**COIT13229 Applied Distributed Systems**

**Term 1, 2024**

**Assessment 3**

Lecturer: Ather Saeed

Tutor: Ather Saeed

Prepared by:

Justin Hastings 12147349

Liam Elsum 12188186

Due Date: 07/06/2024

Table of Contents

[Assumptions 3](#_Toc168691743)

[System Architecture 3](#_Toc168691744)

[Project Setup 3](#_Toc168691745)

[Client to Server Connection 3](#_Toc168691746)

[Database Design 4](#_Toc168691747)

[Build Instructions 5](#_Toc168691748)

[GitHub Repo 5](#_Toc168691749)

[Applications 5](#_Toc168691750)

[Database Configuration 5](#_Toc168691751)

[Project Configuration 6](#_Toc168691752)

[Tests 8](#_Toc168691753)

[Synchronisation 25](#_Toc168691754)

[References 27](#_Toc168691755)

# Assumptions

While investigating the requirements of the task, it was apparent that some assumptions would be required to clarify some ambiguous aspects. These assumptions included:

* There are 5 initial postcodes and 5 days of the week, so each postcode will have its own delivery day until an admin or staff user creates new delivery schedules.
* Each order line will be saved with GST included in its total.
* There is an infinite supply of products, hence we don’t need to handle out of stock occurrences.
* Since payment processing is not required, a simple button click will simulate the payment process.
* Customers will have the option to select a preferred delivery time between 8am and 5pm.
* Users must register as a customer in order to place orders.
* Admin and staff users cannot place orders, they must create their own customer accounts in order to place orders.

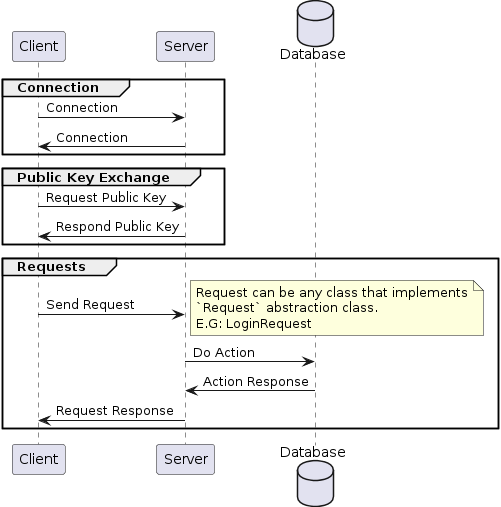
# System Architecture

## Project Setup

The project uses Gradle to manage and build everything. The project is separated across three projects, server, shared and client. Server and client both use the shared project.

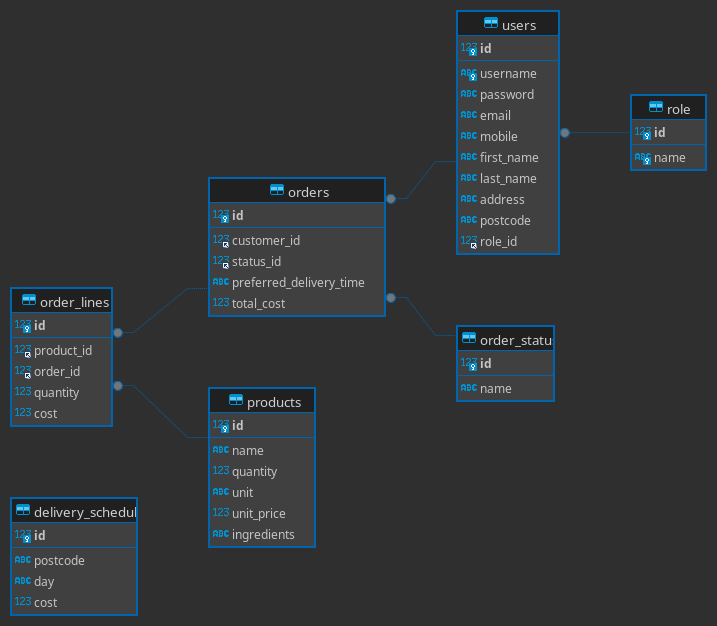
## Client to Server Connection

The client and server communicate to each other using TCP. All requests are done using objects that inherit from an abstract class called “Request”. A diagram depicting standard request flow is below.



## Database Design

The ER diagram for the database that this project uses is below.



# Build Instructions

## GitHub Repo

The GitHub repository can be found here:

<https://github.com/Voltstro/COIT13229-Assessment3>

## Applications

These are the versions of apps that we recommend for building and testing the application.

MySQL 8

Java 17

Latest version of NetBeans

## Database Configuration

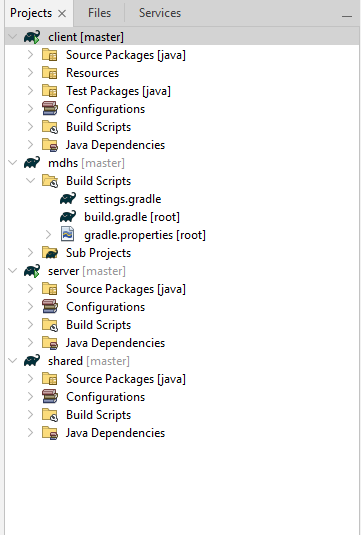
The project relies on a database. The default user the connection that it will try to use is “mdhs” with a password of “Testing123”.

The project will use a database called “mdhs”. The project will automatically create this database. DO NOT PRE-CREATE THE DATABASE Otherwise the application will not create the tables and seed data.

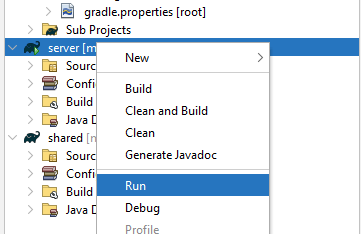
If need be, the database configuration can be changed by going to au.edu.cqu.jhle.shared.database.DatabaseUtility file and editing the constants at the top of the file.

## Project Configuration

Open the root project in NetBeans. NetBeans should show three “Sub Projects”. Open all three by either double clicking, or right click, then open project. Once opened, you should see something like this screenshot.



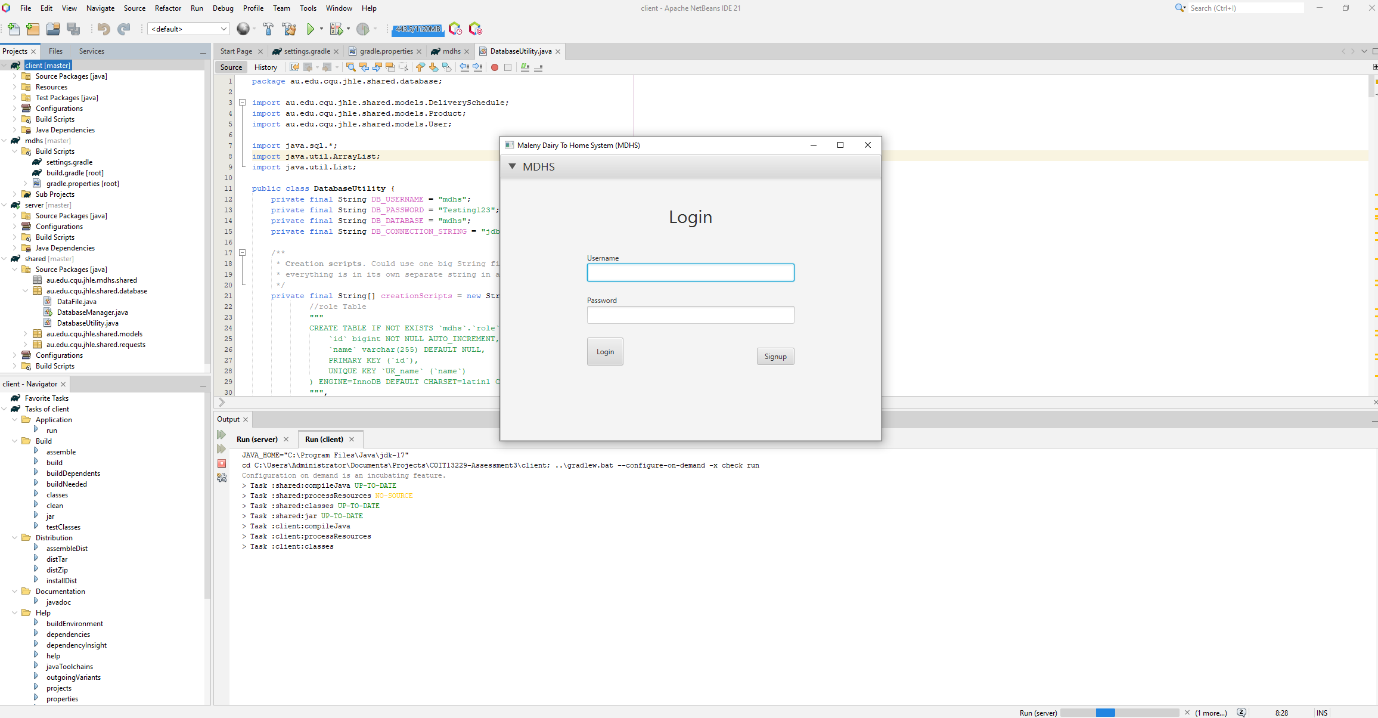
Once ready, right click on server, then either run or debug the project.



The server should now be running.



To start the client, do the same run/debug on the client project. Once running, you should be able to see the client opened.



The default admin user is “test”, with a password of “Testing123”.

# Tests

|  |  |
| --- | --- |
| **Test Name** | User Login |
| **Expected Result** | Admin users can login and get taken to the home screen. |
| **Actual Result** | Admin user provides login details    Successful Login |

|  |  |
| --- | --- |
| **Test Name** | Customer Login |
| **Expected Result** | Customer users can login and get taken to the home screen. |
| **Actual Result** | Customers provide login details    Customer gets logged in. |

|  |  |
| --- | --- |
| **Test Name** | Customer Signup |
| **Expected Result** | Customers can signup |
| **Actual Result** | New customer provides details.    On “Signup”, login happens with new account. |

|  |  |
| --- | --- |
| **Test Name** | Invalid Login Details (Username) |
| **Expected Result** | When an invalid username is provided, an error is shown. |
| **Actual Result** | An error message is shown on invalid username. |

|  |  |
| --- | --- |
| **Test Name** | Invalid Login Details (Password) |
| **Expected Result** | When an invalid password is provided, an error is shown. |
| **Actual Result** | An error message is shown on invalid password. |

|  |  |
| --- | --- |
| **Test Name** | View Orders |
| **Expected Result** | Users can view all orders |
| **Actual Result** |  |

|  |  |
| --- | --- |
| **Test Name** | Create Orders |
| **Expected Result** | User can create orders |
| **Actual Result** | On success, order is created. |

|  |  |
| --- | --- |
| **Test Name** | Create Orders (Invalid Fields) |
| **Expected Result** | On invalid fields (Like invalid data type, or empty fields) an error is shown. |
| **Actual Result** | An error is shown on empty fields |

|  |  |
| --- | --- |
| **Test Name** | View Order’s order lines |
| **Expected Result** | Users can view order’s order lines |
| **Actual Result** |  |

|  |  |
| --- | --- |
| **Test Name** | Create Order’s order lines |
| **Expected Result** | Users can create/add to order’s order lines. This should also add the cost to the order’s total cost. |
| **Actual Result** | Cost is a disabled calculated field. On success, order line is added.    Total cost of all order lines is saved to order. |

|  |  |
| --- | --- |
| **Test Name** | Create Order’s order lines (Invalid fields) |
| **Expected Result** | On invalid fields (Like invalid data type, or empty fields) an error is shown. |
| **Actual Result** | An error is shown on empty fields.    An error is shown on invalid data type. |

|  |  |
| --- | --- |
| **Test Name** | View Products |
| **Expected Result** | User can view products |
| **Actual Result** | User can view all products |

|  |  |
| --- | --- |
| **Test Name** | Create Products |
| **Expected Result** | User can create products |
| **Actual Result** | Users can create new Product.    On success, product is created. |
| **Test Name** | Add Product (Invalid Fields) |
| **Expected Result** | On invalid fields (Like invalid data type, or empty fields) an error is shown. |
| **Actual Result** | An error is shown on empty fields.    An error is shown on invalid data type. |

|  |  |
| --- | --- |
| **Test Name** | View Delivery Schedule |
| **Expected Result** | User can view delivery schedules |
| **Actual Result** | Delivery Schedules can be viewed. |

|  |  |
| --- | --- |
| **Test Name** | Create Delivery Schedule |
| **Expected Result** | User can create delivery schedules |
| **Actual Result** | New delivery schedule details can be provided    New delivery schedule gets successfully created. |

|  |  |
| --- | --- |
| **Test Name** | Create Delivery Schedule (Invalid Fields) |
| **Expected Result** | On invalid fields (Like invalid data type, or empty fields) an error is shown. |
| **Actual Result** | An error is shown on empty fields.    An error is shown on invalid data type. |

|  |  |
| --- | --- |
| **Test Name** | View Users |
| **Expected Result** | Admin users can view users |
| **Actual Result** | All users can be viewed. |

|  |  |
| --- | --- |
| **Test Name** | Create Users |
| **Expected Result** | Admin users can create more users |
| **Actual Result** | New user’s details can be provided.    New user gets successfully created. |
| **Test Name** | Create Users (Invalid Fields) |
| **Expected Result** | On invalid fields (empty fields) an error is shown. |
| **Actual Result** | An error is shown on empty fields. |

# Synchronisation

Synchronisation in this context refers to ensuring the integrity of the data and that it is consistently maintained across the application. It focuses on locking out specific functions, usually database calls, whilst they are in use by a client to prevent conflicts by other clients trying to access or manipulate the same data in a separate transaction. Most distributed systems utilise multiple threads and processes to access data in concurrent transactions. Synchronisation is a necessity for these concurrent transactions to prevent corrupted and inconsistent outcomes (Lenovo, n.d.). If synchronisation is not utilised, or is incorrectly implemented, the application could generate drastically different outcomes for functions. For example, if staff user 1 is editing product 1 to change the price, but at the same time in a separate transaction, staff user 2 removes product 1. This could lead to a couple different outcomes, such as staff user 1’s changes being lost when staff user 2’s delete operation is committed. Another potential outcome is that staff user 1’s changes are now trying to reference a record that has already been deleted and no longer exists because staff user 2’s delete transaction was committed before the update, hence landing the database in an inconsistent state. This scenario is faced similarly by the majority of mutating functions in this application, including but not limited to editing users, delivery schedules, orders etc. Therefore, synchronisation, and the correct implementation of it, is imperative to the success of this application.

# References

Lenovo, n.d., *Whys is Synchronization Important in Distributed Systems & Databases*, viewed 06 June 2024, <https://www.lenovo.com/au/en/glossary/syn/?orgRef=https%253A%252F%252Fwww.google.com%252F>