**Your Books Everywhere!**

**Student: Ban Erno Emmanuel**

**Group: 30238**

Table of Contents

[1](#__RefHeading___Toc1562_2178577591)

[Your Books Everywhere! 1](#__RefHeading___Toc1616_2178577591)

[1](#__RefHeading___Toc1564_2178577591)

[Table of Contents 2](#__RefHeading___Toc1568_2178577591)

[1. Requirements Analysis 3](#__RefHeading___Toc1570_2178577591)

[1.1 Assignment Specification 3](#__RefHeading___Toc1572_2178577591)

[1.2 Functional Requirements 3](#__RefHeading___Toc1574_2178577591)

[1.3 Non-functional Requirements 3](#__RefHeading___Toc1576_2178577591)

[2. Use-Case Model 4](#__RefHeading___Toc1578_2178577591)

[3. System Architectural Design 5](#__RefHeading___Toc1596_2178577591)

[3.1 Architectural Pattern Description 5](#__RefHeading___Toc1608_2178577591)

[3.2 Diagrams 6](#__RefHeading___Toc1610_2178577591)

[4. UML Sequence Diagrams 7](#__RefHeading___Toc1598_2178577591)

[5. Class Design 7](#__RefHeading___Toc1600_2178577591)

[5.1 Design Patterns Description 7](#__RefHeading___Toc1612_2178577591)

[5.2 UML Class Diagram 7](#__RefHeading___Toc1614_2178577591)

[6. Data Model 8](#__RefHeading___Toc1602_2178577591)

[7. System Testing 8](#__RefHeading___Toc1604_2178577591)

[8. Bibliography 8](#__RefHeading___Toc1606_2178577591)

1. Requirements Analysis

# Assignment Specification

The task is to build a book management service. A user should be able to create an account, choose a payment plan and login to search the book library. Payments can be done via a cash only policy and need to be validated by library staff. The library is managed by staff and can be filtered by release date, author, title, genre. If a book is available a user can add it to your library. If not the user can join a waiting list. Once a book has been read by a user it can be returned via the online library return function. This assigns the book to the next user in the waiting list after validation of the return by library staff. The service also provides users with dynamic recommendations based on latest trends (popular borrowed books) or user defined interests by genre or topic.

# Functional Requirements

The functional requirements of the project are:

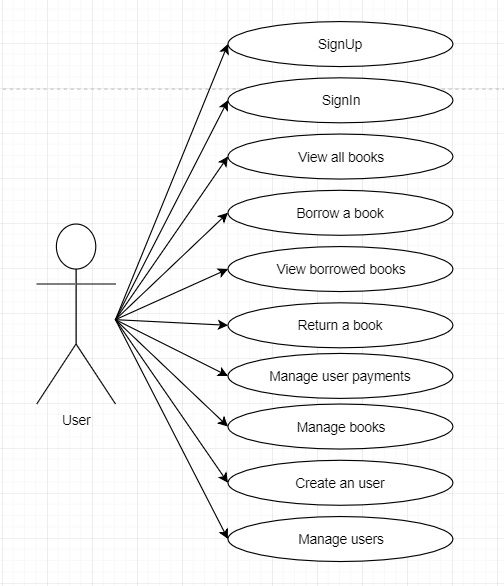
* Users:
  + Create an account
  + Choose a payment plan
  + Log in the account
  + Borrow a book
  + View borrowed books
  + Return a book
  + Book reccomendations
  + Get notification when a wanted book becomes available
* Librarian:
  + Log in the account
  + Manage user payments
  + Manage books
* Admin:
  + Log in the account
  + Create an users and librarians
  + Manage the users and librarians

# Non-functional Requirements

The non-functionat requirements of the project are:

* Security:
  + Password hashing
  + DB security
  + Application security
* Portability
* Good architectural design for easy maintenance

2. Use-Case Model

**

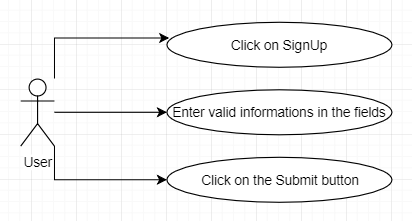
**Use case:** Sign up

**Level:** Sub function

**Primary actor:** User

**Extensions:** All fields contain valid informations → success

else → fail



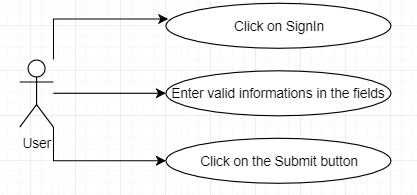
**Use case:** Log in

**Level:** Sub function

**Primary actor:** User/Staff

**Extensions:** Username and password match → success

else → fail

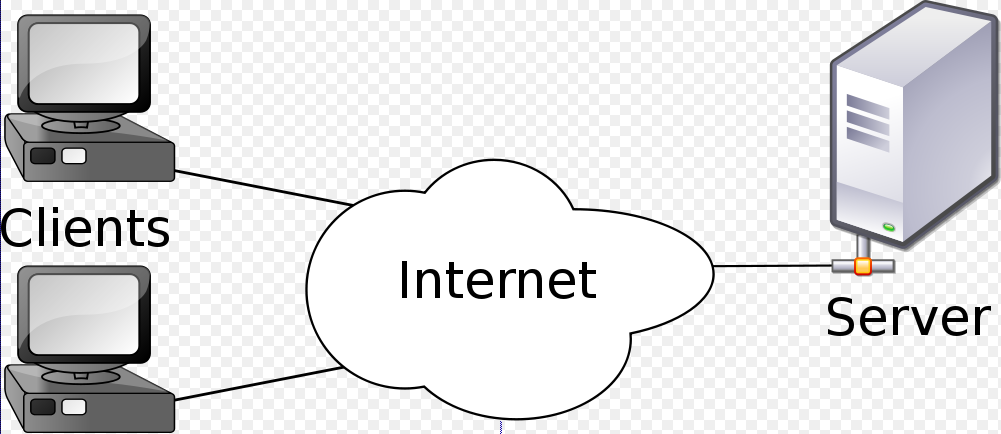


3. System Architectural Design

# 3.1 Architectural Pattern Description

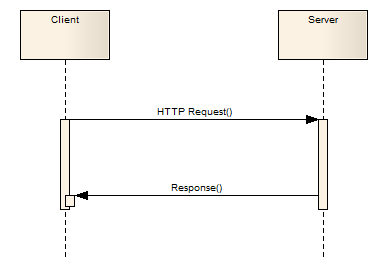
Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about *how* to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.

**

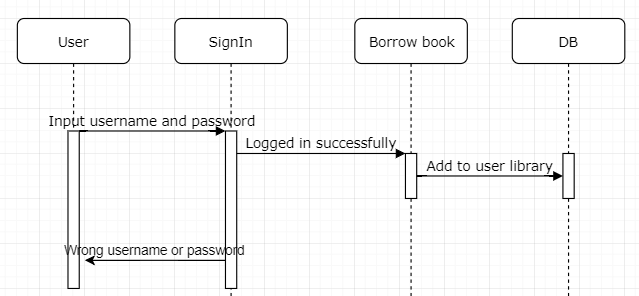
# 3.2 Diagrams

Package diagram:

**

4. UML Sequence Diagrams

Sequence diagram for borrowing an available book:



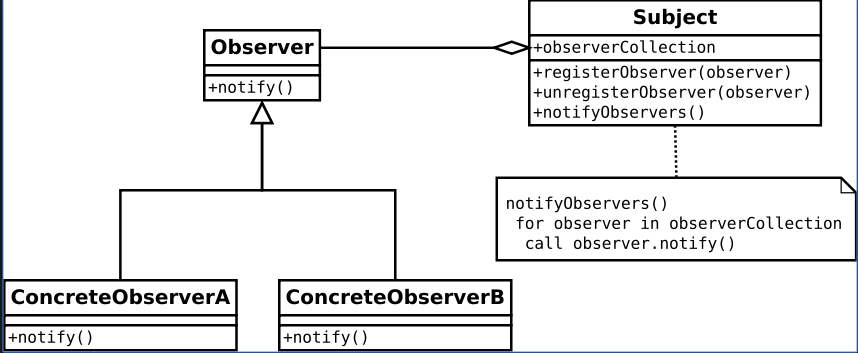
5. Class Design

# 5.1 Design Patterns Description

The factory pattern comes under the creational patterns list category. It provides one of the best ways to create an object. In factory pattern, objects are created without exposing the logic to client and referring to the newly created object using a common interface.

Factory patterns are implemented in Python using factory method. When a user calls a method such that we pass in a string and the return value as a new object is implemented through factory method. The type of object used in factory method is determined by string which is passed through method.

# 5.2 UML Class Diagram

**

6. Data Model

The data models for this project are:

* **waiting\_list:**
  + id
  + id\_book
  + id\_user
* **books:**
  + id
  + title
  + author
  + genre
  + available
* **borrows:**
  + id
  + id\_book
  + id\_user
* **users:**
  + id
  + username
  + password
  + email
  + role
  + price\_plan
  + approved\_user

7. System Testing

The system was tested manually.

8. Bibliography

https://hackernoon.com/dont-install-postgres-docker-pull-postgres-bee20e200198

https://docs.docker.com/samples/library/postgres/

https://stackoverflow.com/

https://www.w3schools.com/

[https://www.tutorialspoint.com/](https://www.tutorialspoint.com/design_pattern/factory_pattern.htm)