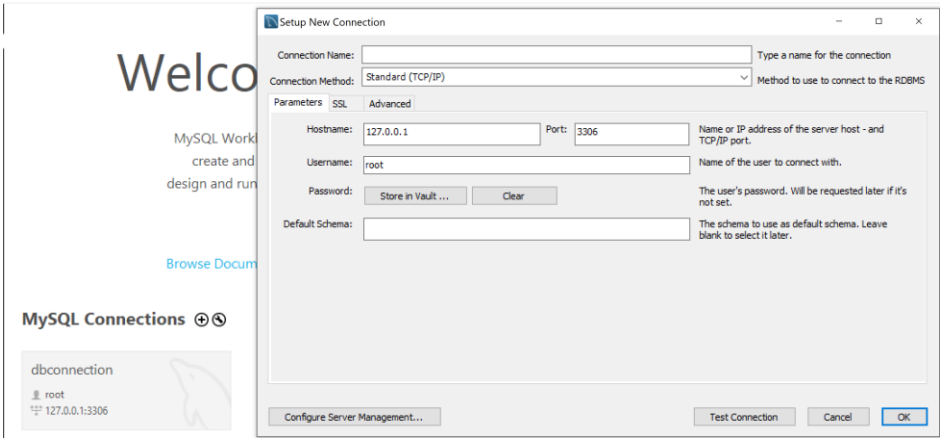
EECS 3311 Final Project Report

How to run the project?

Source Code: Download the zip file. Import it in Eclipse.

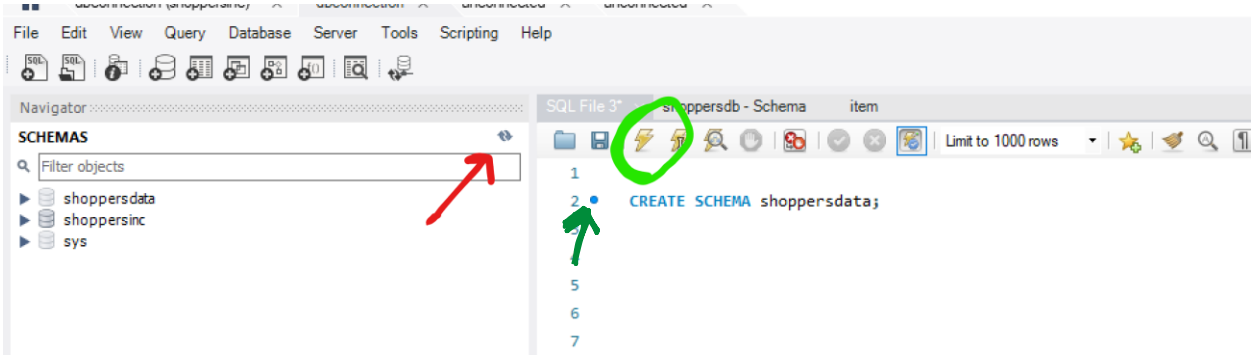
Database: Once the source code has been extracted, now we need to set up the database. I used MySQL Relational database in this project.

Using MySQL Workbench 8.0 CE., create a new connection:



This username and password will be needed later.

Now click on the connection and the following page will open. Click on "Create new schema in the connected server". Alternatively, you can also run the script **CREATE SCHEMA shoppersinc;** and click the thunder button.

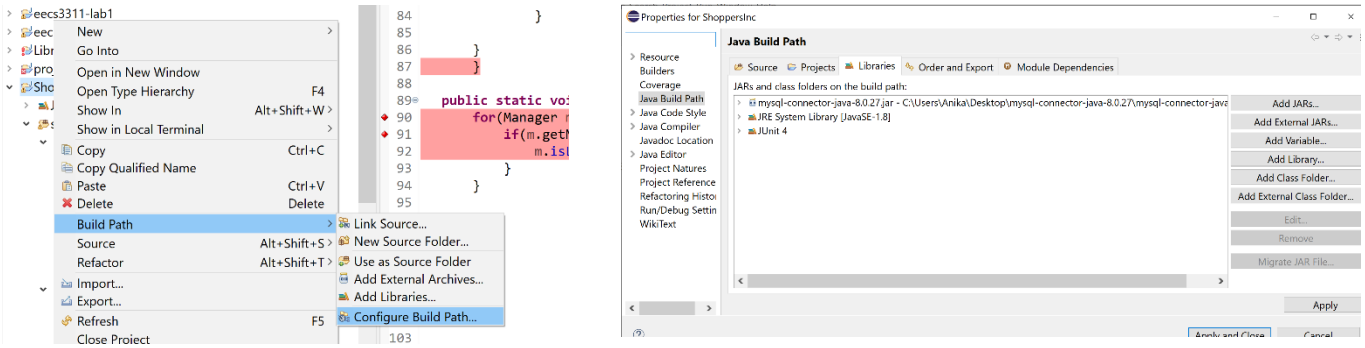


After you click the refresh symbol on the left-hand side, the schema should show up. Then double click the schema to select it. Selected schema appears in bold.

Then run the submitted SQL Script.

Source Code and Database Connection: The database has been created and the data has been entered. Now its time to set up a connection to the database from the source code. To do this, go to the DBConnection.java class under database package. The highlighted parameters have to be modified according to your environment.

1. Add the mysql connector jar file to the project. Right click on the project > Build Path > Configure Build Path > Add External jar. Add the mysql-connector-jar.



2. The first parameter will be changed to [port number which is usually 3306]/[your schema name]

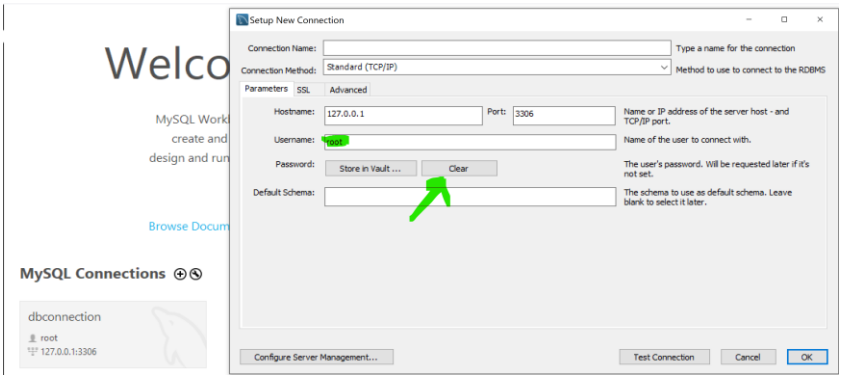
```
package database;

import java.sql.Connection;

public class DBConnection {
    public static Connection getConnection(){
        Connection con=null;
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            con=DriverManager.getConnection("jdbc:mysql://localhost:3306/shoppersinc", "root", "password");
        }catch(Exception e){System.out.println(e);}
        return con;
    }

}
```

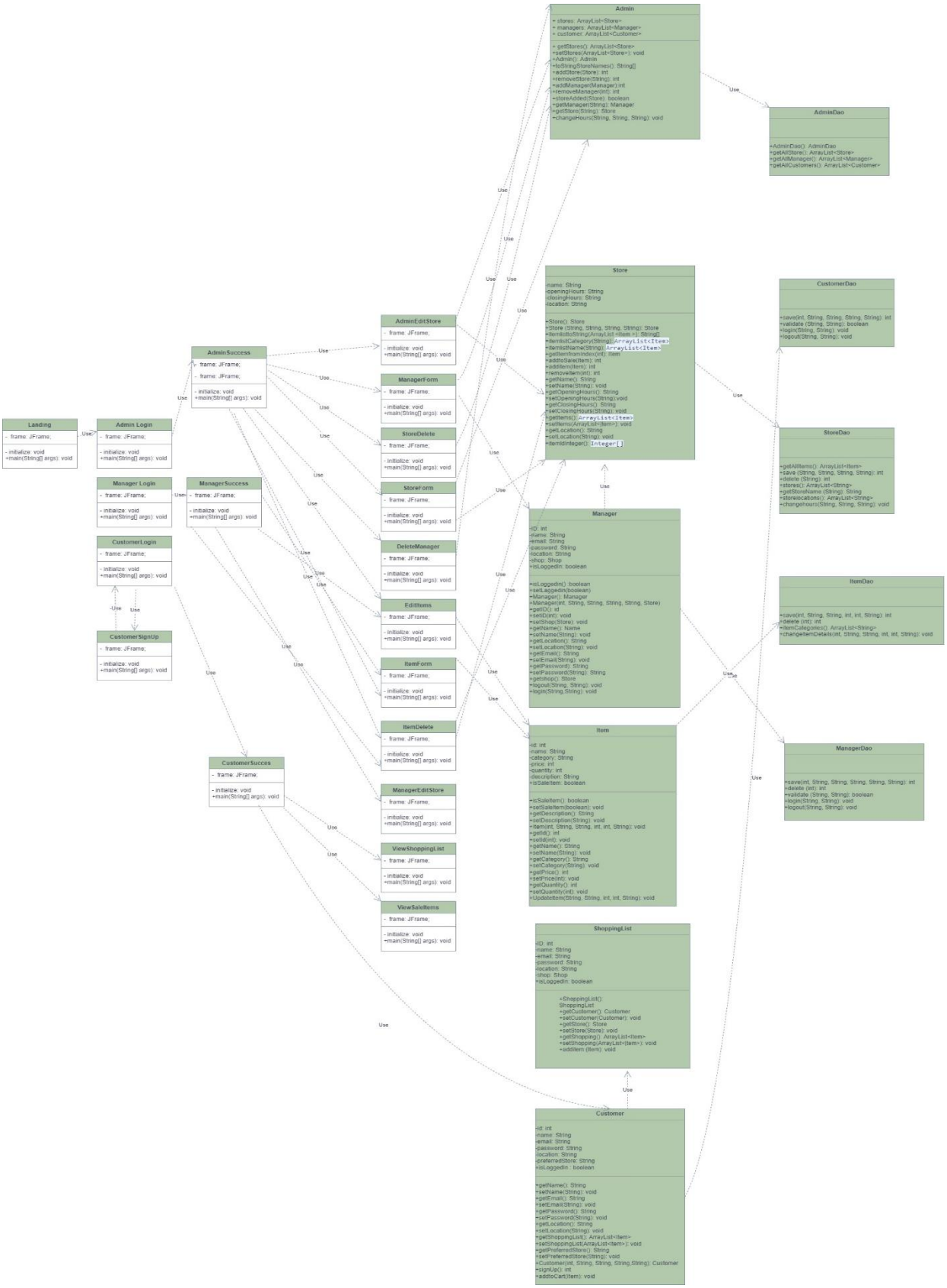
3. The second parameter will remain **root** unless you changed it to something else when setting up the MySQL Workbench Connection.
4. The password is set by you when setting up a connection first time. You can "clear" it and reset it.



Now the project is ready to be run!
Please follow the demo video to see how the source code is run.
Admin Username: admin , Admin Password: admin123

Demo Video Link: https://docs.google.com/document/d/1RCac204O_-Fke5Zdi3T1jyW3iCyBUrAv7RQ9GY_hdoY/edit?usp=sharing

UML Class Diagram for Smart Shoppers Inc



Midterm class diagram vs the final implementation:

In the final project implementation, I used Data Access Object (DAO) pattern. In contrast to the midterm, the implementation contains GUI classes. In the final project, the application/business (backend package and windowsgui packages in my project) layer is separated from the persistence layer (the dao package in my project). First the user interface GUI sends request to the backend domain classes. The backend domain classes then send request to the Dao classes to modify the external relational database.

In the midterm, I did not have any GUI class or any dao classes. So, for every method, the domain classes/ backend logic was changing but it was reflected to the user interface or in the database.

Requirements

4.1.3

New User sign up by creating a new Login instance. The login is a unique combination of userId and password. There can only be one instance of a particular login object. The login method allows the user to access the system, successful execution of the method sets isLoggedIn to true. If +isLoggedIn is false, an error message is generated by the login method itself and the user cannot execute setLocation method which is a non-abstract method within the abstract user class.

4.2.3

If the isLoggedIn boolean is true, the user can then go on to change their username, password (+setPassword(login: Login, password: String), +setUsername(username: String). They can also go on to change the location by using setLocation. If the Login method is not successful, the program will throw exception within that method and user will be repromoted.

4.3.1.3

isAuthorised method in the Manager class shows if the Manager can access the Store and its items. IsAuthorized checks if the store's manager is same as the Manager instance calling the methods. Only if isAuthorized is true, the manager can add, remove and update the item in store. Each of these methods call the given Store class's +addItem(item: Item), +removeItem(item: Item) methods which in turn update Store's availableItems HashMap. This map stores the Item and their quantity. When quantity becomes zero, the Item will no longer be returned in the getAvailableItems()'s arraylist.

Managers can also add an Item into the sales list of the store's sales items by using the method +addtoSales(store: Store, item: Item). This method accesses Store's +saleItems: ArrayList<Item> array and adds an item to it.

Managers can modify a particular item's attributes. It makes sense to change an item's description or price. Manager does so by calling the methods +modifyPrice(price: int, item: Observer) and +modifyDescription(description: String, item: Observer). Usage of Observer design pattern notifies the Item of these changes and alters their attributes as well. +notifyObserver(): method is used for that.

Administrators, like managers extend the abstract class Users. Administrators also have very similar implementation for adding, removing, updating and modifying items, except that the isAuthorized() method is not present in the administrator class. So, Administrators have access to all stores. The only

difference in the implementation of Administrator's methods and Manager's methods is that Manager's methods need to verify first is isAuthorised() is true.

4.3.2.3

In addition to the User's methods, an administrator uses the +addStore(store: Store) and +removeStore(store: Store) methods to add/ remove new stores from the system.

Administrators use the +setOpeningHour(store: Store, time: Time), +setClosingHour(store: Store, time: Time) methods to change the hours of any store in the system. And update store map using updateStoreMap() method. Managers have similar functionality but can only do so with the assigned store (again, using the isAuthorised() method for managers). StoreMap is one of the public attributes of the Store that stores the category and a list of items associated with each category in a HashMap.

4.3.3.3

The administrator uses the +addManager(e: Employee) and +removeManager(e: Employe) methods to add or remove managers from the system. The parameters are of Employee type, which is the parent class of Managers and Administrators- it contributes to the simplicity of the system. Administrator class has the +changeManager(store: Store, e: Employee) method, this method fetches the given Store from the system's database and call the setManager method (in Store class) on that store. Every store has a Manager. As the store's manager changes, the manager will no longer pass the isAuthorized() method test of the old store, ad will only be authorised to access functionalities of the new store that they are assigned to.

4.4.1.3

In Customer class, the method +searchStores() returns arrayList of stores whose locations are within within the identified distance provided by the customer. Customer can save/unsave the address of the store. These methods call the getAddress() method in Store. Customers can also +setLocation(location: String) at any time of the session.

4.4.2.3

Customer class uses the search interface to search items by either item name or item category. Search interface is implemented by the searchItembyName and searchItembycategory classes. The +searchItem(item: String): and +searchCategory(category: String) methods in the Customer class implements these methods respectively to fetch a list of Items that the Customer might be looking for, from the system.

The Customer class has the +getItemDetails(item:Item), where the parameter item is an item from the list of available items from the store that the customer is shopping in. This method returns the details (name and description, price and size, availability) of that item in a string format.

4.4.3.3

Every Customer have a shoppingList for each store they are shopping in, which is an arrayList of Items. Customers can add or remove from the shopping list using the +addItemtoShoppingList(item: Item) and +removeItemfrom ShoppingList(item: Item) methods. These methods in the Customer class are implemented by calling add and remove methods in the ShoppingList class. When Customer class calls the setLocation, a new ShoppingList is initiated.

Customer can have 4 different states CustomerNewSession, CustomerActive, CustomerInactive, CustomerLocationChange. Each of these concrete states are implemented from the CustomerState interface. Customer class implements the CustomerState interface. Depending on the state of the

customer, the +getUpdatedAvailabletemsInStore(): ArrayList<Item> returns a differently updated list of available items.

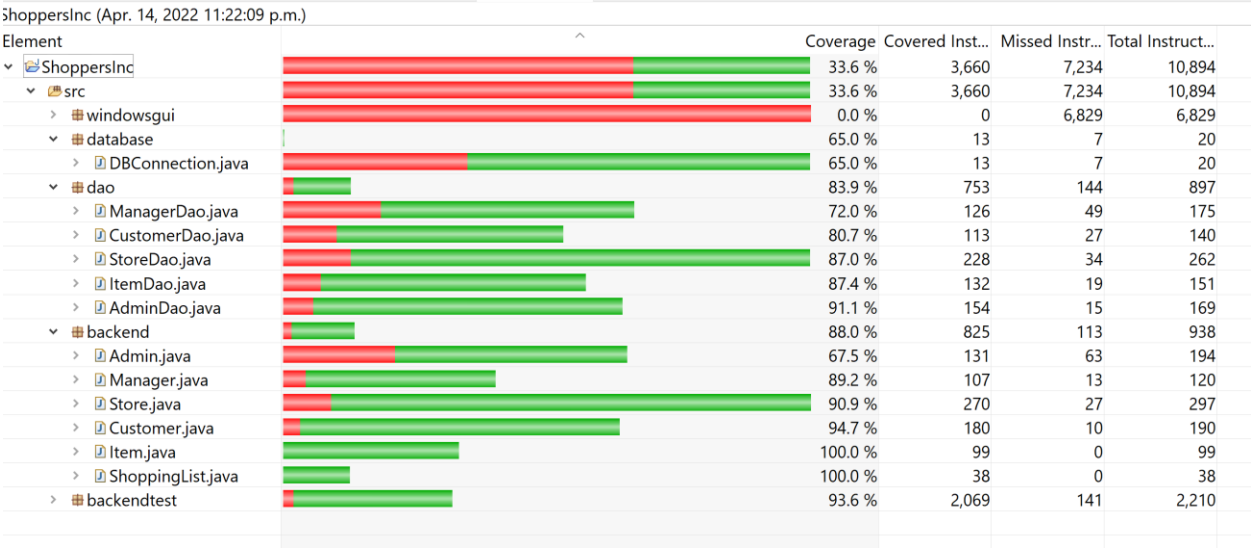
4.4.4.3

In Customer class, the +getShoppingOrder() method is implemented by taking into account the store's internal structure and by using the customer's real-time shoppingList

4.5.3 The +getAllSalesItems(): ArrayList<Item> method in customer class fetches the saleItems array from Store the Customer is shopping in. Everytime Customer calls the +searchItem(item: String) or +searchCategory(category: String) methods, these methods update the searchItem hashMap in Stores. This hashMap keeps track of the number of times an item within a store is being searched. Finally, the getRecommendedItems curates a list of recommended items and returns the list.

Testing

The project is tested using Junit 4. The code coverage for the backend and dao classes are more than 80%



Element	Coverage	Covered Inst...	Missed Instr...	Total Instruct...
ShoppersInc	33.6 %	3,660	7,234	10,894
src	33.6 %	3,660	7,234	10,894
windowsgui	0.0 %	0	6,829	6,829
database	65.0 %	13	7	20
DBConnection.java	65.0 %	13	7	20
dao	83.9 %	753	144	897
ManagerDao.java	72.0 %	126	49	175
CustomerDao.java	80.7 %	113	27	140
StoreDao.java	87.0 %	228	34	262
ItemDao.java	87.4 %	132	19	151
AdminDao.java	91.1 %	154	15	169
backend	88.0 %	825	113	938
Admin.java	67.5 %	131	63	194
Manager.java	89.2 %	107	13	120
Store.java	90.9 %	270	27	297
Customer.java	94.7 %	180	10	190
Item.java	100.0 %	99	0	99
ShoppingList.java	100.0 %	38	0	38
backendtest	93.6 %	2,069	141	2,210