Tiem Ban Nuoc

Book Lovers
Software Architecture Document

Version 4.0

Revision History

Date	Version	Description	Author
23/10/23	1.0	Initial ideas - PA0	Group 11
25/10.23	2.0	Finalize ideas - PA1	Group 11
12/11/23	3.0	Design basic interface	Group 11
22/11/23	4.0	Software Architecture Document	Group 11

Table of Contents

Software Architecture Document

1. Introduction

- This document represents an overview of the architectural design of the Book Lovers system. It outlines the key architectural decisions, components, and relationships that define the overall structure and behavior of the system. The SAD serves as a blueprint for the development team, ensuring that all stakeholders have a shared understanding of the system's architecture and its implications for implementation and maintenance.
- The following acronyms and abbreviations are used in this SAD:
    + SAD: Software Architecture Document
    + API: Application Programming Interface
    + MVC: Model-View-Controller
    + DB: Database
- This project does not reference any document or specification

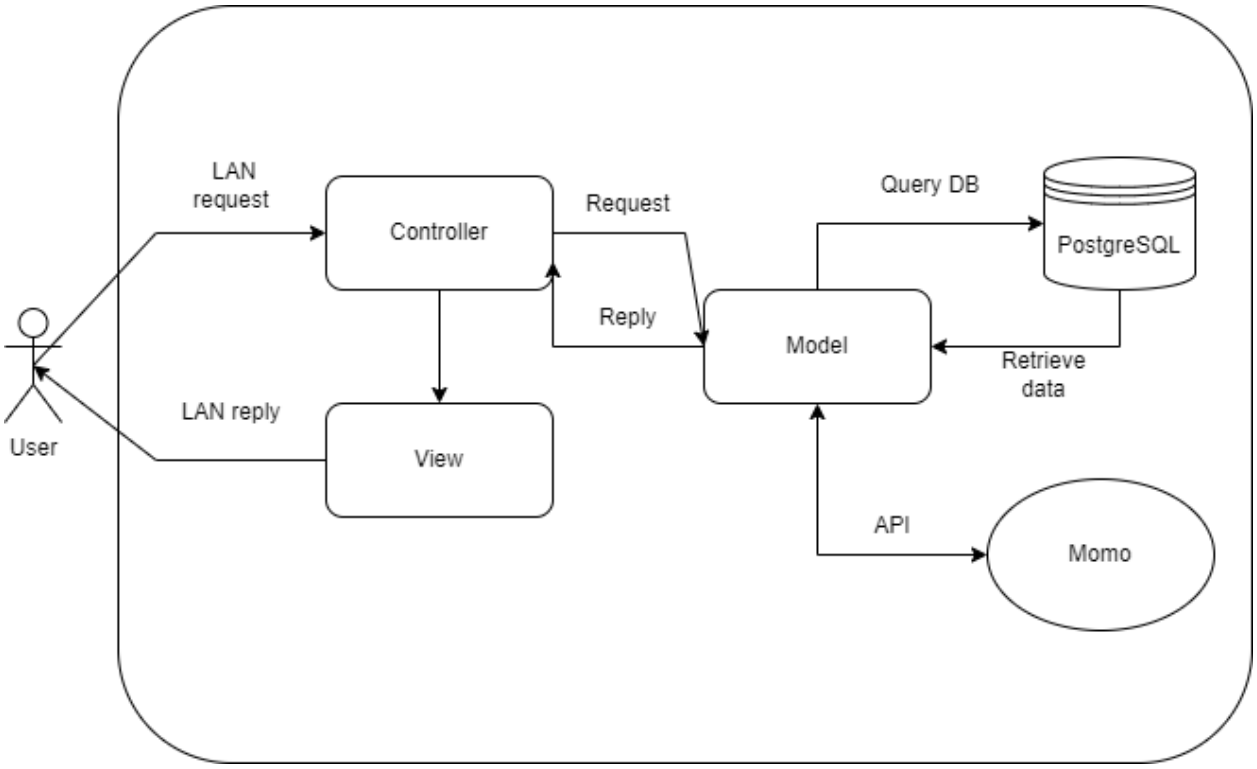
2. Architectural Goals and Constraints

- Environment: Web browsers.
- Front-end: HTML, CSS
- Back-end: JavaScript, NodeJS
- Database: PostgreSQL
- Payment methods: Momo. The system controls the flow and protects the transactions made by the users.
- Privacy: The system secures users' information from other users and outsiders.
- The system must be easy to maintain and upgrade in the future
- The system must be fast, stable, and accurate

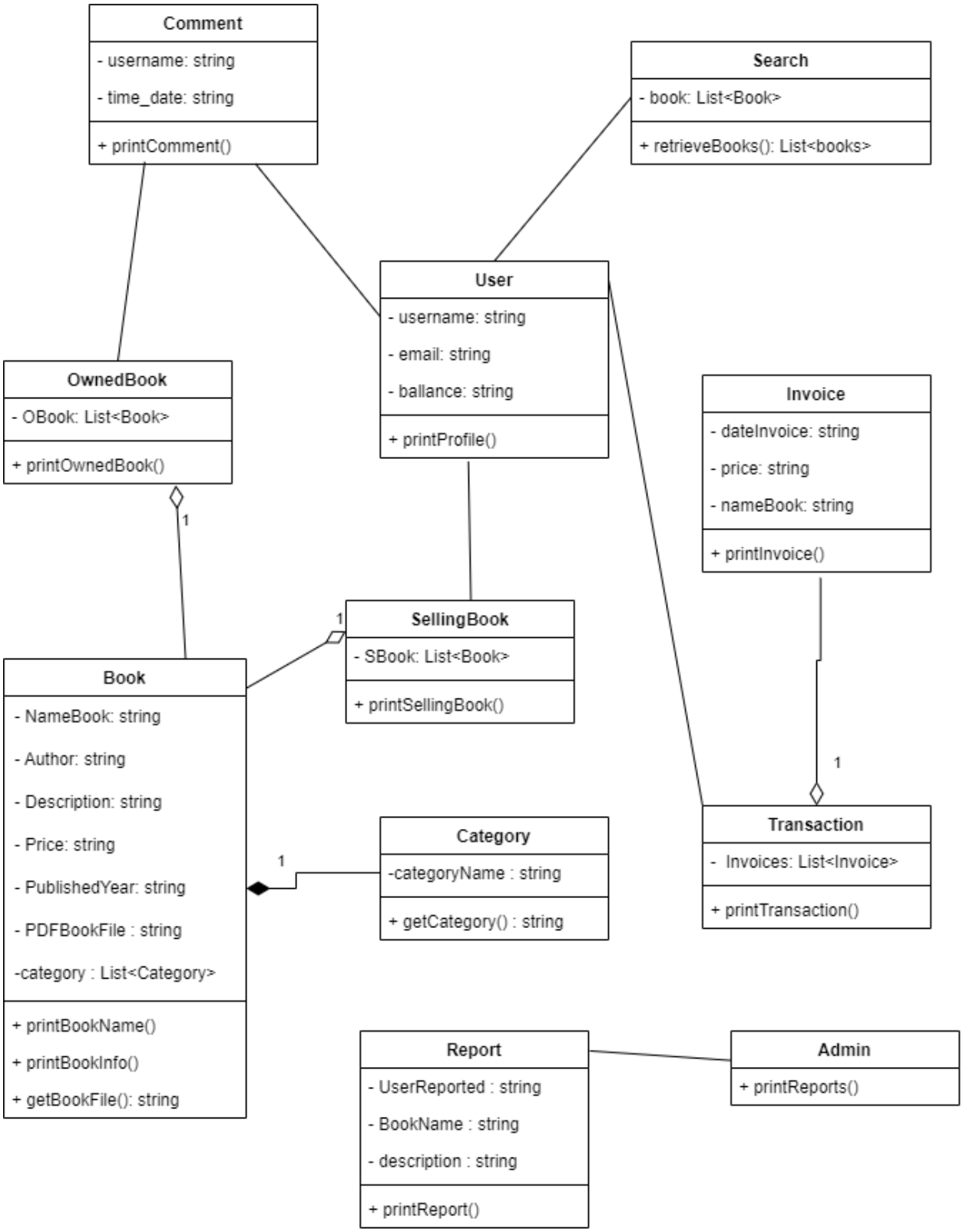
3. Use-Case Model


Download books
<<extend>>
Read books
(Not free)
<<include>>
Buy books
Add book to shopping
cart
Remove book from
shopping cart
Buyer
Comment on books
Read books
(Free)
Deposite money
Withdraw money
Arrange meetings
Join meetings
Registered user
Review profile user
Report user
Sell books
Remove book
Seller
Login
Forgot password
Sign up
Search for
books/documents/references
Share books
Recommend books
User

4. Logical View



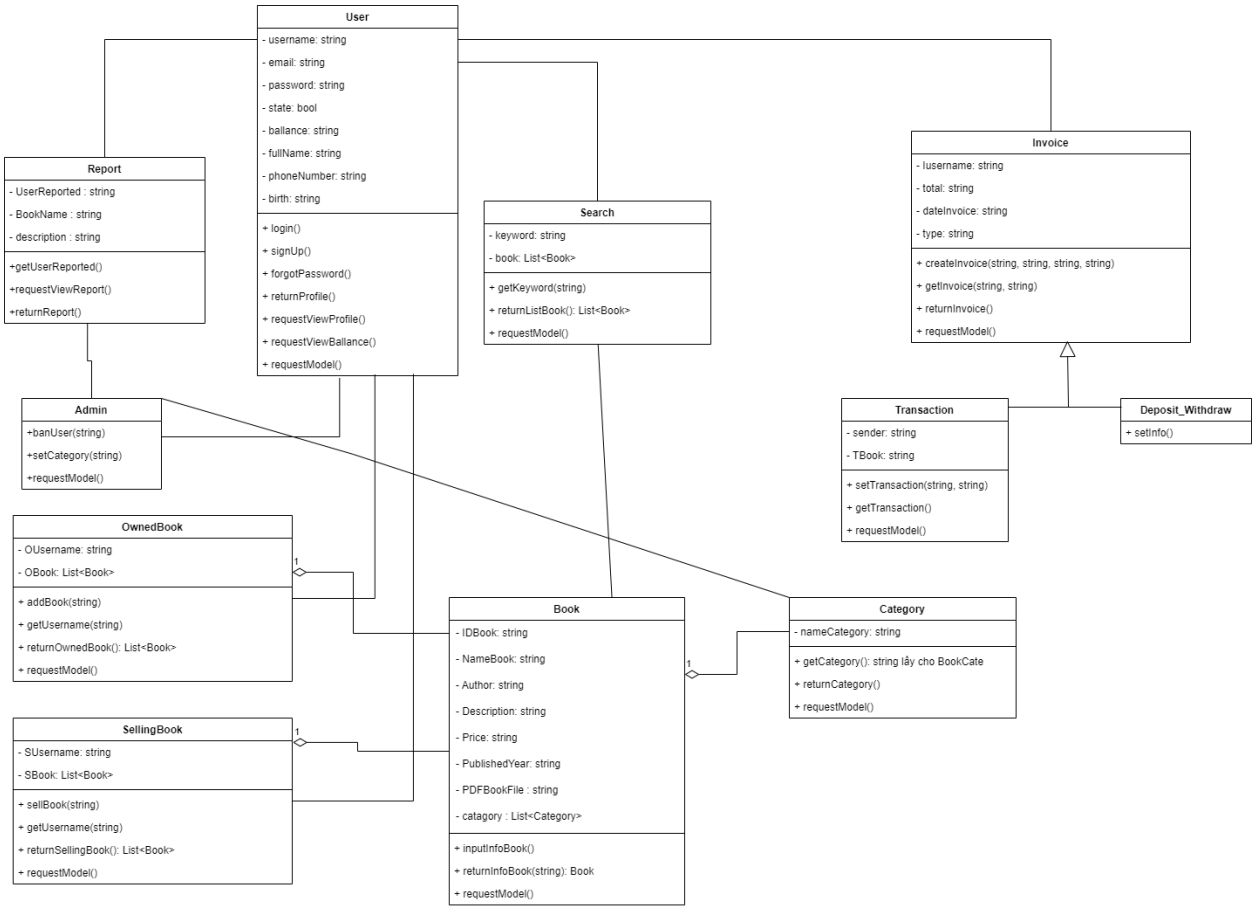
4.1 Component: View



This component is responsible for how objects are displayed to the users such as users' information, books information, owned book list, search bar, etc.

- **Class User:** give information about the user's profile.
    + printProfile(): when a user clicks on the button Profile in horizontal bar, a little vertical list will appear and show his/her profile including username, email and balance.
- **Class Report:** print report content of a user for a book.
    + printReport(): when an admin want to read a content of a report by an user for a book
- **Class Admin:** an admin has permission to read all report contents.
    + printReports(): an admin uses it when wanting to list reports.
- **Class Book:** print book's information or only name when user views it or it is listed when user finds on search bar with related keywords. It also gives the book file for reading.
    + printBookName(): print the book's name into the list of searching or for the detail when viewing its page.
    + printBookInfo(): when a user views this book's page, all information will be shown.
    + getBookFile(): after the user purchased this book, he/she can get the link of the book.
- **Class OwnedBook:** an user can access "My Books" and see all his/her books that are bought.
    + printOwnedBook(): when the user accesses "My Books", the list of books which was purchased will be shown for him/her.
- **Class Comment:** when a user views a book site, if a book has any comment, it will be shown, if not there is no comment will be shown.
    + printComment(): when the user visits a book site. If the book has any comment, each comment will be shown with the username of the commenter, comment content and the time when the commenter commented about this book.
- **Class SellingBook:** A marketplace where users can view which books are on the market and who sells them.
    + printSellingBook(): print out all the books that are being sold and who sells them.
- **Class Invoice:** When users make transactions such as buy/sell/withdraw/deposit then an invoice is created and they can view it.
    + printInvoice(): print out the details of an invoice requested by the user.
- **Class Transaction:** all users can access the "history of transactions" to see all bills that they have purchased in the past.
    + printTransaction(): the list of all bills will be showed in "history of transactions" section in order by descending by time
- **Class Search:** The search bar is shown to the users and they can type in the keywords about the books they want to search.
    + retrieveBook(): List<Book>: return and show a list of books that match the keyword that is input by users.
- **Class Category:** A book can belong to one or more categories, which can be viewed by the users.
    + getCategory(): string: return a line of text which contains the name of the categories where the books belonged.

4.2 Component: Controller



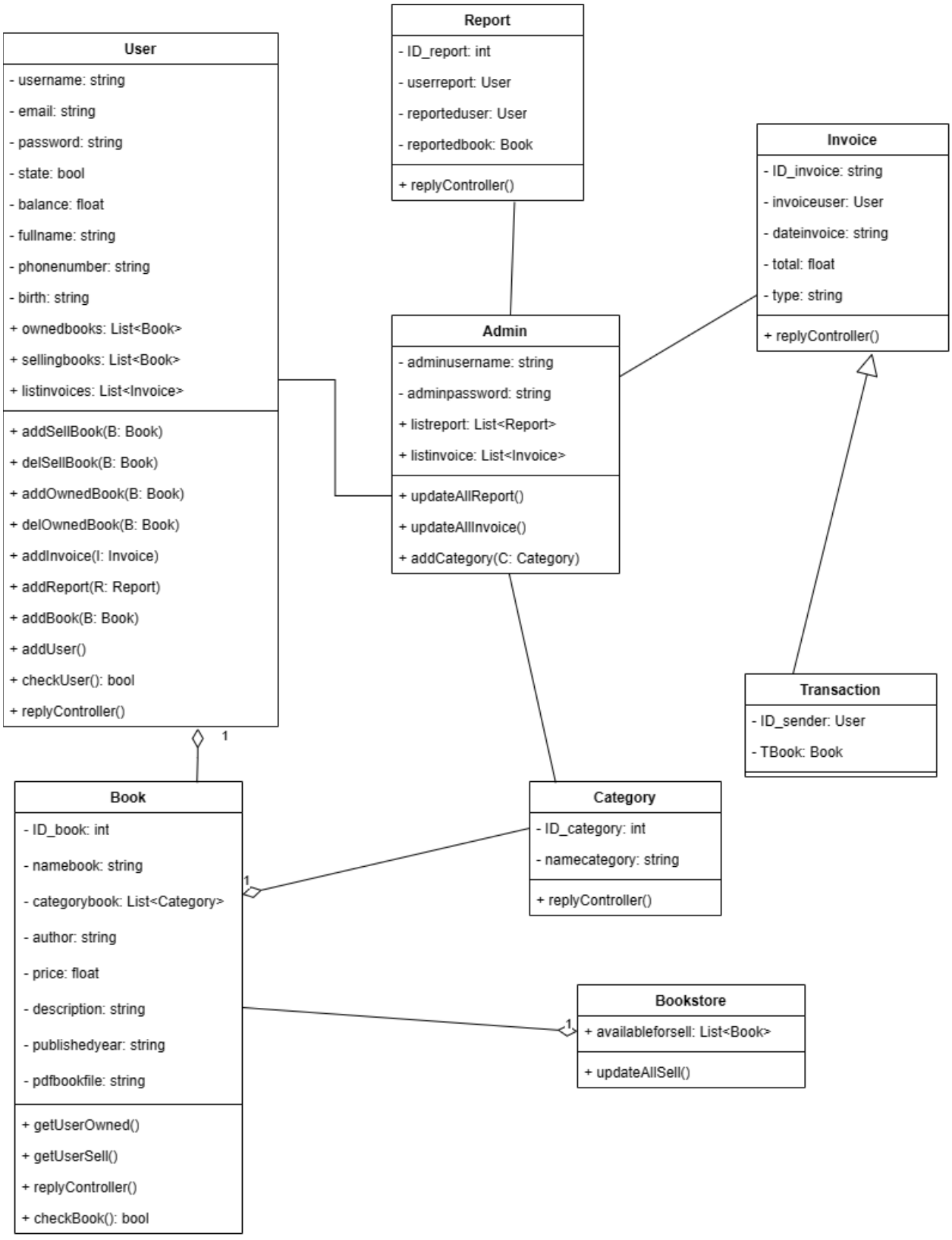
The Controller component serves as the orchestrator, managing interactions between users and the underlying data model. It plays a crucial role in processing user requests, handling administrative tasks, and facilitating communication with the Model component. Let's delve into the classes within the Controller component:

- **Class Admin:** Manages administrative tasks within the system.
  - + banUser(string): ban an user from the system.
  - + setCategory(string) : Sets category.
  - + requestModel(): Requests model interaction for administrative tasks.
- **Class Report:** Handles reporting functionality.
  - + getUserReported(): Gets reported from users.
  - + requestViewReport(): receive a request from Admin to view all the reports.
  - + returnReport(): back to view for admin to view the reports.
- **Class User:** user information about account, personal information and ballance
  - + login(): Handles user login.
  - + signUp(): Handles user registration.
  - + forgotPassword(): Manages password recovery.
  - + returnProfile(): Responds user’s profile to view.
  - + requestViewProfile(): Receives a request from a user who wants to watch his profile.
  - + requestViewBallance(): Receives a request from a user who wants to watch his balance.
  - + requestModel(): Initiates a request to the model.
- **Class Search:** Handles search functionality.
  - + getKeyword(string): Receives a keyword from the user.

    + returnListBook(): Returns a list of books which are based on the keyword that is received.
    + requestModel(): Initiates a request to the model.
- **Class Deposit_Withdraw:** Manages user deposits and withdrawals.
    + setInfo(): Sets information of deposit or withdrawal.
- **Class Transaction:** Handles transaction-related tasks .
    + setTransaction(string, string): Sets transaction information.
    + getTransaction(): Gets transaction information.
    + requestModel(): Initiates a request to the model.
- **Class Invoice:** Manages invoice creation and retrieval.
    + createInvoice(string, string, string, string): Creates an invoice.
    + getInvoice(string, string): Gets user’s request to view invoice history.
    + returnInvoice(): Returns the information based on the user's request.
    + requestModel(): Initiates a request to the model.
- **Class OwnedBook:** Manages books owned by a user.
    + addBook(string): Adds a book to the user's collection.
    + getUsername(string): Gets the username to print all the books that are owned by that user.
    + returnOwnedBook(): Returns the user's owned books .
    + requestModel(): Initiates a request to the model.
- **Class SellingBook:** Manages books available for sale.
    + sellBook(string): User uploads a book to sell
    + getUsername(string): Gets the username to print all the books that are being sold by that user.
    + returnSellingBook(): Returns the user's selling books.
    + requestModel(): Initiates a request to the model.
- **Class Book:** Handles book-related tasks.
    + inputInfoBook(): Inputs all the information of the book
    + returnInfoBook(string): Returns book information.
    + requestModel(): Initiates a request to the model.
- **Class Category:** Manages book categories.
    + getCategory(): Gets all categories.
    + returnCategory(): Returns book’s categories.
    + requestModel(): Initiates a request to the model.

4.3 Component: Model

The Model component forms the backbone of the application, managing data storage and retrieval. It encapsulates the core entities and their interactions. Let's explore the classes within the Model component:


User
- username: string
- email: string
- password: string
- state: bool
- balance: float
- fullname: string
- phonenumber: string
- birth: string
+ ownedbooks: List<Book>
+ sellingbooks: List<Book>
+ listinvoices: List<Invoice>
+ addSellBook(B: Book)
+ delSellBook(B: Book)
+ addOwnedBook(B: Book)
+ delOwnedBook(B: Book)
+ addInvoice(I: Invoice)
+ addReport(R: Report)
+ addBook(B: Book)
+ addUser()
+ checkUser(): bool
+ replyController()
Report
- ID_report: int
- userreport: User
- reporteduser: User
- reportedbook: Book
+ replyController()
Invoice
- ID_invoice: string
- invoiceuser: User
- dateinvoice: string
- total: float
- type: string
+ replyController()
Admin
- adminusername: string
- adminpassword: string
+ listreport: List<Report>
+ listinvoice: List<Invoice>
+ updateAllReport()
+ updateAllInvoice()
+ addCategory(C: Category)
Transaction
- ID_sender: User
- TBook: Book
Book
- ID_book: int
- namebook: string
- categorybook: List<Category>
- author: string
- price: float
- description: string
- publishedyear: string
- pdfbookfile: string
+ getUserOwned()
+ getUserSell()
+ replyController()
+ checkBook(): bool
Category
- ID_category: int
- namecategory: string
+ replyController()
Bookstore
+ availableforsell: List<Book>
+ updateAllSell()
1
1
1
1

- **Class User:** Query users' information.
  - + addSellBook(B: Book): insert Book into SELLINGBOOK table of database and into sellingbooks attribute. if the Book object is not in BOOK table of database then the Book is also added into the BOOK table using addBook(B: Book)
  - + delSellBook(B: Book): delete Book from SELLINGBOOK table and from sellingbooks attribute.
  - + addOwnedBook(B: Book): add Book into OWNEDBOOK table of database and into ownedbooks attribute. Same as addSellBook(B: Book) when the Book object is not in the table yet
  - + delOwnedBook(B: Book): delete Book from OWNEDBOOK table and from ownedbooks attribute.
  - + addInvoice(I: Invoice): add Invoice into INVOICE table and into listinvoices attribute.
  - + addReport(R: Report): add Report into REPORT table.
  - + addBook(B: Book): add Book object into BOOK table of database.
  - + addUser(): initial self User object into USER table of database.
  - + checkUser(): bool: check if the User object is already existed in USER table of the database.
  - + replyController(): Initiates a reply to the controller.
- **Class Admin:** Administrator access rights
  - + updateAllReport(): update all new reports from usersupdate the all users' reports to the attribute listreport from REPORT table.
  - + updateAllInvoice(): update all new invoice from users update the all users' invoices into the attribute listinvoice from INVOICE table.
  - + addCategory(): add category object into CATEGORY table of the database.
- **Class Report:** Reply all reports from users
  - + replyController(): store the admin's reply for the user's report about a book or a other user
- **Class Book:** Query books' information
  - + getUserOwned(): get all the usernames who purchase this book from OWNEDBOOK table.
  - + getUserSell(): get all usernames who sell this book from SELLINGBOOK table.
  - + replyController(): Initiates a reply to the controller.
  - + checkBook(): bool: check if the Book object is already existed in BOOK table of the database.
- **Class Bookstore:** Query books' information for Home page
  - + updateAllSell(): update all the selling books to the availableforsell attribute from SELLINGBOOK table.
- **Class Category:** Query implemented Category
  - + replyController(): Initiates a reply to the controller.
- **Class Transaction:** All the transactions between sellers and buyers
- **Class Invoice:** All the Invoices which include transactions, deposit, and withdrawal history.
  - + replyController(): send the demand invoices from database to the controller

5. Deployment

[Leave this section blank for PA3 if you are writing this document for PA4.

In this section, describe how the system is deployed by mapping the components in Section 4 to machines running them. For example, your mobile app is running on a mobile device (Android, iOS, etc), your server runs all components on the server side including the database]

6. Implementation View

[Leave this section blank for PA3 if you are writing this document for PA4.

In this section, provide folder structures for your code for all components described in Section 4.]