Group report
Group two

N.Allen, A.Azeez, X.Chen, M.Grabara, M.Lee, W.Nujum, J.Tong, A.Wood.

May 13, 2019

As a team we have designed, developed, implemented and tested a Stock Management System for a new restaurant chain Chicken Lovers United Kingdom. This document includes a technical description of the architecture, functionality and testing of our solution.

1 System Architecture

1.1 Database

All system data is stored in a MySQL relational database. This database is split into twelve tables derived from the ER diagram [1]. The database makes use of foreign keys to enforce referential integrity. Vulnerable data, such as passwords, is hashed using the SHA256 hash function for security. The requests to the database are setup to protect against SQL injection. An admin account with full access to all areas of the database has been supplied with the system to allow the business to make adaptations in the case of future domain changes.

1.2 Back-end

For the backend of our system we used Java methods and classes to implement the functionality outlined in the specification document. Each Java method retrieves data from the database through the use of SQL queries and the official MySQL Connector/J 8.0.15. Web requests sent by the front end are handled by Jersey 2.28 framework, a reference implementation of Java API for RESTful Web Services. The backend always responds using JSON objects, or standardized error messages. The frontend sends requests to the backends' specific endpoints. GET requests contain additional specific information attached to the request header, while POST requests consist of a request body, which is a JSO.

1.3 Front-end

The frontend of the web app was built using the Angular 7 framework, the Angular Material component library and Node.js 8 LTS. The Angular frame generates app component objects that dynamically interact with the Java back-end system. For example, when a warehouse manager navigates to the total stock component on the web app, this component will call the relevant java method to request the total stock in the warehouse for a particular food stock item (I.e. cheese slices). The returned data is then displayed to the user in a dynamically refreshed data table. Each of the approximately 30 app component objects are inserted into the web browser's DOM using a root app component that is at the top of the hierarchy tree data structure. Each of these app components consist of a html file, a CSS file and a TypeScript file (Angular 7 uses a superset of JavaScript called TypeScript to control the behavior of the component). These three data files are compiled into an app component using the Node.js runtime environment.

2 Functionality

2.1 Accounts

Our app includes an authorisation system. Before accessing any further functionality a user must log into the system. An account can be either a driver, warehouse or restaurant account. Warehouse and restaurant accounts can have additional manager privileges. Our system includes the functionality to create and delete accounts, check and update account info, and check and update account permissions. Each account type has access to different system functionality. An account of each type has been supplied with the system.

2.2 Warehouse

When a user logs onto the system they are presented with a navigation bar. For users with warehouse permissions this navigation bar displays the following functionality:

- **Add Stock:** Add stock items to warehouse inventory.
- **Remove Stock:** Used to removes lost or damaged items form warehouse inventory.
- **View Pending Orders:** Displays custom orders placed by restaurants. Allows a user to Approve or decline orders. When an order is approved the stock for that order is removed from the warehouse inventory. If the warehouse does not contain enough stock an error message is displayed.
- **Total Stock:** Shows total stock held at the warehouse. Highlights stock items below minimum threshold.
- **Dispatch report:** Graphical report displaying the quantity of stock items sent to each restaurant location over a specified time period.
- **Logout:** Logout of the system.

2.3 Restaurant

Users with restaurant permissions have access to the following functionality:

- **Total Stock:** Shows total stock held at the restaurant. Highlights stock items below minimum threshold.
- **Receive Stock:** Displays the deliveries due to arrive today. Allows for confirmation of delivery. This increases the restaurant stock level by the contents of the order.
- **Remove Stock:** Used to remove lost or damaged items form restaurant inventory.
- **Sales:** Point of sale interface. Allows menu items to be created. Reduces restaurant stock accordingly.
- **Order Stock:** Allows users to create a standard or custom order. Standard orders are automatically approved. The warehouse stock level is checked to ensure the order can be delivered. If the stock is two low an error message is displayed. Custom orders must be approved by a warehouse user. Deliveries are restricted to a maximum of two per week per restaurant. Deliveries must also be at least three days apart. These conditions are checked, and the delivery date is automatically set to the next available working day.
- **Logout:** Logout of the system.

2.4 Manager

Users with manager permissions have access to the following additional functionality:

- **Minimum Stock:** View and update the warehouse or restaurant minimum stock threshold.
- **Account Manager:** Create and delete accounts. View and update account permissions.

2.5 Driver

We created a google maps routing system to display the delivery schedule for each day. The system checks the database to obtain a list of the restaurants that will be receiving deliveries today. The Google Maps API is then used to auto-generate a time and fuel efficient route to each of these restaurants. The routing system also tracks the journey time to ensure that deliveries do not take over 9 hours and occur within the restaurants opening times. If a journey time exceeds 4.5 hours a compulsory rest break is added. When a user with driver permissions logs into the system the days' delivery route is displayed.

We have also developed an android app that displays the location of the warehouse and restaurants. The app allows for locations to be selected and plots a delivery route.

2.6 Functional requirements not satisfied

Out of the 26 functional requirements listed in the specification document, the group has successfully met 25 of them. All of the high priority and compulsory requirements have been successfully met. However, the group was not able to complete the optional Point of Sale interface that is suggested as an extension. The group prioritized the compulsory requirements and left the low priority Point of Sale extension until the end of the project. The group did make a start on it, with the html framework being developed. However, there was not enough time available to make it functional using TypeScript and inserting it into the Angular framework.

3 System Testing

Each backend endpoint has been tested in isolation to ensure that it returns the correct JSON data. The Angular web app has been continuously tested as additional web app components have been added. The TypeScript files have been tested using the test classes that are auto-generated by the Angular system. The functionality of the android app has been fully tested via an Android Mollie and Emulator. The final system has been black box tested by users to ensure all functionality is working as intended.

4 Effort allocation

All members of the team have contributed work and effort to the project. We have collectively decided to award each member of the team an equal share of the marks. Each person will receive a one-eighth split of the percentages (I.e. 12.5%)

Name	Effort allocation
N.Allen	12.5%
A.Azeez	12.5%
X.Chen	12.5%
M.Grabara	12.5%
M.Lee	12.5%
W.Nujum	12.5%
J.Tong	12.5%
A.Wood	12.5%

References

[1] CSC8005 Team 2 Requirements Specification Section 10.