4413 ShoeStore Project Report

## **Team** C**omposition**

The description of our team is as follows:

* Team name: ShoeStore
* Team member 1’ s name and student id (in parenthesis): Steven Haddad (217181454)
* Team member 2’ s name and student id (in parenthesis): Eric Ngo (217091190)
* Team member 3’ s name and student id (in parenthesis): Neven Huynh (217091091)
* Team member 4’ s name and student id (in parenthesis): Matthew Wong

## **Individual Contributions**

Team member 1 (Steven Haddad): Created the SHOESTOCK table in the SQLite database, and randomized the stock from 1 to 5 for all shoes. Created tables for USERS and PURCHASES in the database. Manually collected Nike branded shoe images. Created Purchase JavaBean and implemented CartDAO to checkout and store purchases in the database. Implemented all of the admin associated jsp, controller, and DAO files. Modified the header.jsp to conditionally show admins a link to admin view and customer view. Modified the home.jsp in the customer home page to pull the shoes from the database. Performed some general testing of the entire web app.

I learned about the contributions of other members firstly through communication on discord about what they are going to implement. Then, once they committed their changes to the github I tested their implementations and looked at the newly committed source code to familiarize myself with how it worked. That made me capable of bug fixing other people’s code if need be.

Team member 2 (Eric Ngo): Collected Under Armour branded shoe images. Saved all the images of the shoes and named them so they could be used in the program. Created and tested code for many controllers and DAOs as well as the jsps. Controllers: implemented CartController, implemented LoginController, implemented ProfileController and edited/implemented ShoeController. DAOs: edited CartDAO, implemented LoginDAO, edited ShoeDAO, edited PurchaseDAO. Jsps: implemented and edited profile, register, cart, checkout, header, home, list and login. Did some testing of the web application as well as testing to make sure the Docker pull was working correctly.

I learned of the contributions that the others have made via pulling their work from github as well as communicating to them using discord. When significant enough changes were made by each person they would commit their changes to github to allow the rest of us to pull their changes and test for any kind of issues. We also split the work in a way that we could essentially work on different controllers and DAOs and then combine all the features together to one web application.

Team member 3 (Neven Huynh): Created SHOETYPES table in the SQLite database and inputted the values of each shoe as rows. Manually collected Puma branded shoe images. For the Model, created Shoe and Cart JavaBean. In the Controller, implemented methods in the ShoeController and ProfileController. Implemented methods in ShoeDAO and implemented regular expressions for verifying correct credit card format as well as restricting special characters in usernames for SQL injection attack mitigation. For the view, implemented shoe jsp and modified profile jsp, register jsp. Performed general testing of the entire web application and bug fixing. Deployed web application on Docker.

I learned about the contributions of other group members by communicating with one another on Discord as well as reading the committed code on GitHub. I learned elements of the project by testing the newly added features when pulling the features from GitHub. One example of something that I learned from my group member was to pass the servlet context in a servlet to a DAO class as a parameter. In the DAO class, it was possible to create a constructor to set the passed context and use that to get the path of the SQLite database file.

Team member 4 (Matthew Wong): Did not contribute

## **Signatures of Affirmation**

I, Steven Haddad, attest to the individual contributions reflecting the reality of the project contributions.



I, Eric Ngo, attest to the individual contributions reflecting the reality of the project contributions.



I, Neven Huynh, attest to the individual contributions reflecting the reality of the project contributions.



## 

## Table of contents

[Team Composition 1](#)

[Individual Contributions 1](#_qbafgauqm1ws)

[Signatures of Affirmation 2](#_q9sjl1vpodw4)

[**1. Introduction 4**](#_jcwdf1ccfo7q)

[**2. Architecture description 4**](#_nth458gowgld)

[2.1 Architecture Diagram 5](#_wxyltotbj1fz)

[2.2 Use Case Diagram 6](#_9e6hwjvioye)

[**3. Design description 7**](#_q428n6t2ln40)

[3.1 UML diagram 8](#_5qk432ie3uqo)

[3.2 Database schema diagram 8](#_bze6aqo8u2qu)

[**4. Advanced and distinguished Features (Beyond the requirements) 8**](#_twrzldki5rcv)

[**5. Implementation 9**](#_w41ktswxp3e)

[**6. Deployment efforts 10**](#_54xyj02k2nth)

[**7. Conclusion 11**](#_5jsqylst438b)

# Introduction

We decided to make a shoe store for the team project. In our web application users will be able to view multiple different shoes using the different methods that we have implemented. Shoes are viewable on the home screen or as a list with their details such as brand, name, colour and price. The users can also search for shoes based off of the brand of the shoe as well as the colour and model name. Once a user is able to find the shoe or shoes that they want, they are able to add it to their cart, in their cart they are able to remove any shoes that they have added or check out. However for the user to be able to check out they must have an account which they can choose to register for or login at any time while they browse the web application. The user must also have entered at some point an address and credit card to be able to check out. Logged in users will have access to a feature that allows them to access their profile that lets them change their account details such as their username, password, address or credit card. On this page they will also be able to access their purchase history to view the past purchases they have made, that page will show what they purchased, the card and address they used as well as the date that they have made that purchase. Admin users are also implemented into the web application, they are able to perform and view extra actions that normal users cannot access. Some of these features are to view all users in the database along with their information, change the stock and price of shoes, and view all of the purchases in the database.

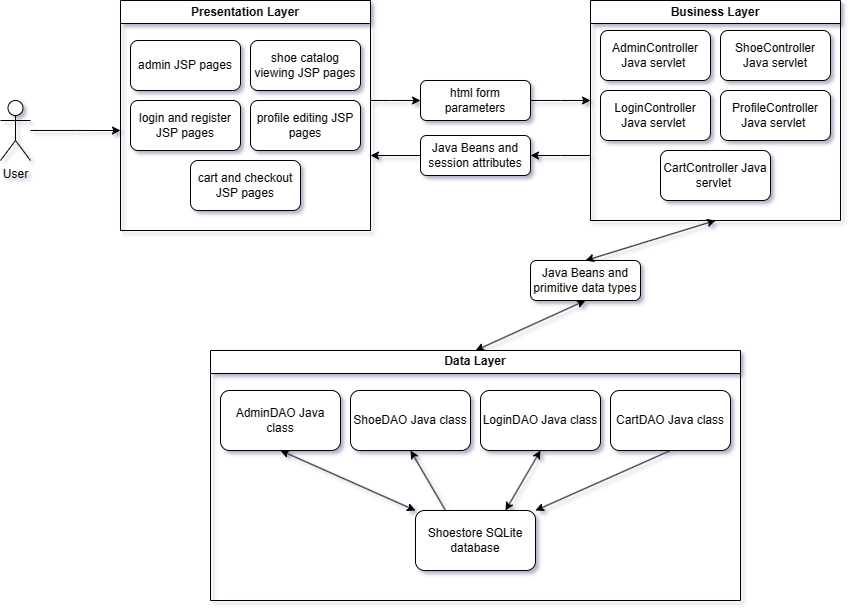
We used MVC, DAO and observer design patterns to create our project. As for the technologies most if not all of the group members used eclipse and a tomcat server to create and test the web application, for our database we used SQLite. We also used Docker to deploy the project. The MVC was used to separate the display that the users see and the logic code. The DAO allowed us to interact with our SQLite database. The observer design pattern allows us to set up observers to see what action the user takes to then return a certain result.

A weakness of our project is that our web application does not have a way to sort the many different shoes by their price or alphabetically in terms of model name. Our web application also does not inform the user or disable the add to cart button when they select a shoe that has 0 stock left, it simply lets the user add the item to the cart but the item will not show up. Also we did not implement any kind of encryption for the user’s information, so that would be another weakness of the project. Our strengths in this is that we have regular expressions for the input of credit cards to ensure that they are in the right format as well as usernames to prevent users from creating usernames with special characters. We also use PreparedStatements for the database interactions. Our code for the most part is somewhat easy to understand although it can be a bit messy due to the mass amount of buttons that are accessible on some pages of the web application.

# Architecture description

We used a 3 tier-architecture pattern to create a clear separation between the **Presentation layer**, **Business layer** and **Data layer** for our web app. Every single jsp file in our web app is tasked solely with presenting the information to the customers and admins, our web app displays the dynamic information, like shoes, shoestock, purchases, and logging in, through session attributes or Java Beans that were created in the Business layer. There is no business logic being performed in the jsp, the jsp just passes the input information to the controller servlets in the Business layer through html forms and the Business layer handles setting session attributes or setting Java Bean attributes. The DAO classes (Data layer) are used as an interface to the SQLite database, to access the Data layer. Each DAO class is specifically designed to include methods used by only one specific controller class, and the DAO classes are the only ones that access the database directly through JDBC queries. The resulting information from the SQLite queries or statements are passed back to the controller through JavaBeans or through primitive data types, and ResultSet objects are not directly returned

## 2.1 Architecture Diagram

****

## 2.2 Use Case Diagram

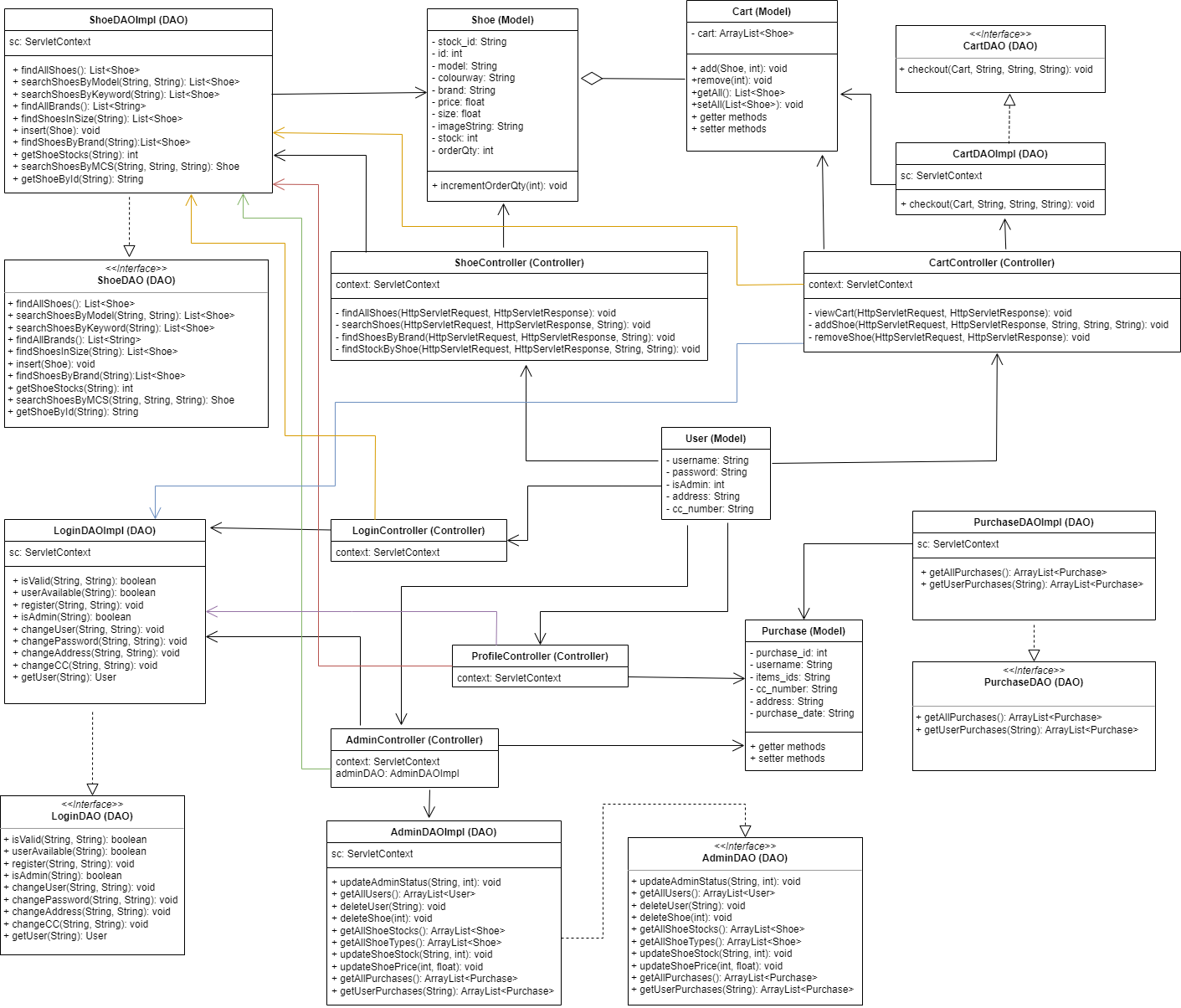
# Design description

We used a few design patterns which include MVC, DAO and observer. The design patterns we chose all merged together to create what we have for our web application. For the interaction between the user and the backend we chose to use MVC so that we could separate the different parts of code. For example, we had all of the code that was used to display the web application in one section, and that code did not include any kind of logic code. We also had all of the code that would detect whether a certain action was done in another area of the project, this would be considered MVC and observer. For our database interaction, we chose to use DAO to completely split the code that handles the database interactions from all of the other code to prevent the frontend from seeing how we interact with the database.

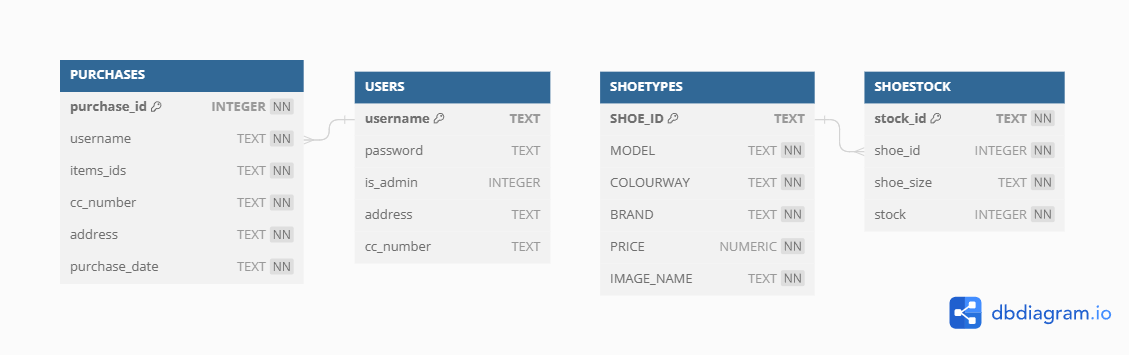
At first within our home screen we first chose to manually input each of the images along with the references so that when an image of a shoe was clicked it would bring the user to the right page, but after creating the rest of the web application we realized that it would be possible and much easier to use a for loop to automatically create the list of shoe images. We didn’t really think of any design patterns whilst making the web application, we mostly just kept implementing the features in the ways we knew it would work. The same goes with any main design decisions, we never really compared different designs of the web application and just went with the first one we had. We also don’t have any trade-offs due to choosing and using the first thing that had worked for us.

For the UML we have decided to not include the setters and getters for the model classes indicated by the “(Model)” beside the class name, as well as the init, doGet and doPost methods for the controller classes labeled “(Controller)”, as well as the getConnection and closeConnection methods for the DAO classes labeled “(DAO)”. This is due to space constraints and the fact that if we were to add all of these methods for every single class that we have, all of the text within the image would be too small to be able to read. The colourful arrows are just a way to distinguish between the lines that have to overlap.

## 3.1 UML diagram



## 3.2 Database schema diagram

****

# Advanced and distinguished Features (Beyond the requirements)

**Design Patterns**:

* Observer design pattern in Controller servlets to call functions depending on the action of the user

**Customer**:

* Update user information (Address, Credit Card number)
* Change username and password
* Have the option of using a temporary credit card and address without having to save the information on their account
* View inventory information of individual shoes (quantity remaining)
* See account page with customer information and purchase history
* Search shoes by keyword (brand, model, colour)

**Administrator**:

* Update stock of shoes and delete shoes from the store
* Update users of their admin status (can create new admins as well as remove admins)
* Search for specific users in sales history

**Other features**:

* SQL injection mitigation with prepared statement calls to the database instead of regular statements
* Regular expressions to restrict usernames from containing special characters to further mitigate SQL injection as well as simplify username search in AdminView (find purchases by username)
* Regular expressions to ensure valid credit card numbers
* Users able to see the stock of shoes for the corresponding size of shoe they select

# Implementation

For the front-end implementation, our group decided to use HTML, CSS, and JavaScript. Due to not every group member being familiar with other front-end technologies such as React or Angular, we decided that it was best to use what was covered in class so that every group member was able to understand and contribute to the project. As such, our group also used Servlets and JSPs for the back-end instead of other technologies such as Springboots or Node.js to encourage contribution from all members. Although the usage of other technologies may have improved the appearance of the web application, the trade-offs to using other technologies was that it would have taken more time to learn the technologies. Thus, it would have taken more time to complete the project.

The database systems our group explored consisted of SQLite and MySQL with SQLite being chosen for the project. We decided to use SQLite as it would simplify the database access for all members of the group. Instead of each member having to create and access a MySQL database, it was more straightforward to include a SQLite database file within the project. Not only does this allow for better accessibility among the group members, but changes within the database can be reflected on the repository. Instead of having to manually update the database to reflect new entries or changes, pulling the project from the repository does this all automatically. Additionally, it was easier to deploy the web application on Docker without having to worry about using a MySQL database. A SQLite database allowed the application to be run on the server right away. One trade-off of using SQLite as opposed to MySQL is that it is better suited for single user access to the database. Although it is suited to an application on the scale of our project, a real shoe-store application with real customers should not use SQLite.

# Deployment efforts

Our group was successful in deploying the project and to do that, we used Docker. We compared Docker with the other option of AWS or Amazon Web Services and we found that using Docker was easier. In order to deploy the project, we followed a tutorial which first instructed us to download Docker Desktop. After registering an account, we began to build our docker image. We created a DockerFile with the following text:

FROM tomcat:9.0-jdk21-openjdk

COPY ./*name-of-project.war* /usr/local/tomcat/webapps

In our case, we named our project 4413\_Project.war. We built the image with the following command in the command line:

docker build -t web-service .

And checked the image with:

docker image ls

Finally, we ran the container using the following command:

docker run -d -p 8080:8080 web-service

Within Docker desktop, we copied the Id of the image and logged in to Docker hub. We created a public repository with the name *web-service* and used the following command in the command line:

docker tag *paste-Id-here* user-name/repo-name:1

with username ‘nvn64’ and repo-name ‘web-service’ used in the command of our project. The pull command can be found within Docker hub and with that, other users can use the command to pull the docker image of our project. In order to run our group’s project with Docker, the following commands are to be used to run the docker image locally in a container:

docker pull nvn64/web-service:1

docker build -t web-service .

docker image ls

docker run -d -p 8080:8080 nvn64/web-service:1

Then finally, we were able to visit [http://localhost:8080/4413\_Project/](http://localhost:8080/Project-name/) to open the web application in our browser.

# Conclusion

To conclude this report, our team was able to successfully utilize a 3 tier architecture, incorporating MVC, DAO, and observer design patterns to create a functional and maintainable web application. The team used technologies like HTML through JSP, CSS, and JavaScript, for the front end ; while Servlets, JDBC and SQLite, were used for the backend. We were able to successfully deploy our Tomcat 9 server on to Docker and our application is stored in a Docker repository ready to be pulled and run whenever. The project's strengths lie in its clear separation of concerns through the use of design patterns and clear designation between each layer of the architecture. We also implemented additional features that we considered necessary for an ecommerce shoe store, being able to choose different size shoes and having different stocks for those different sizes.

To speak about the development process with our team, our strengths were in our effective and constant communication between team members and our ability to lay out a comprehensive plan early on in the project development. This allowed us to outline the different views that we would need for the front end and what information the tables of the database would need to include. We were all consistently on the same page and it was easy to refer back to the outline when we were confused.

However, weaknesses in the project include the absence of encryption for user information, the lack of sorting options for shoes on the web application, as well as the inability to modify all of the user information in the admin pages. One of the challenges faced included starting late in the project timeline, leading to a concentrated effort in the final days. From this we learned the necessity of completing a larger portion of the project ahead of time was important, because there is always an unforeseen amount of work that well-intentioned planning could not account for. Another problem we had was there was not a clear designation of who was working the front-end and who was working the back-end at one point because we were only left with 3 people actually contributing to the project, as opposed to the 4 members we had, which made the plan flawed and we were not diligent enough to correct it. This led to some initial miscommunication and code duplication that could have been avoided had we made a new plan right away.

Despite the challenges that we face, the outcome of our project demonstrates the advantage of collaborative work. Each team member decided what they wanted to work on based on their knowledge, strengths and interests regarding the technologies used. Our team was able to divide the more menial tasks, like manual data collection, into a shared responsibility so that it would be easier to complete, as opposed to one person doing it on their own. Working in a group with people who implemented different parts of the web application made it easier to cross-scrutinize and test each part of the website as it was being deployed and led to less bugs overall than an individual project. However, we did have some issues with the Eclipse IDE's compatibility in a collaborative setting with all the different cache files being stored in the project, if given the chance to use a better IDE more suited for team collaboration we would choose to develop our future projects in an IDE like Visual Studio.

Overall, the ShoeStore project showcases the team's ability to work collaboratively, implement complex features, and address challenges in the development process. We are satisfied with the web application that we designed.