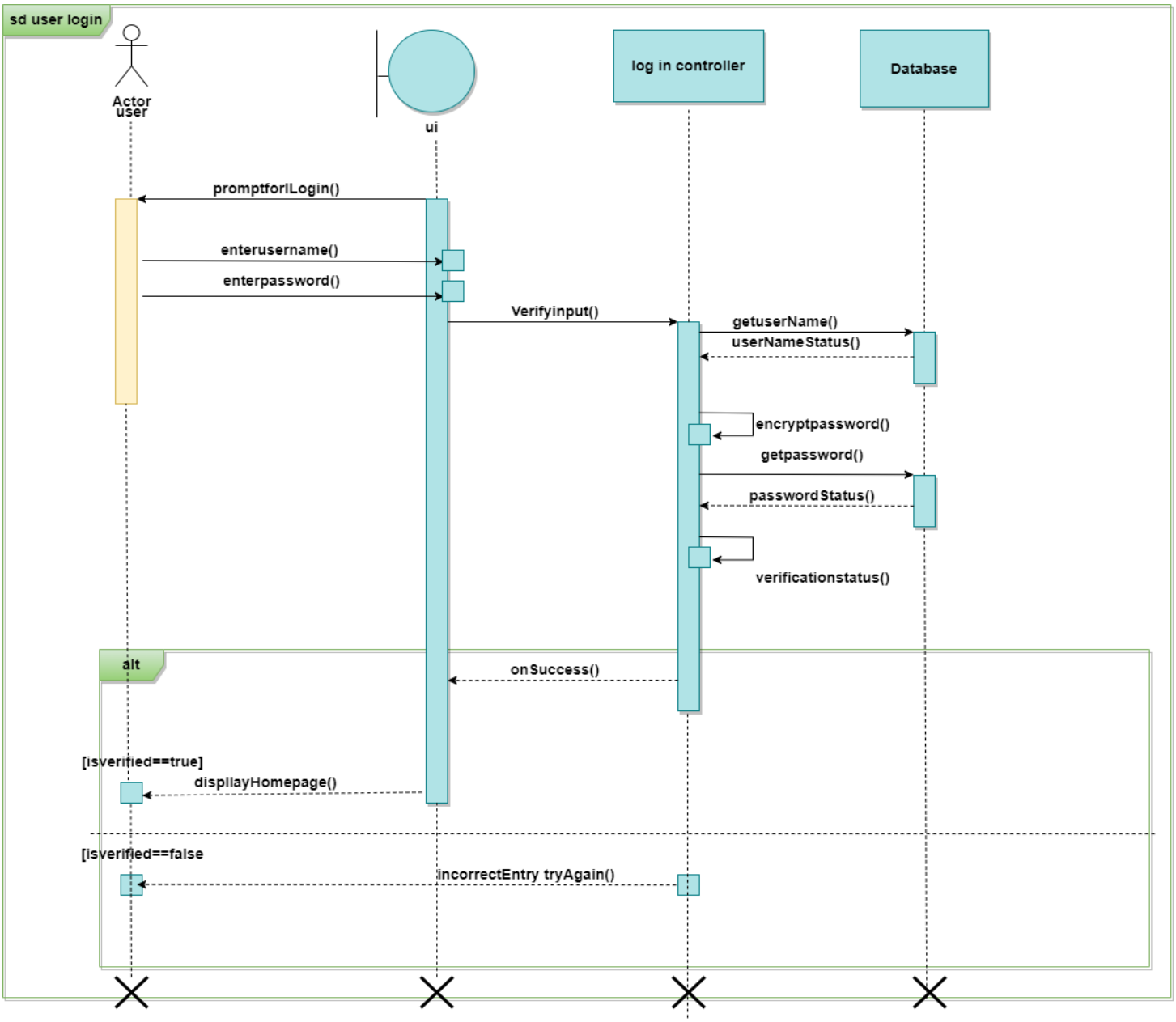

Figure 4- 3: Sequence Diagram user login

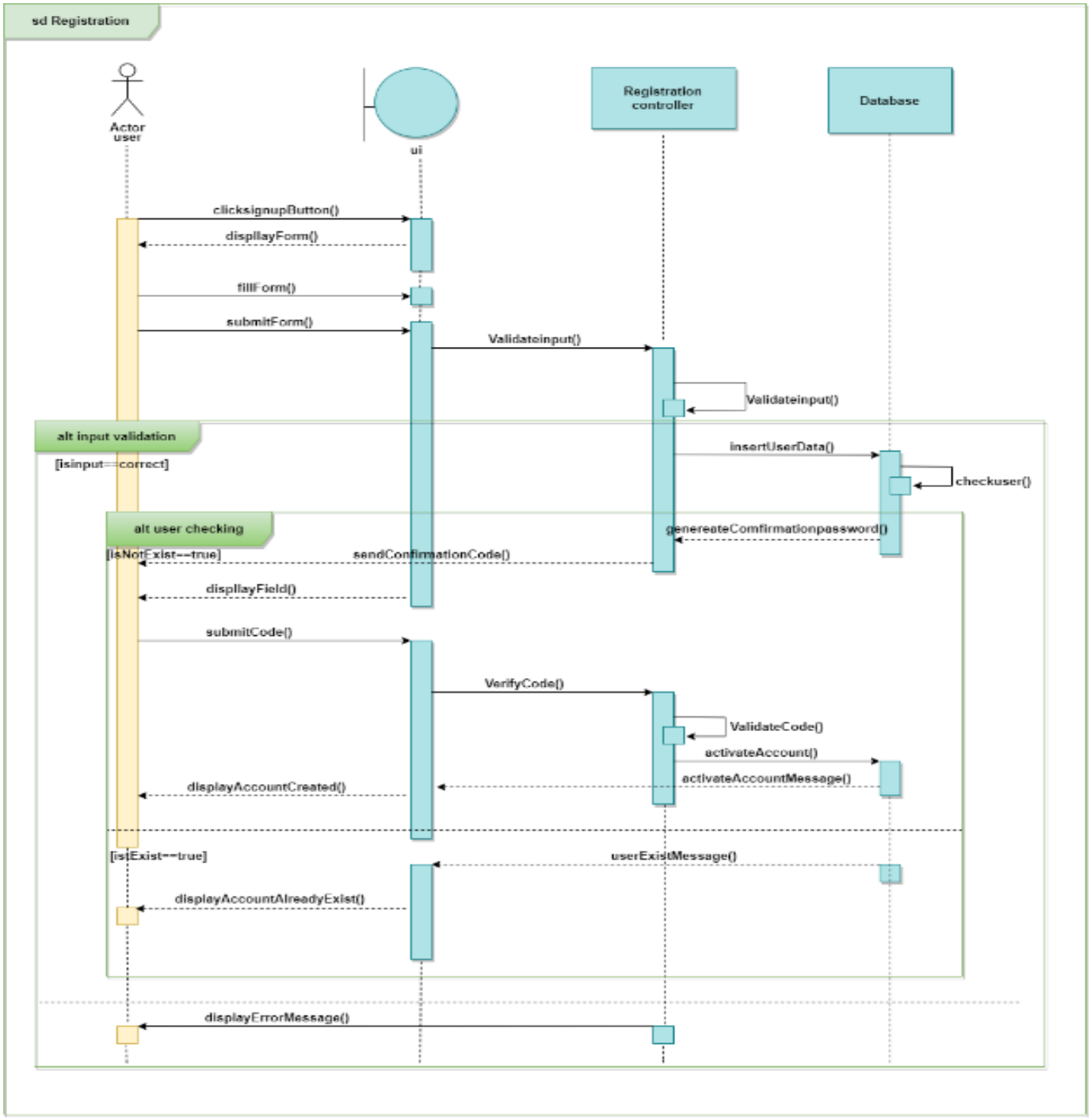

Figure 4- 4: Sequence Diagram Registration

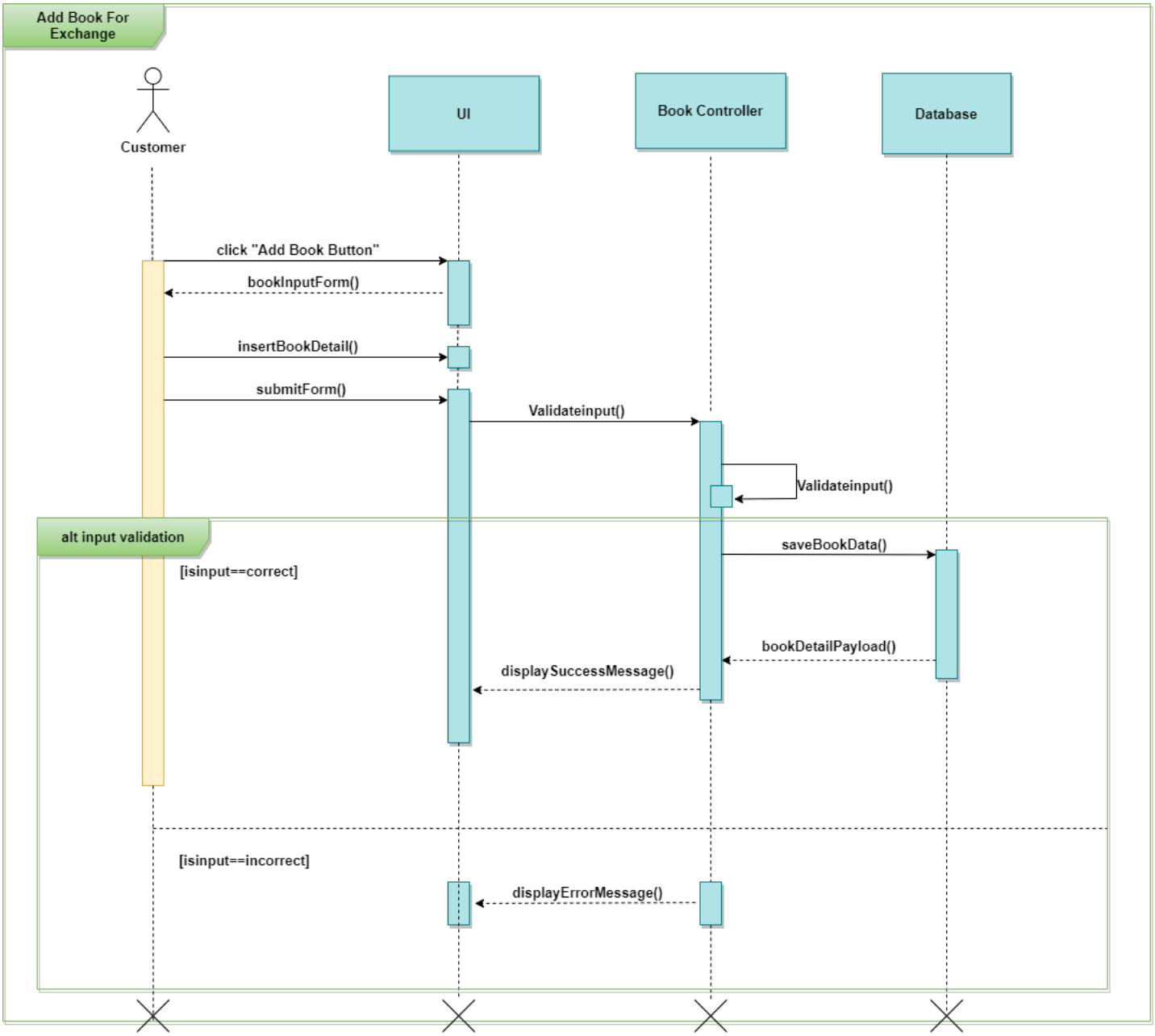

Figure 4- 5: Sequence Diagram add book for Exchange

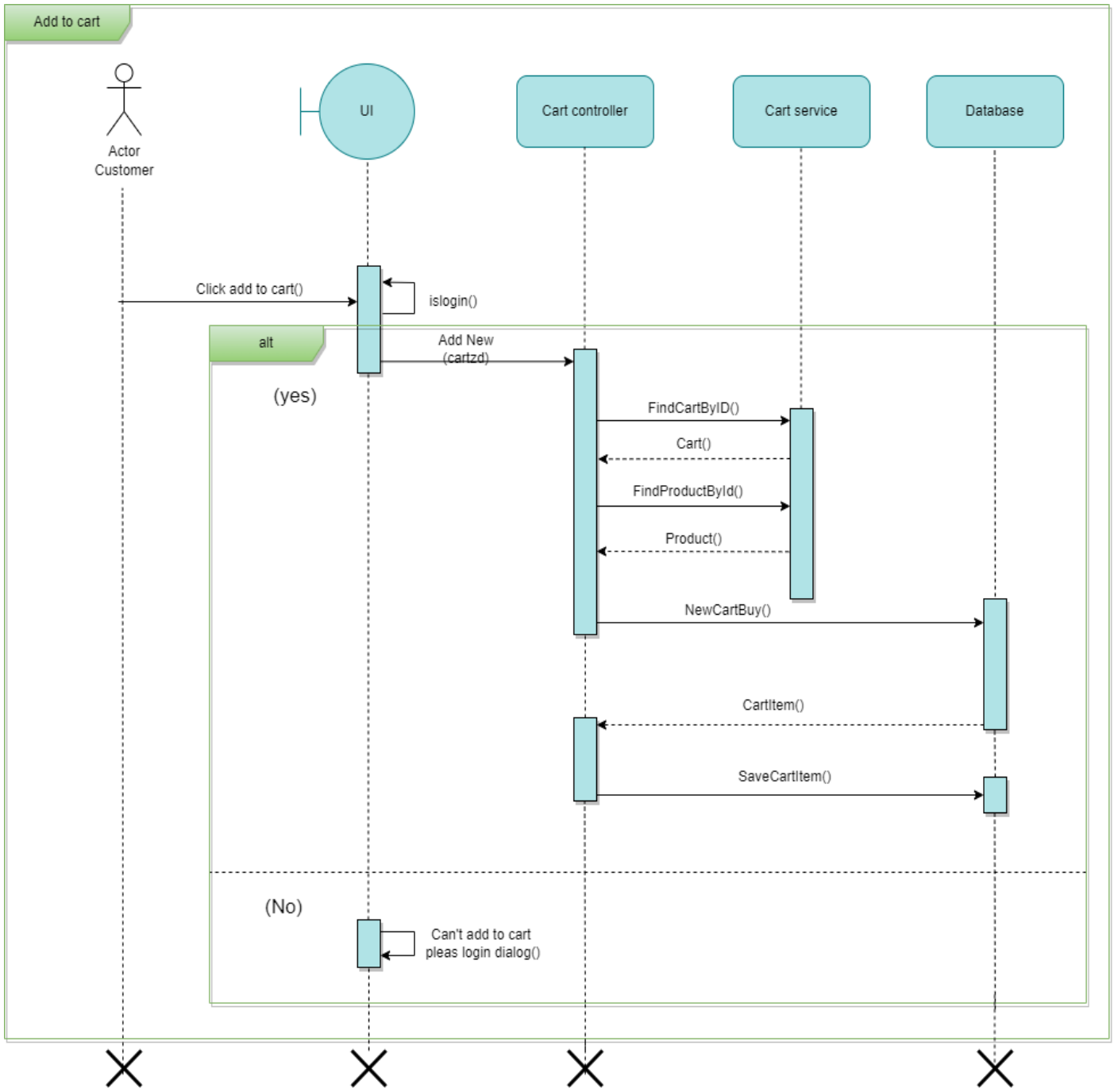

Figure 4- 6: Sequence Diagram add to cart

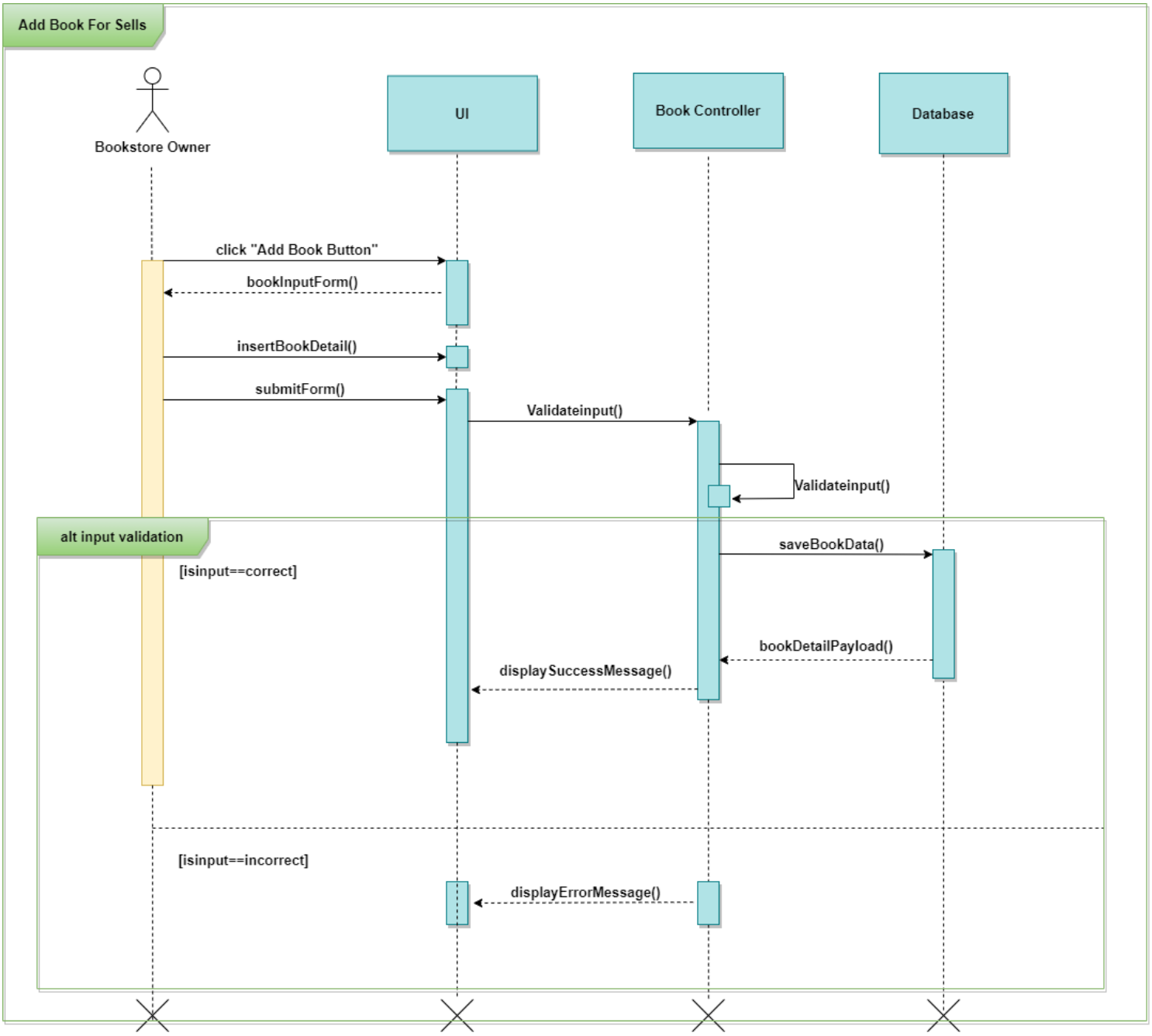

Figure 4- 7: Sequence Diagram Add Book for sells

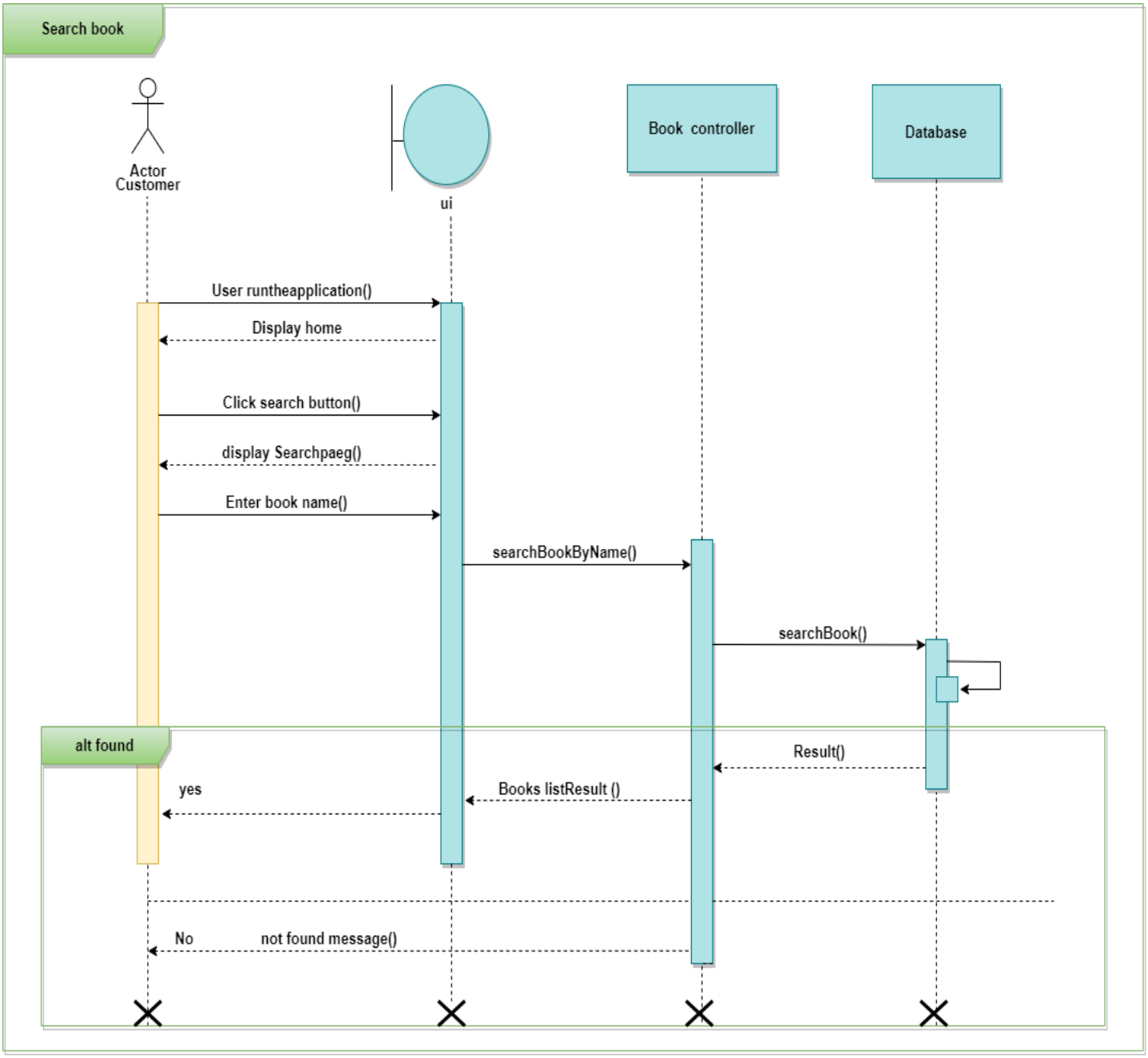

Figure 4- 8: Sequence Diagram search Book

4.3.2. Activity Diagram

We used activity diagram to show the flow of control or object with emphasis on the sequence and conditions of the flow. The actions coordinated by activity models can be initiated because other actions finish executing because objects and data become available, or because some events external to the flow occur. [13]

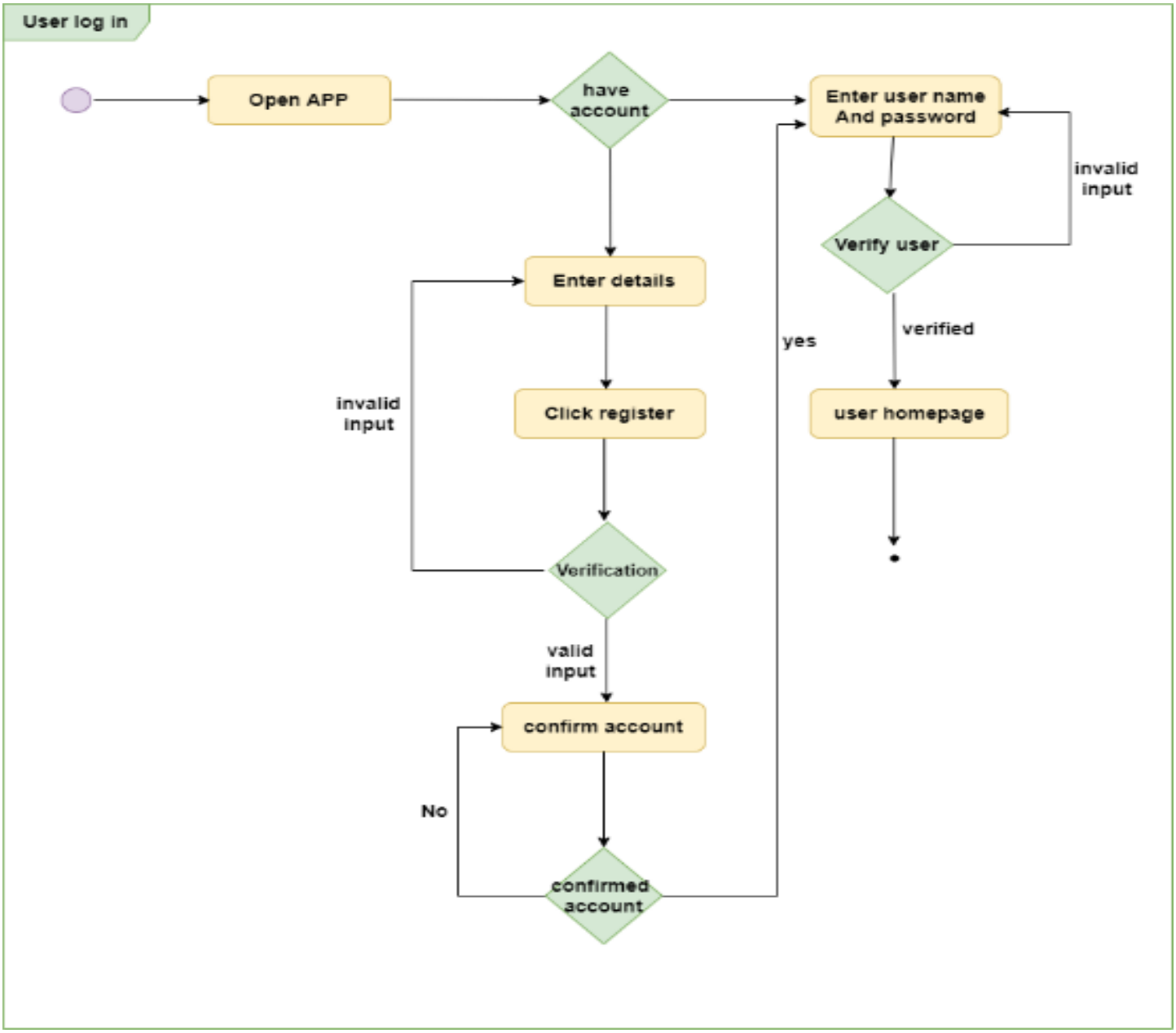

Figure 4- 9: Activity Diagram user login

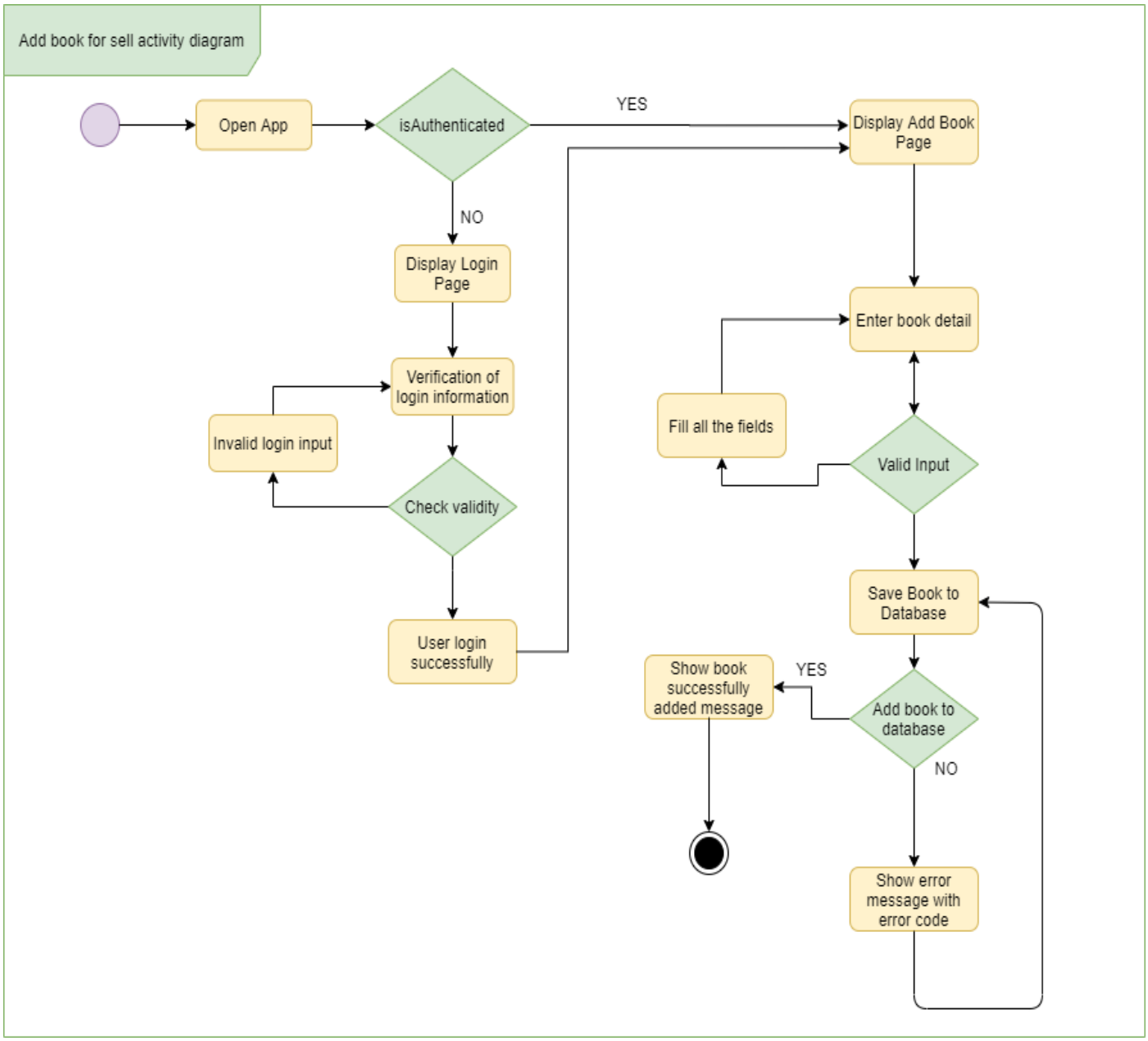

Figure 4- 10: Activity Diagram Add book for sells

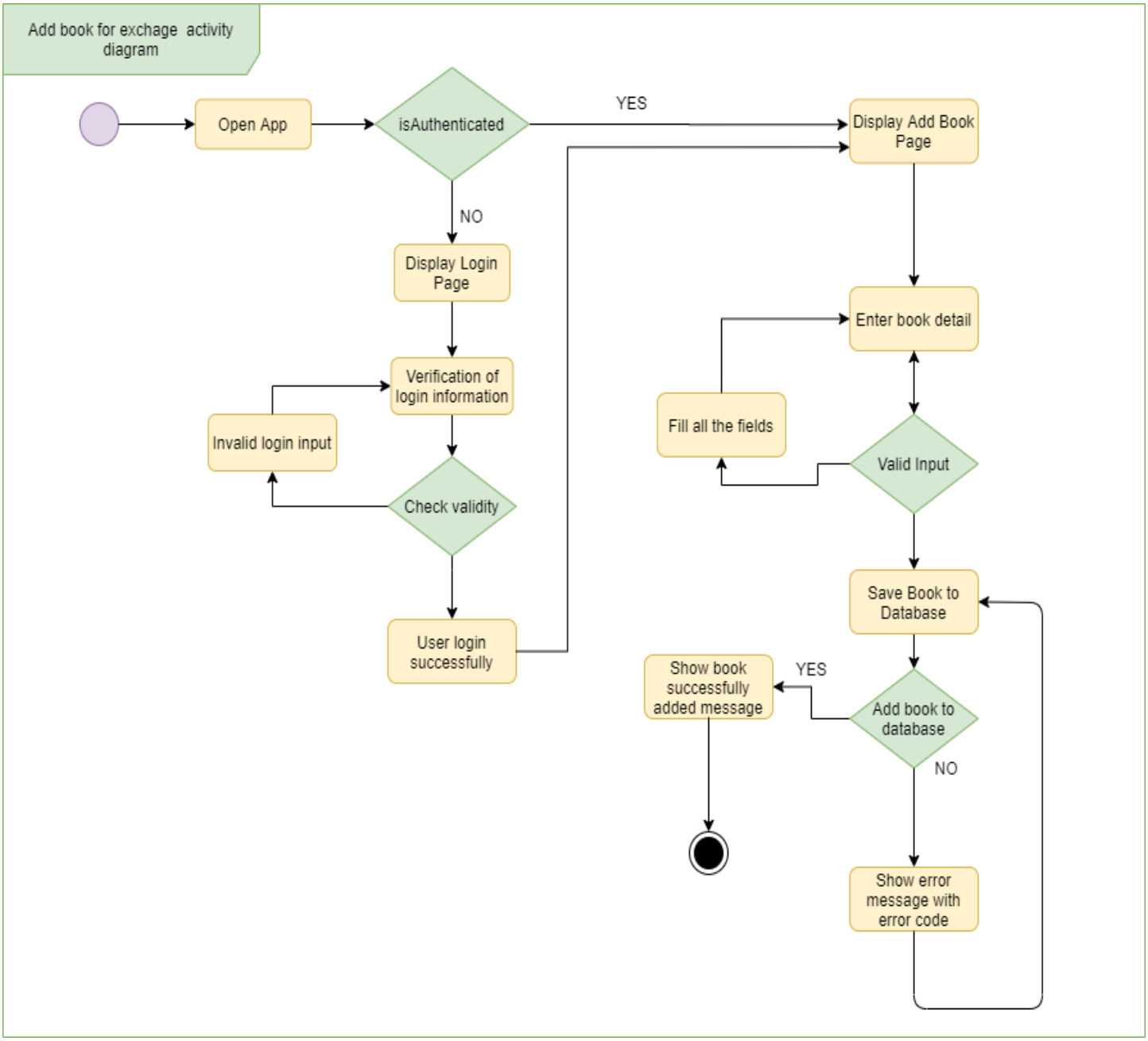

Figure 4- 11: Activity Diagram Add book for exchange

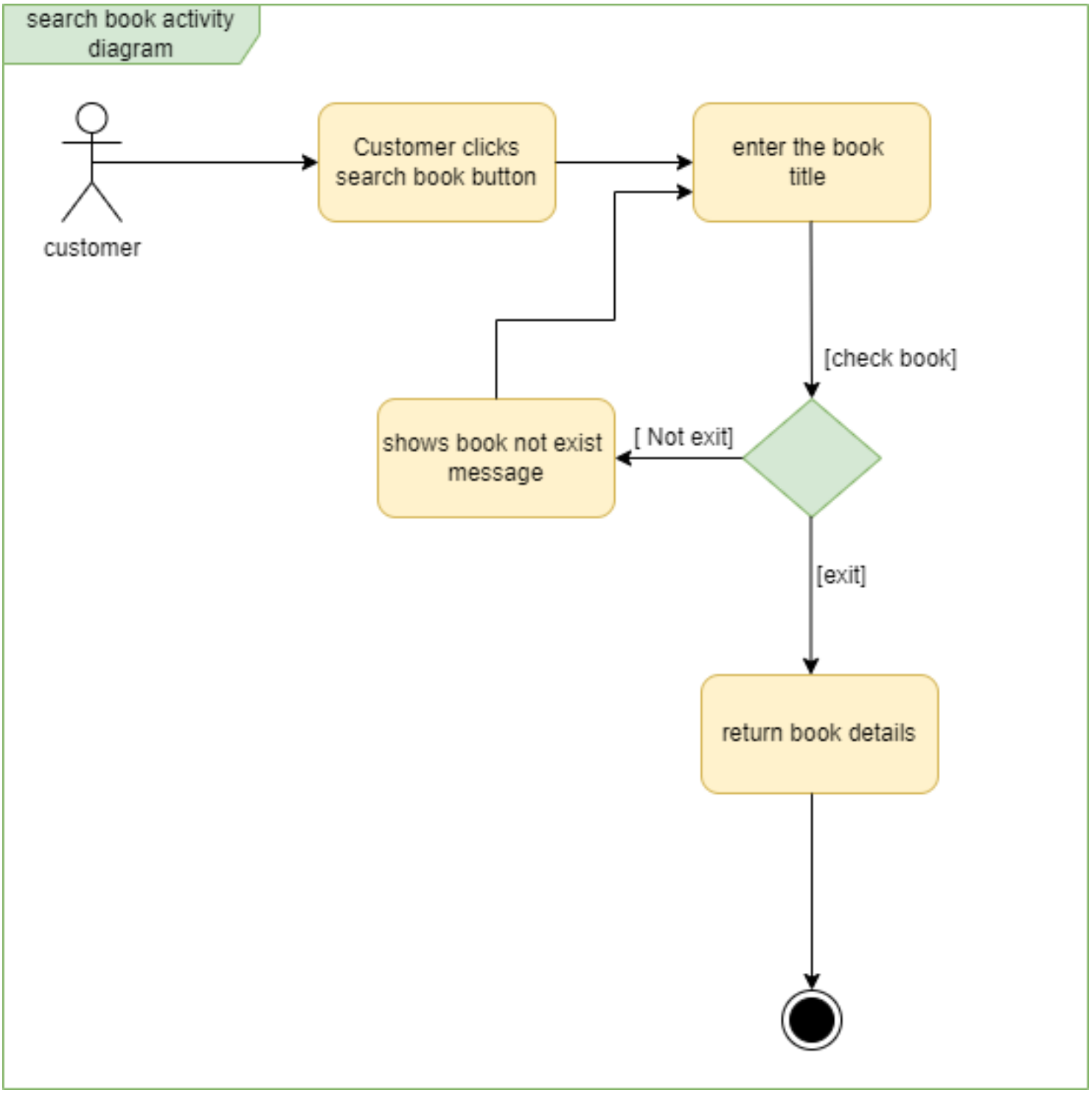

Figure 4- 12: Activity Diagram search book

4.3.3. State Chart Diagram

A state chart diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State chart diagrams are also referred to as State machines and State Diagrams. These terms are often used interchangeably. So simply, a state chart diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli. We can say that every class has a state but we don't model every class using State chart diagrams. We prefer to model the states with three or more states. [14]

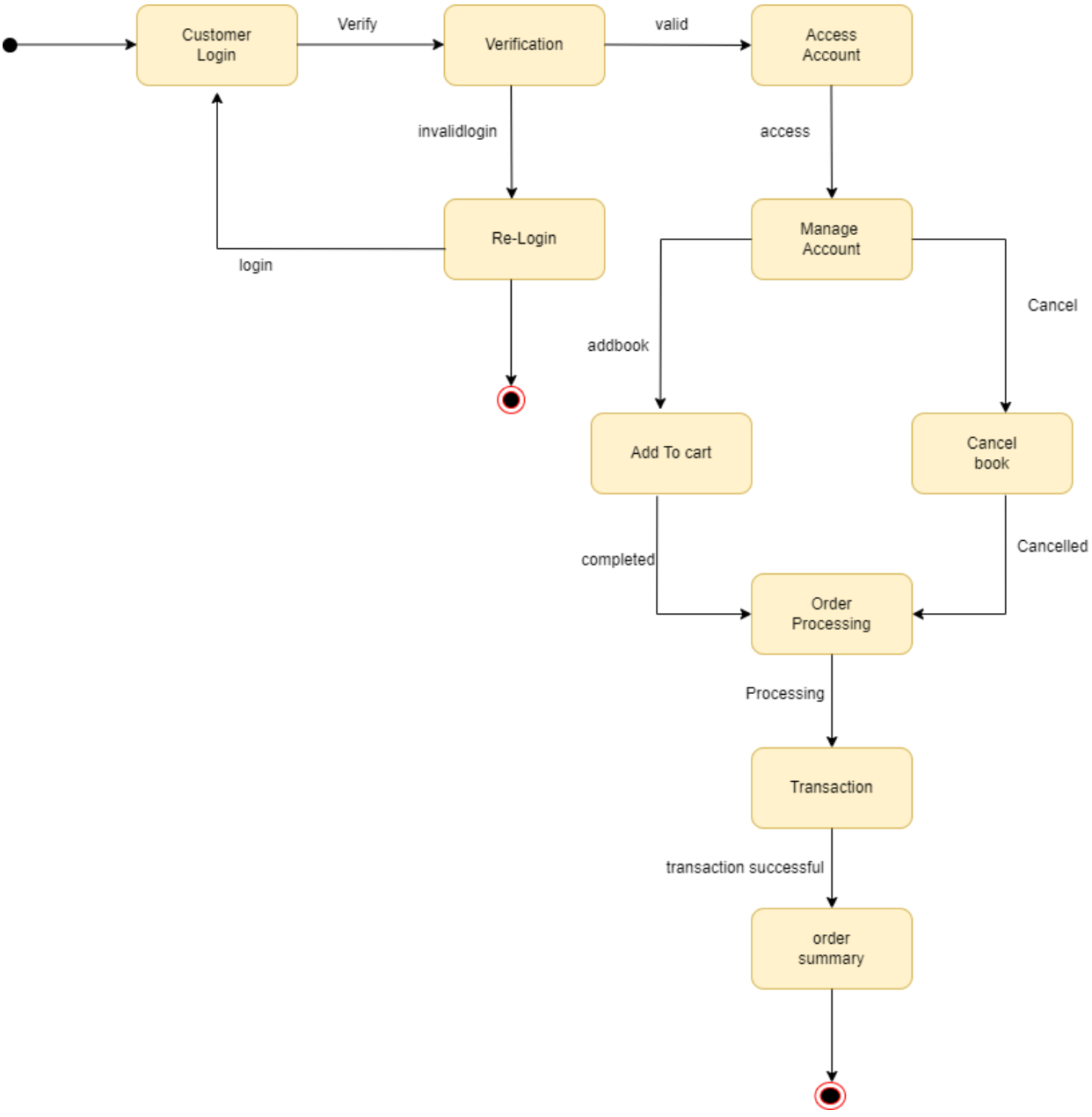

Figure 4- 13: State Chart Diagram log in

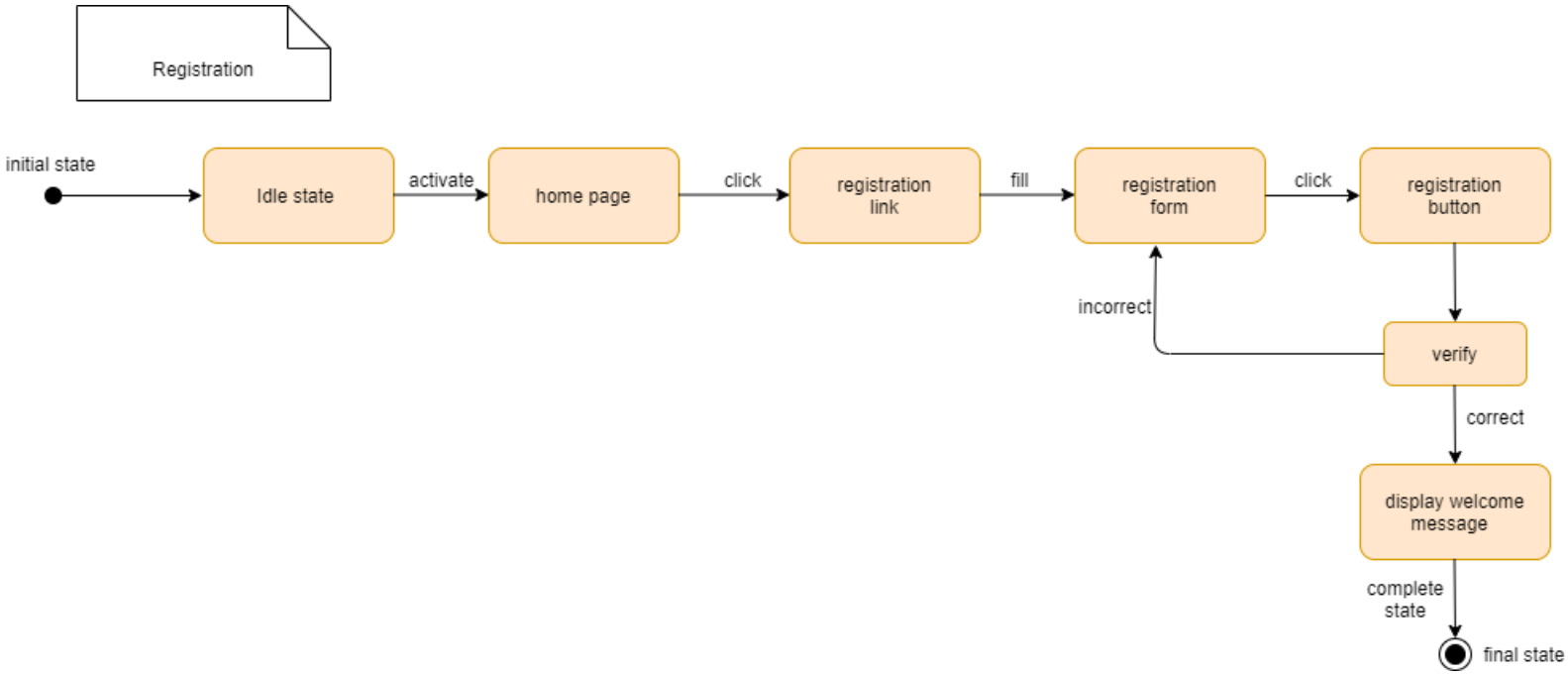

Figure 4- 14: State Chart Diagram Registered

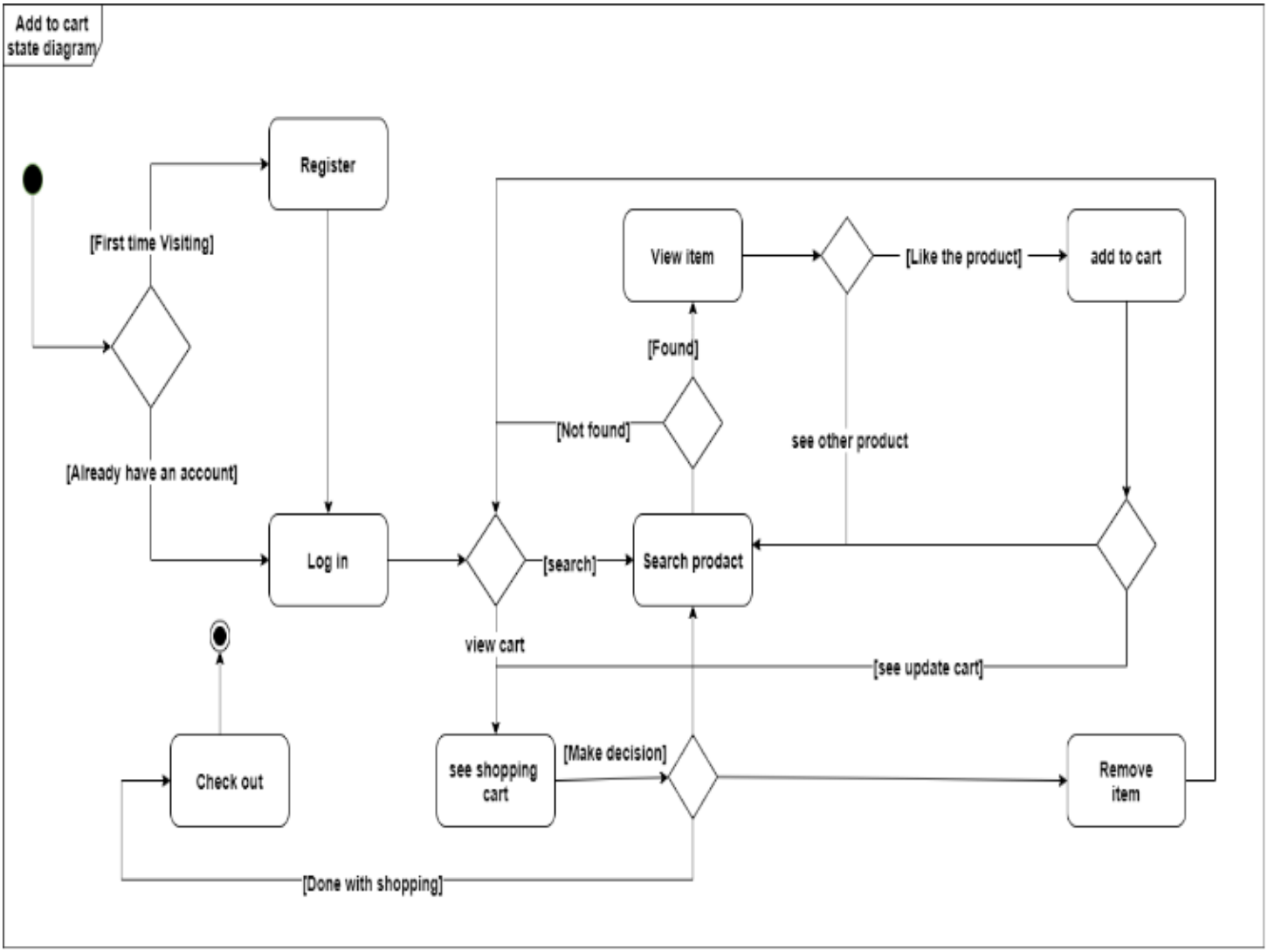

Figure 4- 15: State Chart Diagram add to cart

CHAPTER FIVE: SYSTEM DESIGN

5.1. INTRODUCTION

In an Android-based bookstore and exchange platform system our system design modeling will fill the gap between the system specification produced during requirement elicitation and analysis which is concentrated on the purpose and the functionality of the Android-based bookstore and exchange platform system. In the design phase of the Android-based bookstore and exchange platform system, system decomposition, component modeling, database design, class mapping, deployment diagram and the exact architecture of the proposed system which is the user interface prototyping will be showed in detail.

5.2. Design Goals

In our system development process system design part is very important to make the implementation of the proposed system very easy. The different types of the system modeling techniques that are used to make easy the implementation of the system such as deployment and component modeling are shown in detail. Not only the system modeling techniques but also some system design techniques such as system decomposition design are cover in detail in this phase.

Some of the design goals are: -

- **Security-** The system should be secured that unauthorized users cannot access the data that does not concern them. This prevents the entrance of some malicious data in to our system
- **Accessibility-** Accessibility One of the best feature of the proposed system is its accessibility. Users can access the current information being everywhere in the country as well as on the internet.

  To trace some of its best features related with accessibility like: -

  • It's accessible without any geographical location limitation

  • It is accessible without time limitation – user can share their idea every time.

  • Information or the same information accessed by multiple users at the same time.
- **Fault Tolerance: -** The system should be able to give a response (error message) when the user enters incorrect input. This recommends the user enter the correct input.

- **Robustness: -** The system can survive wrong applicant inputs. Besides this end, applicants that use an Android-based bookstore and exchange platform have limited access to info about the customers (like phone no, email address).
- **Modifiability: -** The system should be modifiable to modify different services depending on the need of the user. The system can be modified by adding some addition futures like book genres.
- **Usability: -** Android-based bookstore and exchange platform provides an easy user-friendly interface for users of the systems. It also provides a help menu that gives a brief description of how to use the system so that users can be able to use it easily.
- **Memory: -** Android-based bookstore and exchange platform requires the following space to run the system. Any smartphone with an Android 4.0+ operating system, 1GB ram and 50MB available space. Internet connection is also required.

5.3. Proposed System Architecture

The system architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

We use Three-tier architecture for our system. Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

Architectural style

Layered – Client Server

Client-server architecture is an architecture of a computer network in which many clients or remote processors request and receive service from a centralized server or host computer. Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them.

Our system implements client-server architecture in which clients or users request and servers respond. This architectural style provides a secure, accessible, and reliable distributed application structure.

For example, when a customer tries to tries to find a book to buy, they enter the book name and presses search button, then the client sends a request to the server to find the search key in the database then the server replies the result to the client and displays it to the user screen. Similar processes are handled using this mechanism.

Architectural pattern

Model-View-Controller (MVC)

The Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Each architecture component is built to handle specific development aspects of an application.

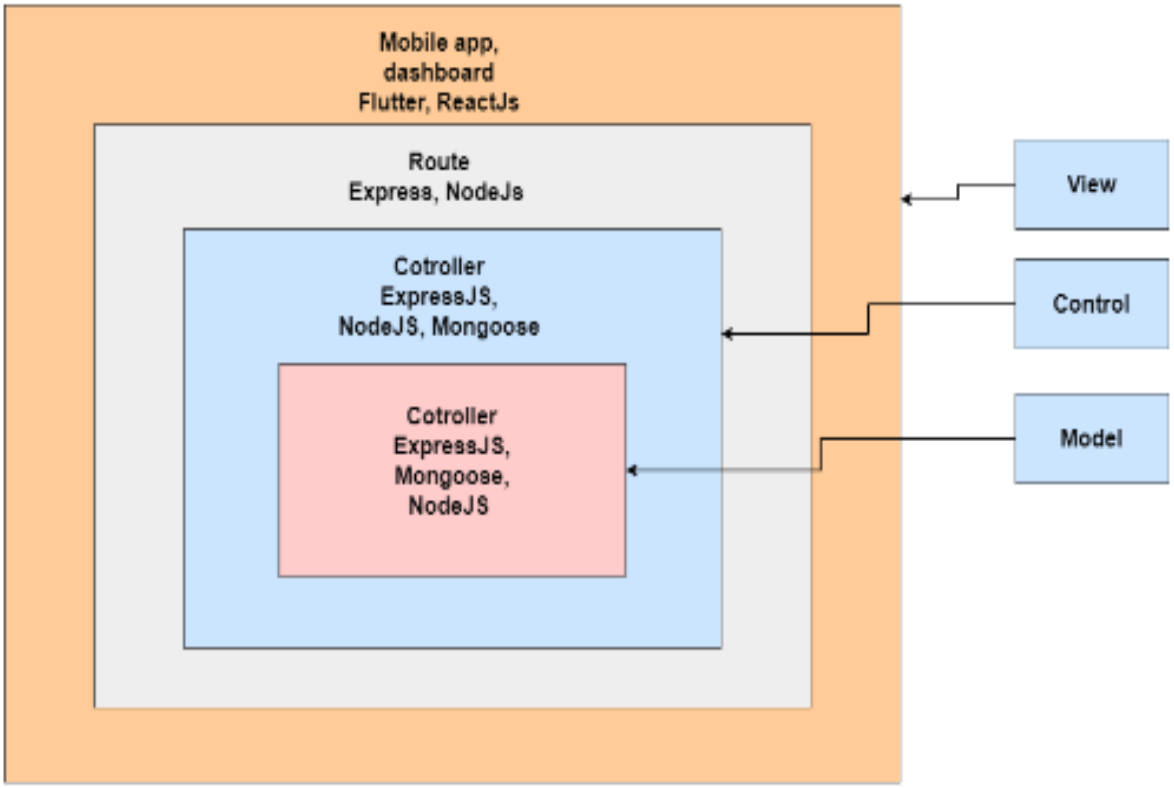

Figure 5- 1: Model view control diagram

Model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

View

It means a presentation of data in a particular format, triggered by a controller's decision to present the data.

Controller

The controller is responsible for responding to the user input and performing interactions on the data model objects. The controller receives the input, validates the input, and then performs the business operation that modifies the state of the data model.

5.3.1. Subsystem Decomposition and Description

System decomposition begins by decomposing the system into cohesive, well-defined subsystems. Subsystems are then decomposed into cohesive, well-defined components. Our system components are decomposed into cohesive, well-defined sub-components as follows:

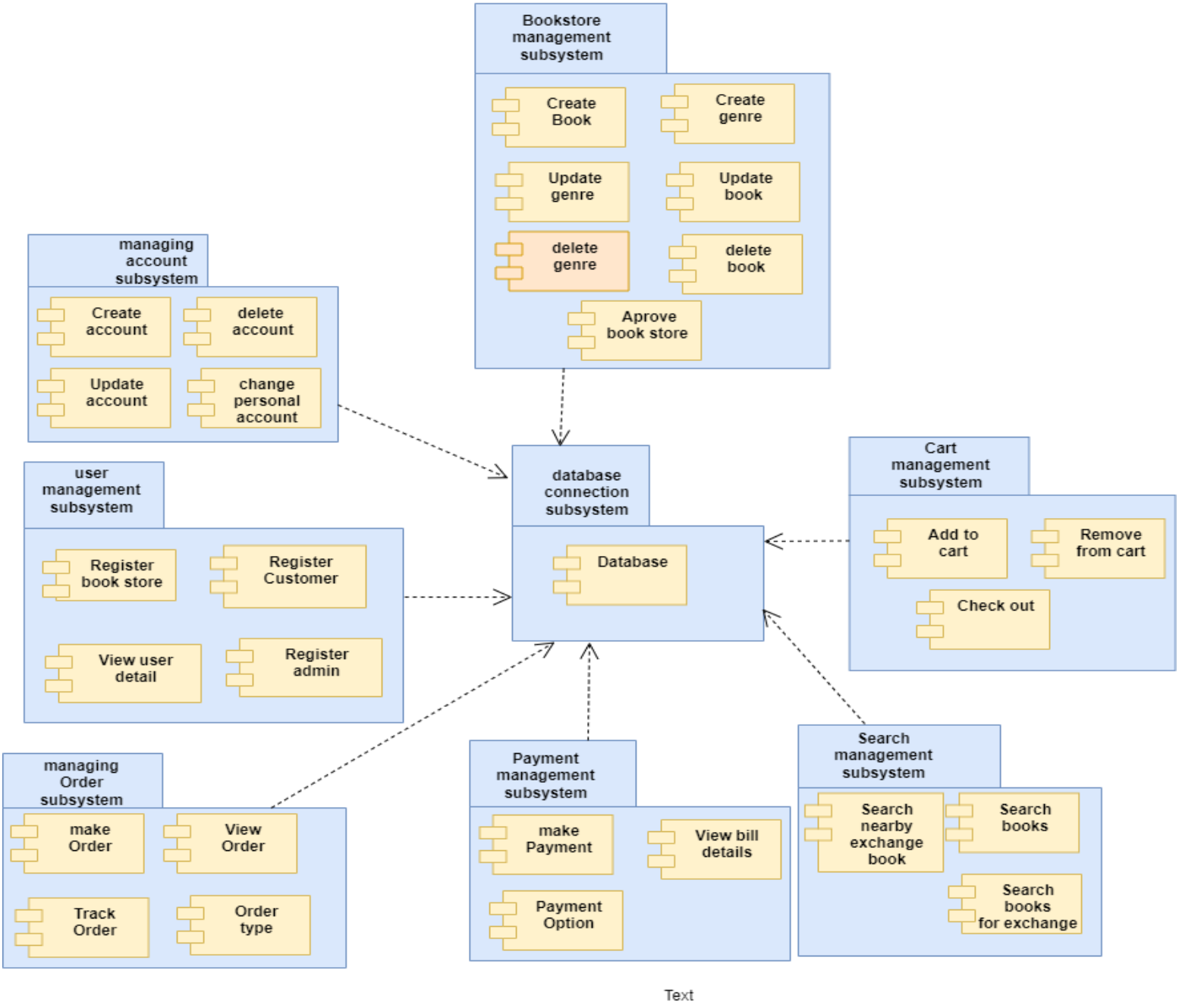

Figure 5- 2: Subsystem Decomposition

5.3.2. Hardware/Software Mapping

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes, such as, hardware or software execution environments, and the middleware connecting them. Deployment diagrams are used to describe the static deployment view of a system. As in the diagram below we have depicted how we want to deploy our proposed system.

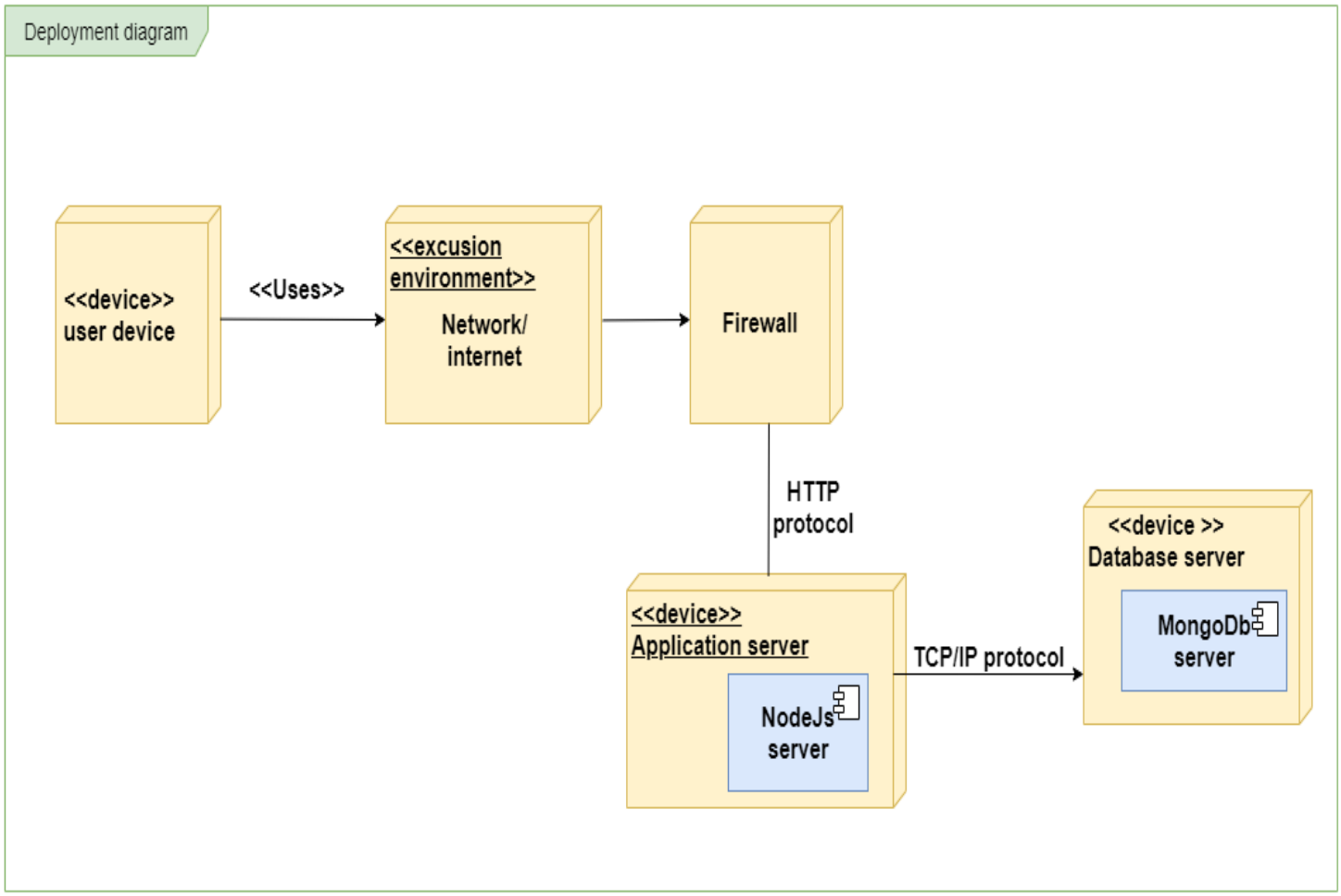

Figure 5- 3: Deployment diagram

5.3.3. Detailed Class Diagram

The class diagram is the main building block of object-oriented modeling. We used it for general conceptual modeling of the structure of our application, and for detailed modeling, translating the models into programming code. Class diagrams can also be used for data modeling.

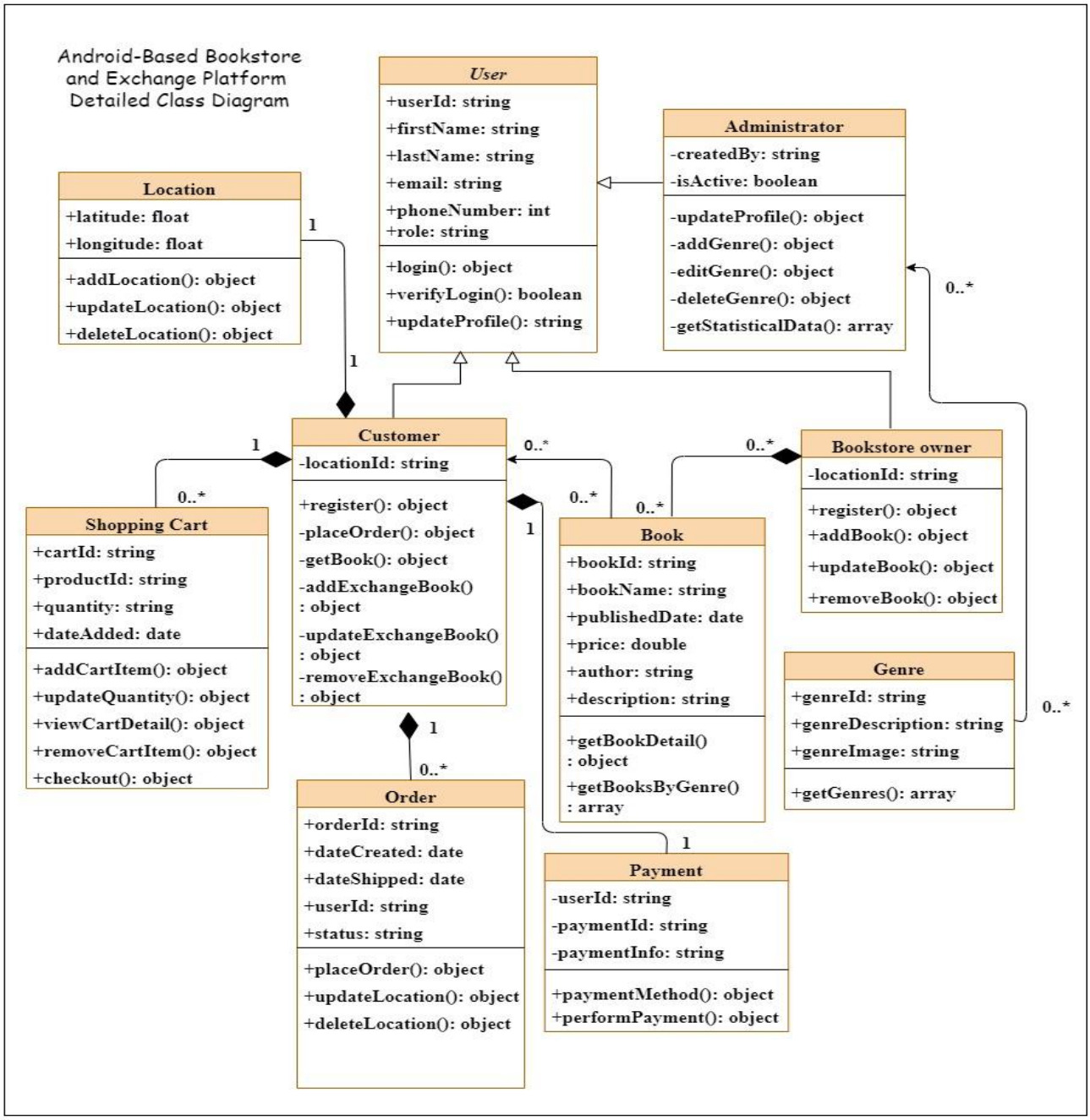

Figure 5- 4: Detailed Class diagram

5.3.4. Persistent Data Management

A database is a collection of information that is organized so that it can be easily accessed, managed, and updated. Database design is the organization of data according to a database model. It involves classifying data and identifying interrelationships. It is the activity of representing classes, attributes, and relationships in a database. Entity-Relationship (ER) diagram, also known as an entity-relationship model, is a graphical representation of an information system that depicts the relationship among people, objects, places, concepts, or events within that system. It is a data modeling technique that can help define business processes

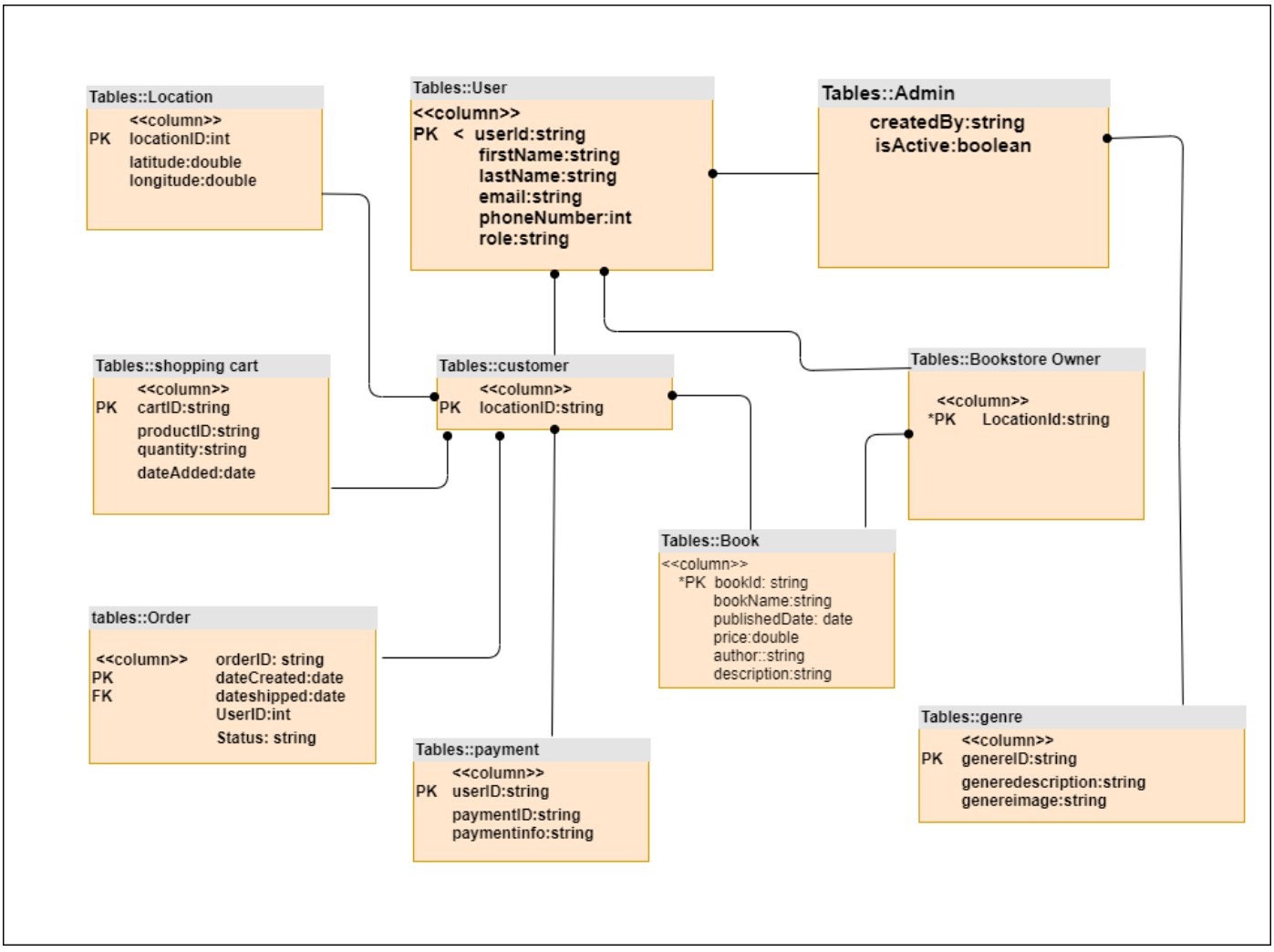

Figure 5- 5: Persistent Database Diagram

5.3.5. Access Control and Security

Access control is a fundamental component of data security that dictates who's allowed to access and use company information and resources. In our case we have three roles and each roles have different access controls. Through authentication and authorization, our access control policies make sure users are who they say they are and that they have appropriate access to the system.

Table 4-27: Access control with role table

Operations	**Roles with access rights**		
	Customer	**Bookstore owner**	**Admin**
Sign up	√	√	√
Sign in	√	√	√
Reset password	√	√	√
Add book for exchange	√		
Add book for sell		√	
Get data about the bookstore		√	
View book	√	√	√
Update book		√	
Delete book		√	
Exchange book	√		
Search book	√	√	√
Get bookstores	√		√
Approve bookstore			√

Get data about the app			√
Manage customers		√	√
Manage bookstore owners			√
Register admins			√

5.4. Packages

In these sections, the decomposition of subsystems into packages and an overview of each packagedependencies with other packages can be depicted. A UML package diagram is a style of diagramthat only shows packages and the dependencies between them.

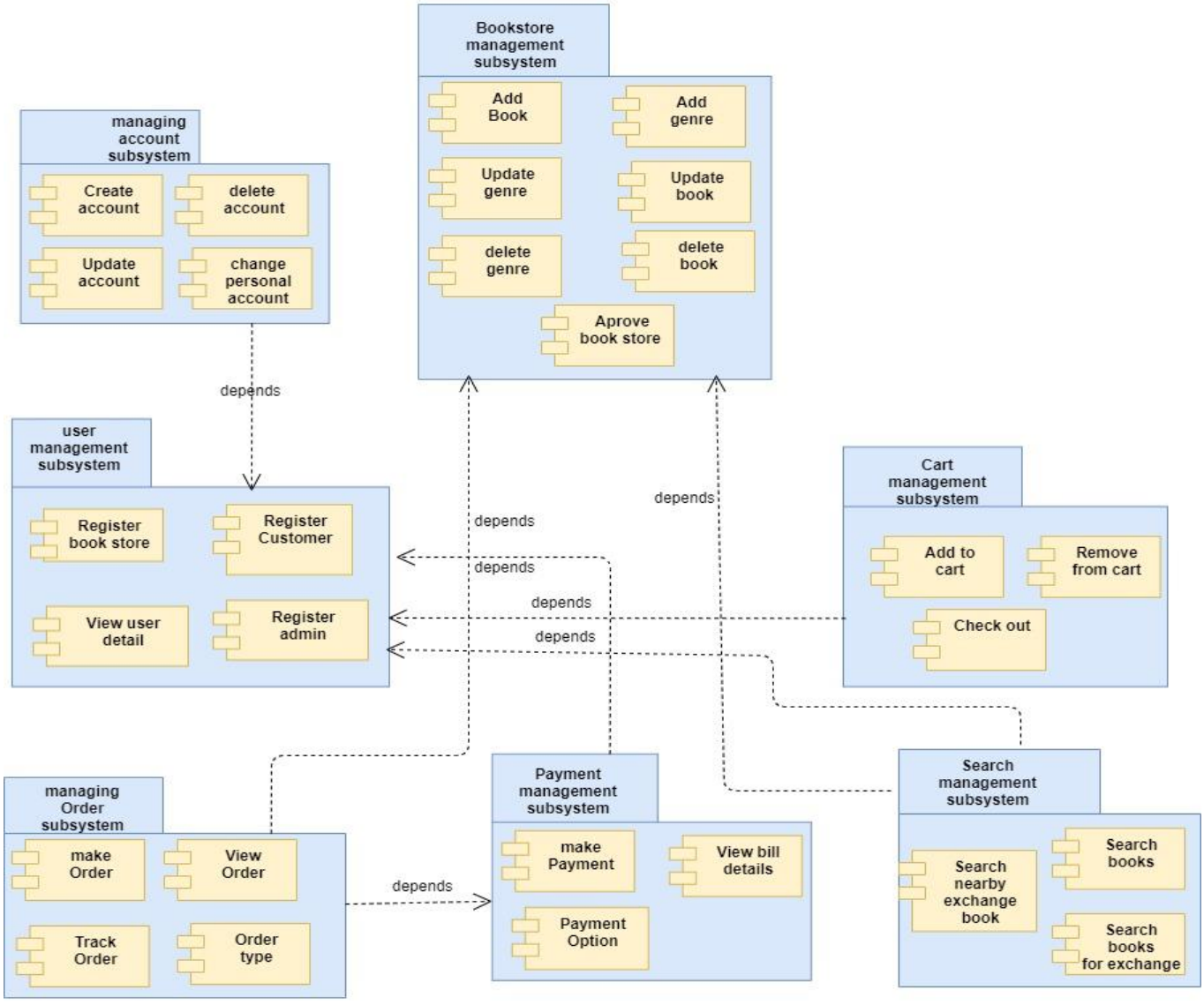

Figure 5- 6: Package diagram

5.5. Algorithm design

Some algorithms of our project are the following.

Algorthm1: Login ()

This method enables the user to log in to the system:

BEGN

Login (username, password)

INPUT: Username and Password97

IF (User exist)

THEN

READ Password FROM database

IF (Password == Entered Password)

Login successful

ELSE

PRINT “incorrect password “

END IF

ELSE

PRINT “incorrect Username or password “

END IF

END

Algorithm2: CreateAccount ()

This method enables the user to create an account.

BEGIN

CreateAccoun(firstName, lastName, password, phoneNumber)

INPUT: (firstName, lastName, password, phonNumber) into the database

IF (user exists)

PRINT “Account already exists try again with other email or phone”

ELSE

PRINT “Successfully Created!”

ENDIF

END

Algorithm3: AddBook()

This method enables the user to create an account.

BEGIN

AddBook(bookName, author, publishedDate, price, description)

INPUT: (bookName, author, publishedDate, price, description) into the database

IF (bookAdded)

PRINT “Book successfully added”

ELSE

PRINT “Error adding a book!”

ENDIF

END

Algorithm4: RequestExchangeBook()

This method enables to exchange books with users

BEGIN

RequestExchangeBook (bookId, exchangerId, exchangeRequesterId)

INPUT: (bookId, exchangerId, exchangeRequesterId) into the database

```
SEND: request(exchangerId, exchangeRequesterId)
IF (bookApproved)
    PRINT “Exchange request approved”
ELSE
   PRINT “Exchange request disapproved”
ENDIF
     END
```

5.6. User interface design

The proposed system has a graphic user interface to interact with the user. Below different user interfaces that are visible to all users of the system are described. In addition, can get information posted and different links to other pages.

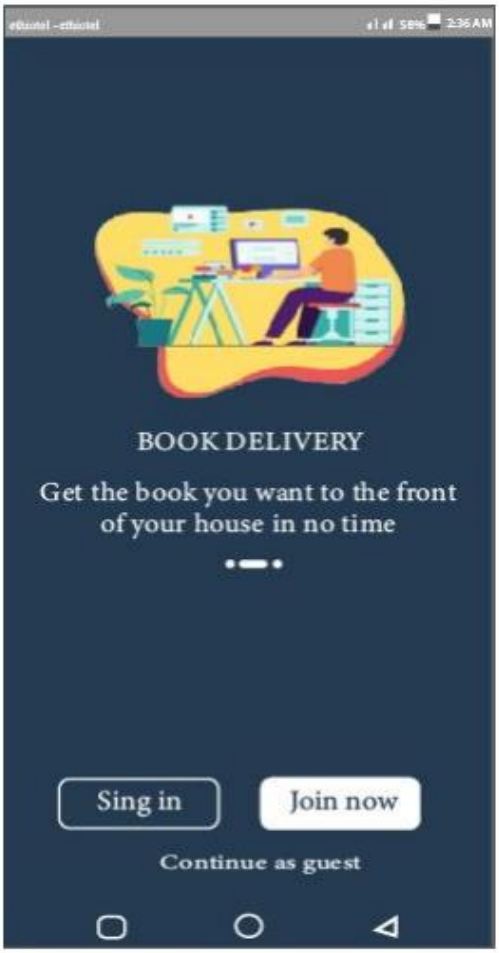

Figure 5- 7: Dark mode Splash screen UI

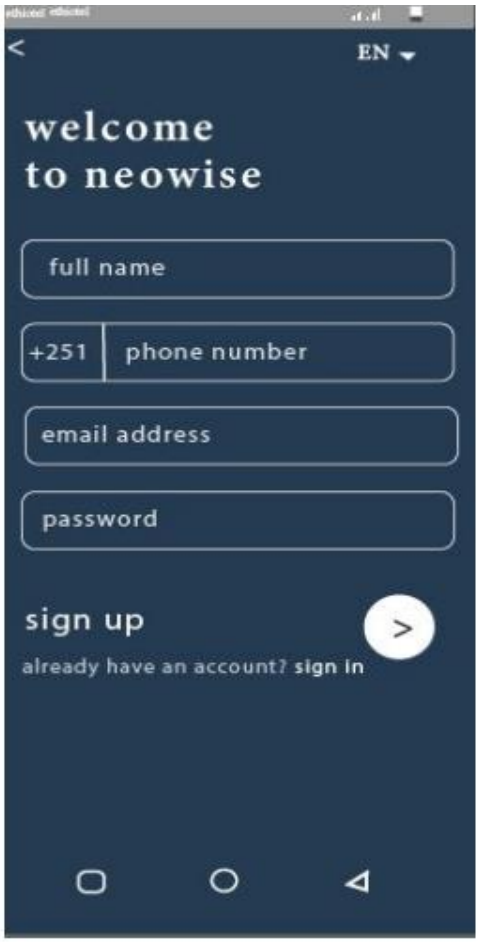

Figure 5- 8: Dark mode signup screen

Figure 5- 9: Dark mode Home screen UI Figure 5- 10: Dark mode product detail screen UI

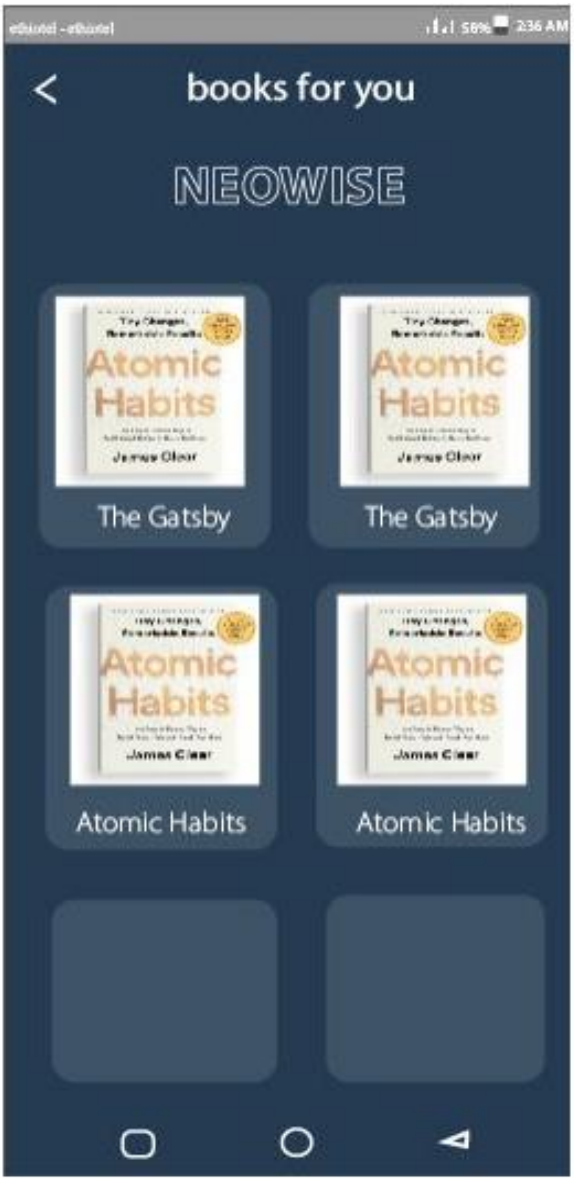

Figure 5- 11: Dark mode genre screen UI Figure 5- 12: Dark mode books for you screen UI

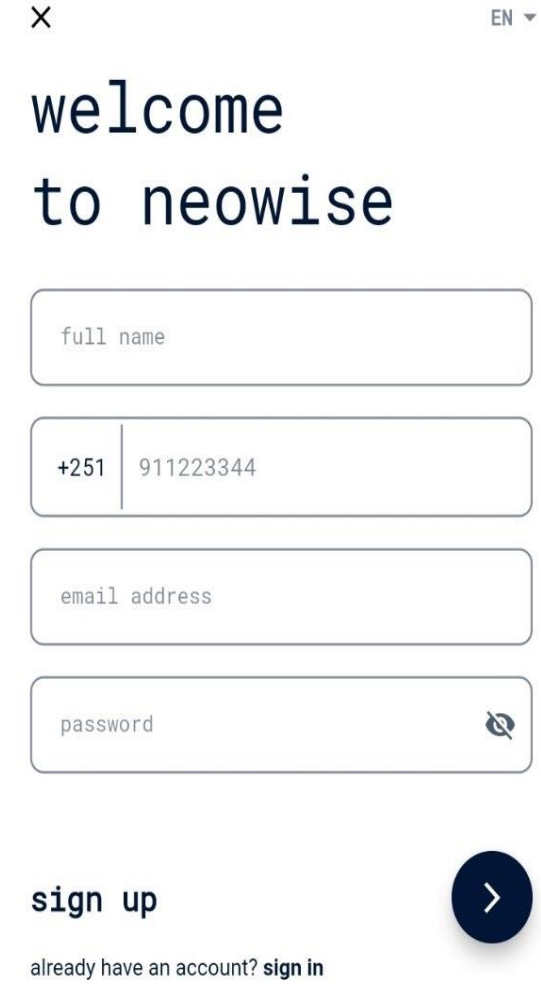

Figure 5- 13: Light mode sign up screen UI

Figure 5- 14: Light mode trending books screen UI

CHAPTER SIX: IMPLEMENTATION AND TESTING

During this phase physical design specification must be turned into working computer code, and provide help for current and future users and take care of the system. And then the code is tested until most of the errors have been detected and corrected. The purpose of this activity is to convert the final physical system specification into working model with reliable software and hardware

6.1. Implementation of Database

We use Mongo DB database to manage our system which assists to insert, update, delete and view data which is stored in the database.

Book Model

```
import mongoose from 'mongoose'
const bookSchema = mongoose.Schema(
 {
  title: {
   type: String,
   required: true,
  },
  description: {
   type: String,
   required: true,
  },
  author: {
   type: String,
   required: true,
  },
  genre: {
   type: [String],
   required: true,
  },
  frontCover: {
   type: String,
```

```
  required: true,
},
backCover: {
  type: String,
  required: true,
},
price: {
  type: Number,
  required: true,
},
pageCount: {
  type: Number,
  required: true,
},
seenCount: {
  type: Number,
  default: 0,
},
 status: {
   type: String,
   enum: ['pending', 'approved'],
  default: 'pending',
 },
bookstore: {
  type: mongoose.SchemaTypes.ObjectId,
  required: true,
  ref: 'Bookstore',
},
publishedAt: {
  type: String,
  required: true,
```

```
    },
  },
  { timestamps: true }
)
const Book = mongoose.model('Book', bookSchema)
 export default Book
```

User Model

```
import mongoose from 'mongoose'
import bcrypt from 'bcryptjs'
import locationSchema from './locationSchema.js'

const userSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      lowercase: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    phone: {
      type: String,
      required: true,
      unique: true,
```

```
  },
  profilePicture: {
    type: String,
  },
  locationName: {
    type: String,
    default: 'Unknown',
  },
  telegramUsername: {
    type: String,
    required: false,
  },
  firebaseUserId: {
    type: String,
    required: false,
  },
  favourites: [
    {
      type: mongoose.SchemaTypes.ObjectId,
      ref: 'Book',
      addedDate: Date.now(),
    },
  ],
  location: locationSchema,
  role: {
    type: String,
    enum: ['user', 'bookstore', 'admin'],
    default: 'user',
  },
},
{ timestamps: true }
```

```
)

userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password)
}

userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) {
    next()
  }

  const salt = await bcrypt.genSalt(10)
  this.password = await bcrypt.hash(this.password, salt)
})

userSchema.index({
  location: '2dsphere',
})

const User = mongoose.model('User', userSchema)

export default User
```

6.2. Implementation of Class Diagram

Implementation of exchange book class

```
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:get/get.dart';
import 'package:neowise/models/exchangeBookModel.dart';
import 'package:neowise/services/exchangeBookServices.dart';
import 'package:neowise/services/localStorageServices.dart';
import 'package:neowise/services/locationServices.dart';

class ExchangeBooksListController extends GetxController {
  final LocationServices _locationServices = LocationServices();
```

```
final LocalStorageServices _localStorageServices = LocalStorageServices();
final ExchangeBookServices _exchangeBookServices = ExchangeBookServices();
RxList<ExchangeBookModel> nearbyBooks = <ExchangeBookModel>[].obs;
RxBool isLoading = true.obs;
String lat = '';
String lng = '';

Future getExchangeBooksList() async {
  isLoading.value = true;
  var res = await _exchangeBookServices.getBooksForExchange(lat, lng);
  print(res);
  print('response');
  if (res != null) {
    if (res.statusCode == 200 || res.statusCode == 201) {
      var responseProperties = res.data;
      List<ExchangeBookModel> jsonResponse = (responseProperties as List)
          .map((book) => ExchangeBookModel.fromJson(book))
          .toList();
      nearbyBooks.clear();
      nearbyBooks.addAll(jsonResponse);
      isLoading.value = false;
    }
  } else {
    isLoading.value = false;
    Get.snackbar('Error!', 'Server error, please try again',
        snackPosition: SnackPosition.BOTTOM,
        backgroundColor: Colors.redAccent.shade100,
        margin: EdgeInsets.all(5));
  }
}

getCurrentLocation() async {
  bool locationServiceEnabled = await _locationServices.checkServiceEnabled();

  if (!locationServiceEnabled) {
    var user = await _localStorageServices.getUserInfo();
    var location = user['location'];
    lng = location[0].toString();
    lat = location[1].toString();
    getExchangeBooksList();
  } else {
    Position location = await _locationServices.determinePosition();
    lat = location.latitude.toString();
    lng = location.longitude.toString();
    getExchangeBooksList();
  }
```

```
  }

  @override
  void onInit() {
    getCurrentLocation();
    super.onInit();
  }
}
```

Implementation of Exchange Books List Screen Body Class

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:liquid_pull_to_refresh/liquid_pull_to_refresh.dart';
import 'package:neowise/config/palette.dart';
import 'package:neowise/controllers/exchangeBooksListController.dart';
import 'package:neowise/routes/appRoutes.dart';
import 'package:neowise/screens/exchanegeBooksList/components/exchangeBookCard.dart';

class ExchangeBooksListScreenBody extends StatelessWidget {
  ExchangeBooksListScreenBody({Key? key}) : super(key: key);
  final ExchangeBooksListController controller =
      Get.put(ExchangeBooksListController());
  @override
  Widget build(BuildContext context) {
    return Obx(() => Scaffold(
          body: LiquidPullToRefresh(
            onRefresh: controller.getExchangeBooksList,
            child: ListView.builder(
                itemCount: controller.nearbyBooks.length,
                shrinkWrap: true,
                physics: BouncingScrollPhysics(),
                itemBuilder: (context, index) =>
                    ExchangeBookCard(book: controller.nearbyBooks[index])),
          ),
          floatingActionButton: Padding(
            padding: const EdgeInsets.all(8.0),
            child: FloatingActionButton.extended(
                onPressed: () => Get.toNamed(AppRoutes.ADDEXCHANGEBOOK),
                label: Text('Exchange Book')),
          ),
        ));
  }
}
```

Implementation of Exchange Book Services Class

```
import 'package:dio/dio.dart';
import 'package:neowise/services/constant.dart';
import 'package:neowise/services/localStorageServices.dart';

class ExchangeBookServices {
  final dio = Dio();
  final LocalStorageServices _localStorageServices = LocalStorageServices();

  Future getBooksForExchange(lat, lng) async {
    try {
      print('$BASE_URL/bookexchange/nearby/?lat=$lat&lng=$lng');
      Response response =
          await dio.get('$BASE_URL/bookexchange/nearby/?lat=$lat&lng=$lng');
      return response;
    } on DioError catch (error) {
      if (error.response != null) {
        return error.response;
      } else {
        error.message;
      }
    }
  }

  Future addBookForExchange(data) async {
    try {
      var token = await _localStorageServices.getToken();
      dio.options.headers['authorization'] = 'Bearer $token';
      Response response =
          await dio.post('$BASE_URL/bookexchange/add', data: data);
      return response;
```

```
  } on DioError catch (error) {
    if (error.response != null) {
      return error.response;
    } else {
      return error.message;
    }
  }
}

Future getBookDetail(id) async {
  try {
    print('$BASE_URL/bookexchange/detail/$id');
    Response response = await dio.get(
      '$BASE_URL/bookexchange/detail/$id',
    );

    return response;
  } on DioError catch (error) {
    return error.response;
  }
}

Future getMyExchangeBooks() async {
  try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    print('$BASE_URL/bookexchange/mybooks');
    Response response = await dio.get('$BASE_URL/bookexchange/mybooks');
    return response;
  } on DioError catch (error) {
    if (error.response != null) {
```

```
      return error.response;
    } else {
      return error.message;
    }
  }
}

Future deleteMyExchangeBook(id) async {
  try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    Response response = await dio.delete('$BASE_URL/bookexchange/delete/$id');
    return response;
  } on DioError catch (error) {
    if (error.response != null) {
      return error.response;
    } else {
      return error.message;
    }
  }
}

Future searchMyExchangeBooks(searchKey) async {
  try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    Response response = await dio.post('$BASE_URL/bookexchange/mybooks');
    return response;
  } on DioError catch (error) {
    if (error.response != null) {
      return error.response;
```

```
    } else {
      return error.message;
    }
  }
}

Future updateMyExchangeBook(data, id) async {
  try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    Response response =
        await dio.put('$BASE_URL/bookexchange/update/$id', data: data);
    return response;
  } on DioError catch (error) {
    if (error.response != null) {
      return error.response;
    } else {
      return error.message;
    }
  }
}

Future requestBook(data, id) async {
  try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    Response response = await dio.post('$BASE_URL/request/$id', data: data);
    return response;
  } on DioError catch (error) {
    if (error.response != null) {
      return error.response;
```

```
    } else {
      return error.message;
    }
   }
  }

  Future getMySentRequests() async {
   try {
    var token = await _localStorageServices.getToken();
    dio.options.headers['authorization'] = 'Bearer $token';
    Response response = await dio.get('$BASE_URL/request/sent');
    return response;
   } on DioError
catch (error) {
    if (error.response != null) {
      return error.response;
    } else {
      return error.message;
    }
   }
  }
}
```

6.3. Configuration of the Application server

We use Node.Js server because it built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

6.4. Configuration of Application security

Our system will use user validation during login to ensure that the user is valid or not. Access to the software is given only to valid operators. We need a username and password to get access to the software. And the system will have a modulus that prevents NoSQL injection attacks, HTTP

header attacks, XXS attack prevention. The System should have a high authentication function. while login into the system should generate a dynamic web token that is encrypted with the user's login information and encryption key which is stored securely and all user's API requests will be authorized according to that web token.

6.5. Implementation of User Interface

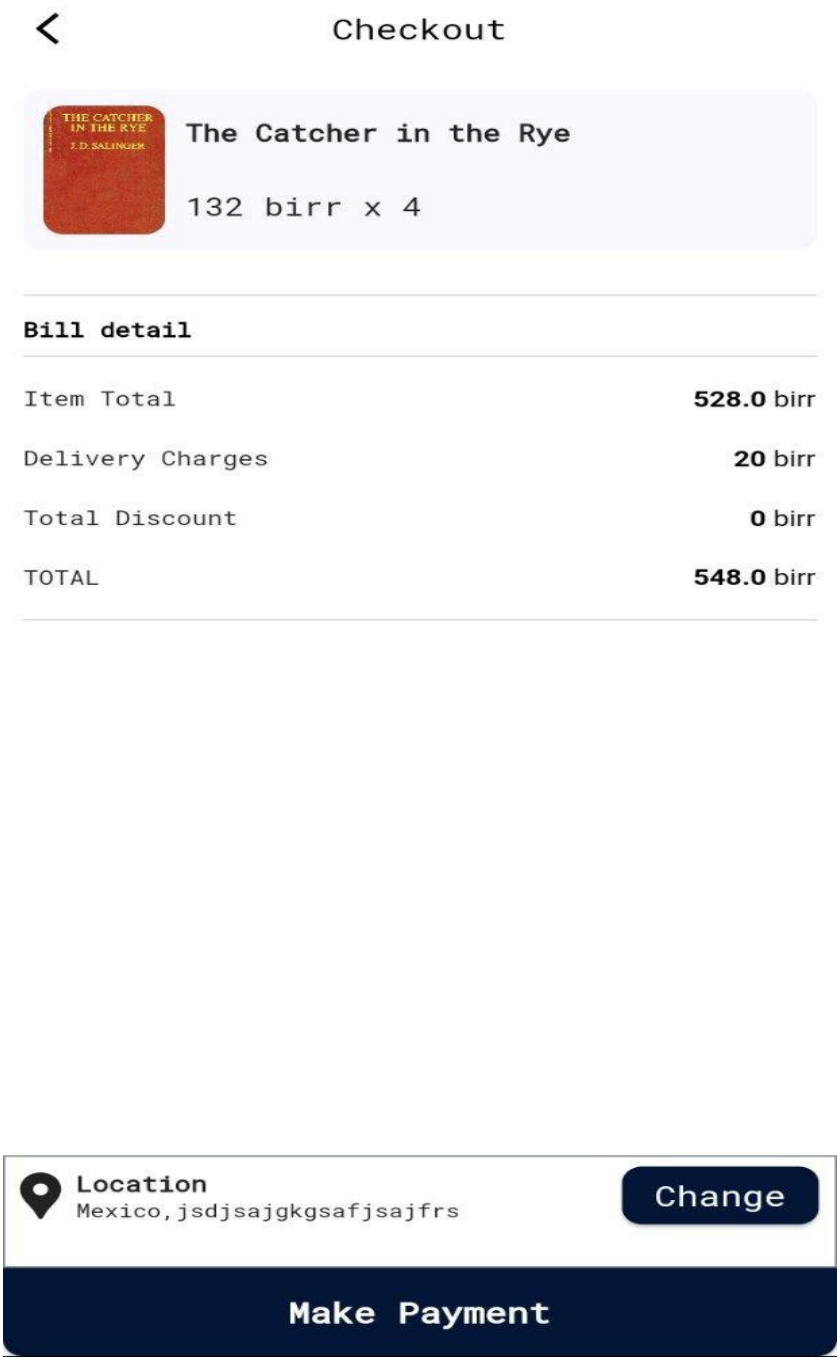

Figure 6.1 Check out screen

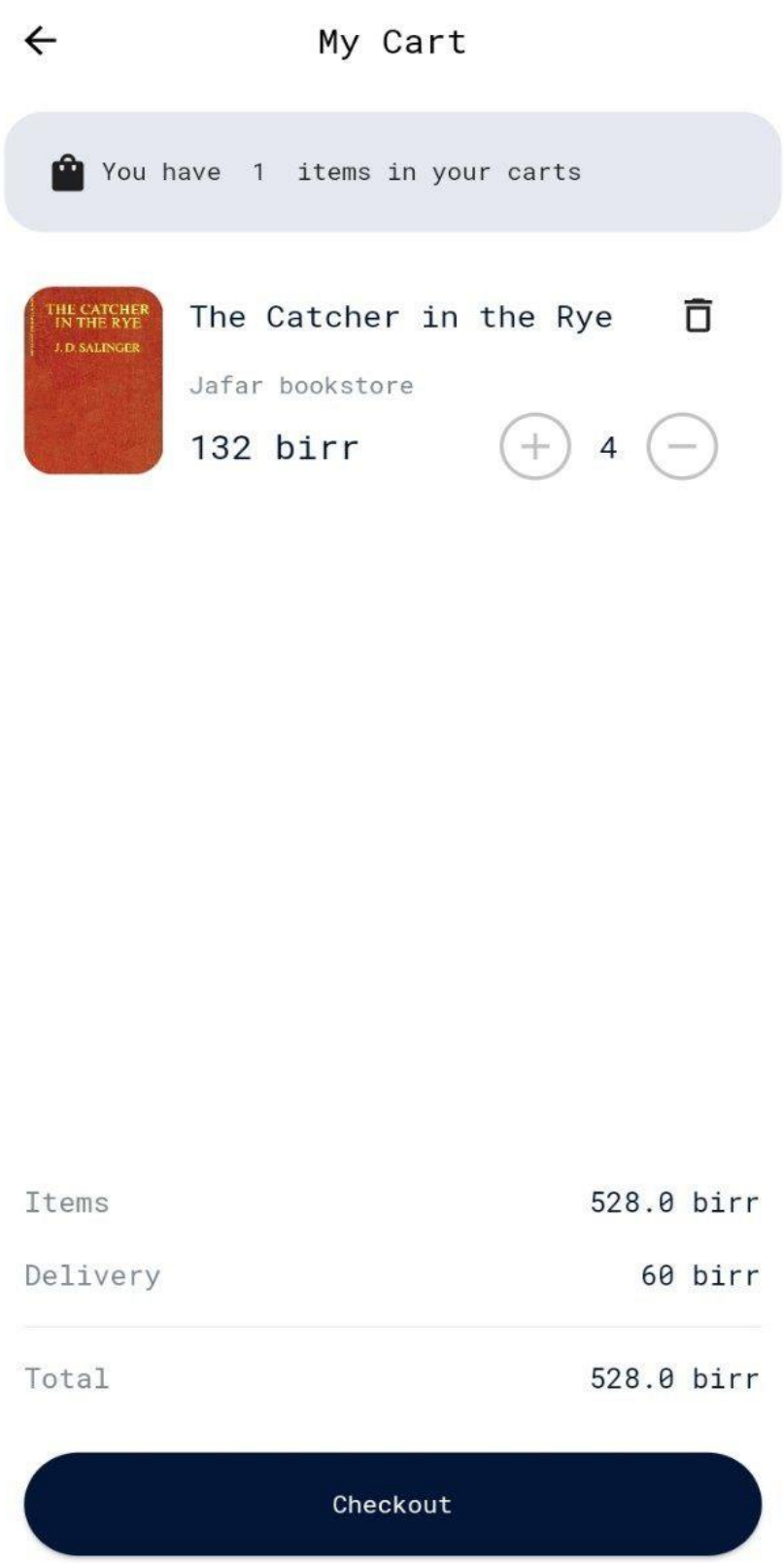

Figure 6.2 Cart Screen

Figure 6.3 Book List Screen

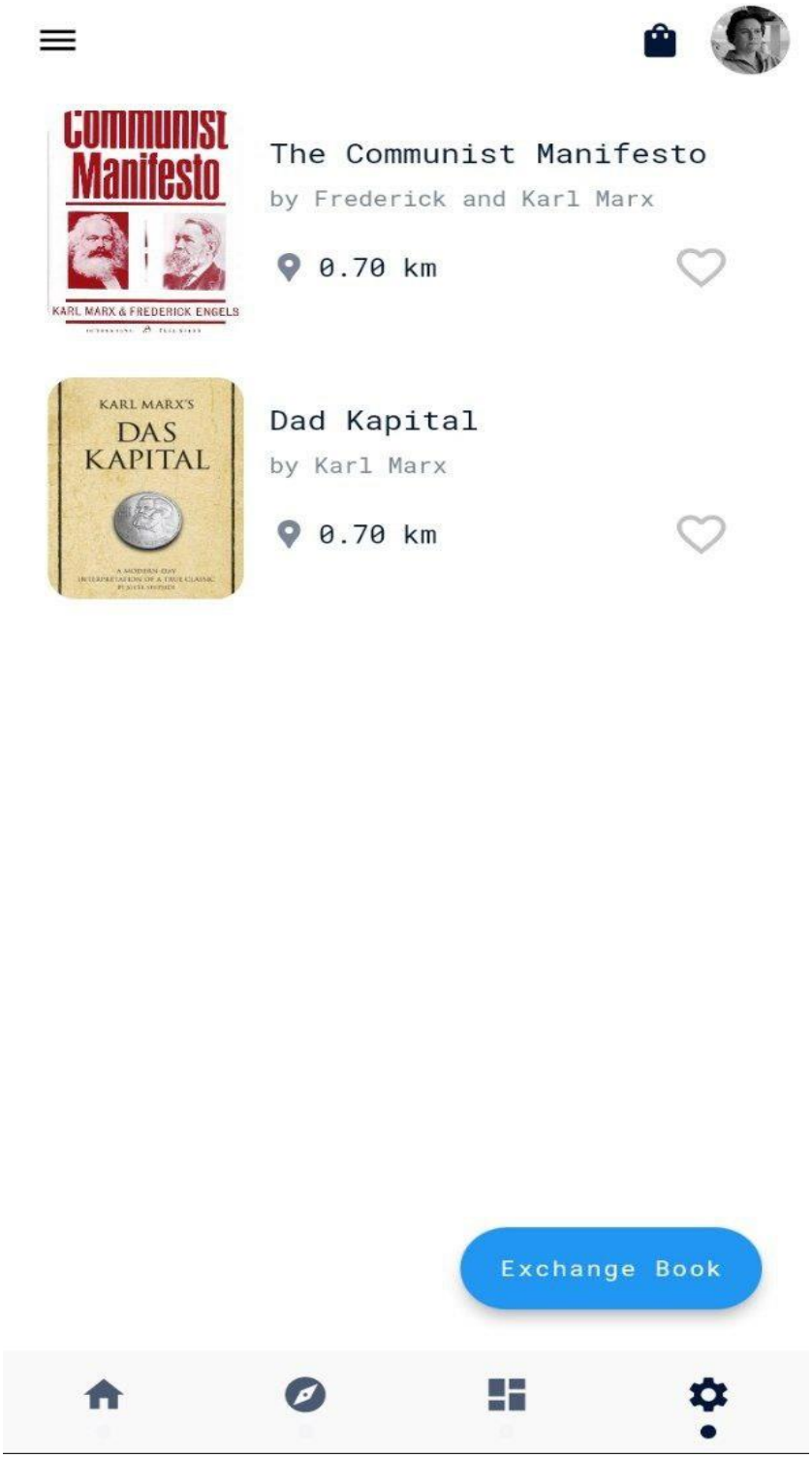

Figure 6.4 Exchange book Screen

Figure 6.5 Add book Screen

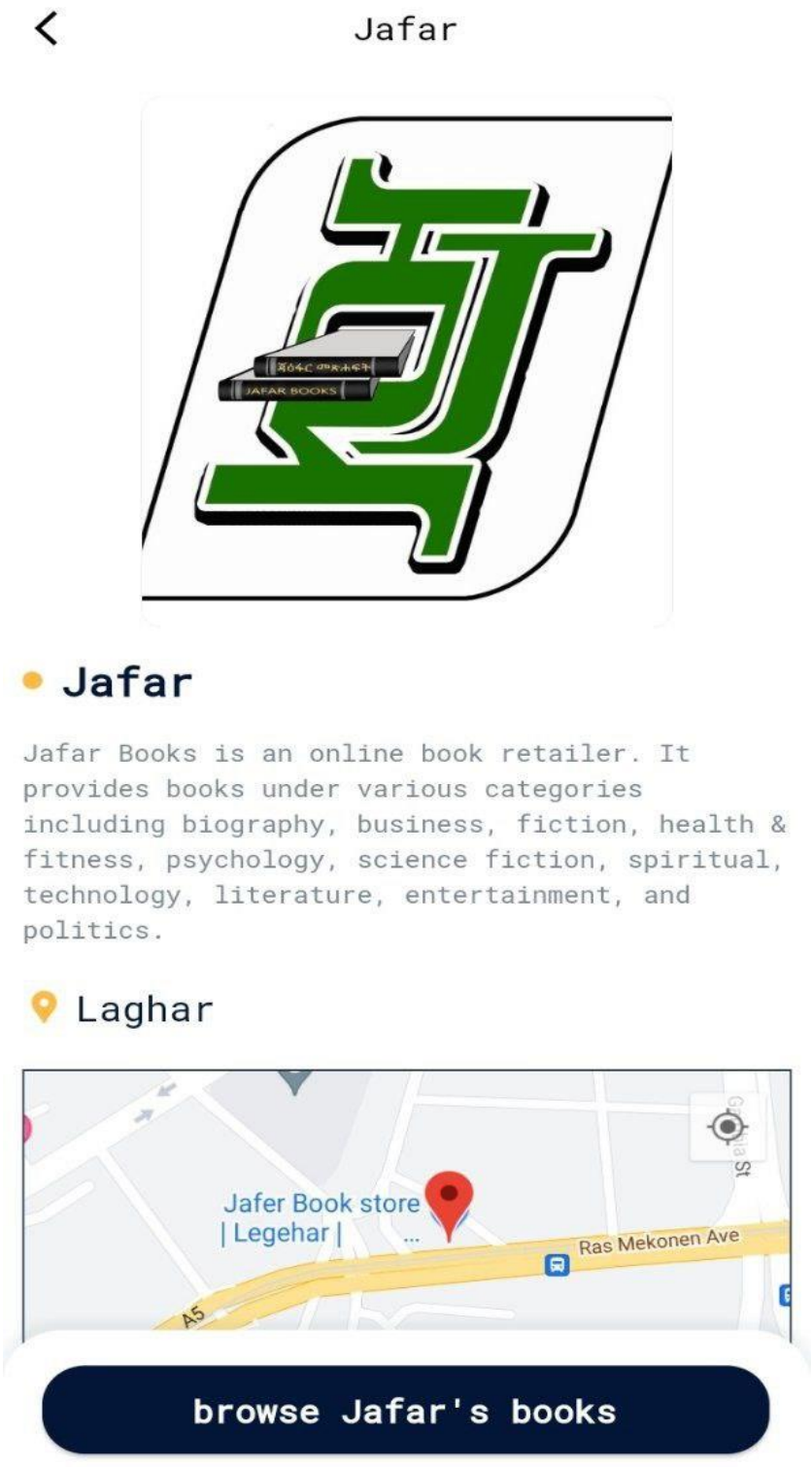

Figure 6.6 Book Store Detail Screen

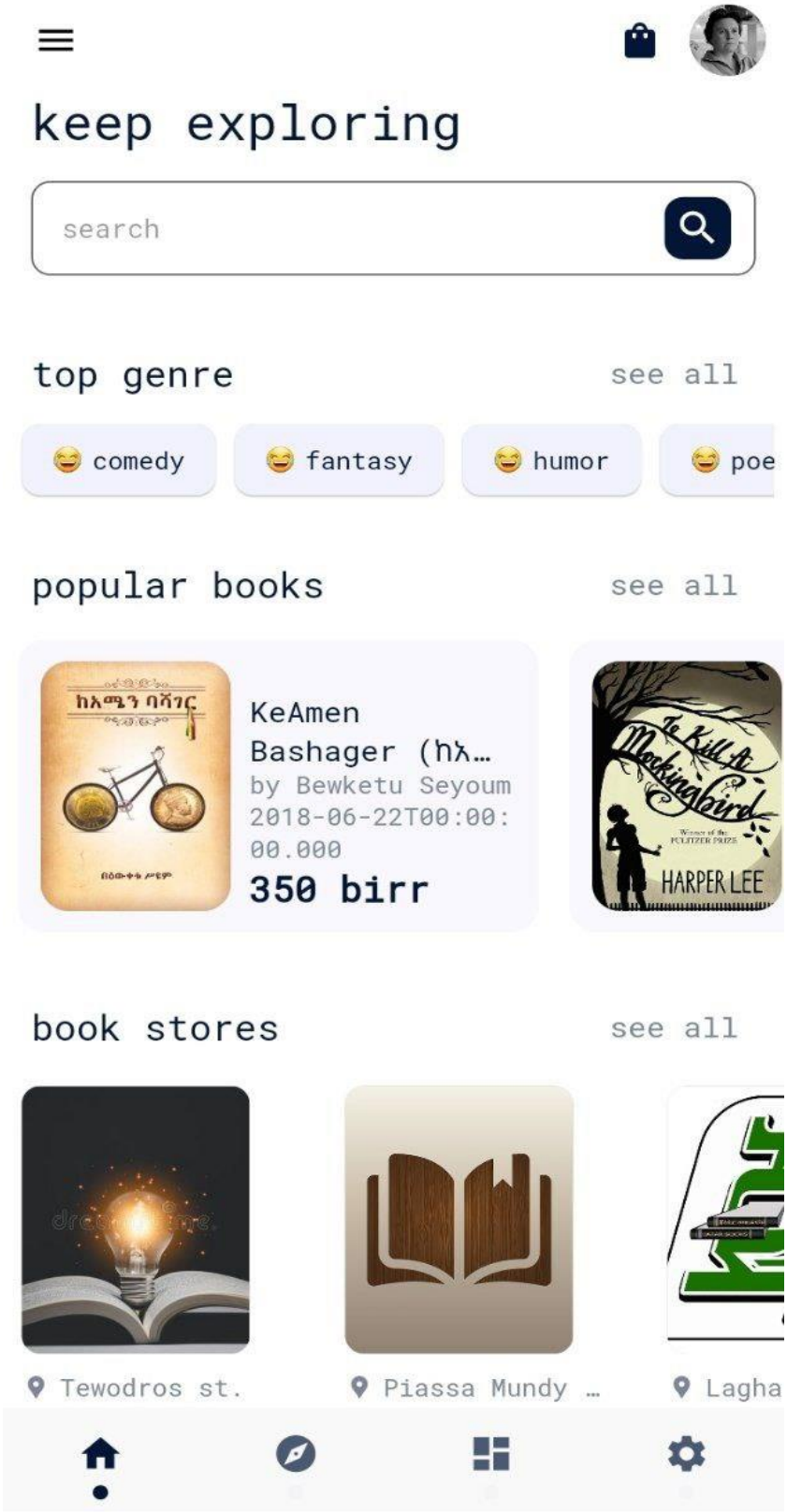

Figure 6.7 Home Screen

6.6.Testing

Testing is a process to show the correctness of the program and designed to analyze the logic used in the implementation of the System. In the case of our project we use unit, user acceptance, and integration testing method, which is briefly stated in proposal.

6.6.1 Unit testing

Each module is tested alone in an attempt to discover any errors in its code. In unit testing, each module (roughly a section of code that performs a single function) is tested alone in an attempt to discover any errors that may exist in the module's code. We tested the system as shown below in the appendix.

6.6.2 Integration testing

The process of bringing together for testing purposes all of the modules that a program comprises. Modules are typically integrated in a top-down, incremental fashion. If an error occurs, the process stops, the error is identified and corrected, and the test is redone. The process repeats until the entire program—all modules at all level is successfully integrated and tested with no errors. After the team tested using unit testing the next step is integrating testing. In integrated testing the team tested the system all modules that a program contains.

6.6.3 Acceptance testing

It is the process whereby actual users test a completed information system, the end result of which is the users' acceptance of it once they are satisfied with it. Acceptance refers to the fact that users typically sign off on the system and "accept" it once they are satisfied with it. Testing the system in the environment where it will be used. The purpose of acceptance testing is for users to determine whether the system meets their requirements. In acceptance test the teams test the system by different user that satisfies the requirements. The user tested by inserting real data.

6.6.4 System testing

System testing is simply expanded integration testing, where you are testing the interfaces between programs in a system rather than testing the interfaces between modules in a program. System testing is also intended to demonstrate whether a system meets its objectives. It is the final step of testing. In this step the team members tests the entire system as a whole with all forms, code, modules and tests all the functionalities in the System. This form of testing is popularly known as Black Box testing or System tests. All errors in the forms, functions, modules are tested.

CHAPTER SEVEN: CONCLUSION AND RECOMMENDATION

7.1. Conclusion

In this project, we have developed a mobile based Bookstore and Exchange application and web dashboard that facilitates various book exchange activities to customers, book owner and for book stores. The system developed in the project consists of mobile and web applications. These are two different applications on the same database. The mobile application enables customers make order, add to cart books, track their order, get nearby book stores, get nearby books for exchange. The web application facilitates to handle managing bookstore information, approve bookstores, get number of total bookstores and users registered on the system and so on. Our model contains analysis model which contains the Functional and Non-functional requirements, use case, sequence, state chart, activity diagrams, conceptual modeling of classes, and identifying user interface prototypes. And also contains designs model which consists subsystem structuring, component structuring, deployment model and the persistent modeling of database.

7.2. Recommendation

The Android-based Book Store and Exchange Platform, as we developed it, is only for Ethiopia. We intend to expand this project to east Africa and beyond. We also intend to integrate with artificial intelligence in order to predict the reader's interests and recommend books based on previous reading history.

The payment process is control with manual system. In the future, we plan to implement to integrate with other payment systems like CBE birr, yene pay. We will make it more efficient and effective by modifying some of its parts, by changing the technologies we used, and by using newly efficient and updated technologies

References

[1] G. Bracha, The Dart Programming Language.

[2] 9 January 2021. [Online]. Available: https://flutter.dev.

[3] M. T. Thomas, "React in Action," 2018.

[4] A. Mardan, "Practical Node.js: Building Real-World Scalable Web Apps," in *Practical Node.js: Building Real-World Scalable Web Apps*, 2014, p. 519.

[5] MongoDB, 2022. [Online]. Available: https://docs.mongodb.com/.

[6] L. Moroney, The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, 2017.

[7] K. Simpson, in *You Don't Know JS: Scope & Closures*, 2021, p. 224.

[8] Git, 2 June 2021. [Online]. Available: https://git-scm.com/doc.

[9] Github, January 2022. [Online]. Available: https://docs.github.com/en.

[10] portswigger. [Online]. Available: https://portswigger.net/web-security/host-header.

[11] portswigger. [Online]. Available: https://portswigger.net/web-security/cross-site-scripting.

[12] geeksforgeeks. [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/.

[13] geeksforgeeks. [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams.

[14] geeksforgeeks. [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/.