EECS4413
2020-2021 Winter Term
Design Document
April 9, 2021

Xuan Xiong	214951404
Qasim Ahmed	214809719
Yonis Abokar	215133457
Yash Dhamija	215250343

Table of Contents

1 Architecture

The architectural design for this project is based on Model-2 architecture of the Model-View-Controller (MVC). This architectural pattern efficiently decouples the business logic, model and scalable web applications. In our project design, the controller contains the logic to determine the requests coming from clients. We additionally integrated further design principles such as the Data Access Object (DAO) to decouple the business logic from the data retrieval, and the chain-of-responsibility design pattern to implement filters to enforce certain restrictions of access.

1.1 Patterns Used

1.1.1 MVC

In the model, we implemented a data structure representation of all the elements in a bookstore, such as a login form, a checkout menu, and reviews. This component of the application is where the business logic resides, and any intercepted user requests are answered.

The view component is the application's visual representation in the form of a UI and implemented in HTML (JSPX), CSS, and JavaScript. This layer is where a user is able to generate user requests to the controller component and see the results.

Lastly, we implemented the controller in the form of various servlets. Each servlet handles the requests and responses of its corresponding view. This is also where validation is handled for user requests.

1.1.2 DAO

The DAO package implements a design pattern we used to provide an interface from our underlying database to our model component. In the DAO, we were able to create multiple DAO classes that connect to a cloud-hosted MySQL database using Database Connection Class using JDBC exhibiting a singleton. The DAO allows us to extract sensitive data and present it in a form that is still useful to the model without revealing the exact queries used or the response from the queries.

1.2 Design Decisions

1.1.1 MVC

The reason why we chose to implement our project following the principles of MVC was because of maintainability, scalability, and testability representing the most important aspects of application development. Since working on all components in parallel does not affect one another, we were able to increase work efficiency.

1.1.2 DAO

The sole purpose of using the DAO pattern was to separate a data resource and the implementation to access it. This way, any resource that called for a request of data would not need to worry about any changes behind the scenes because the DAO interface provided a guaranteed behaviour in its output. Furthermore, testing any of the data retrieval methods would be easier because it would not have dependencies on other components.

1.1.3 Filter

In our application, the administrator would have access to analytics that expose customer details. A normal user clearly should not be able to also access such details. In order to solve this problem, we decided to implement a filter. By using filters, we could easily validate and change where a user request is delegated. Since we were already using the javax.servlets package, javax.servlets.Filter was easy to integrate as well. Specifically, customer/partner names are anonymized replacing their names with X symbols when displaying the administrator reports. In addition, when admin logins, they provide the request *qasimahmed.me/BookLand/AdministratorLogin* which uses a filter mechanism for all other types of users beside admin sending the request to the Admin servlet.

1.3 Trade-offs

Although maintainability, scalability, and testability are clear advantages of using MVC, DAO, and filters, they also create multiple layers of abstraction. As a result, performance will ultimately suffer. As requests are received, it would have to pass through additional layers of processing or validation compared to if no design patterns were used and all the code were mixed into one file. However, this disadvantage is heavily outweighed by the increased efficiency in development.

1.4 Diagrams

1.4.1 Use Case Diagrams

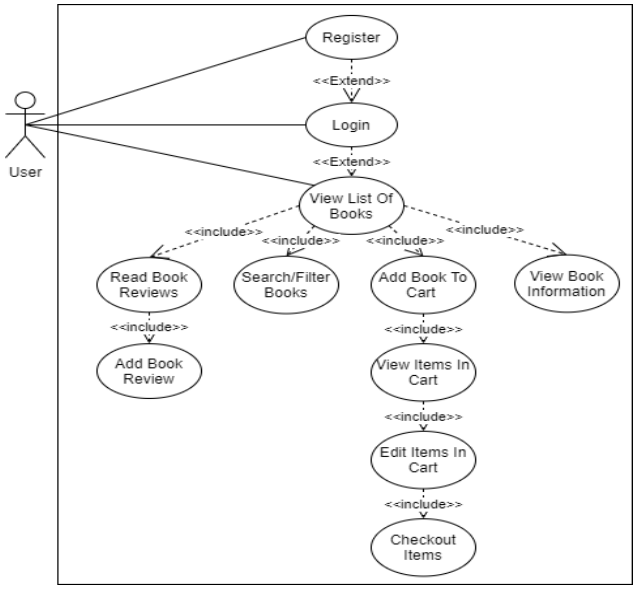

Figure 1: Use case diagram for a user

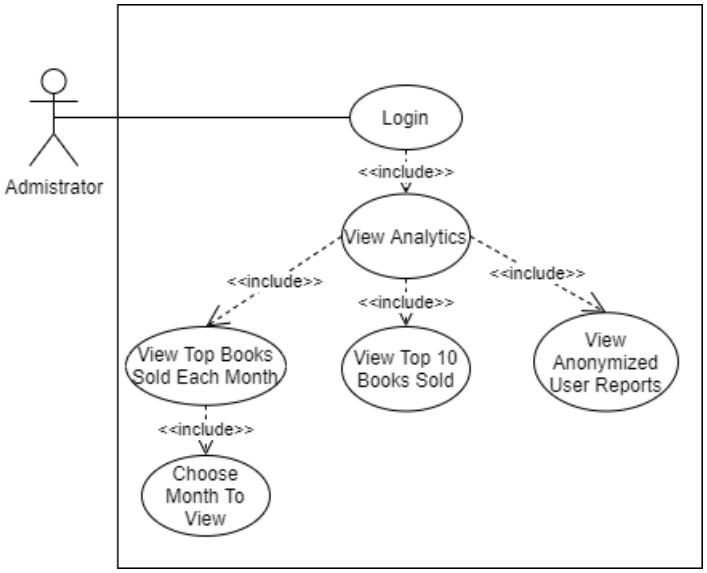

Figure 2: Use case diagram for an administrator

1.4.2 Class Diagrams

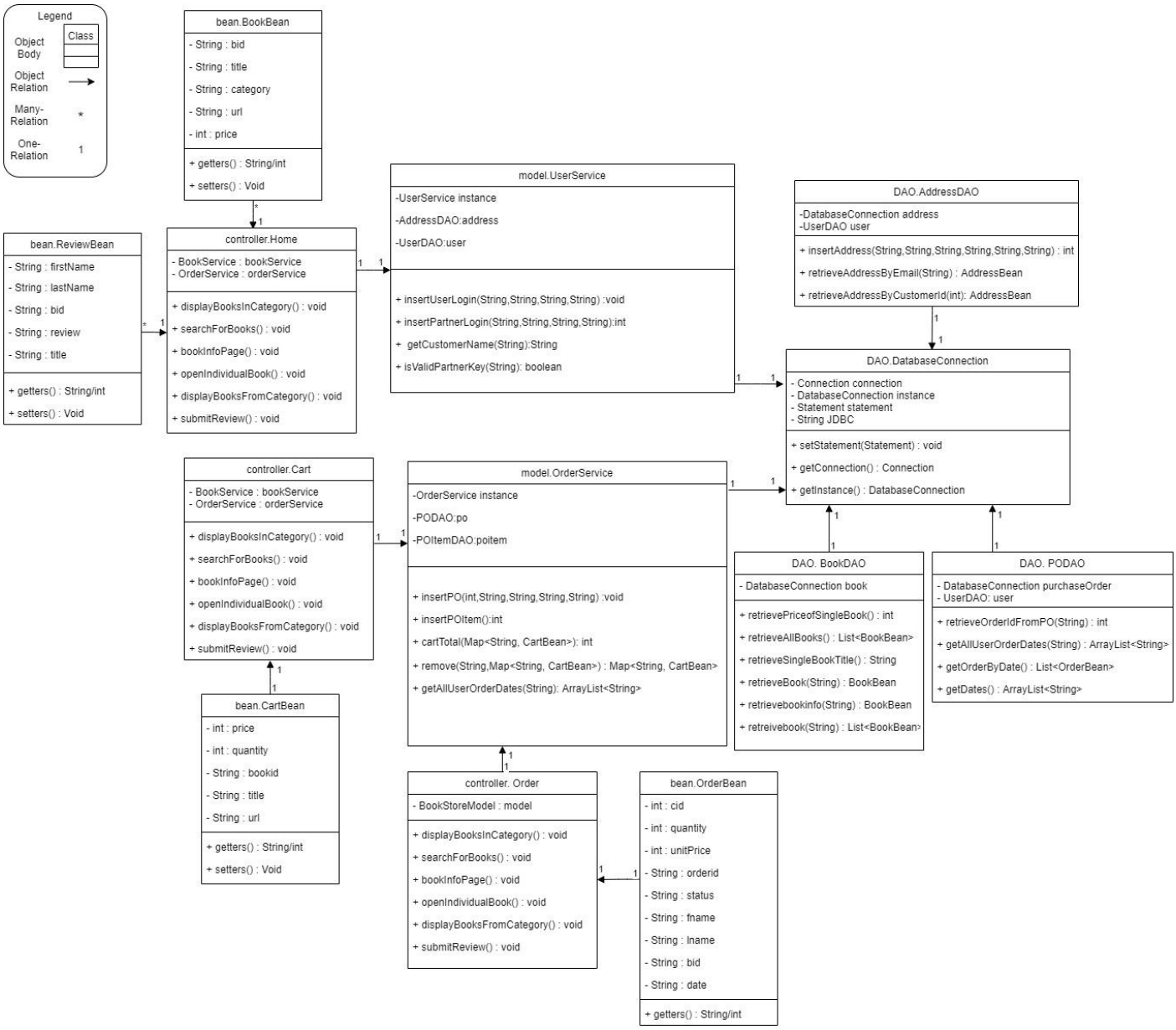

Figure 3: Class diagram of the bookstore

1.4.3 Sequence Diagrams

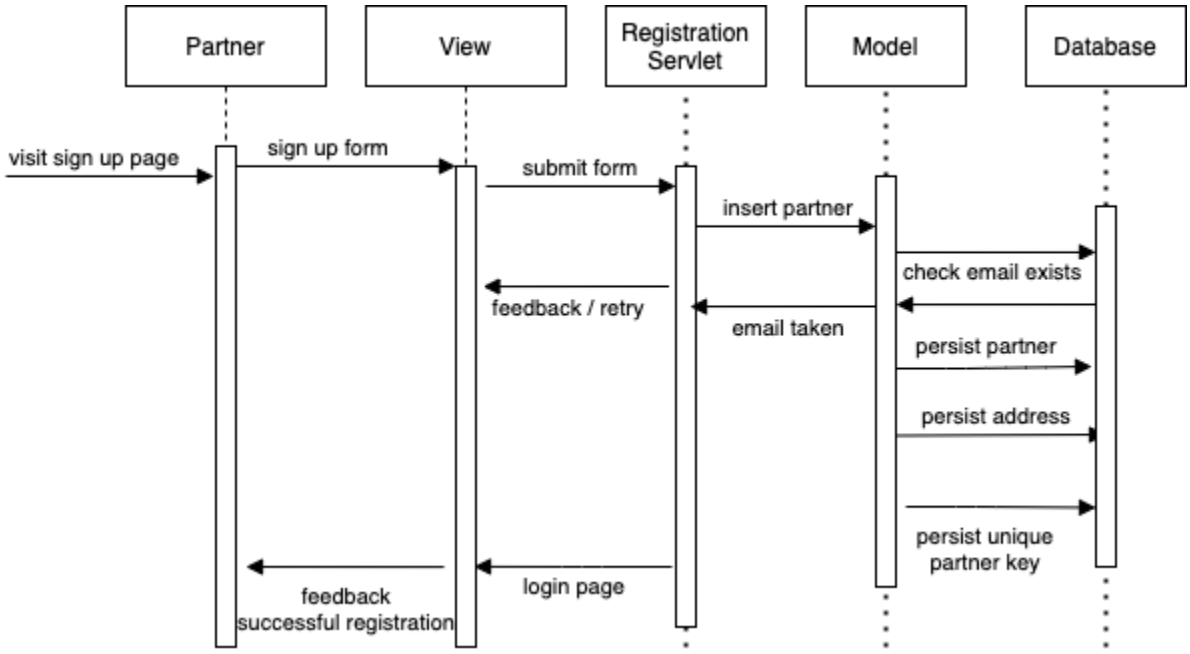

Figure 4: Registering as a partner

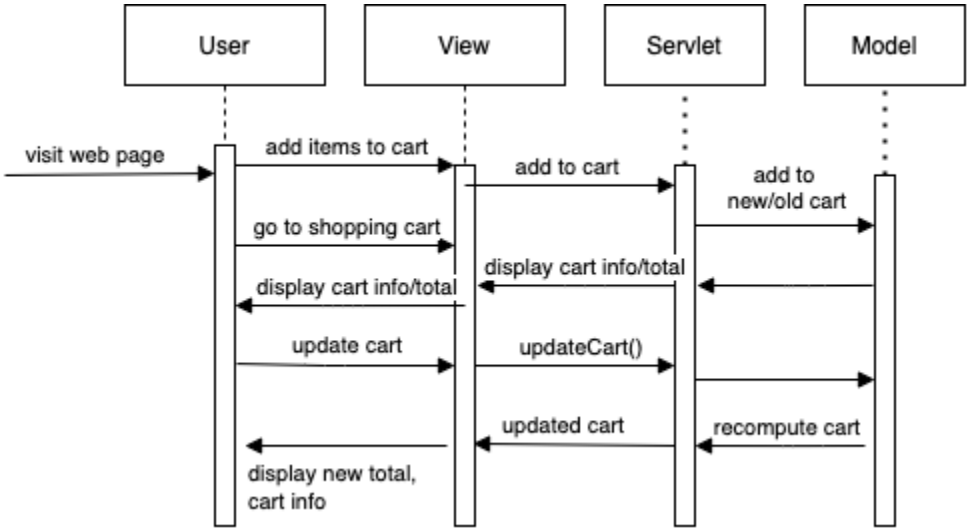

Figure 5: Adding Items to/updating cart

2 Implementation

2.1 Implementation Decisions

2.1.1 Front End

To implement the view component of our application, we utilized Java's web framework, jspx. Since our server-side implementation of the controller used Java web servlets, jspx showed clear advantages. Aside from plain HTML, jspx allows us to use expression language (EL) in order for the client-side pages to access attributes and variables set on the server-side. As a result, we were able to dynamically update the web pages more efficiently with this form of communication between the view and the controller. Standard CSS was used to style the HTML and JavaScript was used for certain client-side functionalities such as validation.

2.1.2 Back End

Adhering to MVC principles, we separated our backend into the model and controller components, with each of these using other subcomponents like the DAO and bean object.

In the controller, we created numerous servlets that handle various use cases like registration, login, and payments as examples. In each servlet, they share one instance of the model, called the BookStoreModel, and handles all the business logic such as distinguishing between administrators, partners, and normal users.

In our model implementation, we utilized the DAO interface and bean objects to set the attributes associated with our bookstore. Specifically, the model calls the DAO methods to retrieve data from the database and stores them as a bean object to return to the controller.

The DAO classes are implemented such that they connect to our cloud-hosted database using the JDBC driver. When executing queries, they use Java's PreparedStatement functionality as a security measure against SQL injection attacks.

Our REST service is only available to partners. In order for partners to successfully request the REST service, they are given a generated API key associated with their account. By passing in their unique key, they can be authenticated as a valid partner and can access the REST endpoint.

2.1.2 Deployment

The war file was produced in eclipse by using maven build in eclipse. Docker and EC2 instance was used to deploy this project on the cloud. The advantages of Docker deployment is that deployment was fast and easy, its ability to run anywhere and its flexibility that war files can be easily deployed again and again. Domain name qasimahmed.me was linked to the IP address generated by EC2 and we used cloudflare

to make the site secure and add in extra protection against attacks. One other benefit of cloudflare is that it provides SSL.

2.2 Trade-offs

As previously stated, the DAO uses prepared statements to execute queries as a preventative measure against SQL injection attacks. The implementation of this functionality ultimately leads to performance penalties. Like all major security threats, SQL injection attacks create much larger issues in the bigger picture and having prepared statements outweighs the slight performance penalties it comes with. In addition, in the early stages of the project we were considering Hibernate Framework so the base classes of the all beans can be easily stored and retrieved. Yet, the group concluded that JDBC performance testing was better than Hibernate, choosing performance over OOP.

2.3 Limitations and Testing

Some of the classes in the DAO were tested for accurate query retrieval/insertion and detecting the security feature of the prepared statement with SQL injections . Since there was not enough time to do the entire testing using junit, we tried testing the bookstore using localhost each time we added a new feature/functionality. Lots of bugs/issues were fixed during the testing phase and code was fixed and updated code was pushed to git.

2.3.1 Security Vulnerability

For security purposes we tried doing SQL injection on various forms and query parameters to essentially get into the database. We also tried doing HTML injection but that doesn't have any effect on the database. We also tried doing Cross site scripting(XSS) and we were not successful in getting into the database and destroying it which was good. We made sure we used prepared statements for user inputs which prevents SQL injection and sanitizes the input.

2.3.2 SQL injection test

We specially tried SQL injection on the search bar in the bookstore and after changing to prepared statements the inputs were sanitized and the database was safe. SQL injection was also tried on others places e.g when we open a book by clicking on title of book and when we tried an injection on query parameter bookinfo, it gave us a 404 page which is good in this case and is expected because the book ISBN doesn’t exist in the database.

2.3.3 What Was Not Tested

Detailed testing was done for each component of the bookstore. We also tried going to URL’s which didn’t exist and thus we handled those special cases by either redirecting to 404 page or exception page.

The model component was not tested as its implementation was based on the DAO. Since we thoroughly tested the DAO, we indirectly tested the model as well. Testing the model again would just create unnecessary overhead work.

3 Performance Testing Report

In order to test the performance of our application, we used Jmeter to set up the testing environment and to extract data. To simulate traffic to our website, threads were created that acted as users who made REST calls to our API endpoint. We specifically analyzed two metrics: the response time and the throughput. One assumption we made when testing the server load was that each user had 3 seconds of “think time”, meaning each user (or thread) would send a request every 3 seconds. The arrival rate would then be equal to the total number of users divided by the think time. For example, if we tested with 102 threads, the arrival rate would be 34 req/s. To ensure the robustness of our application, we tested for a wide range of users, steadily increasing the total server load after each iteration of testing. Using this method, we were able to test from 1 user to 400 concurrent users. The upper boundary of our test case was chosen such that the server CPU utilization reached 60%.

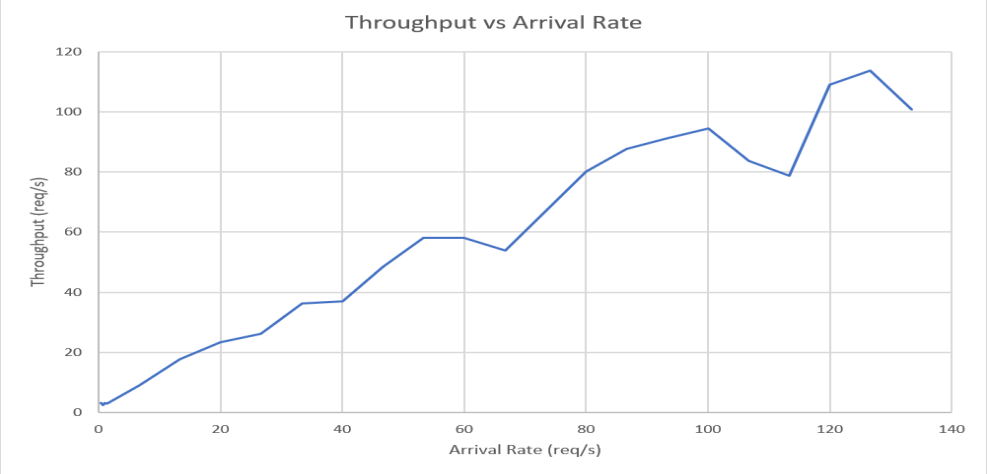

Figure 6: Graph of throughput vs arrival rate

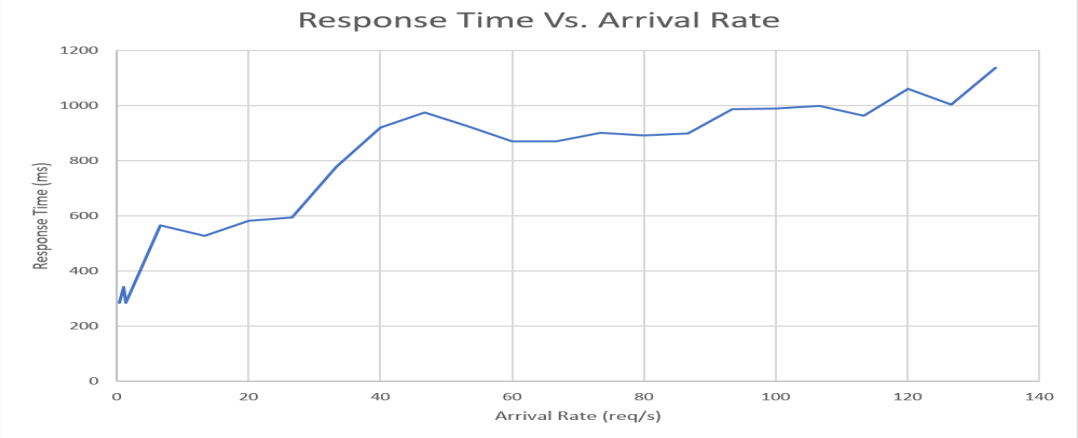

Figure 7: Graph of response time vs arrival rate

From these results, it can be concluded that throughput and response time both increase as the arrival rate increases. However, arrival rate seems to have a greater effect on throughput compared to response time, which has a slower increase in the domain of [40,133].

4 Team Member Contributions

This project was an amazing experience for all of us during this semester. As a group, to stay connected with each other, we implemented an Agile software development technique of pair programming providing feedback to each other thus becoming better developers.

Qasim

Contributions

Worked as the developer for the book store, implemented the main home page of bookstore, search feature, catalogues, Cart page, payment page, setting up database and DAO, reviews and ratings, Login Page, Register Pages,also implemented extra features such as My orders for visitors and partners and a Manage Reviews tab for Admins. Worked on Limitations and Testing part of report, UML use case diagram and also tested the bookstore thoroughly including SQL injection and fixed errors/bugs. Also deployed the bookstore war file using Docker and AWS EC2.

Learned

Learned how to deploy war files using docker and AWS EC2. Also learned how maven based projects work and how to add dependencies and configure java build path according to the server requirements.

Yash

Contributions

I worked on the order processing and product services, and payment component. For rest calls api for partners, I created a unique access key upon partner registration which is used to authenticate partners (passed in url) and fulfill their request. I also worked on servlets; home, login, book information; maintaining the session information flow and refactoring servlets and DAOs, and on front-end jspx navigation bar, and cross-scripting testing, and design; use case and sequence diagrams.

Learned

This project helped me understand how different components interact in an e-commerce application and REST web service, using MVC architecture to write modular code. I learned the need and technique of testing for xss injections from my team mate Yonis.

Xuan

Contributions

I was able to implement the analytics page for administrators. To accomplish this, I first had to modify the database to include purchase order items and the purchase order itself. I then created data retrieval methods within the DAO that adhered to the three use cases as specified in the project specifications and connected it to the model component in order for it to be used in the administrator web servlet. Within this servlet, I created additional methods to authenticate a user as the administrator when they try to access the analytics page by using a filter. Finally, I created the analytics web page to display the results.

Learned

I learned about how the various components in a typical web application interact with one another without being directly connected. One example would be how the servlets are able delegate certain tasks to others and are also able to forward requests and responses as well.

Yonis

Contributions

During this project, I contributed to refactoring code in the controllers and DAO maintaining modularity. In addition, I wrote some test classes for the DAO. I assisted in creating the payment structure of our code and implemented the logic for the several DAO classes connecting them to the model. Also, I assisted with creating login and register servlets.

Learned

I learned how to collaborate with individuals and work on github to effectively merge code. As well, I learned the entire layout of building a web application, especially the back-end component.

Qasim Ahmed Henry Xiong  Yonis Abokar