PART 1

Overview & Software Requirements Specification

1) Introduction

a) Purpose

Our E-Book System works around make it easier for users to suffer various of books online that has been uploaded by another users, select the required books and order them.

So, we are talking about a way to connect buyers with sellers through the internet which has been nearly used in every home in the world in the last years. The idea is that you can either be a seller and gain money by uploading your books online to be shown by hundreds, thousand or even million of users according to the reach of the website.

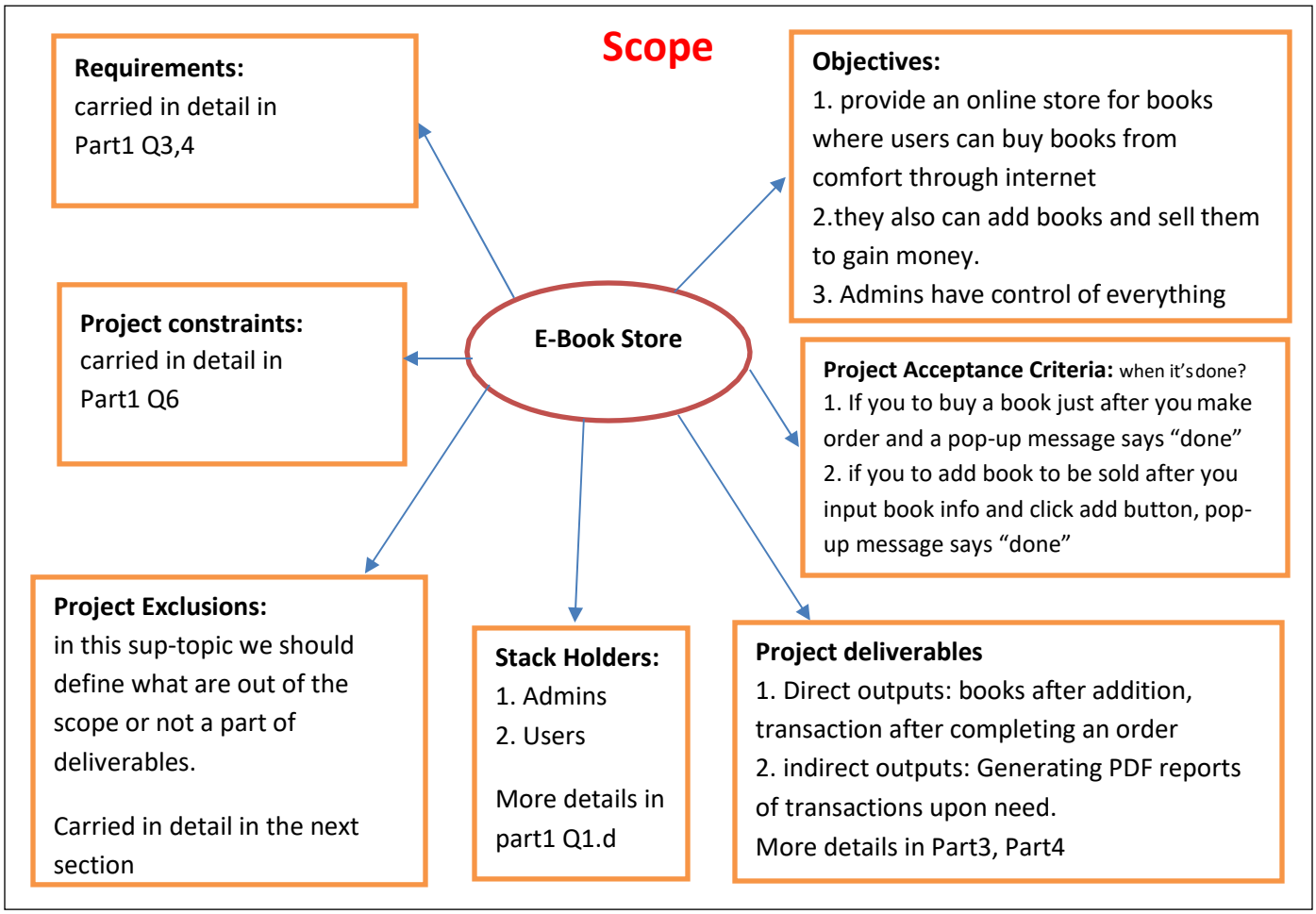
You can be a buyer for sure and select your favorite book out of hundreds of categories where each category contains thousands of books with enough details about each book.

User just need to register an account on the system and with some little clicks the job is done!

That was according to the user view or its purpose to join our website, According to the admins of website it's a way to gain money for sure from collecting taxes on every order and they also have the control to check users on the system, categories and book and can delete any user if he goes upon the site rules. Although admins have information about every user and every transaction that has been made to improve the quality of the system and make it secure.

b) Project Scope

Project Scope



Out of Scope (Project exclusions)

This project **will NOT accomplish or include** the following:	1. Admins Can't register accounts (Add throw database) 2. No insertion or removing for Categories (Add, Remove throw database) 3. Users can't update their profile data 4. No one can access the website except after complete login even if they enter the links of other pages, they will be directed to login page again (made to improve security)

c) Glossary and Abbreviations

1. E-Book ❼ Electronic Book
2. JS ❼ Java Script
3. SRP ❼ Software requirement pattern
4. SE ❼ Software engineering

d) List of the System Stakeholders

1. Admins
2. Users

NOTE: our system is designed that it can hold many stake holders that's is achieved by just change the values of table users in database which contain column user type, can read, can write, can update, can delete. And we later control that in our project with **state design pattern.**
(0,1) acts as Boolean value except for user type column we give value for each new stack holder.

So, for example

User type	0
Can read	1
Can write	0
Can update	0
Can delete	0

User

User type	1
Can read	1
Can write	1
Can update	1
Can delete	1

Admin

User type	2
Can read	1
Can write	1
Can update	0
Can delete	0

Client (new type)

e) References

- Faculty lectures
- Book: Software Engineering 9th edition
- https://www.journaldev.com/1827/java-design-patterns-example-tutorial
- https://businessanalystlearnings.com/blog/2016/8/18/a-list-of-requirements-prioritization-techniques-you-should-know-about
- https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

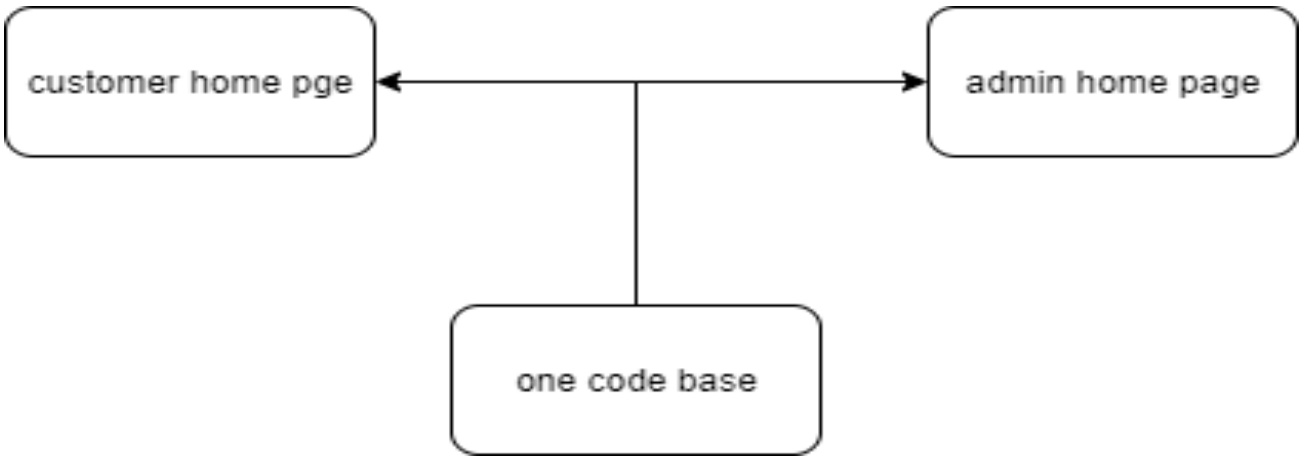
2) What is a Software Requirement Pattern (SRP)?

Software Requirements patterns: the reuse in software applications usually is addressed to the reusing of implementation details (i.e code), but SRP is the reuse of high level abstraction (requirements analysis stage), which comes with many benefits for the overall product and for the customers/developers ,like shortens the reduces the development cost and effort , improves the product quality ,reduces the time needed for the product to be in the market.

Requirement pattern for itself is a way or template used to write a requirement, and SRP is the same of design patterns but in a high-level approach than design patterns.

How SRP affected our architecture?

We used the concept of reusing in SRP and merged the common pages/features for the customer and the admin in a reusable ,customizable way that can implement specific features for each one of them individually, for example, both admin and customer can view the list of books ,login with username and password , so we made one common page for login and use it for both of them , so we didn't need to make two architectural designs for each one of them.



3) Functional Requirements

a) User Requirements Specification

Users FRs:

Requirement ID	Requirement Statement	Comments	Grouping Assignment & Explanation
U01	Customers can register with id, password, name, email	User must use a valid id and password, email	**Critical** - because user can't use our app without it
U02	Customers can login with their email, password	The name can be used for login	**Critical**- because user can't use our app without it
U04	Users can view all the books with their details	details: description, seller if exist, book condition, category, price and name	**Critical**- because it's one of the main features in the system, and let the user see different categories with different books
U05	Users can surf books and add a book to the cart	Books in the cart still not bought yet he need to make order first to buy it	**Moderate**- it's important for the user to make order consist of more than one product
U06	Users can see their cart details and the total amount	Details include: all the added items in that time with their prices and their information	**Critical**- users must see their cart before they make order.
U07	Users can make order to buy the selected books in the cart	The order contains all the (selected) books from the cart in that time	**Critical**- it's the main business of our System

U08	Users can add books to be sold in the website	Upload the name, category, ISBN, publisher, quantity, price, book condition, description, cover image	**Critical**- because it's one of the main features in the app, people must have the ability to add books.
U09	Users can upload books to be sold in the website	They upload a pdf of their book.	**Critical**- because it's one of the main features in the app,

Admin FRs:

Requirements ID	**Requirement statement**	**comments**	**Grouping Assignment & Explanation**
A01	Admin can login with his email, password	All data of the admins will be added from the database (not through the website)	**Critical**- the admin can't use our app without it.
A02	Admin can view all the books with their details	Book viewed same as user view	**Critical**- Admins must be in touch of what going inside the site
A03	Admin can display all the registered users in the system with their registration information	Information include their name, email, phone number	**Critical** -the admin must be able to see the users, it's the main feature of the admin
A04	Admin can delete any user when he needs	Users that's goes above site rules or use scamming or cheating must be deleted	**Optional**- its not a critical requirement nor an optional requirement because system can work without it but it must be included

b) System Requirement Specification

1. The system can let users register an account after filling the registration information and they can login with email and password.
2. The system shows all books to customers with different categories.
3. The system has search feature which allows users and admins to search for a specific book using search bar.
4. The system has ability to let customers see their cart, total price.

3) Non-Functional Requirements

a) the General types/Categories of non-functional requirements

There are different types/categories of non-functional requirements:

1. Constraints
2. External interface requirements
3. Performance
4. Quality attributes

We implemented some of these categories in our app, we'll discuss them in more detail in the next point.

b, c) Non-Functional Requirements Specification, Fit-Criteria for each one

we gathered the non-functional requirements we considered and how we tested and tried to implement them in the same point).

1. The first non-functional requirement we implemented was the load time of the app, which falls under the performance category, we used google speed test insight for making sure that the load time of our website is efficient and meets the standards for most users.
2. The second one was response time, which falls under the same category as number one (performance), according to Jakob Nielsen, there are 3 main metrics for response time, which we had followed. For more about those 3 metric click here.
3. The last non-functional requirement we had considered was the portability, which falls under the category of quality attributes, we aimed to reduce the size of our app as much as possible, so it can run on old devices, we used TinyPng to reduce the size of the images, which helped decreasing the size of the app dramatically.

d) How would the above-mentioned Non-Functional Requirements affect the overall Architecture of the System?

To implement those non-functional requirements we tried from the beginning to achieve that, so we have used the MVC architecture pattern to ensure high loading, and response time ,as this architecture is used by big companies and many websites, we have models for our data, services to do operations on them , and controller for showing them to users.

4)Domain Requirements Specification

- System can't let users or admin accesses any page without login first even if he tries to write its link in internet browser search bar he will be directed to login page to complete login first.
- The user can add a book on condition that he has intellectual property rights
- user can make order and see all his orders and cancel the orders in any time
- user can buy any book with any quantity included between 1 and the book quantity on the system, if there is stock of the book
- there are ways for payment method and for every order the payment method and details will be stored and displayed and purchase page.
- two users can add and sell the same book if they have intellectual property rights.
- When a quantity of a book is over user can't add it to cart and the button will be disabled and turns to "out of stock"
- admin can see all user and delete any users upon need.

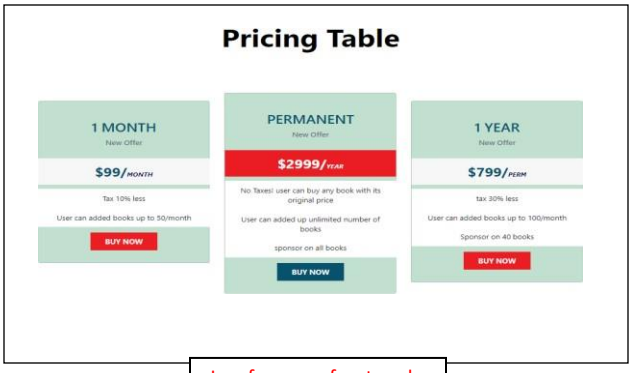
5)Design & Implementation constraints

- System back end implemented using PHP & MySQL.
- System Front End implemented by Html, Css, JavaScript, jQuery, and Bootstrap.
- Design must keep sure that it doesn't affect the system performance badly
- System design must be user friendly and provide preferences
- System must be secured as we got important information about each user this information includes visa information email, phone ...etc.
- We used **State** design pattern in the implementation to differentiate between user, admins and any new stack holder that may be added. these controlling include nav-bar changes and many other pages to be used by user, admin
- We used **factory** design pattern to achieve the re-use of code and make the design better as we grouped functions of Book, order, costumer classes in one class as the shape of factory design pattern.
- We used **MVC** pattern (model, view, control) to make the system easier and more collective as we use the control as an intermediate between view and models. so, when making an action in view model its effect will be made after the system call controller and then go to the model and get the result back
- We used Delegation design pattern to be able to use the methods of a class in another class by creating an object from this class in the method that we want to duplicate and use it better than copy, paste the function or inherit from the class.

6) System Evolution

a) Anticipated changes

1. **Subscribe**
   Soon we will put more features if the user made membership with us, the payment tax on each order will decrease or if he made permanent membership, he can buy books without any tax on any order



Img from our front-end

2. **Speed our site**

if we have many users in the website in the same time then the website will become slow, so we need to make the web faster more details about this how to make it faster in point b

3. **Increase website Storage**

now we work with low storage and will be ended soon

so, we need more storages to keep website work perfectly

4. **Payment methods**

the website has 2 ways to pay (visa or cash).

We will put more types to pay in the future like (PayPal, master, fawry).

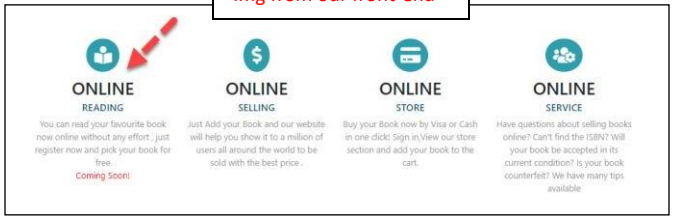
5. **Trade books(product)**

this feature means the customers can exchange books together without buy or sell the book

6. **online reading**

Users can pay for reading and downloading the books online

Img from our front-end



7) What are the requirements discovery approaches that you'll rely on?

The requirements discovery is about collection information for the requirements, collecting user and the system requirements and it used to prepare solution for the problem, example:

Stakeholder.

- User: the customers that will enter the website and select the books to buy or can add book to sell them.
- Admin: the manager that has the ability and responsibility see all the user information and all the transaction process

Interviewing.

- Closed interview: the list of pre-determined required example for user (register, login, view books, buy book, add books, upload books, my order, view buyer) and for admin (login, view books, view transaction, view user).

Scenarios.

- Are example of how the website will be used. so,
  For admin: the admin will login by enter valid credentials so, he can login to the system, he can check and see the books and all its details. he can also view the transaction process and its details. he can also see the registered users' information and can delete any user

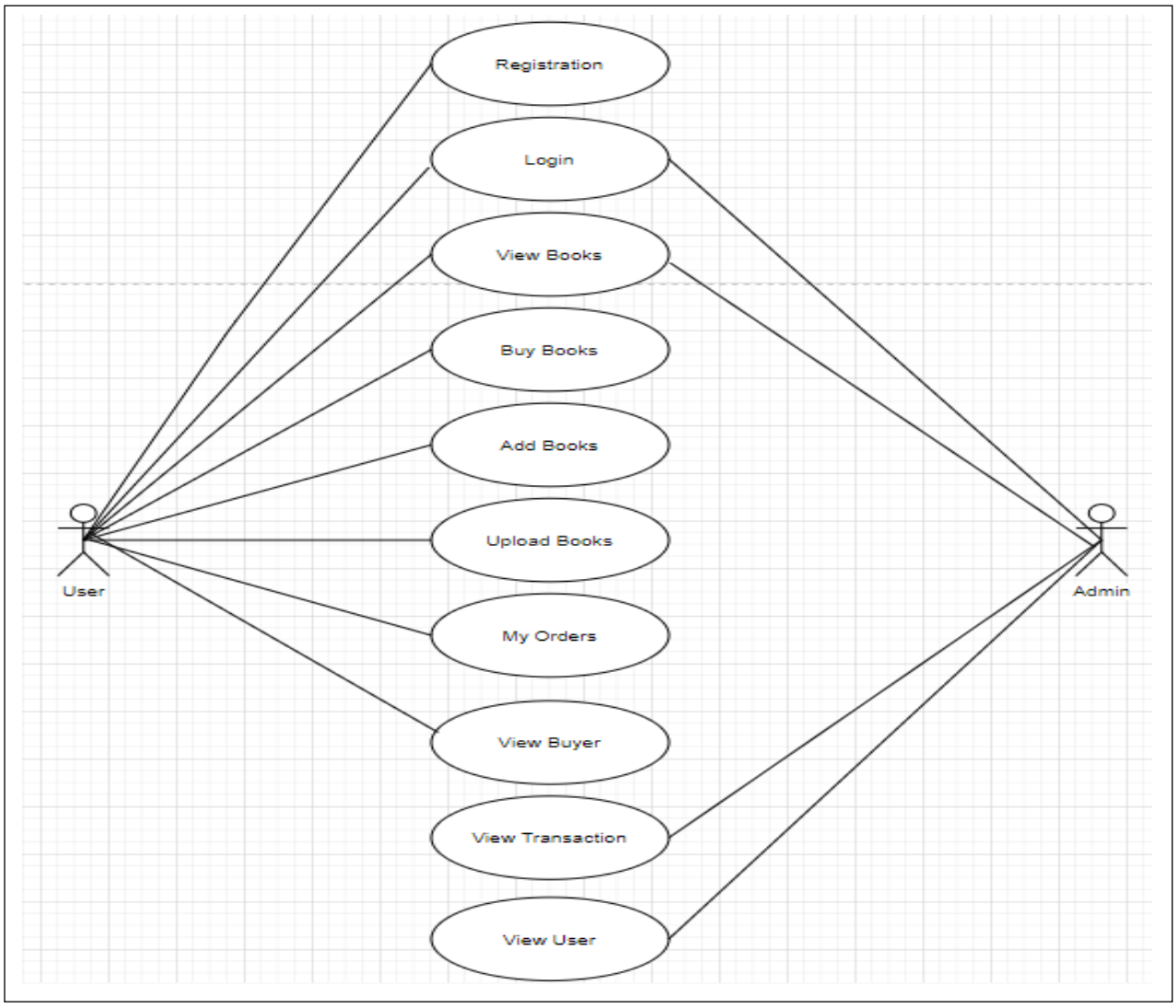
- For user: he will start by register then he will be able to login after that he can search for the category he want which will help to find the book then he can choose any books from different category or from the same category then he can add those to cart and complete order process.
  he can also add or upload books to be sold.

Ethnography.

- This type is get by see how the system work and doesn't need to ask to get function to help the system to become easier to use and more helpful example (we can make function to calculate the total of what the user buy from the website all the time).

Use Case:

- This type used to give the actor an interaction and describe the interface.



8) What are the requirements validation techniques that you'll employ/use?

It is the process of checking that requirements defined for development, it helps us define the system that the customer really needs. In order to check issues related to the requirements, we perform what is called **requirements validation**.

Now there are several techniques which are used either alone or with other techniques to check entire system or just a part of it.

We used a technique called **Prototyping.**

In this technique for validation the prototype of the system is presented to an end user, this end user experiments with the whole system and sees if it is what they need and if there is something missing or if there's an error. This technique is generally used for collecting feedback to help us find what customers need most.

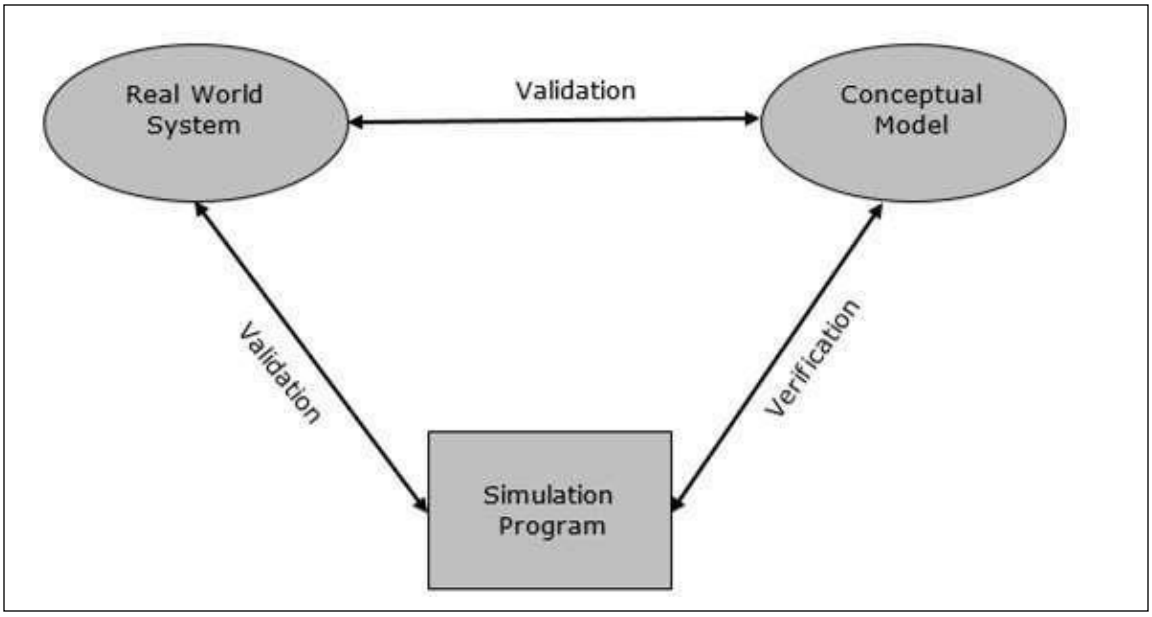
Another technique that we used is **Test Case Generation.**

All requirements are checked and tested to see if they have any errors either in the data the user entered or the requirement field itself. There are many checks in the SRS document which are testable, like Completeness checks, Validity checks, Realism checks and Verifiability.

with coding we've implemented a number of checks from the checks we previously mentioned, for example, in Completeness checks we check if there is missing input data entered by the user such as (Name, email, Credit

Card and other info) and if there is something missing the user is faced with a message explaining what is missing in his data. Another check we used is the Realism check, which checks if the data entered is imaginary or unreal, for example, if the user entered that he was born in the year 2020 then he will receive a message telling him that the data is not real. We also implemented Verifiability checks in order to verify user's data, for example, passwords and usernames, we check if the username and password are stored in our database, or if a user was trying to create an account for the first time he is asked to enter a password and verify it.

Overall, we check every possible requirement and see if there any errors or bugs to help us fix any problems that we face.



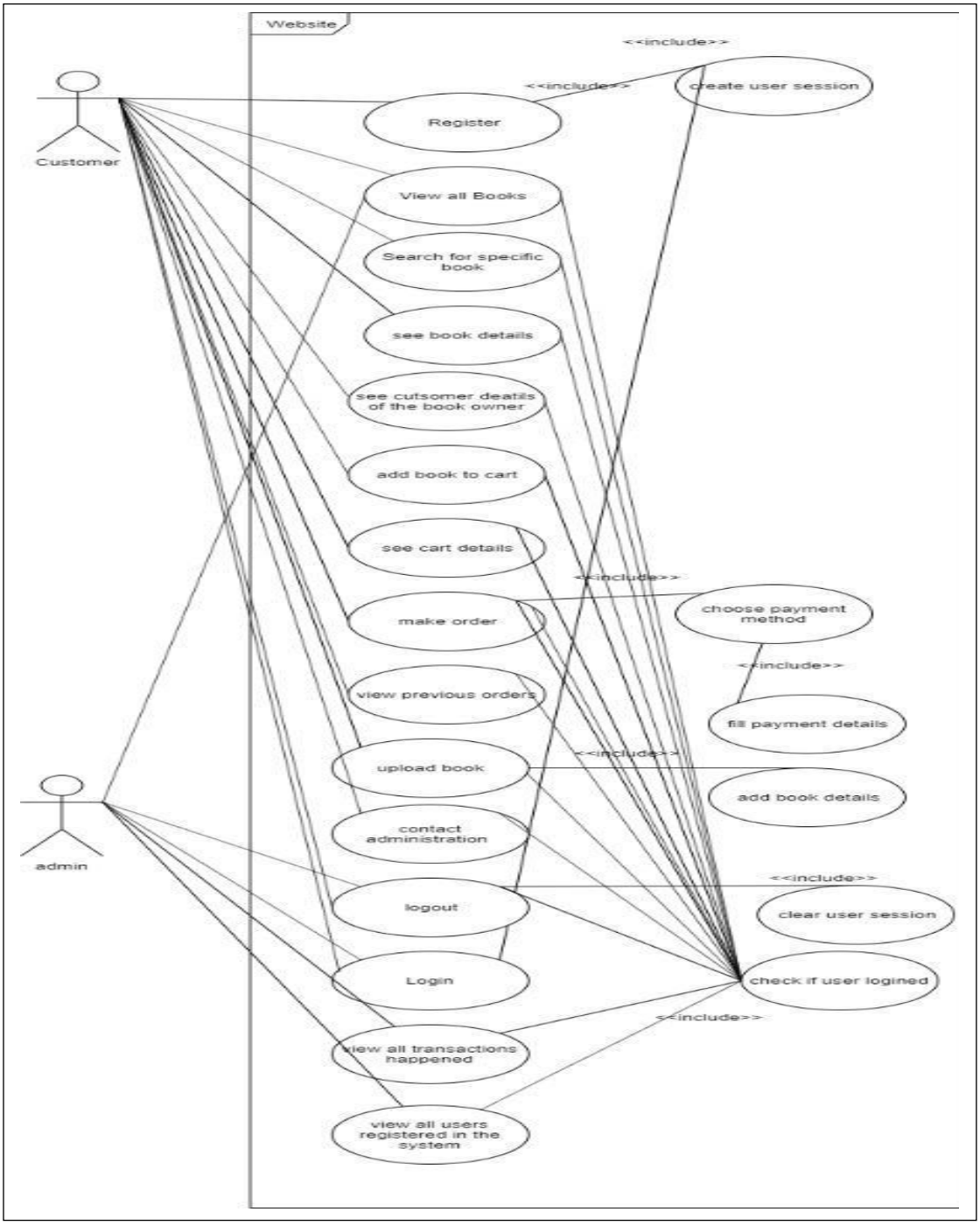
END OF PART 1

PART 2

System Design & Models

9) Functional Diagrams

a) Use-Case Diagrams including all the Use-Cases for the System



b) Detailed Use-Cases Description

- customer registration case

ID:	**01**
Title:	Customer registration
Description:	for someone to use our app, he must register first
Primary Actor:	Customer uses the app for first time
Preconditions:	none
Postconditions:	the user will have an account for our app in order to use it.
Main Success Scenario:	the user enters the required information correctly, press register button , create user session with the user id, and then navigate to the home page
Extensions:	show error message if the user didn't enter correct data
Frequency of Use:	often

- view all books case

ID:	**02**
Title:	view all books
Description:	view all books in the system.
Primary Actor:	customer or admin
Pre-conditions:	must have account and logged in the app
Post-conditions:	have a list with all the books in the system
Main Success Scenario:	user (customer or admin) enters the home page, which requests all the books in the system to see it
Extensions:	if the user isn't logged in, the app will navigate to the login screen
Frequency of Use:	always

- search for a book use case

ID:	**03**
Title:	search for a specific book
Description:	the customer can search for a specific book in the system by its name the result will contain all the books that contains the string he entered
Primary Actor:	customer
Preconditions:	the customer must be logged in the app
Postconditions:	a list will all the books that contains the search string
Main Success Scenario:	the user enters the name he wants, gets the list of the result, find the book he wants
Extensions:	none
Frequency of Use:	often

- book details case:

ID:	04
Title:	show book details
Description:	the customer should be able to see the book details of any book, the details includes data like the book owner name, isbn, price ,category , etc.
Primary Actor:	Customer
Preconditions:	customer must be logged in the system; the book must exist in the system
Postconditions:	the app should be in the book details page with the correct information for the book
Main Success Scenario:	the user opens the home page. the user sees a book and presses on it. the user sees the book details of the book that he pressed on.
Extensions:	error page will appear if the book is not found on the system
Frequency of Use:	often

- Book owner details case:

ID:	**05**
Title:	see the details of the book owner
Description:	the user should be able to see who uploaded or added this book to the app.
Primary Actor:	Customer
Preconditions:	the user should be signed in, and the book should have book owner The book owner should exist.
Postconditions:	the customer should be in the page of the book owner details
Main Success Scenario:	user press on book details. the name of the owner appears in the details. the user presses it. the app navigates to the page of the book owner
Extensions:	if the book owner isn't found the error page will appear.
Frequency of Use:	unlikely

- add book to cart case:

ID:	06
Title:	Add book to cart
Description:	the customer should be able to add book to his cart, if it's already there it will increase the total number of books
Primary Actor:	Customer
Preconditions:	customer should be logged in the system; the book exists on the system
Postconditions:	adding the book to the user's cart, updating the total price of the cart.
Main Success Scenario	the user presses the add to cart button of a specific book. the app adds the book to the cart and updates it the app navigates to the cart page and shows the updates.
Extensions:	checks if the customer logged in or not
Frequency of Use:	usually

- Cart details case:

ID:	**07**
Title:	show Cart details
Description:	the customer should be able to see his cart details, the books inside it , the total price etc..
Primary Actor:	Customer
Preconditions:	the customer should be logged in the system
Postconditions:	the customer should see his cart with its details
Main Success Scenario:	the customer presses the cart button. the customer sees the cart details and total price.
Extensions:	none
Frequency of Use:	usually

- Make order use case:

ID:	**08**
Title:	Creating order
Description:	the customer should be able to create an order with the books inside his cart
Primary Actor:	Customer
Preconditions:	customer must be logged in the system, has at least one book at his cart, fills the order details
Postconditions:	creates an order with the books inside the cart, with a report
Main Success Scenario:	the user presses make order. the customer fills the order details. the customer presses confirm order. the app shows a success message.
Extensions:	checks if the customer logged in the system and has at least one book in his cart
Frequency of Use:	often

- upload book use case:

ID:	**10**
Title:	upload book use case
Description:	the customer uploads the book in pdf format and its details
Primary Actor:	customer
Preconditions:	user logged in the system, has the pdf file of the book, has its details
Postconditions:	the book is uploaded to the system with its details, other customers can view and buy it
Main Success Scenario:	the customer presses upload book. the user chooses the pdf file and uploads it. the user is asked to fill book details. the user presses add.
Extensions:	show validation error messages if the users leave empty or invalid fields that are required
Frequency of Use:	often

- Contact administration use case:

ID:	**11**
Title:	Contact administration
Description:	the customer can contact the app administration for any feedback or issues that they want to submit.
Primary Actor:	customer
Preconditions:	user must be logged in the system
Postconditions:	the message is sent to the administration and success message is shown to the customer
Main Success Scenario:	the customer presses contact button. the app navigates to the contact form. the user fills the contact information. the user presses the sent button. the message is sent successfully.
Extensions:	None
Frequency of Use:	not often.

- view all customers use case:

ID:	**13**
Title:	view all customers
Description:	the admin can view all customers that are registered on the system
Primary Actor:	Admin
Preconditions:	admin has to be logged in the system
Postconditions:	a list with all customers in the system
Main Success Scenario:	the admin presses the customers button. a list with all customers is shown to the admin
Extensions:	none
Frequency of Use:	usually

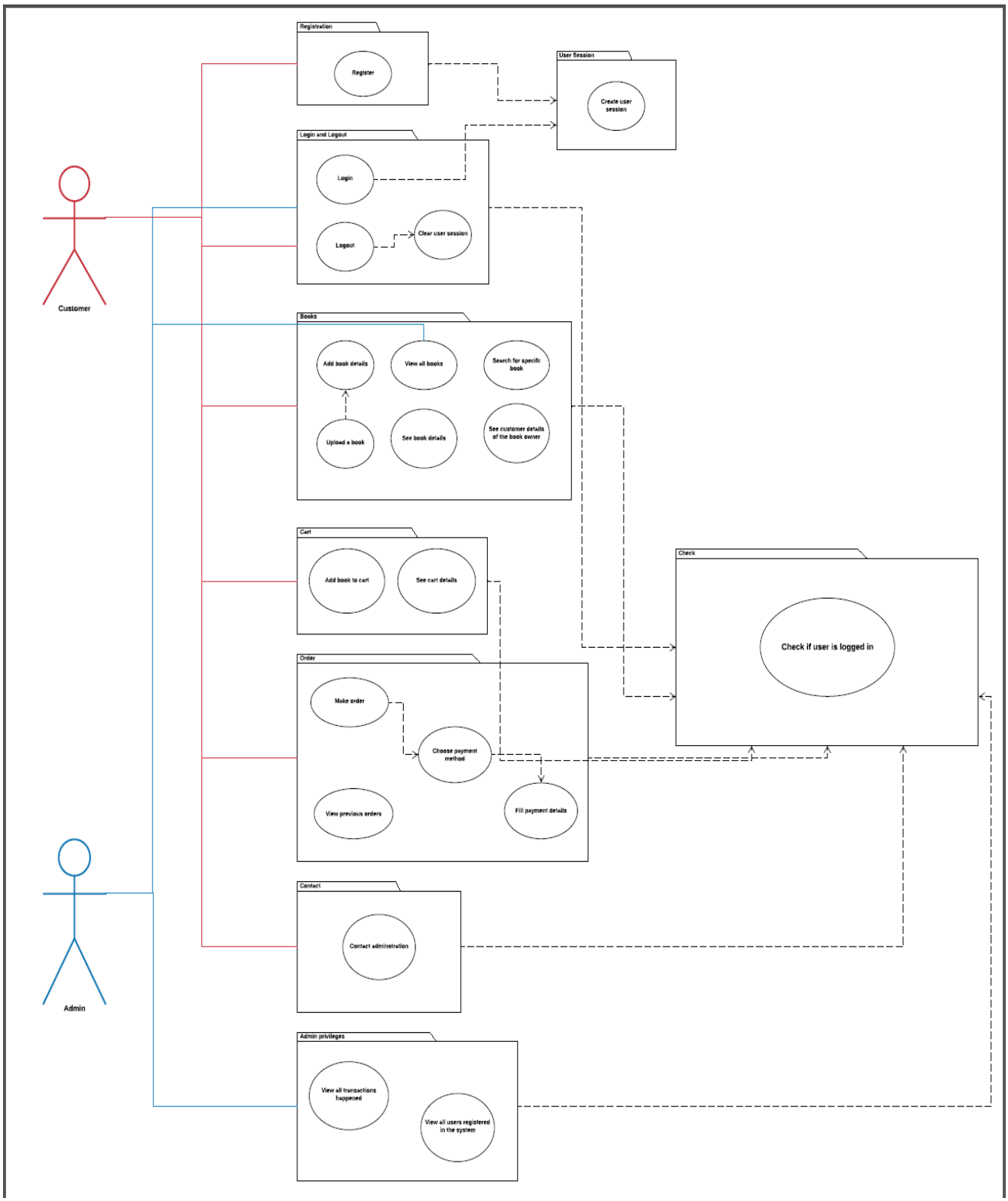
- Delete customer use case:

ID:	**14**
Title:	admin deletes customer
Description:	the admin can select customer from the customers list and deletes him
Primary Actor:	admin
Preconditions:	the admin must be logged in the system. the customer he wants to delete must be registered on the system.
Postconditions:	a success message that the user is deleted. a new list of customers without the deleted one
Main Success Scenario:	the admin opens the list of customers. the admin chooses the customer he wants to delete. the customer is deleted successfully.
Extensions:	none
Frequency of Use:	not often

- login use case

ID:	**15**
Title:	user login
Description:	the user can login with his account into the system
Primary Actor:	admin or customer
Preconditions:	must have an already existed account. there's no open session in the app at the time of login, i.e : he is not logged in yet.
Postconditions:	the app navigates to the home page and creates a session for the user.
Main Success Scenario:	the user (customer or admin) enters his login details. presses login button. the login is success and the app navigate to the home page
Extensions:	checks if the user email and password are valid or not
Frequency of Use:	always

c) Package Diagram grouping relevant Use-Cases into Packages

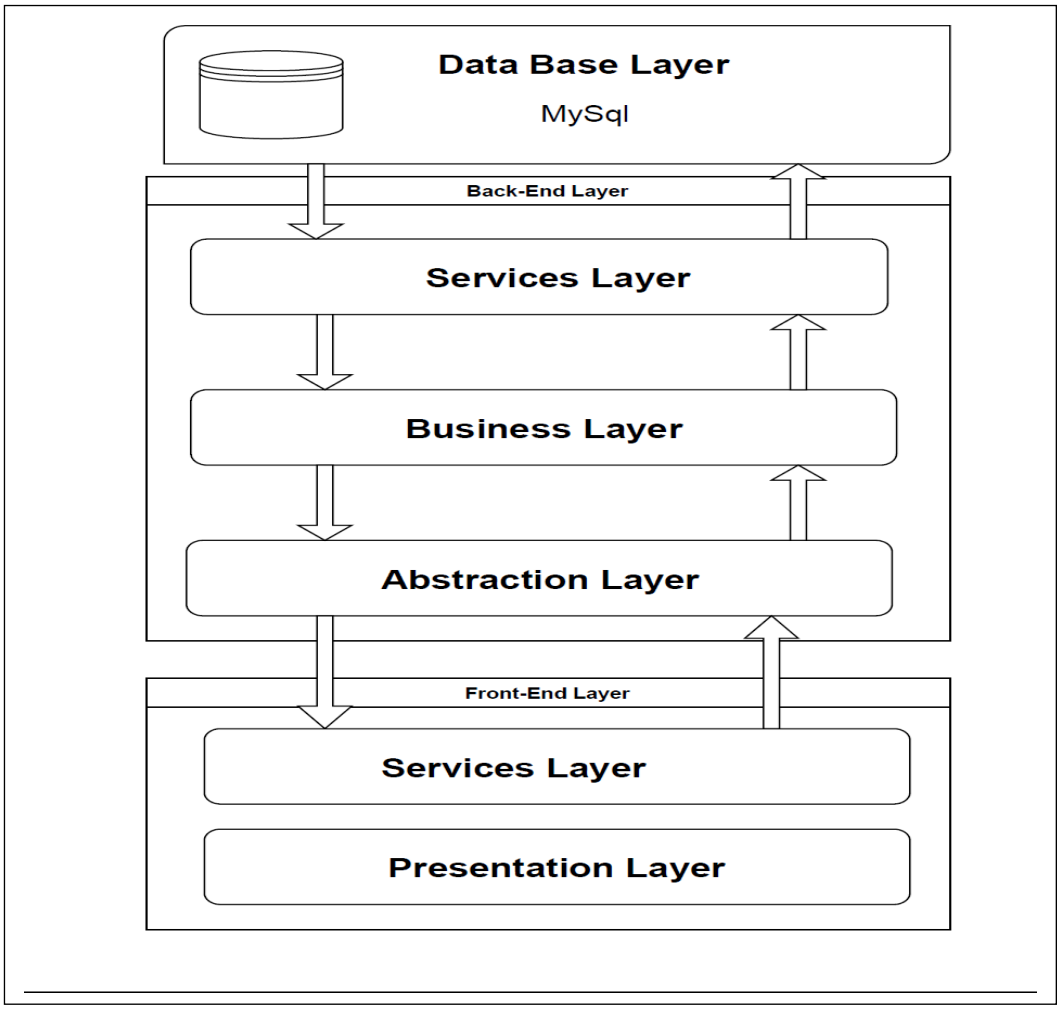


10) Structural & Behavioral Diagram

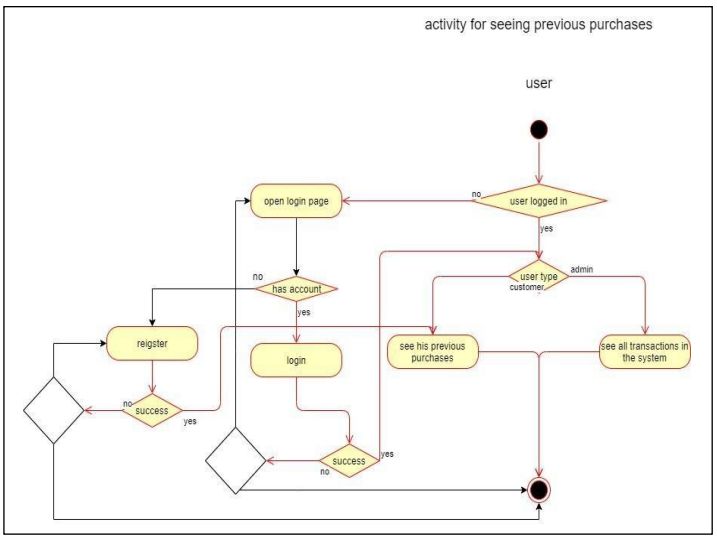
a) System Architecture

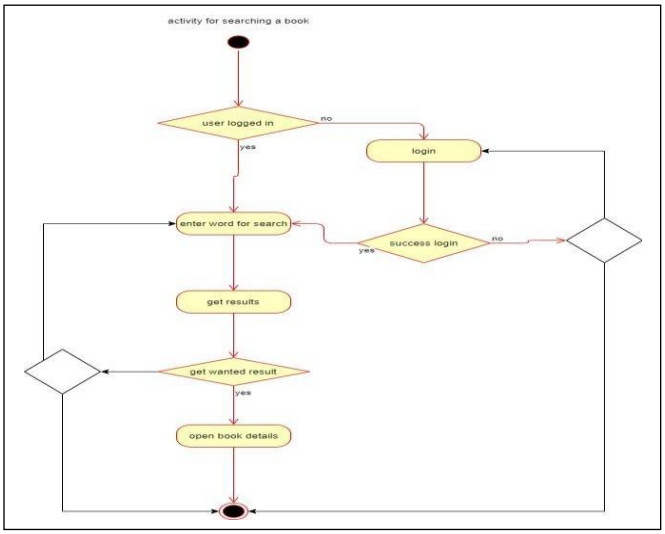
we used the MVC architectural pattern for our app , MVC stands for Model View Controller, models are the data representation that's used in the app , View is the UI that's appeared to the user of the app , Controller is the link between Model and View , it sends that data to the View , and sends actions and mutate the state of the model .

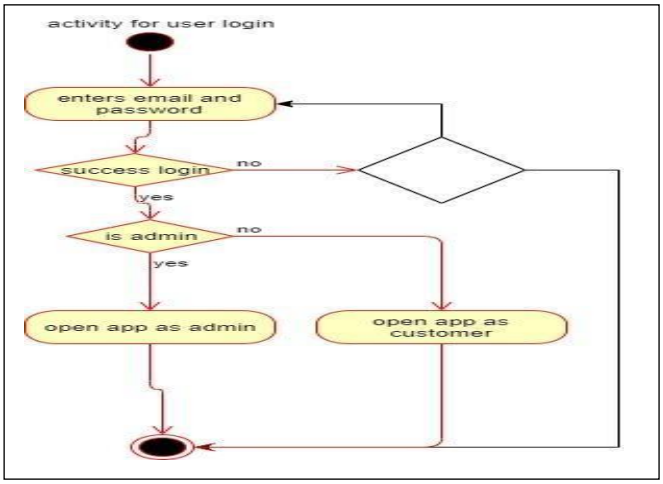
MVC keeps the code clean, easy to maintain and to understand, and its very popular pattern in web development so we adopted this pattern to be consistent with the industry, and any other developer can get familiar with our project as soon as possible.

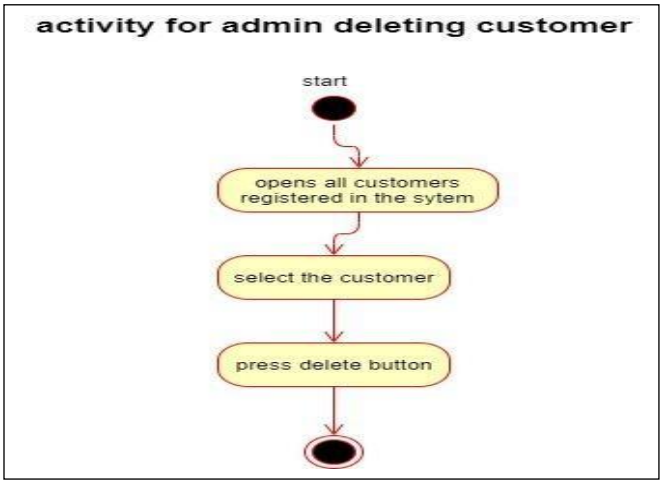


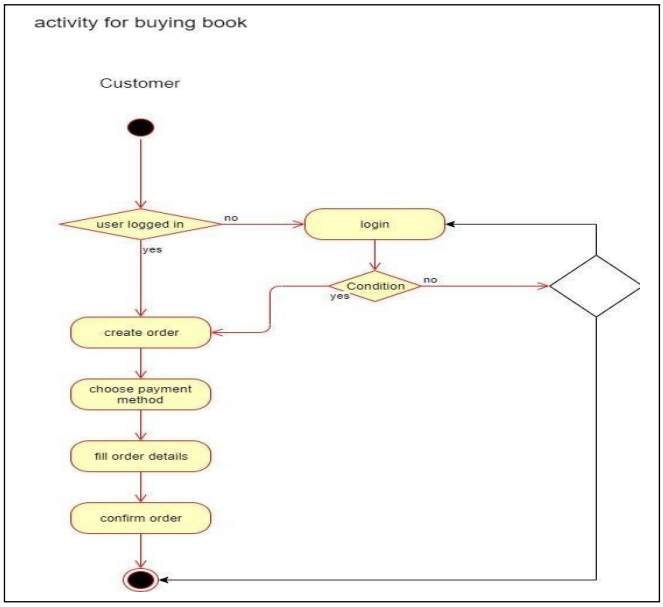
b) Activity Diagrams

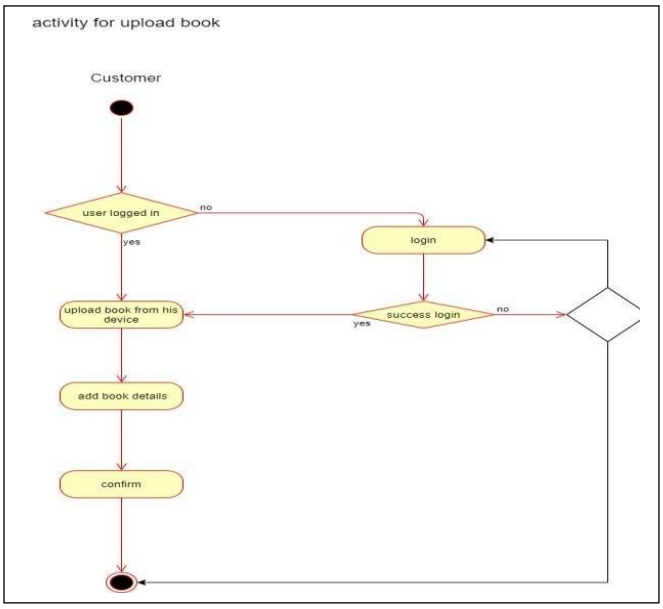






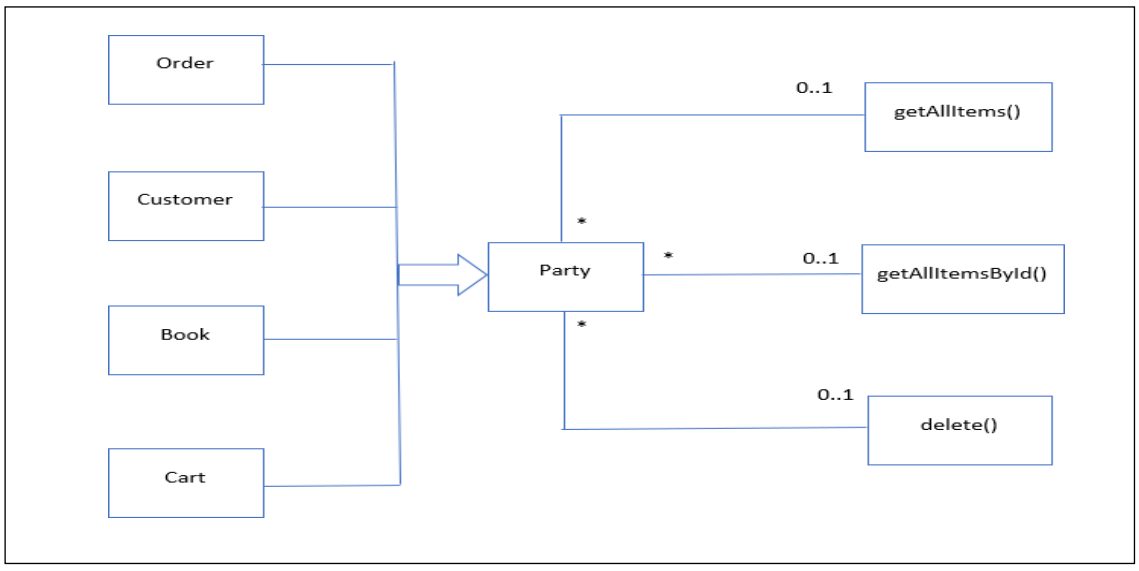






c) What is a Software Analysis Pattern? Apply at least one analysis pattern while analyzing and designing your system

- The analysis pattern is group of concepts that represents the common construction in the system.
- The type of analysis pattern:
  1. Accountability: this type used to describe the relation and responsibilities between the parties.
  2. Observation and Measurements: this type used to the facts.
  3. Referring to objects: this type used when referring to object.
  4. Inventory and Accounting: this type used to record the way how the money and goods are moved between the parts and to make sure to record in the properly way.
  5. Planning: this type used to describe the plan and protocol that used to record the resources.
  6. Trading: this type used in the situation where the price is Changeable, and we need to understand how the price will affect the profit.
- The analysis that will be used is Accountability and the pattern called party:
  - The problem: we have many classes have the same the function.
  - The solution: create common class has this function for this classes.

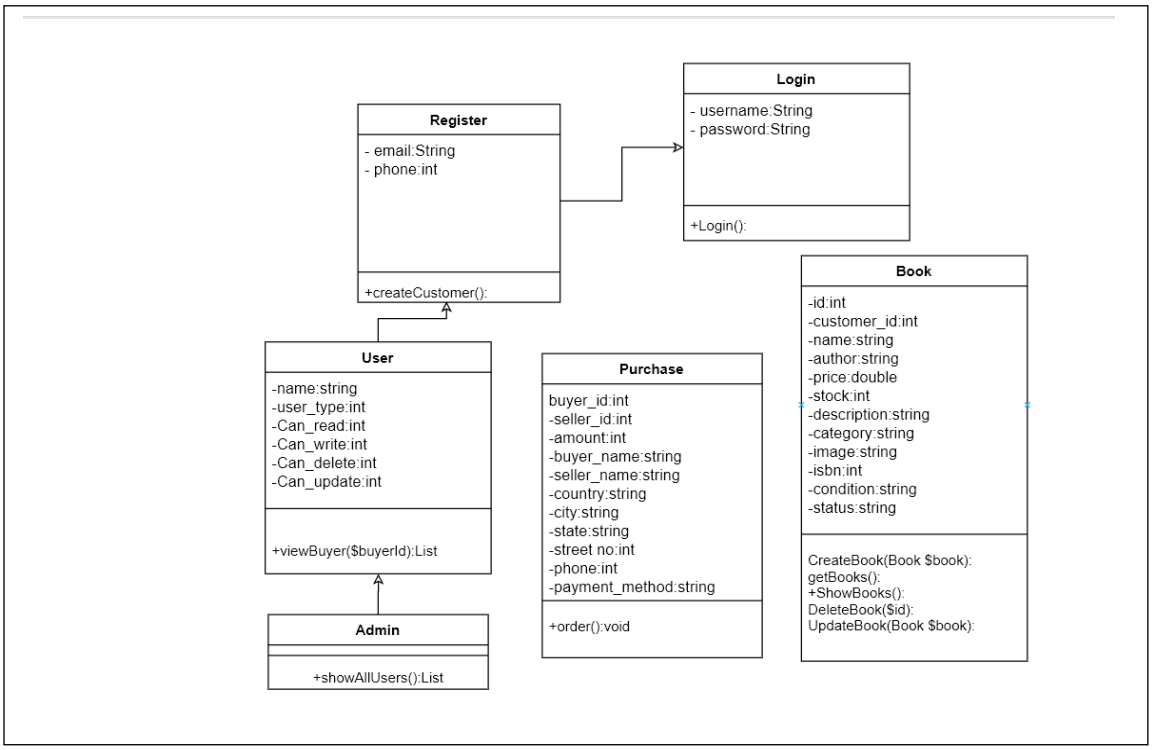


NOTE: We used factory design pattern to achieve this in our code

d) Class Diagram 1: An initial version based on the requirements and Use Case/Activity diagrams

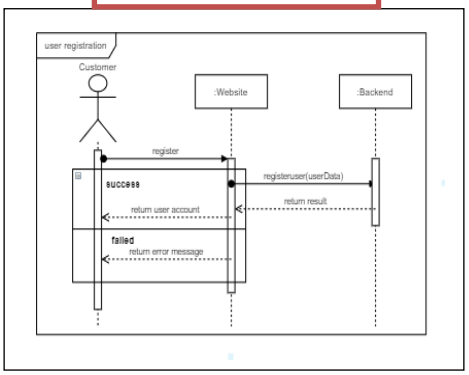
we created the first-class diagram initially based on requirements so, it's very simple where it only consist of the initial attributes and basic operation needed to achieve the requirements

Class Diagram V1

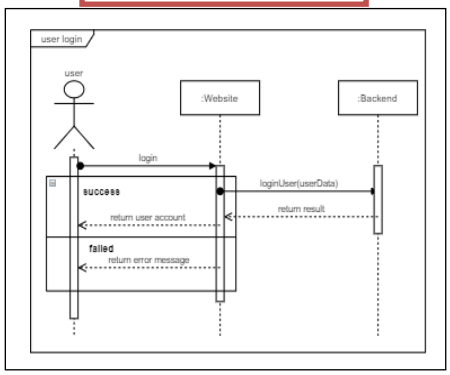


e) Sequence Diagram(s)

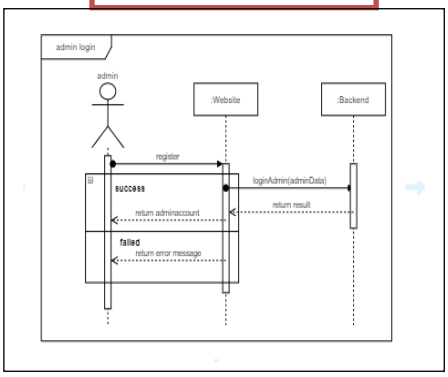
User Registration



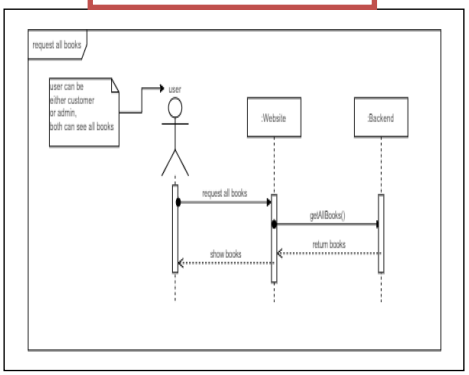
User Login



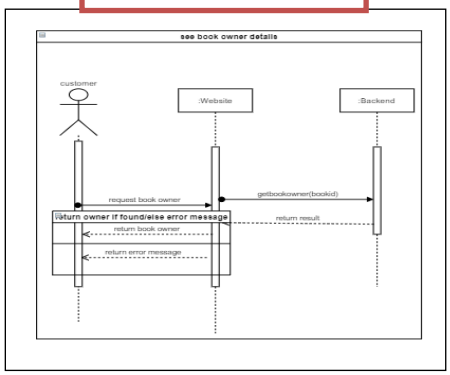
Admin Login



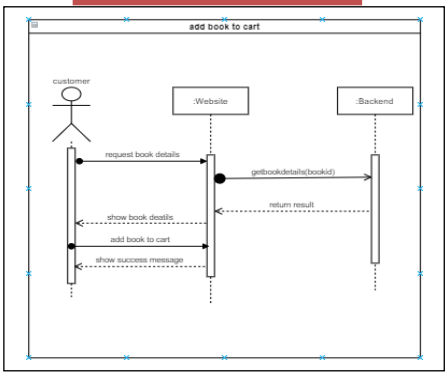
Request All Books



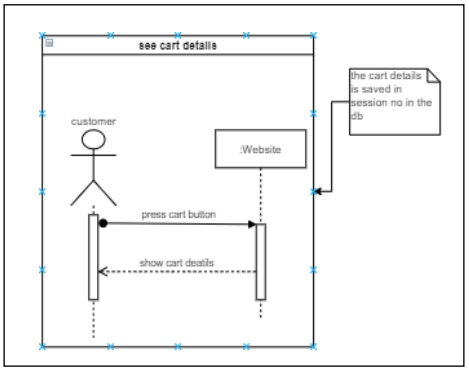
See Book Owner Details



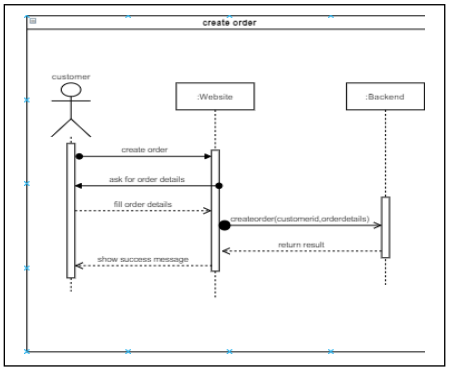
Added Book to Cart



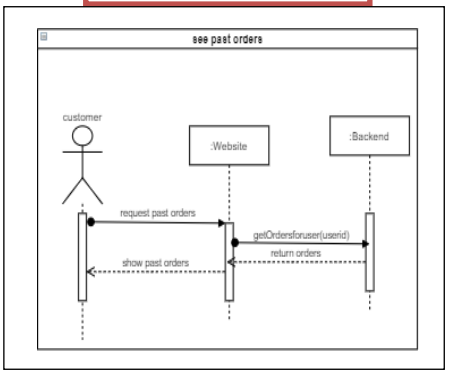
See Cart Details



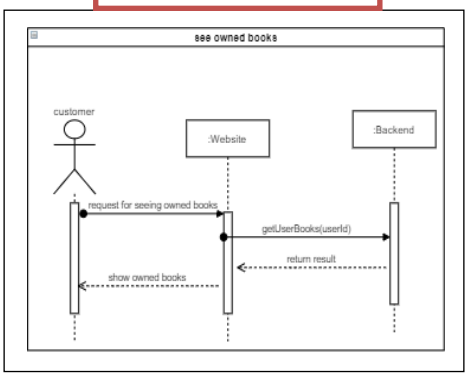
Create Order



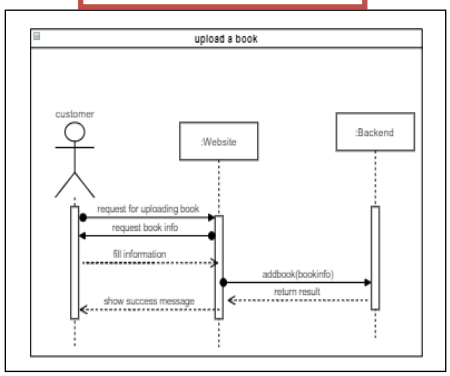
See Past Offers



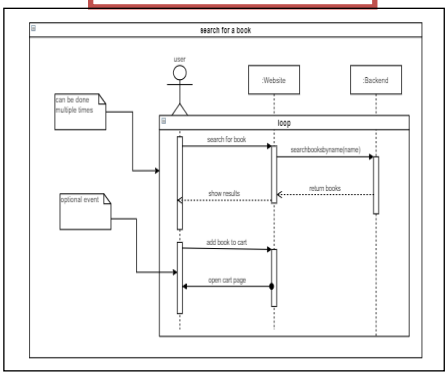
See Owned Books



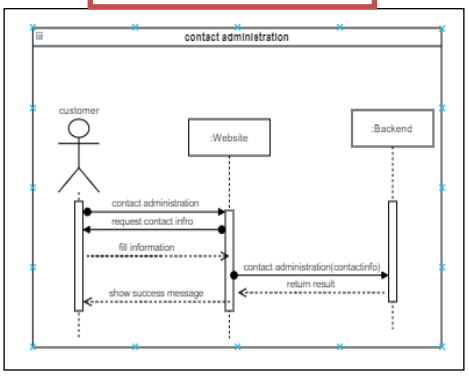
Upload a Book



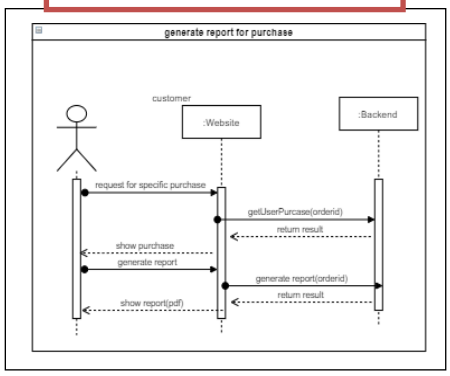
Search for a Book



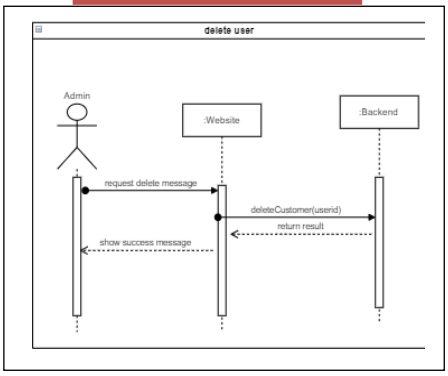
Contact Administration



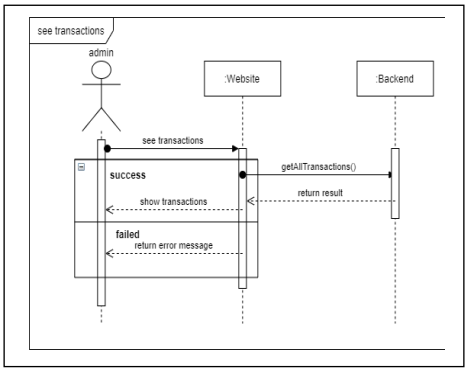
Generate Report for Purchase



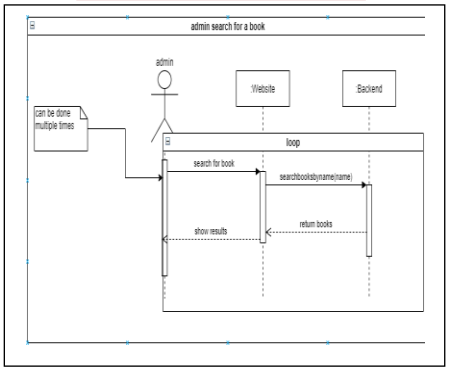
Delete User



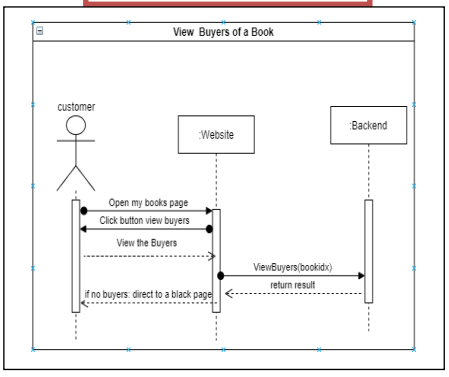
See Transactions



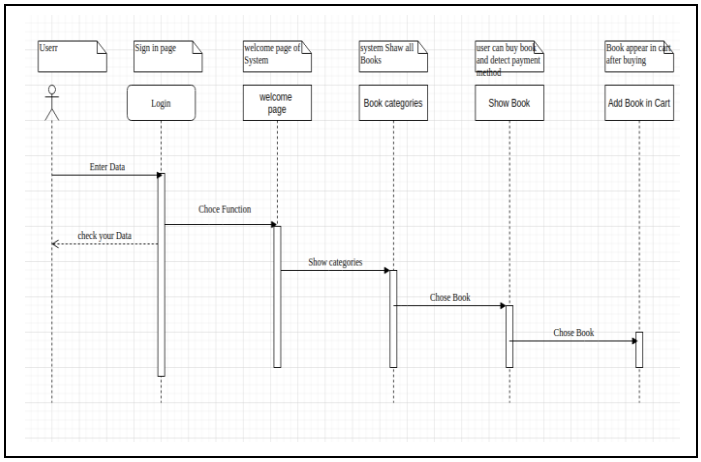
Admin Search for a Book

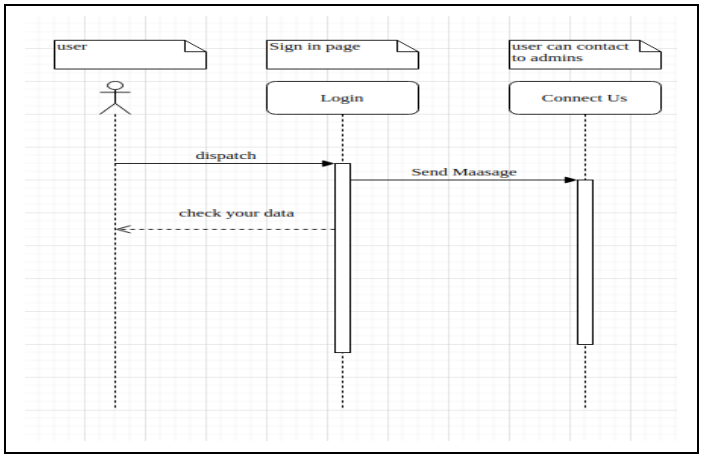


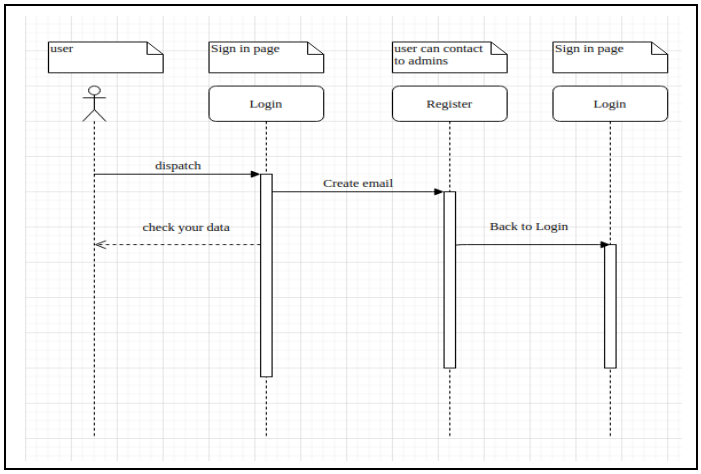
View Buyers of a Book

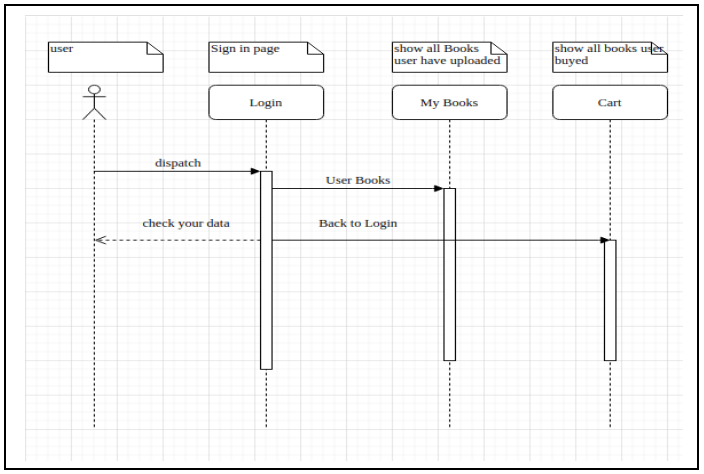


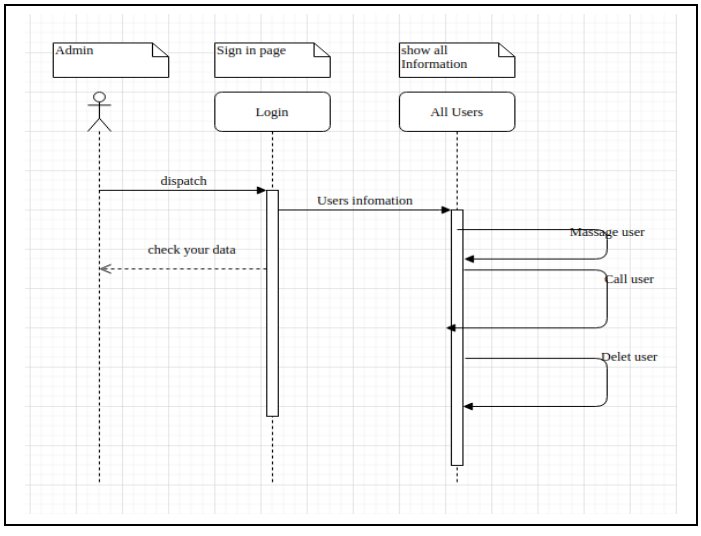
f) System Sequence Diagrams (SSDs)

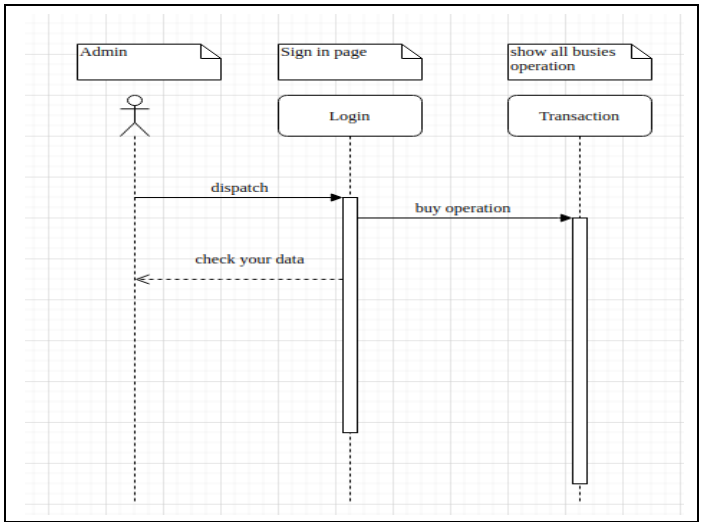




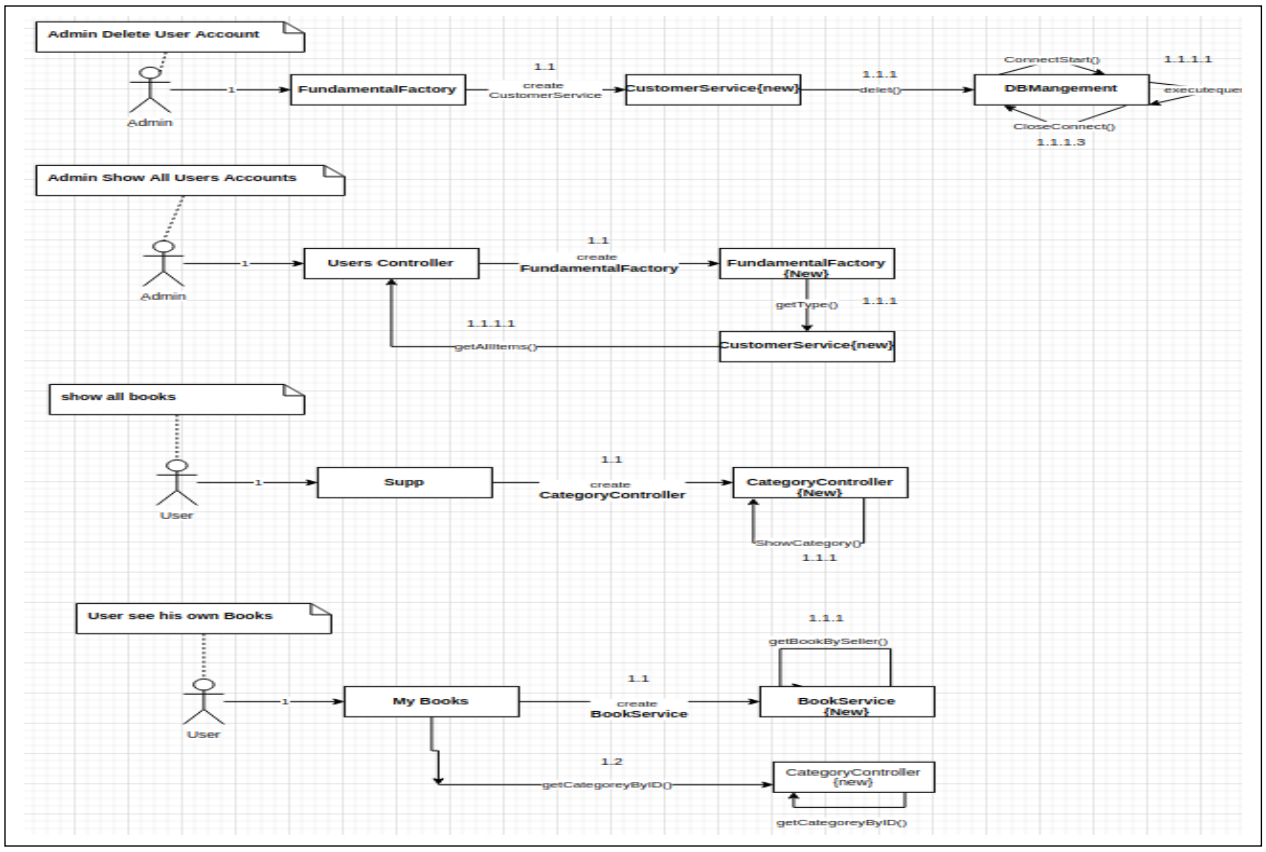








g) Collaboration/Communication Diagram(s)



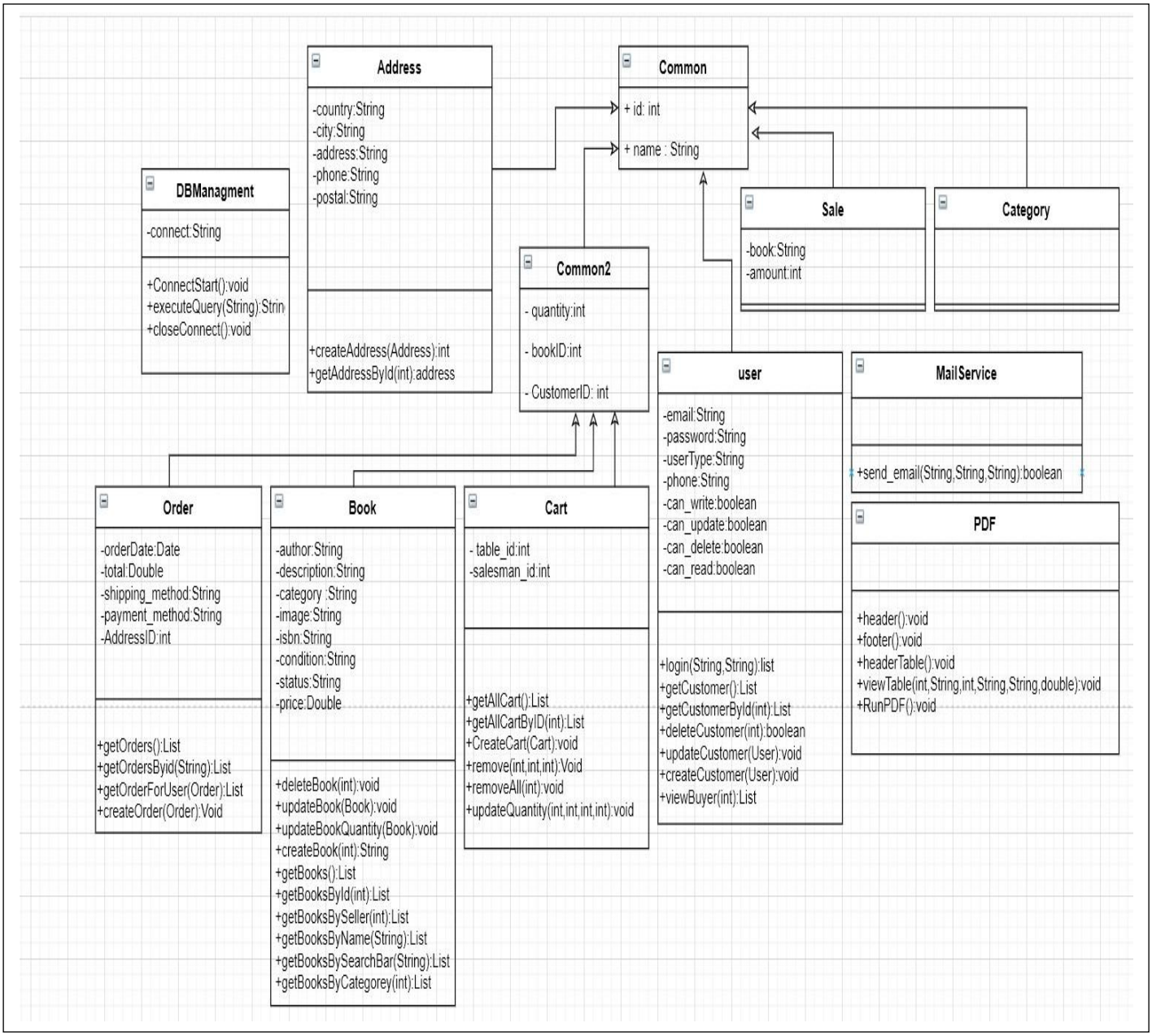
h) Which strategy (or strategies) did you use to implement the use-cases: One Central Class, Actor Class, or Use-Case class.

we used the user case owner strategy in our use case implementation ,we wanted the use case diagram to be clear for the customers, we made it for them, so it should be from their perspective.

its advantages:	Its disadvantages:
1. Customer friendly 2. Easy to understand 3. straightforward	1. Not clear to developers 2. Lake of data representation or visualization in some cases

i) Class Diagram 2: An intermediate version based on the interaction diagrams

We have created class diagram 2 before we apply design patterns and the use the OOP properties so, it contains attributes and classes, some more operations added after version 1.



j) Four Design Patterns Applied

Definition: Design pattern is a well-described and good solution for the common software problems, and it's very popular between developers.

First, we need to demonstrate that the solid principles have been applied clearly in the code to make a well-designed and easily readable code.

We use design patterns to save time and reduce effort because it's already defined before in java and it provides industry standard approach for a specific software problem to be solved, it depends on reusability of what we have to reduce total cost, and because they are already defined they make code easy to understand and make it highly maintainable.

Categories of design patterns: Creational DP, Structural DP, Behavioral DP

In our project, we used four patterns that will be discussed in the next lines

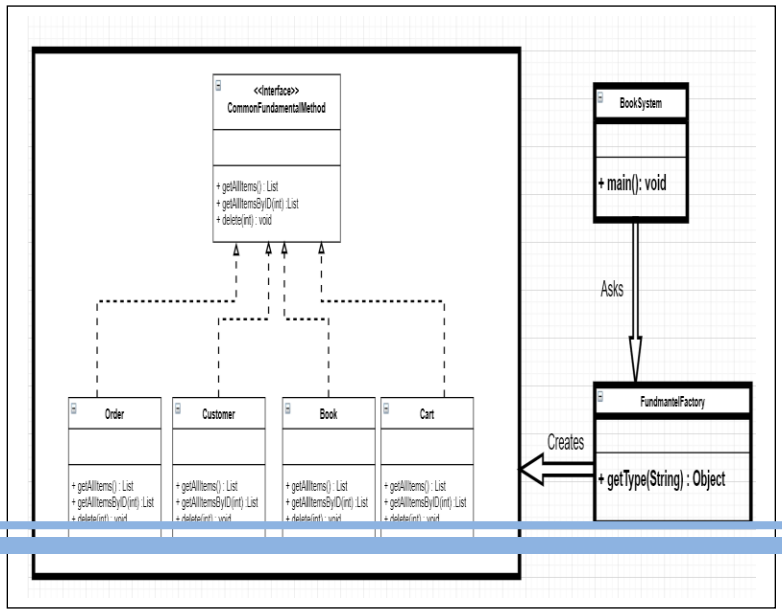
Factory pattern

Description: the aim of this pattern is to get a copy or instantiation from the client program to the factory class.

Problem it solves used when we want to return one sub-class from multiple sub-classes that is grouped in one super class based on input.

Affection on my system design:

We used factory pattern factory pattern to get a copy from the methods in the figure and group them in a CommonFundemntalMethod class to share them with the sub classes Order, Customer, Book, Cart. better than declare them in each class and this how to make a re-

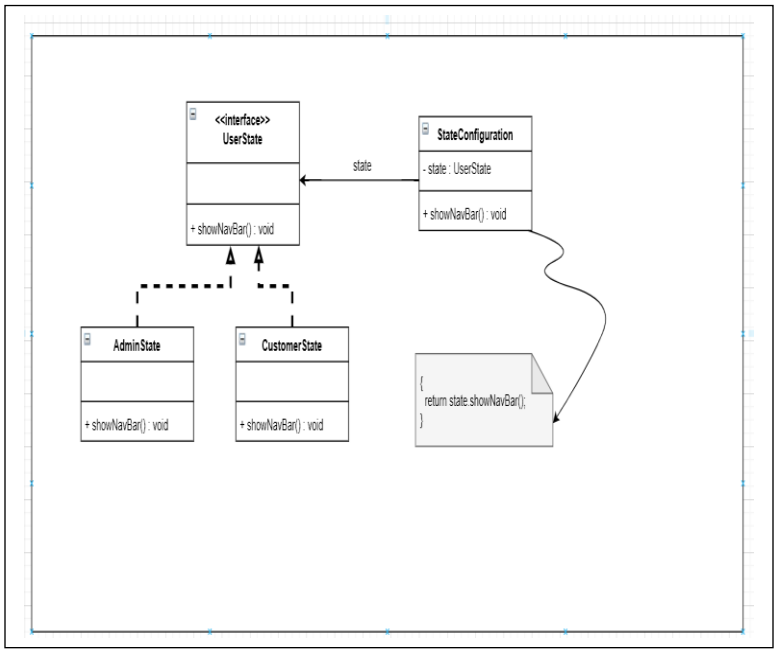


State pattern

Description: State design pattern is used if we have to change the behavior of an object or variable based on its state, we can have a state variable in the Object and use if-else condition block to perform different actions based on the state. State pattern is used to provide a systematic and loosely coupled way to achieve this through Context and State implementations.

Problem it solves very useful at facing problems that related to a state of an object in a part of runtime , the most common example on it is the Tv remote that when we click on its button state of tv changed to turn on , then if we click it again state will back to turn off . with this concept it can help in so many problems that the system want to change a part or a state in it.

Affection on my system design:



We used State design pattern in differentiate between system stack holders (admin, user) as we have a column name “user-type” in the user table in our database this column hold a value for each one who use our system we set the value of user-type of admin equal to 1 , and 0 for users .

So, we used state design pattern to hold the value of object usType and change project functionality according to this value

as example: the navbar of the pages changes according admin or user as shown in the figure. Also, we disable add to cart button for admin using usType value.

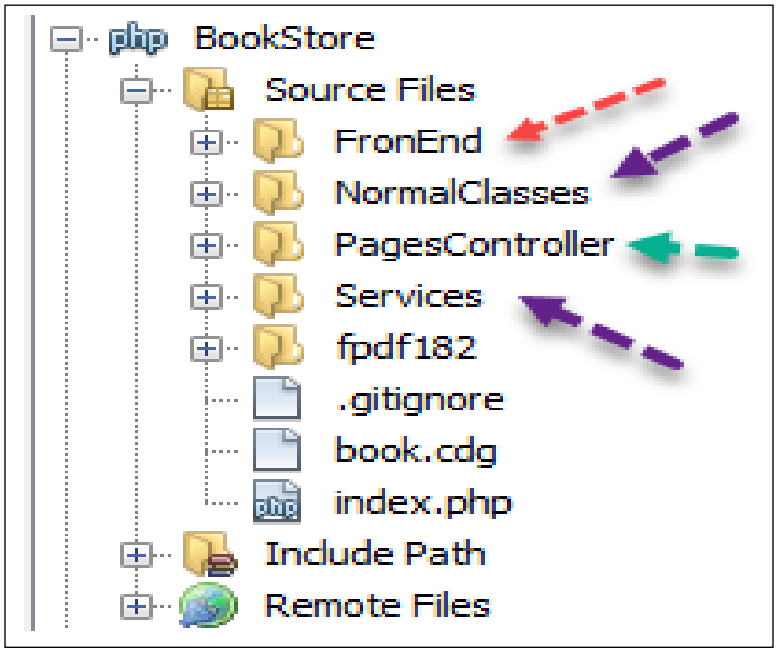
MVC pattern

Description: The MVC architecture pattern is used from a long time ago in software engineering, where almost used by all languages with a little bit different technique.
MVC stands for Model, View, Control where it separates the application into different three parts.

View is the UI that's appeared to the user of the app, Controller is the link between Model and View, it sends that data to the View , and sends actions and mutate the state of the model .

Problem it solves MVC keeps the code clean, easy to maintain and to understand, and it's very popular pattern in web development

Affection on my system design:



We used MVC as shown in the figure we make the package Normal Classes, Services represent the Model
and Front end package represent the view as it holds everything related to shape of the system
and package Pages Controller represent the Controller of the MVC we made this to make the system easier and more collective as we use the control as an intermediate between view and models also, to ensure high loading, and response time ,as this architecture is used by big companies and many websites

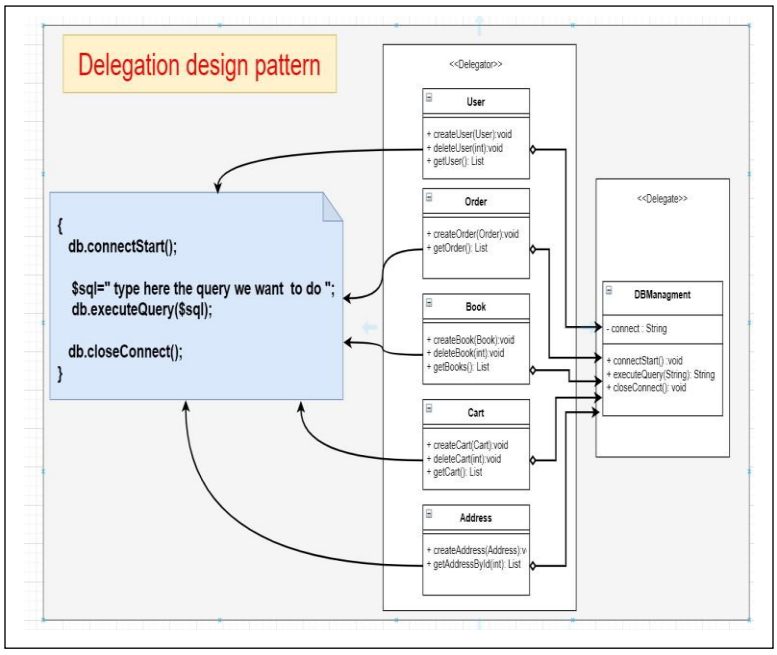
Delegation pattern

Description: Delegation Pattern abstract methods and functionality of a class into another class, not only this but the real power of this pattern comes when there are multiple delegates. The delegator typically has a method for each delegate that will convert the delegator to use that delegate.

It is useful for understanding to compare the delegation pattern to inheritance. Both are powerful reuse techniques with a few of key differences; inheritance is directly supported by today's object-oriented programming languages and enables the use of polymorphism, whereas the delegation pattern allows the delegate to be changed at run-time.

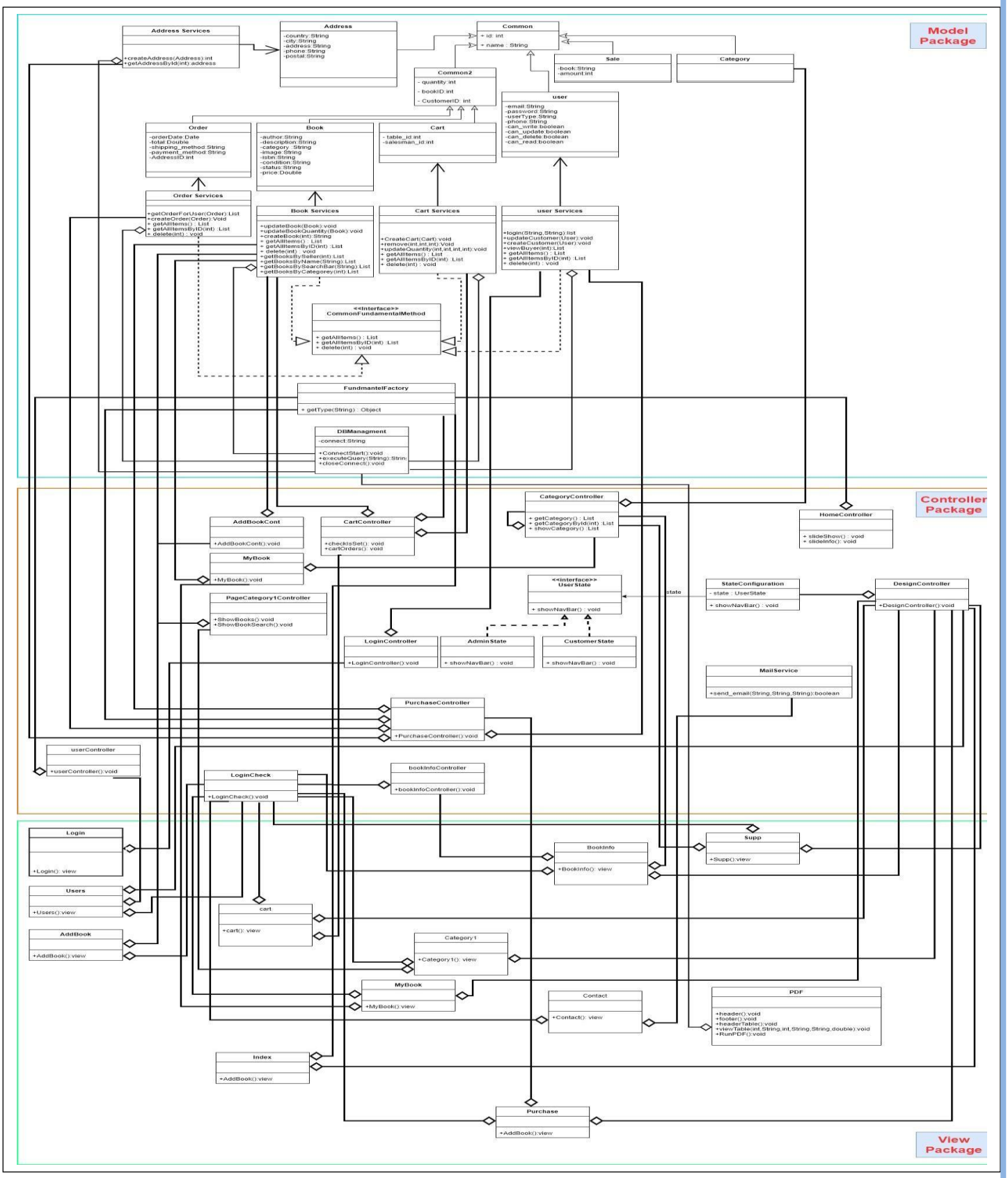
Problem it solves very useful to decrease system run time, increase its flexibility and separates the different set of methods

Affection on my system design:



From the description its clear that we use this pattern when we want to abstract methods of a class so we create object from this class into the classes where we want to use those methods so, as shown in the figure we have class DB management which open connection with database so, this method is very useful and nearly used in every part of the project that why we used delegation pattern here to use these method in another classes better than re-typing method at every class or copy-paste the method.

k)Class Diagram 3: The final version, after applying the design pattern(s) and any other modifications



Design some User Interfaces

The Interaction Design is used to test if the interface of the system is friendly user or not by using the 5 dimensions of it:

1. The Words:

•This means that the word used in the system must be easy to understand and give information about example [home page, back button] but also doesn't contain much information just the information that gives the meaning.

2. The Visual Representations:

•This type means that the icon or the image that used in the system must be easy to understand because this element used to give the information to the user so, they must be easy and friendly.

3.**The Physical Object or Space**:

•This type describes how the user physically interact with this system example [using laptop, mobile phone] and where the spaces that the user use this system example [supermarket, school] so all this information will affect this dimension.

4.**The Time**:

•This type used to define how much does the system take to get what the user need and also how much time the animation take in the system which consider as important major for all the user to make the system doesn't need to much time to obtain what they need.

5.**The Behavior**:

•This type used to show the user can perform his action in the system and what the feedback from the user about this system and it depends on the previous dimension.