SOEN 387 ASSIGNMENT 3 REPORT

Report Presented to
Professor Eric Chan

By
Krishna Patel 40176352
Brianna Malpartida 40045115
Chit Chit Myet Cheal Zaw 40110140
SOEN387 section F

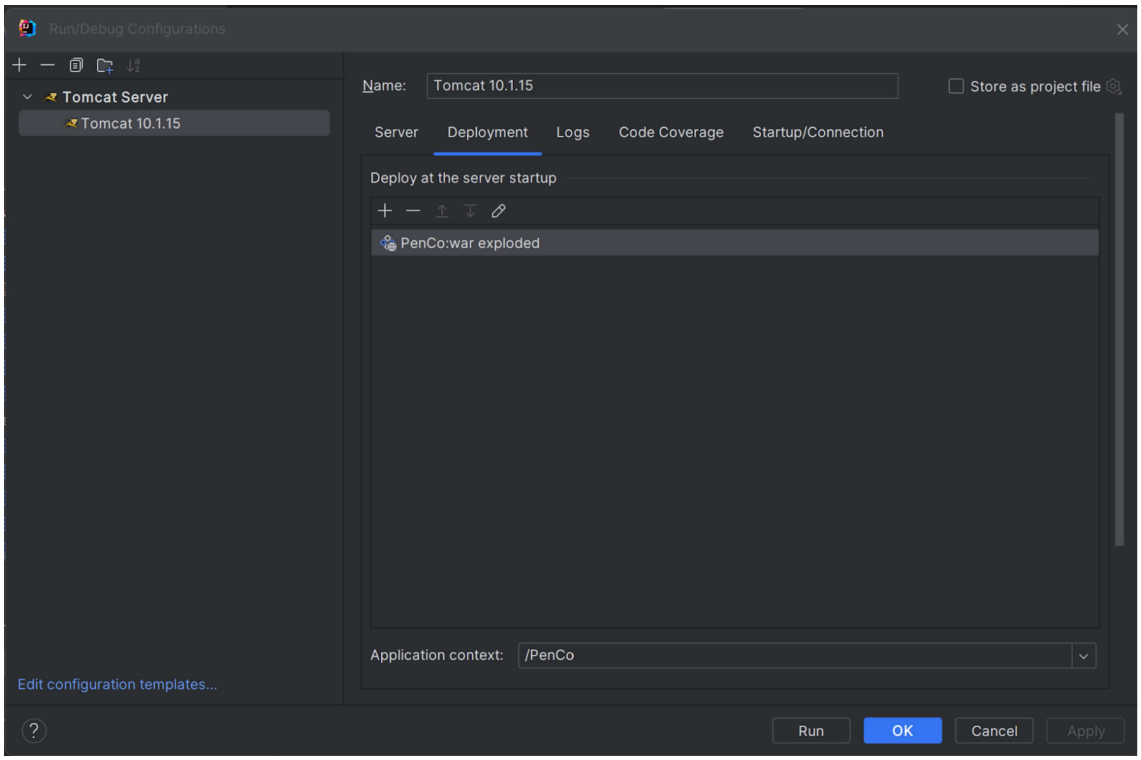
Concordia University
4 December 2023

DEVELOPMENT ENVIRONMENT USED AND HOW TO COMPILE APPLICATION

To work on this assignment, we used IntelliJ as an IDE. With Apache TomCat 10, we were able to build our project using Jakarta EE. Our systems had the latest Java JDK 21 which is also needed for development.

In order to run the application, one must first clone the code of our repository onto their local machine, and open the project in the IDE of their choice.

Once done, make sure that TomCat and all other dependencies of the project are installed and functioning. All Maven build files are available on the repository and will be imported accordingly. It is also important to ensure that the TomCat server configurations have the .war exploded file set up for deployment, just as seen in the picture below:

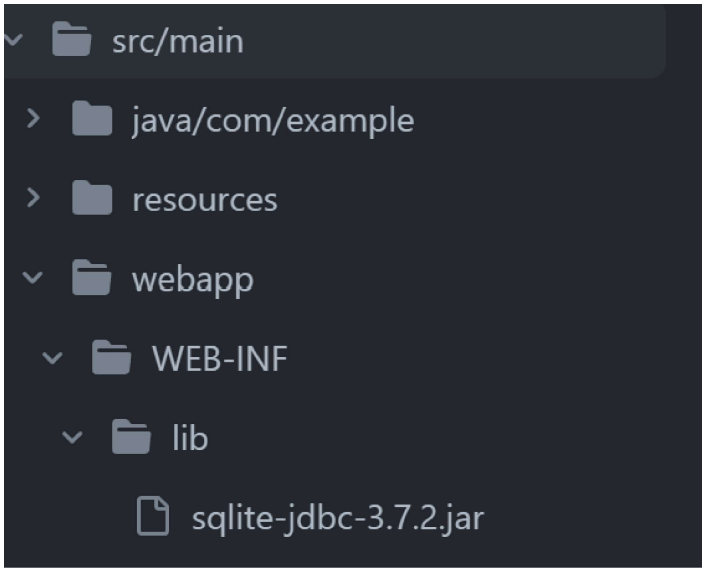


The war file can be found in the main project folder, it is named **PenCo-1.0-SNAPSHOT.war**

This assignment uses a SQLite database in order to display its data.

The connectivity to the database will be explained in a later section of the report.

Moreover, to use JDBC in our application, the appropriate .jar file has been added to the project build. It can be found among the external libraries as well as in the src file, under webapp/WEB-INF/lib:



LOGIN CREDENTIALS

Staff login:

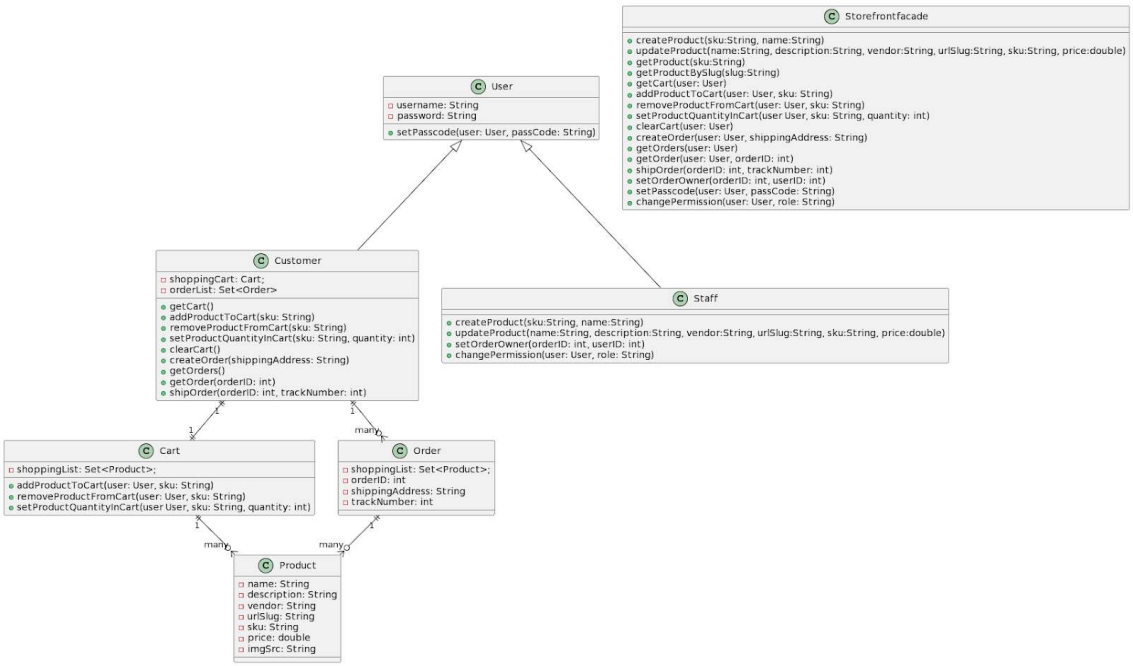
`-user passcode: staff1`

User login:

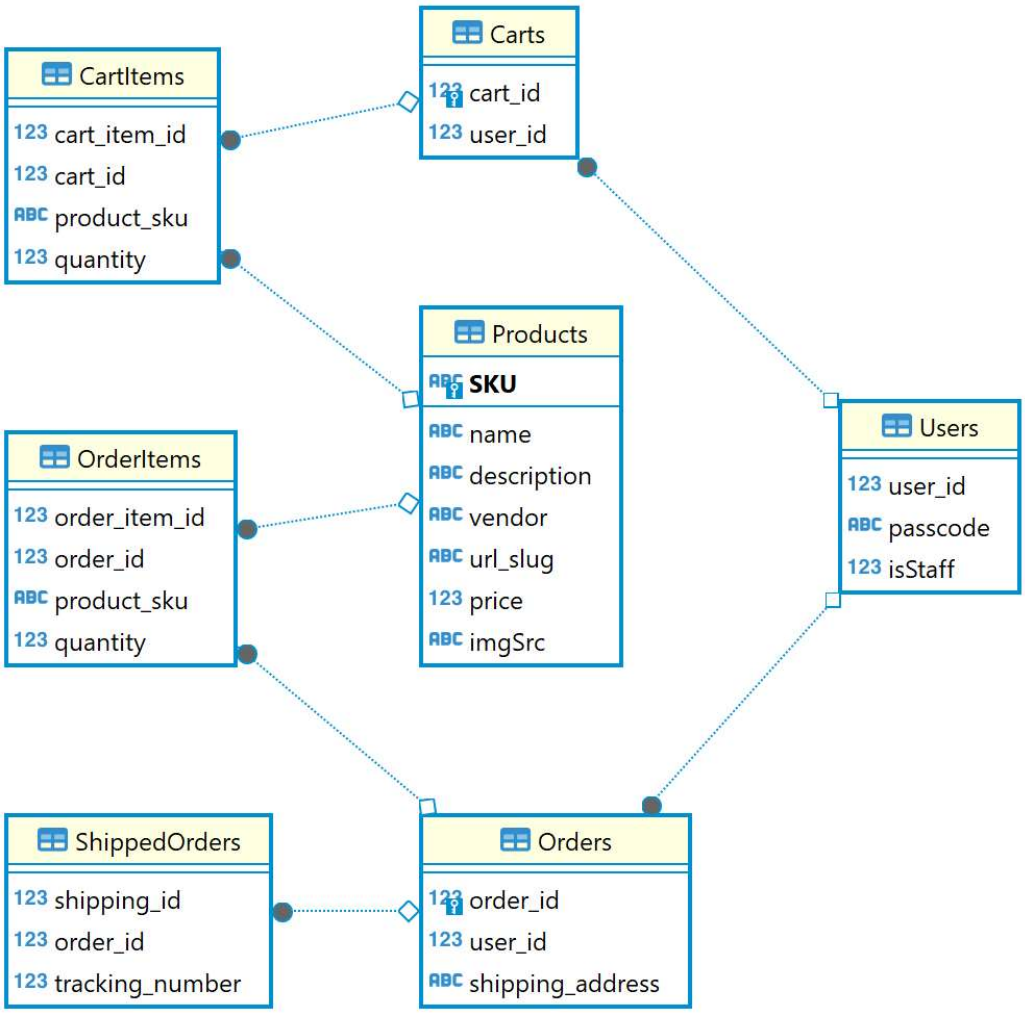
`-user passcode: user1 or user0`

TO SIGN OUT OF USER:
Restart the server.

UML DIAGRAM

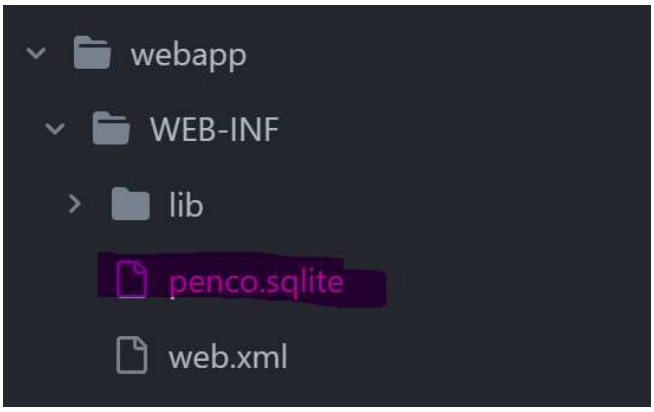

Storefrontfacade
createProduct(sku:String, name:String)
updateProduct(name:String, description:String, vendor:String, urlSlug:String, sku:String, price:double)
getProduct(sku:String)
getProductBySlug(slug:String)
getCart(user: User)
addProductToCart(user: User, sku: String)
removeProductFromCart(user: User, sku: String)
setProductQuantityInCart(user User, sku: String, quantity: int)
clearCart(user: User)
createOrder(user: User, shippingAddress: String)
getOrders(user: User)
getOrder(user: User, orderID: int)
shipOrder(orderID: int, trackNumber: int)
setOrderOwner(orderID: int, userID: int)
setPasscode(user: User, passCode: String)
changePermission(user: User, role: String)
User
username: String
password: String
setPasscode(user: User, passCode: String)
Customer
shoppingCart: Cart;
orderList: Set<Order>
getCart()
addProductToCart(sku: String)
removeProductFromCart(sku: String)
setProductQuantityInCart(sku: String, quantity: int)
clearCart()
createOrder(shippingAddress: String)
getOrders()
getOrder(orderID: int)
shipOrder(orderID: int, trackNumber: int)
Staff
createProduct(sku:String, name:String)
updateProduct(name:String, description:String, vendor:String, urlSlug:String, sku:String, price:double)
setOrderOwner(orderID: int, userID: int)
changePermission(user: User, role: String)
Cart
shoppingList: Set<Product>;
addProductToCart(user: User, sku: String)
removeProductFromCart(user: User, sku: String)
setProductQuantityInCart(user User, sku: String, quantity: int)
Order
shoppingList: Set<Product>;
orderID: int
shippingAddress: String
trackNumber: int
Product
name: String
description: String
vendor: String
urlSlug: String
sku: String
price: double
imgSrc: String
1
*
many

ER DIAGRAM


CartItems
123 cart_item_id
123 cart_id
ABC product_sku
123 quantity
Carts
123 cart_id
123 user_id
Products
ABC SKU
ABC name
ABC description
ABC vendor
ABC url_slug
123 price
ABC imgSrc
Users
123 user_id
ABC passcode
123 isStaff
OrderItems
123 order_item_id
123 order_id
ABC product_sku
123 quantity
ShippedOrders
123 shipping_id
123 order_id
123 tracking_number
Orders
123 order_id
123 user_id
ABC shipping_address

SQLite file

The penco.sqlite can be found in the WEB-INF file.



DATABASE CONNECTIVITY

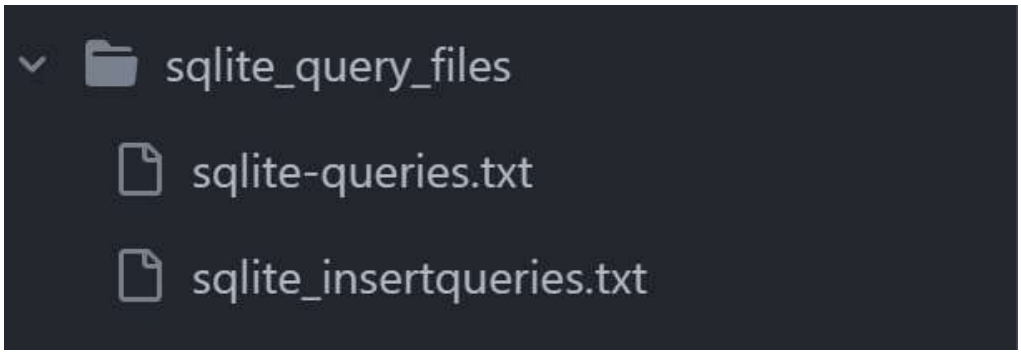
The application uses the latest version of SQLite. To simplify our application, I wrote an SQLConnector class which creates a connection to the database instantly.

```
public class SQLConnector {
    public Connection myDbConn;

    public SQLConnector() {

        try{
            Class.forName("org.sqlite.JDBC");
        }catch(ClassNotFoundException e){
            System.out.print("JDBC NOT FOUND");
        }

        String url = "jdbc:sqlite:C:/Users/Krish/Documents/School/Concordia/Projects/PenCo/src/main/webapp/WEB-INF/penco.sqlite";
        //String username = "cuties387";
        //String password = "Soen387!";

        try {
            myDbConn = DriverManager.getConnection(url);
        } catch (SQLException ex) {
            throw new RuntimeException(ex);
        }
    }
```

String url must be updated to the absolute path of the penco.sqlite file mentioned previously.

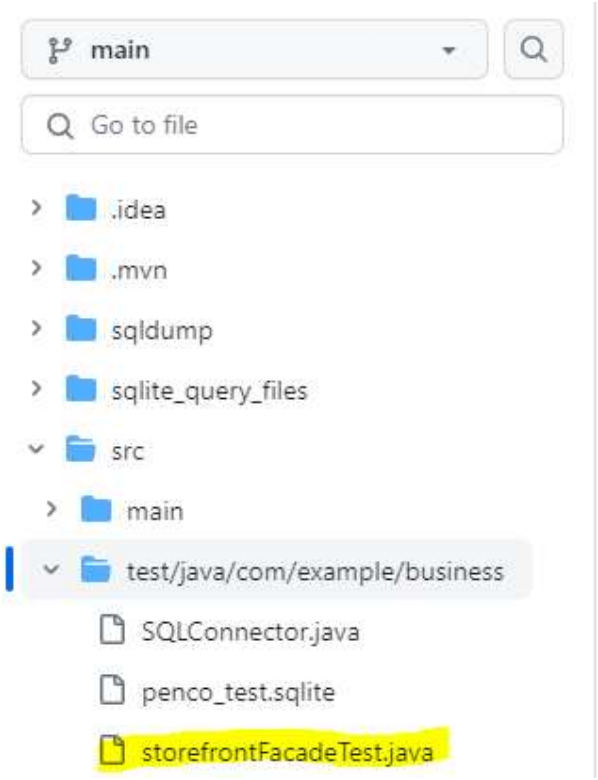
SQL FOR DATABASE CREATION

The queries used to create the DB have been stored in these txt files under the folder sqlite_queries_file:

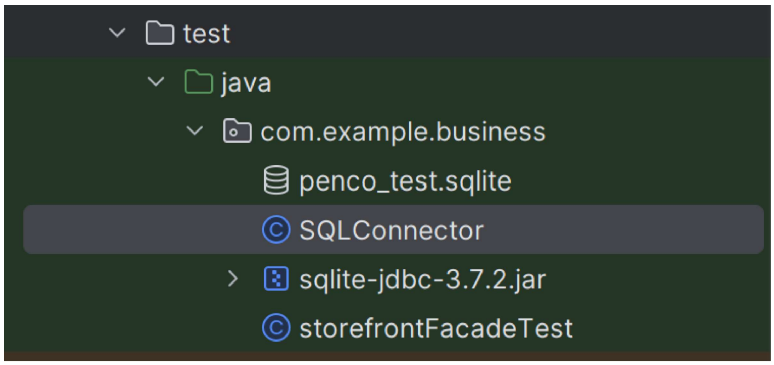


UNIT TEST

The unit test can be found under the src folder, in test.



As we used IntelliJ to create these unit tests, they can simply be run from the IDE and the results are displayed in the console of the program.



In SQLConnector under test, make sure to change the String url to the path of the penco_test.sqlite.

```
1 usage  Krishna Patel
public SQLConnector() {

    try {
        Class.forName( className: "org.sqlite.JDBC");
    } catch (ClassNotFoundException e) {
        System.out.print("JDBC NOT FOUND");
    }


    String url = "jdbc:sqlite:C:/Users/Krish/Documents/School/Concordia/Projects/PenCo/src/test/java/com/example/business/penco_test.sqlite";


    //String username = "cuties387";
    //String password = "Soen387!";

    try {
        myDbConn = DriverManager.getConnection(url);
    } catch (SQLException ex) {
```

The tests can be run by clicking on the run arrow next to the desired test on intelliJ.

```
class storefrontFacadeTest {

    Krishna Patel
    @Test
    void setOrderOwner_alreadyClaimed() {
        // Create a Customer object
        Customer customer = new Customer();
        int orderId = 1000;

        // Create an instance of the class containing the setOrderOwner method
        storefrontFacade facade = new storefrontFacade();

        // Call the setOrderOwner method with the Customer and orderId
        facade.setOrderOwner(customer, orderId);

        // Assert that the order was claimed by the Customer
        assertTrue(orderIsClaimed(orderId, customer.getUserId()));

        // Attempt to claim the order again
        facade.setOrderOwner(customer, orderId);

        // Assert that the order is still claimed (it should not be claimed again)
        assertTrue(orderIsClaimed(orderId, customer.getUserId()));
    }
```

COMMIT HISTORY

The commit history of our project can be found on the public repository of the assignment here on this page.
Insights on contributors can be found here.