Book Store Project

Report

EECS 4413

Team Members:

Yang Zhou:	215804800
ChengXiang Gong:	214321616
Xinqi Chen:	215639545

Table of Contents

Description:

Our project is published on this website:
http://ec2-3-96-57-51.ca-central-1.compute.amazonaws.com:8080/index.jsp.

It is the website we made and use for demo, if there is any question of this website please contact any of our group members for help.

Program

For testing, A visitor will be able to browse any product (books) on our website and check its comments and rating. A visitor can also check books by choosing their categories. However, there is something that can only be accessed by a registered user. For those functions, a visitor can register for a new account on our website, and then they will have access to their own shopping cart for purchase and even leave a comment for any book in the store.

Test Administrator:

- **Username:** admin
- **Password:** 123456

Our program source code is submitted through the moodle submission page as a war file.

The program requires the following dependencies:

activation-1.1.jar aopalliance-repackaged-2.4.0-b10.jar c3p0-0.9.5.2.jar hk2-api-2.4.0-b10.jar hk2-locator-2.4.0-b10.jar hk2-utils-2.4.0-b10.jar jackson-annotations-2.3.2.jar jackson-core-2.3.2.jar jackson-databind-2.3.2.jar jackson-jaxrs-base-2.3.2.jar jackson-jaxrs-json-provider-2.3.2.jar jackson-module-jaxb-annotations-2.3.2.jar javaee-api-7.0.jar javassist-3.18.1-GA.jar javax.annotation-api-1.2.jar javax.inject-2.4.0-b10.jar javax.mail-1.5.0.jar jersey-media-json-jackson-2.17.jar	jersey-server-2.17.jar jsp-api-2.1.jar jstl-api-1.2.jar mchange-commons-java-0.2.11.jar mysql-connector-java-5.1.36.jar osgi-resource-locator-1.0.1.jar servlet-api-2.5.jar standard-1.1.2.jar validation-api-1.1.0.Final.jar javax.ws.rs-api-2.0.1.jar jersey-client-2.17.jar jersey-common-2.17.jar jersey-container-servlet-2.17.jar jersey-container-servlet-core-2.17.jar jersey-entity-filtering-2.17.jar jersey-guava-2.17.jar jersey-media-jaxb-2.17.jar

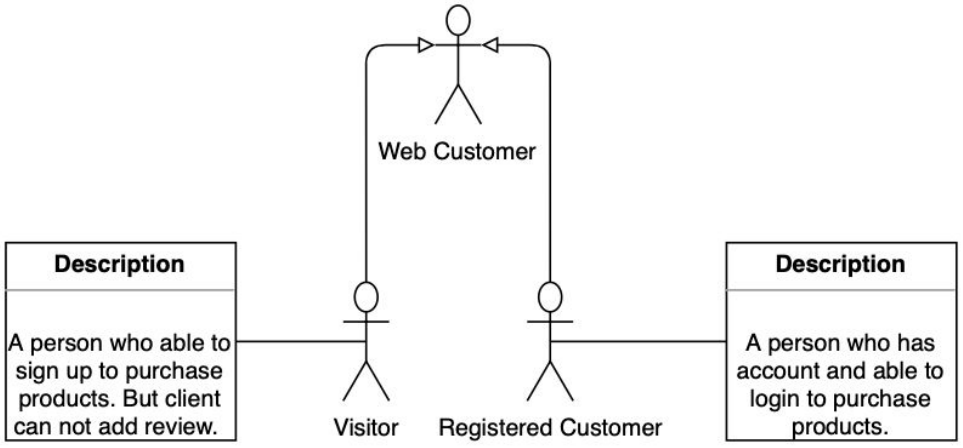
↑ All dependencies included in the submitted war file

Architecture

Program Structure:

UML Use Case:

Web Customer: The person who purchases books online.



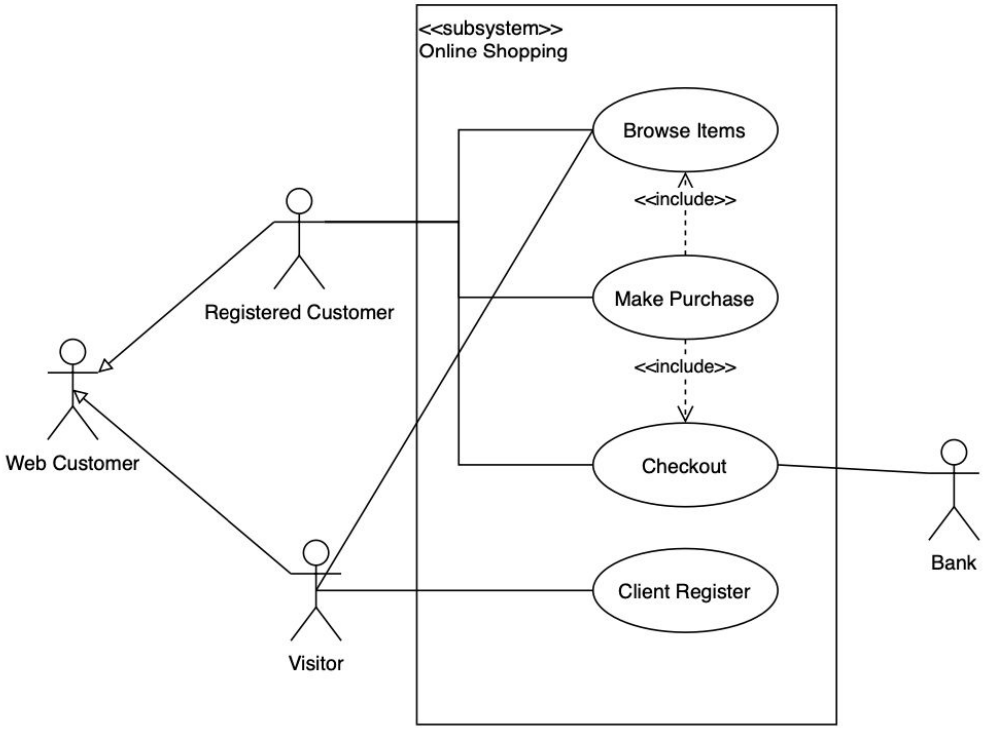
Visitor: The person who can browse Lotus Bookshop and make a purchase after creating a new account.

Registered Customer: The person who has an account and is able to login for purchasing books online.

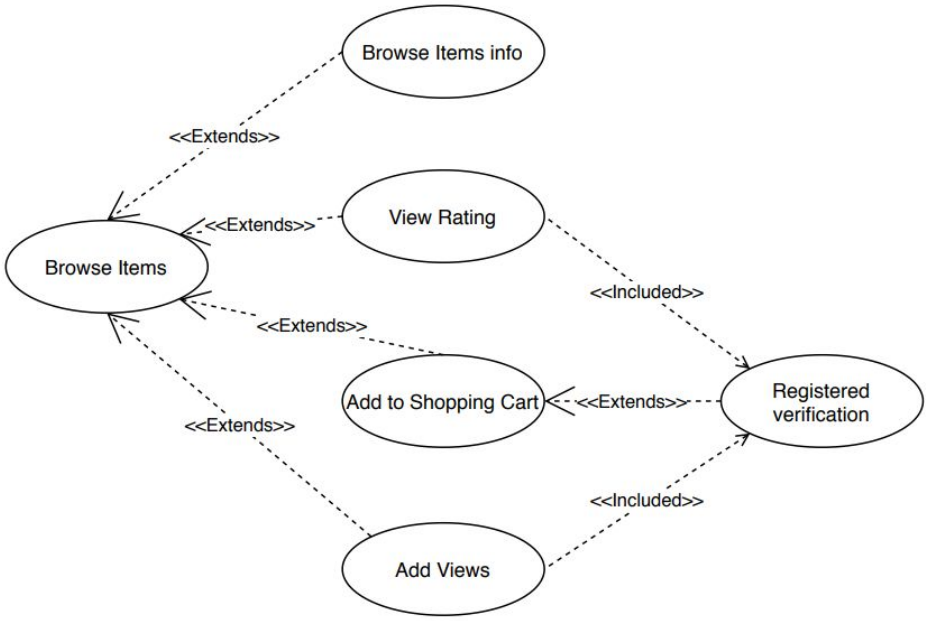
Administrator: The person who can check store reports (Top 10 Sold book & Monthly report)

Bank: The bank gets involved in the payment process.

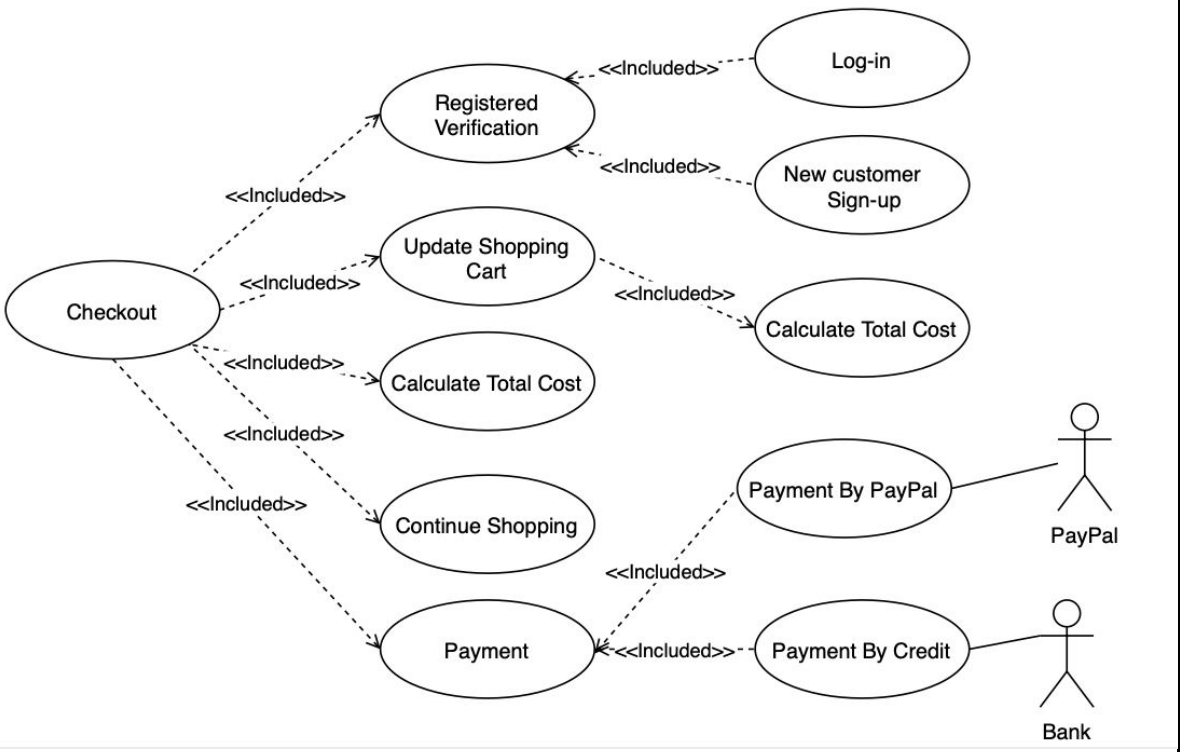
1. **Use Case:** The 3 general use cases are Browse Items, Make Purchase, Checkout and Client Register. Firstly, All the Web customer actors are able to browse products on Lotus Bookshop. Secondly, A client can make a purchase and Checkout. Thirdly, A client can create a new account.



2. **Browse Items:** Browse Items is a primary use case which is extended by several use cases, for example Web Customers may choose different catalog, view items rating, add items to shopping cart and submit book review. These Use cases are additional features which are based on Browse Items.



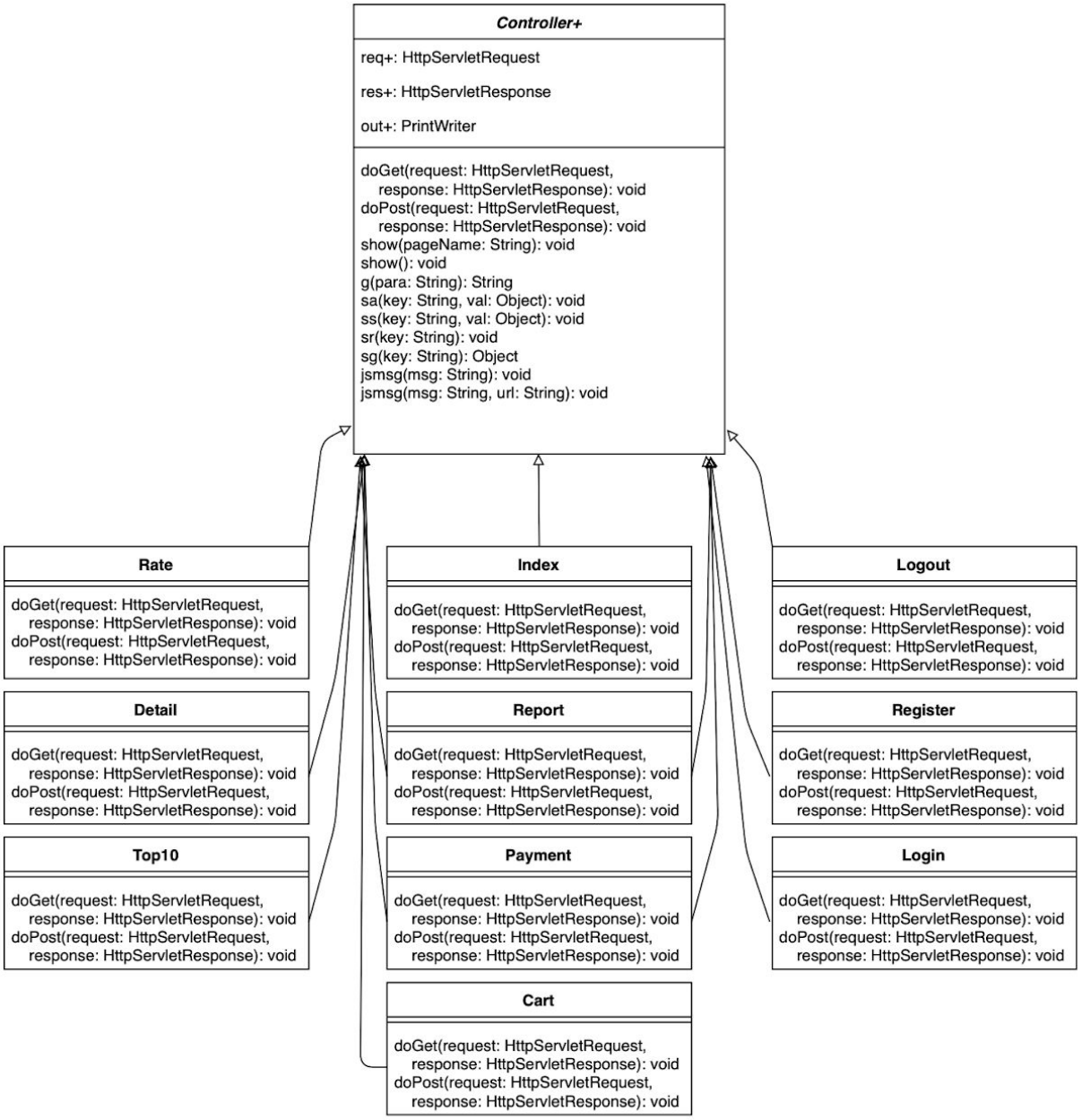
3. **Checkout:** checkout use case is extended by Registered Verification which ensures Web customers in a sign-up state when they checkout. In Registered Verification Use Case, Web customers either Log-in with an account or create a new account (Sign-up). Additionally, Checkout includes updating the shopping cart, Calculating total cost, and Continuing shopping (return to the main page).

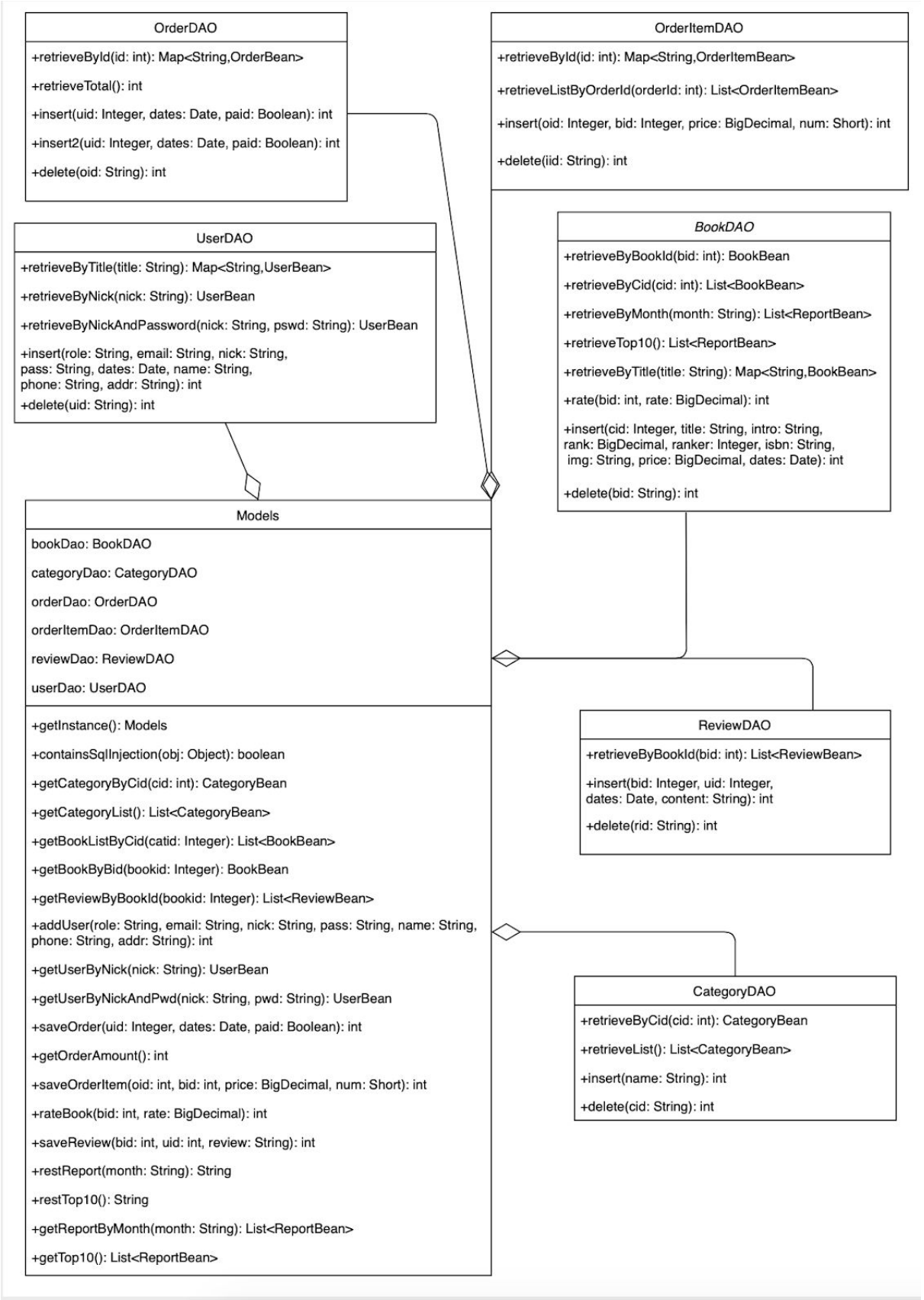


Class Diagram:

Descriptions of all the packages:

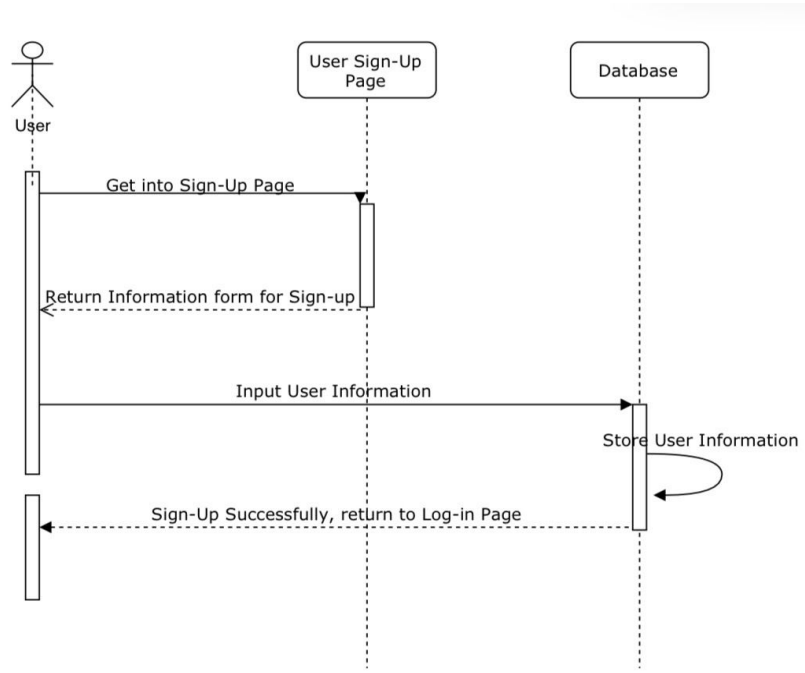
- **Bean:**
  This package contains the fundamental data structures and they are the data structure for the data object that we used in our project.
- **Ctrl:**
  This package contains the controller of the project. All the java classes in these packages are servlets, it checks with the website to goto and what function to call, they also gather necessary components and data for the web page. This is a critical part of our e-commerce system.
- **Model:**
  This package contains all the data access classes for our data object. All the classes will make sure that our object is singletons so that we don't have duplicate connections. Every database connection and operations are atomic which avoid unexpected errors from happening.
- **Rest:**
  This package contains REST methods for generating reports with curl.


Controller+
req+: HttpServletRequest
res+: HttpServletResponse
out+: PrintWriter
doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
show(pageName: String): void
show(): void
g(para: String): String
sa(key: String, val: Object): void
ss(key: String, val: Object): void
sr(key: String): void
sg(key: String): Object
jsmsg(msg: String): void
jsmsg(msg: String, url: String): void
Rate
Index
Logout
Detail
Report
Register
Top10
Payment
Login
Cart

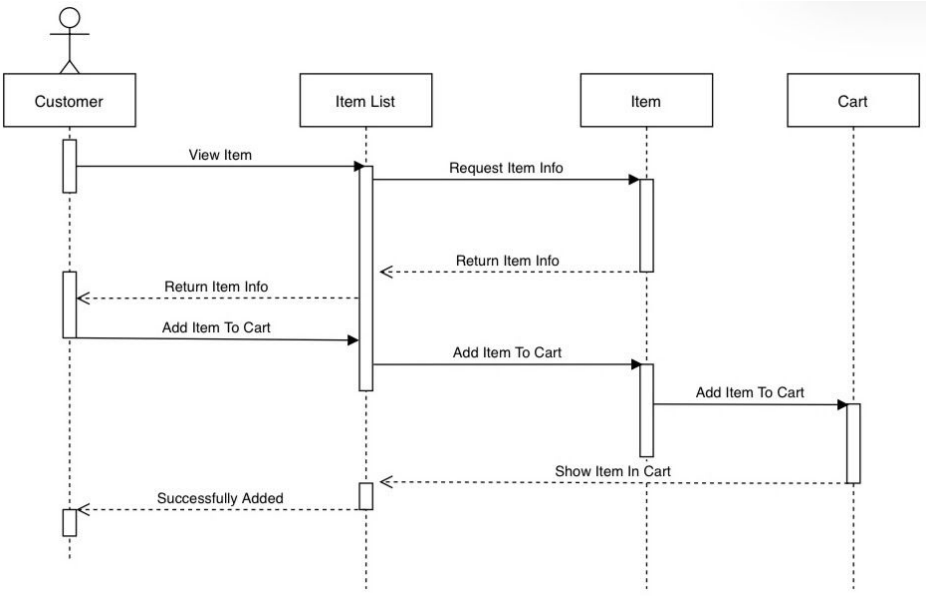

OrderDAO
+retrieveById(id: int): Map<String,OrderBean>
+retrieveTotal(): int
+insert(uid: Integer, dates: Date, paid: Boolean): int
+insert2(uid: Integer, dates: Date, paid: Boolean): int
+delete(oid: String): int
OrderItemDAO
+retrieveById(id: int): Map<String,OrderItemBean>
+retrieveListByOrderId(orderId: int): List<OrderItemBean>
+insert(oid: Integer, bid: Integer, price: BigDecimal, num: Short): int
+delete(iid: String): int
UserDAO
+retrieveByTitle(title: String): Map<String,UserBean>
+retrieveByNick(nick: String): UserBean
+retrieveByNickAndPassword(nick: String, pswd: String): UserBean
+insert(role: String, email: String, nick: String, pass: String, dates: Date, name: String, phone: String, addr: String): int
+delete(uid: String): int
BookDAO
+retrieveByBookId(bid: int): BookBean
+retrieveByCid(cid: int): List<BookBean>
+retrieveByMonth(month: String): List<ReportBean>
+retrieveTop10(): List<ReportBean>
+retrieveByTitle(title: String): Map<String,BookBean>
+rate(bid: int, rate: BigDecimal): int
+insert(cid: Integer, title: String, intro: String, rank: BigDecimal, ranker: Integer, isbn: String, img: String, price: BigDecimal, dates: Date): int
+delete(bid: String): int
Models
bookDao: BookDAO
categoryDao: CategoryDAO
orderDao: OrderDAO
orderItemDao: OrderItemDAO
reviewDao: ReviewDAO
userDao: UserDAO
+getInstance(): Models
+containsSqlInjection(obj: Object): boolean
+getCategoryByCid(cid: int): CategoryBean
+getCategoryList(): List<CategoryBean>
+getBookListByCid(catid: Integer): List<BookBean>
+getBookByBid(bookid: Integer): BookBean
+getReviewByBookId(bookid: Integer): List<ReviewBean>
+addUser(role: String, email: String, nick: String, pass: String, name: String, phone: String, addr: String): int
+getUserByNick(nick: String): UserBean
+getUserByNickAndPwd(nick: String, pwd: String): UserBean
+saveOrder(uid: Integer, dates: Date, paid: Boolean): int
+getOrderAmount(): int
+saveOrderItem(oid: int, bid: int, price: BigDecimal, num: Short): int
+rateBook(bid: int, rate: BigDecimal): int
+saveReview(bid: int, uid: int, review: String): int
+restReport(month: String): String
+restTop10(): String
+getReportByMonth(month: String): List<ReportBean>
+getTop10(): List<ReportBean>
ReviewDAO
+retrieveByBookId(bid: int): List<ReviewBean>
+insert(bid: Integer, uid: Integer, dates: Date, content: String): int
+delete(rid: String): int
CategoryDAO
+retrieveByCid(cid: int): CategoryBean
+retrieveList(): List<CategoryBean>
+insert(name: String): int
+delete(cid: String): int

Sequence Diagram

1. Use case: User Register



2. Use case: Shopping cart



Design Decision:

1. Event-Driven Design

In our project, We are using event-driven design for our Book Store. As the UML sequence diagram shows, each operation is motivated by an event from a servlet. The reason we choose this design pattern is because building a systems around an event-driven architecture simplifies horizontal scalability in distributed computing models and makes them more resilient to failure

2. Error Handling

We are handling all exceptions on the server side to avoid inconsistency. Error handling using Javascript might be blocked on the client side, which makes the program's behavior hard to predict. In this project, all exceptions are handled on the stage of the controller. Therefore, exceptions and alerts will be displayed on the page without interruptions.

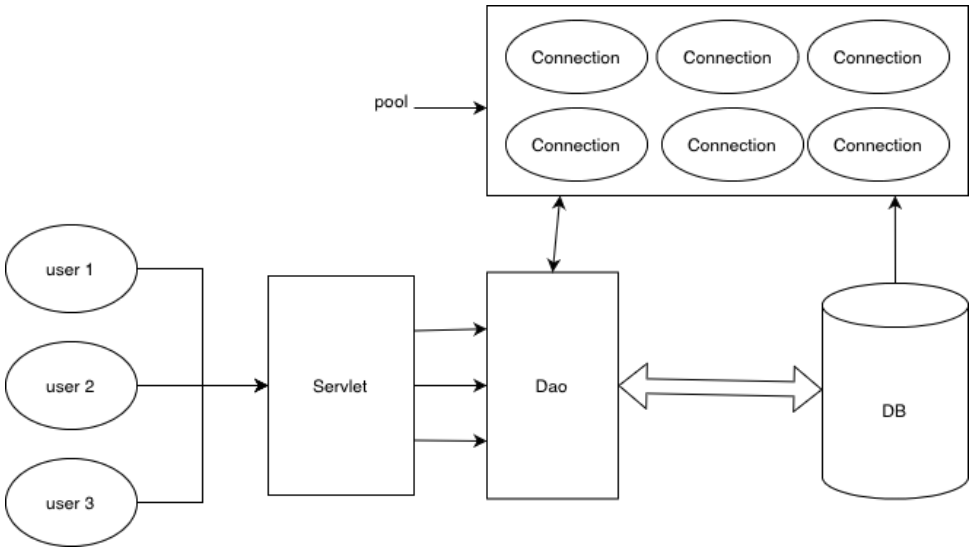
3. MVC Frameworks

We decided to use the Model-View-Controller framework to separate business logic, data and display. This design framework leads to good loose coupling since the three modules are independent so that changing one won't affect the others. If MVC is not used, data accessing and page elements would be crowded together in the controller which breaks the cohesion design principle.

Implementation Decision:

1. Data Access

When implementing database connection, we decided to use the JDBC c3p0 connection pool.



Every user request needs to obtain a connection from the database which consumes lots of resources and causes long overhead. It is important to use a connection pool for a high concurrency and real-time web application to improve performance. c3p0 pool has an extra advantage that the database connection will be closed automatically without explicitly closing which greatly saves resources.

2. Controller Hierarchy

We implemented a controller to each corresponding functionality so that different controllers are responsible for a specific URL. In order to control all the

controllers we created, there is a main controller to coordinate different controllers just like the first class diagram shows.

Limitations:

1. Visitors and customers can rate a book several times.
2. Verify:
    a. Can't verify the correctness of Email from the information user entered, can only check if the email contain @ or not
    b. Since we are not using a real payment system, therefore we cannot check the correctness of the credit card that user entered.
3. No Pictures for items
4. No User profile, Admin Panel
5. UI is not friendly enough
6. Didn't test for XSS

Contributions:

For coding, we collaborate, share ideas and work together and move side by side. All team members in this project in this team are new to the E-commerce system before this semester. Without working or previous experience on developing a huge network like this project we decided to use an inefficient but reliable method of distributing work, which is to work together as close as possible. We used Zoom to share screens and send files or code we wrote to Wechat in order to share what we completed with our teammates. We made a time schedule that we meet almost every evening to work together and code with each other's help, to develop, to solve, to debug as we are one.

Xinqi Chen:

I am the leader of this project and mainly focus on the decision maker for the direction of our project. By studying online and attending lectures, I have a reasonable understanding how and why JavaEE that we are studying is a popular commercial standard.

ChengXiang Gong:

I discussed and studied our code decisions closely with my teammates, listened to their opinions, exchanged ideas, and improved myself so as to improve the progress of our project. I code all work that the leader assigned to me, other than that, I mainly focused on helping my teammates to debug and change their format, also writing comments for some specific method for later debug.

Yang Zhou:

Together with my classmates, I completed my part of the code in each package, discussed with my classmates and completed the tasks assigned by the group leader. In addition I will find loopholes in students' opinions and help students simplify their code.