VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Thi Van Linh Le

INVENTORY MANAGEMNET APPLICATION

Android Mobile Application

Technology
2022

ACKNOWLEDGEMENTS

First of all, I would like to thank to Dr. Ghodrat Moghadampour. Without his patient instructions during the whole process of my final thesis, I could not have finished this thesis in time.

I would also like to thank to my parents. I appreciate their effort and love in bringing me up to be a better individual.

Finally, my dear friends, I would like to thank them very much for spending their precious time helping me with other works and give me time to do my final thesis.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Thi Van Linh Le
Title	Inventory Management Application
Year	2022
Language	English
Pages	64
Name of Supervisor	Dr. Ghodrat Moghadampour

This thesis aimed to develop an Android application, which was designed specifically for retailers, to control and track their inventory remotely from anywhere using smartphones. The application had met all requirements and deployed successfully.

This inventory management application is used as a centralized database for inventory information and allowed users to manage inventory information, track inventory data, label inventory, search item by scanning barcode, track data log history, manage user access, and alert low stock and over stock.

To reduce human error, improve efficiency and accuracy, this application identifies each item by scanning barcode labels. The application uses the cloud-based database system, which allows multi-user access and inventory management, First In - First Out methodology as an inventory display.

Keywords Inventory, management, application, Android, and mobile.

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

API	Application programming interface
MVVM	Model-View-View Model
UI	User interface
JSON	JavaScript object notation
BSON	Binary encoded JavaScript Object Notation
JWT	JSON web token
HTTP	Hypertext Transfer Protocol
ODM	Object Data Modeling
SSPL	Server-Side Public License
IDE	Integrated Development Environment
XML	Extensible Markup Language
UML	Unified Modeling Language

LIST OF APPENDICES

1 INTRODUCTION

In this chapter, the introduction, including background, aims and objectives of the thesis, is described as below.

1.1 Background

An inventory management system is popular nowadays for controlling, tracking inventory instead of repetitive and tedious manual work. An Android application was developed by demand with a cloud-based database providing multi-user access and making inventory management work more convenient.

In this thesis, an Android application was developed using the Java language, with its back-end (Node.js REST API) connected to a database located in the MongoDB Atlas Cloud.

1.2 Objectives

First, the demand of the organization is analysed. Second, based on the result of research, a plan was made. This plan was used as an instruction to show how the application in separate phases. Next, the programming languages and technologies was analysed and chosen based on the research. This phase is a principal option because the tools was provided the best method to achieve the goal. The application was developed and tested before transferring to the company for further tests. The company will return a report which will show the result of practical test as well as the requirement for editing demand. The application will be checked and re-programmed based on the company report. Finally, the application will be sent to the company as a final product.

2 RELEVANT TECHNOLOGY

In this chapter, relevant technologies and libraries used in the whole project are explained here.

2.1 JavaScript

JavaScript (often shortened to JS) is a lightweight, interpreted, object-oriented programing language with first-class functions based on objects which can return a value by passing a function itself to other functions as an argument. It is commonly used as the scripting language for Web pages but is also used in many non-browser environments. In September 1995, Mocha was first developed by a Netscape programmer named Brendan Eich, but quickly became known as LiveScript and, later, JavaScript. (MDN contributors 2022)

2.1.1 JavaScript Versions

In 1997, due to JavaScript's rapid growth, Netscape standardized JavaScript with ECMA (European Computer Manufacturers Association) - a non-profit organization that develops standards in computer hardware, communications, and programming languages. The ECMA specifications were labeled ECMA-262 and ECMAScript languages included JavaScript, JScript, and ActionScript. Since then, different versions of ECMAScript have been released and abbreviated to ES1, ES2, ES3, ES4, ES5, ES6, ECMAScript 2016, ECMAScript 2017, ECMAScript 2018. (Kopecky, 2020)

This project uses the JavaScript version 2015 knows as ES6.

2.1.2 ES6

ES6, also officially known as ECMAScript 2015 or ES2015 is a significant update to the JavaScript programming language. It is the first major update to the language since ES5 which was standardized in 2009. Therefore, ES2015 is often called ES6. Some features of ES6 has been used in this project and will be discussed below.

ES6 and new keywords let and constant: The let keyword declares a variable with block scope, it is not possible to access let variables outside a function expression. Constants can be declared using the const keyword, const variables name cannot be declared twice.

ES6 and new arrow functions: functions expressions can be written in a short syntax using arrow functions. Arrow functions must be defined before use and they do not have their own “this”.

ES6 and promises: Instead of using callbacks, promises in ES6 can avoid “callback hell” (multiple or dependent callbacks). This project uses ES6 promise and call “resolve()” for success operation and “reject()” for failure operation. (Kopecky, 2020)

2.2 Node.js

Node.js is a cross-platform, open-source JavaScript runtime environment. Node.js runs the Chrome’s V8 JavaScript engine outside of the browser. This results in extremely fast processing. Node.js is an asynchronous, a non-blocking and event-driven I/O system, it does not wait for one API call to complete before moving to the next one. Instead, it executes the next event and returns to the previous using a callback function that was specified before. Rather than blocking the thread and wasting CPU cycles waiting, Node.js resumes the operations when the response is received. As a result, Node.js enables to handling of thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs. (Node n.d.)

Node.js was chosen because Node.js offers high-performance for real-time applications, easy scalability and has a large community support.

2.3 Express

Express is a library, a Node.js web application framework that provides a core set of features to develop web and mobile applications. Express provides mechanisms to write handlers for HTTP requests, to set common web application setting such as connecting port / template location, adding additional request processing "middleware". There are plenty of middleware packages can solve almost any web development problem, examples are libraries to work with sessions, cookies, URL parameters, POST data, security headers.

2.4 JSON Web Token

JSON Web Token (JWT) is an open standard (RFC 7519 used to share security information between two parties (a client and a server) as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using ‘a secret’ or ‘a public/private key pair’ (using RSA or ECDSA). JSON Web Tokens consist of three parts, which are: Header, Payload and Signature. Due to its small size, JWT is transmitted quickly and can be sent through a URL, through a POST parameter, or inside an HTTP header. In this project server uses secret string to create JWT and sent to client when login success, client sent HTTP requests included JWT in the header to authenticate itself with the server. (JWT n.d.)

2.5 Bcrypt Hashing Algorithm

Keeping in mind the fact that passwords must never be stored in plaintext, there are plenty of algorithms to safely store passwords such as MD5, SHA1, SHA256, SHA384, PBKDF2, or Bcrypt. This project uses Bcrypt to hash passwords. Bcrypt requires attackers a lot of computational time compared to others.

Bcrypt hashing function allows us to build a password security platform that scales with computation power and always hashes every password with a salt. Known as

a slow algorithm, Bcrypt requires a salt as part of hashing process, therefore it reduces the number of passwords by second an attacker could hash when making a dictionary attack. A salt is a unique, randomly generated string that is added to each password as part of the hashing process. (Arias, 2021)

2.6 Mongoose

Mongoose is an Object Data Modeling (ODM) library that creates a connection between MongoDB and Node.js. It offers a variety of hooks, provides model validation, manages relationships between data, and is used to translate between objects in code and the representation of those objects in MongoDB.

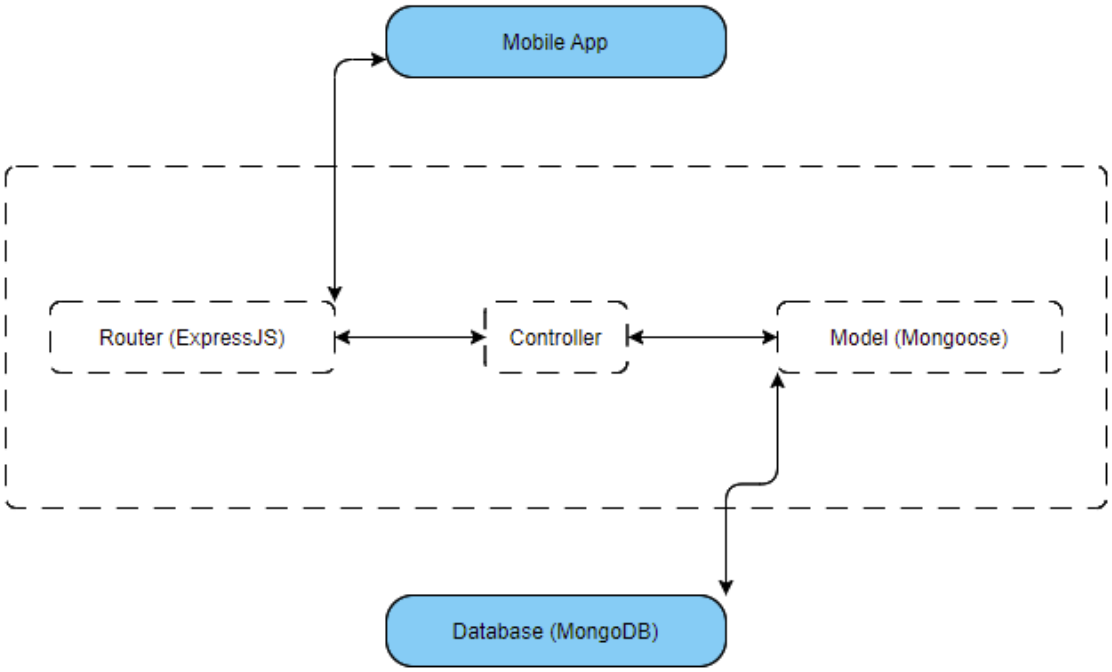

Figure 1. Mongoose in the server-database relationship. (Manning, n.d.)

Figure 1 shows the relationships between parts, the router receives http requests from the mobile application and forwards them to the controller, the controller retrieves the data by working with models which are defined with the Mongoose

library. The router also receives data from the controller and sends it to the mobile application.

2.7 MongoDB

MongoDB is an open-source document-oriented, NoSQL database and is developed by MongoDB Inc, licensed under the 'Server Side Public License' (SSPL). MongoDB is not based on using tables and rows to store data as traditional SQL database, it makes use of collections and documents instead. MongoDB provides a mechanism for the storage and retrieval of data called BSON (which is a binary JSON), it stores data records as documents which are grouped together in collections. Because MongoDB does not require predefined schemas, users can create any number of fields in a document, documents in the same collection can have different properties and key-value pairs. MongoDB provides more flexibility for storing the non-alike data, big data application will get advantages from horizontal scalability: it also improves the speed of database operation and ability to distribute data across a cluster of machines. (MongoDB, n.d.)

2.8 Postman

Postman was designed in the year 2012 and works as an API testing tool. Postman tool allows to design, mock, debug, testing, document, monitor and publish the APIs. In this project, Postman works as an HTTP client that sends requests to the server and receives responses, it also makes API development and testing straightforward. (Postman, n.d.)

2.9 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development. Android studio contains all the Android tool to design, test, debug, and profile application. Android studio version 4.0.1 is used to develop this project which has features including: Gradle-based build support, Android Virtual

Device to run and debug, help to build up for all devices and more. (Android Developers, n.d.)

2.10 Java

Java is a general-purpose, concurrent, strongly typed, class-based, object-oriented programming language, created in 1995 and is used widely around the world. Java is popular, reliable and is used for developing different type of applications included: Mobile application (Android application), Web applications, Desktop application, Server application, Games and many more. Java code can run on different platforms such as Windows, Mac, Linux, and Raspberry Pi. Java applications are normally compiled to bytecode and can run in any Java Virtual Machine. Once Java code is compiled, it can run on all platforms that support Java without the need to recompile. (Java, n.d.)

2.11 MVVM (Model-View-ViewModel) Architecture Pattern

MVVM is a structural design pattern apply for developing application, it separates the data presentation logic(Views or UI) from the business logic part. The separate code layers of MVVM are:

Model: This layer represents the data and the business logic of the application.

View: This layer consist the UI code(Activity, Fragment) with the purpose is to send the user's action to ViewModel. This layer observes the ViewModel and does not contain any application logic.

ViewModel: It interacts with model and exposes those data streams which are relevant to the View and serves as a link between the Model and the View. (Geeks for Geeks, 2021)

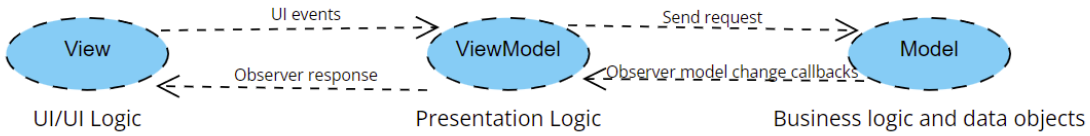

Figure 2. MVVM model.(Patrzyk, Rycerz and Bubak, 2015)

There are two ways to implement MVVM design pattern in Android projects:

- Using the DataBinding library released by Google
- Using any tool like RxJava for DataBinding.

This project applies the MVVM model on some main packages and using the DataBinding library released by Google.

2.12 Retrofit

Retrofit is a REST Client for Java and Android and is an open-source library which simplifies HTTP communication by making remote APIs declarative, type-safe interfaces. It allows retrieving and uploading JSON (or other structured data) via a REST based web service. Retrofit allows users to specify the converter that is used for data serialization. Typically for JSON use GSon but can add custom converters to process XML or other protocols. Retrofit uses the OkHttp library for HTTP requests.

3 APPLICATION DESCRIPTION

This chapter show the detailed description of the application.

3.1 General Description

The purpose of this thesis is to build a Management Inventory Application with cloud-based database. This application stores database at MongoDB Atlas Cloud. The user's information needs to be registered on the register page before logging in to access all the important features embedded in the application. In order to improve the security, this project uses JWT (JSON web token) to verify client requests send to the server. The application includes the following services:

- Authentication and authorization
  Login with a hashed password, secured with a JSON web token (JWT), limited access where the user is not a manager. Changeable password through the user's email, new user registration .
- Inventory summary
  Generates the summary information of the current stock, calculates the total value of the current stock. Shows the alert and list items which are under stock.
- Manage inventory
  Able to get/add/delete/edit/adjustment inventory information. Data request secured by a JSON web token (JWT).
- Data logs history
  Can show the list of history actions, whose action, what kind of action, time of action.
- Labeling item
  Using a mobile phone camera to add the barcode of an item, can search an item from the inventory by scanning the barcode.

3.2 Quality Function Deployment

The requirements of this application can be categorized into three types based on the priorities: must-have requirements, should-have requirements, nice-to-have requirements.

3.2.1 Must have requirements

The must-have requirements list of the application are introduced in Table 1.

Table 1. The must-have requirements list

Requirements
Database on cloud
Register user using hashed password
Signing in with JSON web token checking
Can change – reset password
Authorization (access control)
Checking summary quantity of inventory
Generate the total value of current inventory
Alert and show list items which are low stock/over stock
Checking stock information
Add stock information
Edit stock information
Delete stock information
Apply first in first out (FIFO)
Secure method when requesting data
Can scan and add a barcode by a camera
Can scan and search item information by a barcode
Can edit the max stock of each item
Checking the profile of the user

3.2.2 Should-have Requirements

The should-have requirements list of the application are introduced in Table 2.

Table 2. The should-have requirements list

Requirements
Can list data log history
Sort data log history by date time descending

3.2.3 Nice-to-have Requirements

The nice-to-have requirements list of the application are introduced in Table 3.

Table 3. The nice-to-have requirements list

Requirements
With Responsive UI
Propose re-order list

3.3 Use Case Diagram

The Use Case diagram of the application can be seen in Figure 3.

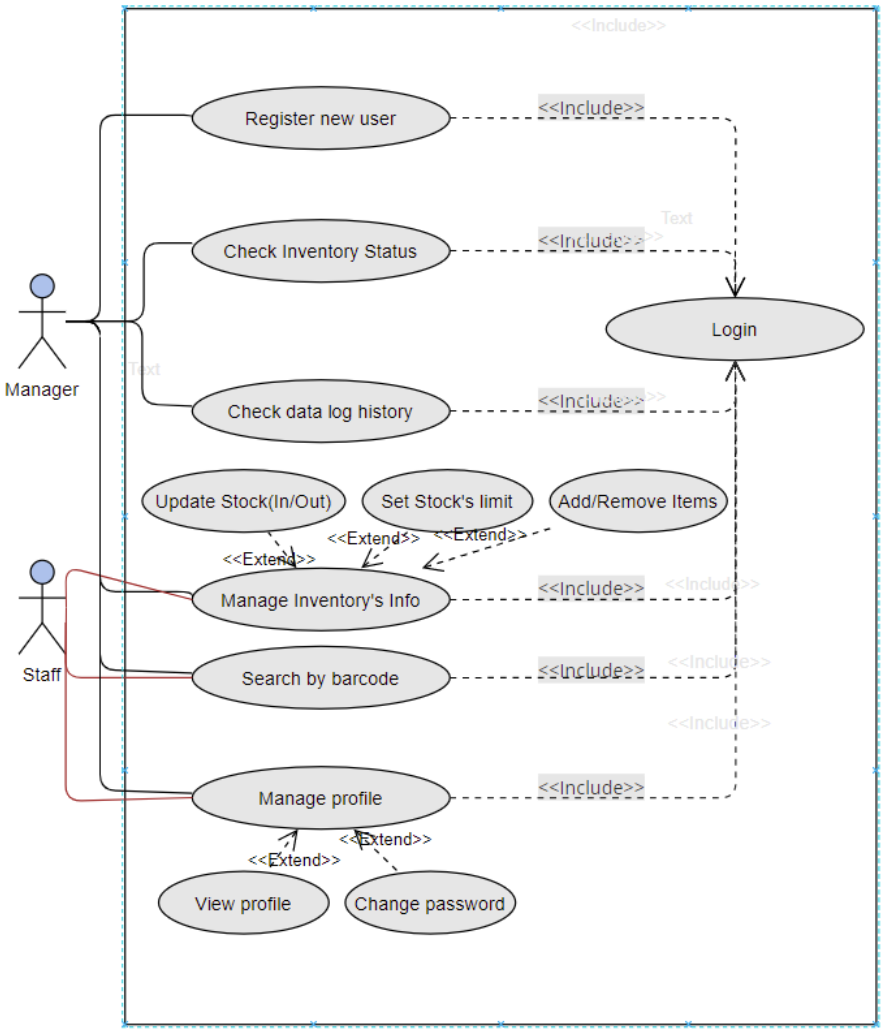

Figure 3. Use Case diagram for Inventory management.

As shown by the Use Case diagram, the manager has full rights access when the staff get limited access; there can be many manager users and many staff users. The user must be registered before logging in and accessing the application features. Users can manage the inventory and use the scanning barcode operation but only the manager user can register a new user, check the inventory status, edit the item price and check data logs.

3.4 Class Diagram

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that describes the structure of a system by showing its classes, their attributes, operations (or methods), and relationships between them. Five main packages in this project are: Login, Dashboard, Item management, Data log, Account setting. These five packages are described here.

3.4.1 Login Package

The login class description is given in Figure 4.

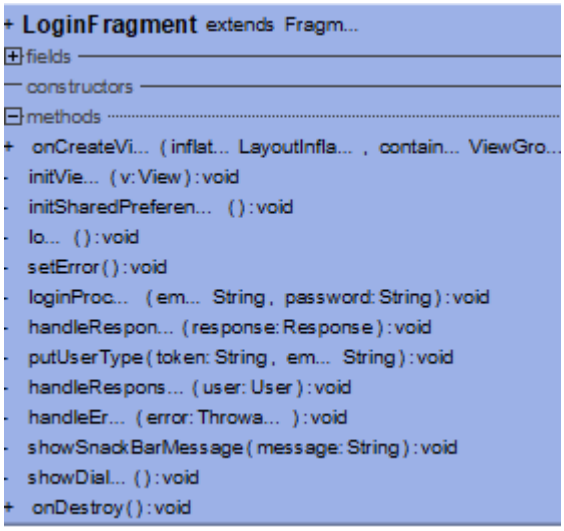

Figure 4. Class diagram for login

LoginFragment class presents the method for login to the application. The login request is sent to the server, the login's parameters are combined, encoded and set to the header variable authorization using OkhttpClient. When the login response returns successfully, the client will receive login information and a token (JWT) from the server as a response.

3.4.2 Dashboard packages, Type package, Item package, Detail package, Data Log package

Packages present from Figure 5 to Figure 9 includes classes along with their attributes, methods and relationships. These packages using MVVM model as a

part of their architectural patterns (Some methods haven't change completely to MVVM model). Repository modules provide a LiveData object for ViewModel. ViewModel class calls the Repository API. Fragment classes gets LiveData list from ViewModel, once the list data is retrieved, it is passed to Adapter classes (the RecyclerView adapter) and display the list data in the RecyclerView component.

The dashboard package is given in Figure 5.

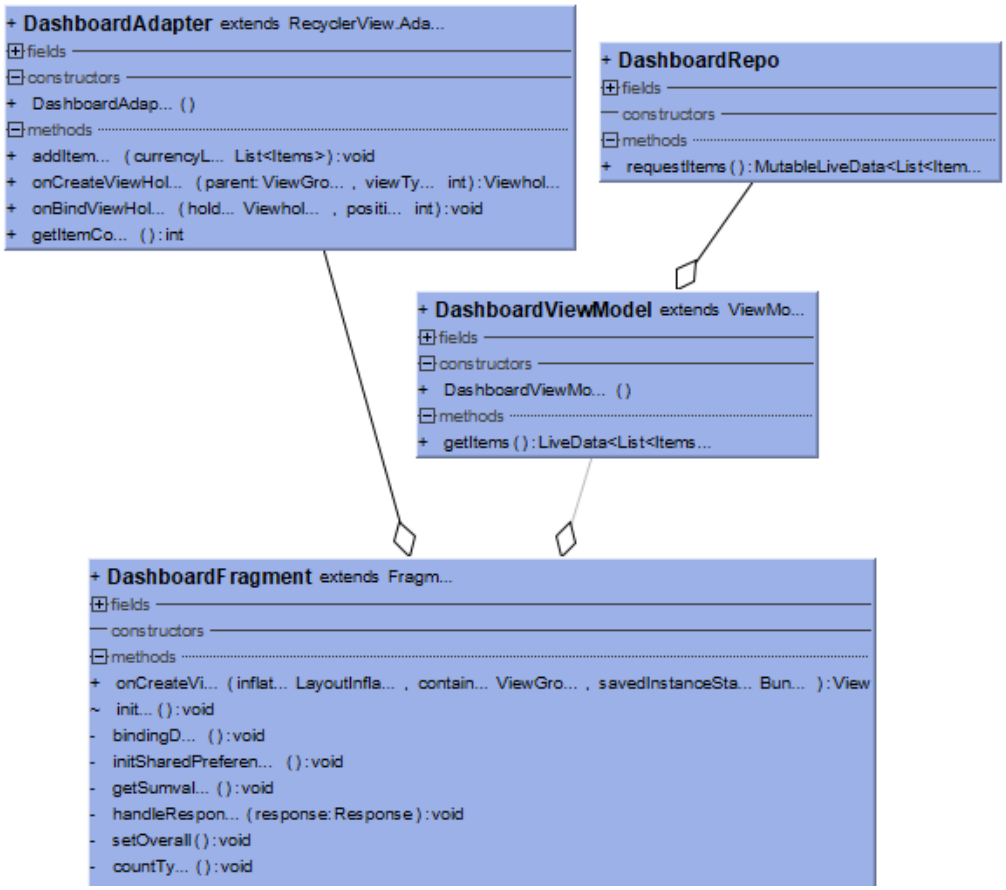

Figure 5. Class diagram for Dashboard package

The Dashboard package After the user logs in successfully, the application will display Dasboard UI included: summary status of inventory, alert list from "low stock list" and "over stock list".

The Type package is given in Figure 6.

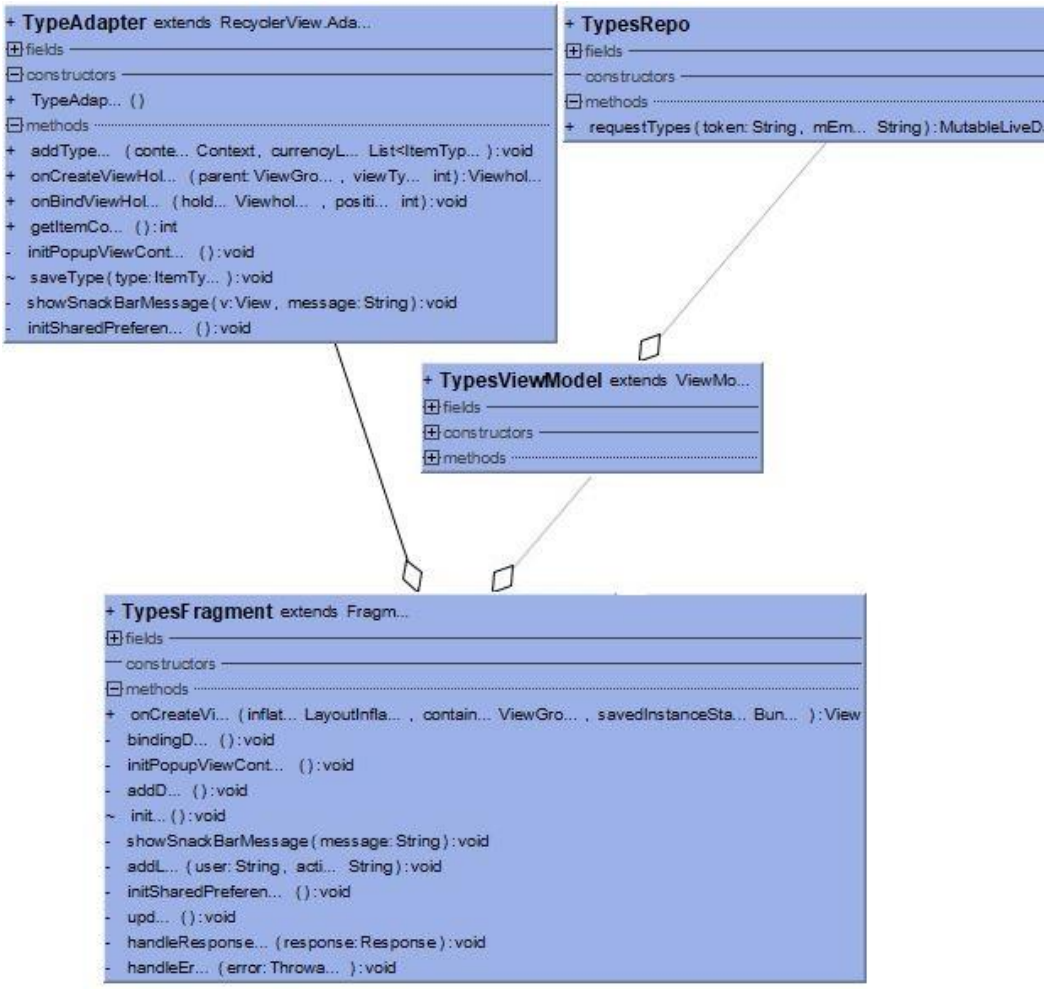

Figure 6. Class diagram for Type package

The Type package displays the list of type in inventory. The user can add, update and delete each type on the list.

The Item package is given in Figure 7.

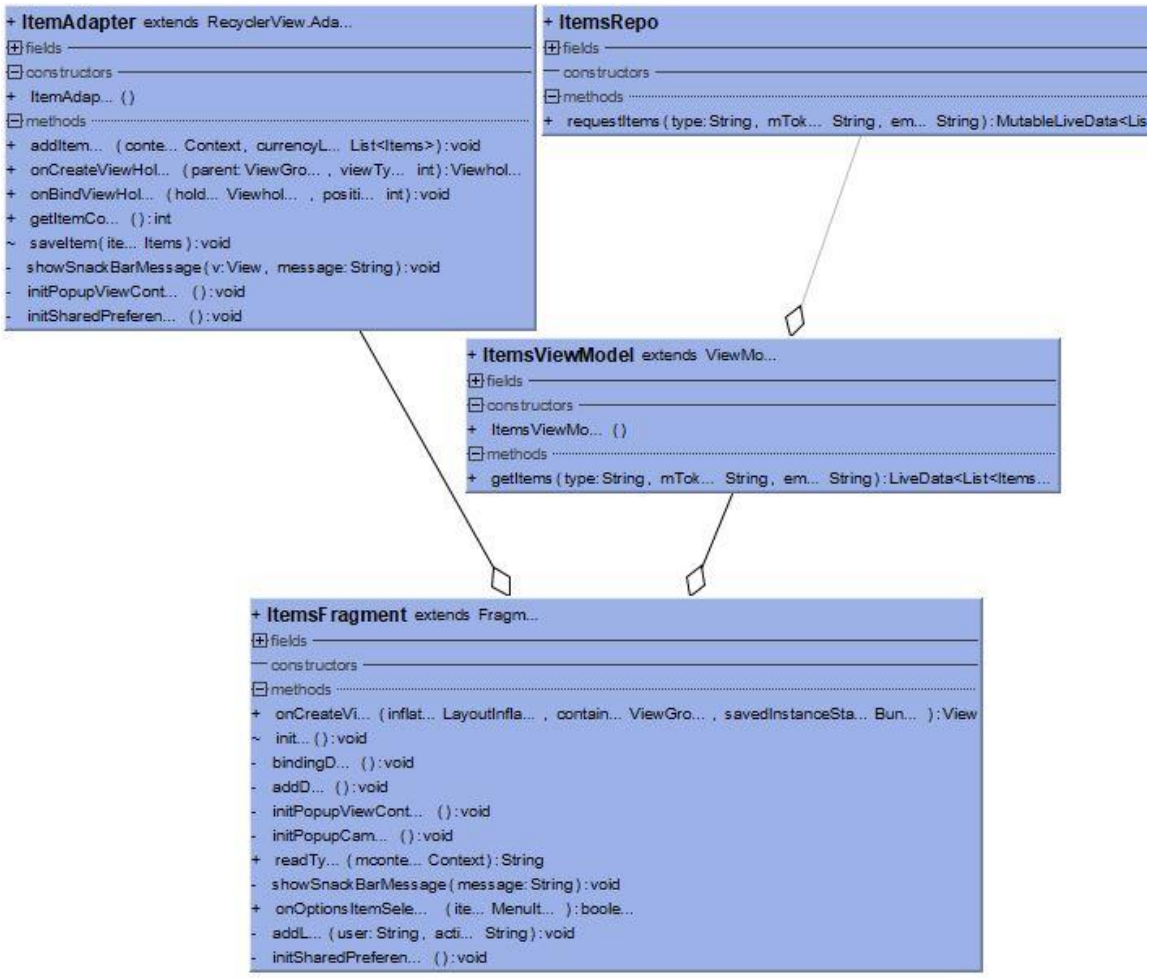

Figure 7. Class diagram for Item package

The Item package includes the listing items of a type. The user can add, update and delete each item on the list.

The Detail package is given in Figure 8.

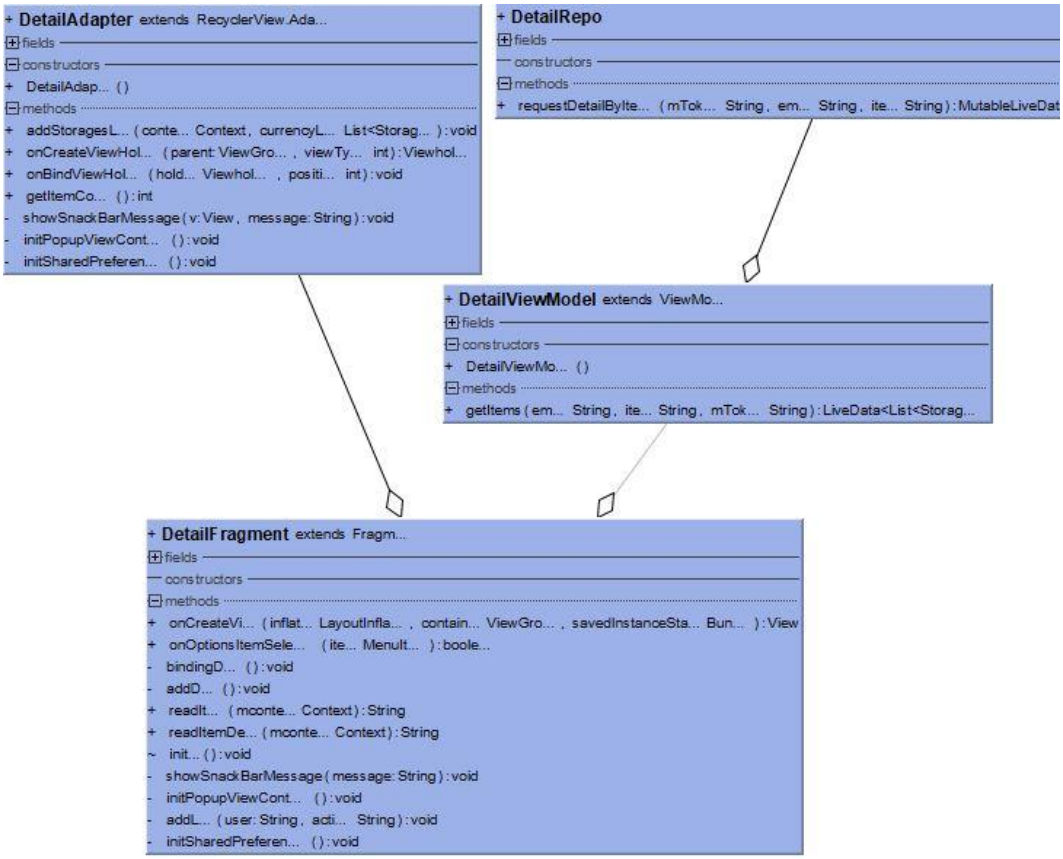

Figure 8. Class diagram for Detail package

Detail package: Listing detail information of in/out stock of each item. User can add, update and delete each detail on the list.

The Log package description is given in Figure 9.

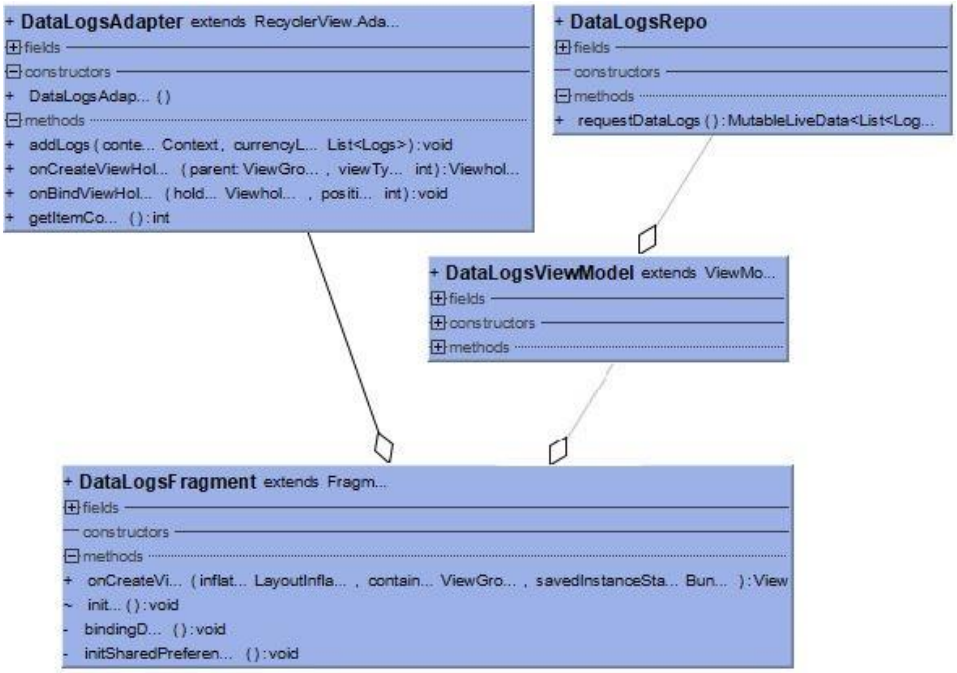

Figure 9. Class diagram for Data Log package

The Data log package Listing history of every change of data including user information, data change information, time of change.

3.4.3 Network Util

The NetworkUtil class is given in Figure 10.

Figure 10. NetworkUtil class diagram

There are getRetrofit() static methods with different parameters manages the process of receiving, sending, creating HTTP requests and response.

3.4.4 Retrofit Interface

The Introduce to “RetrofitInterface.class” is given in Figure 11.

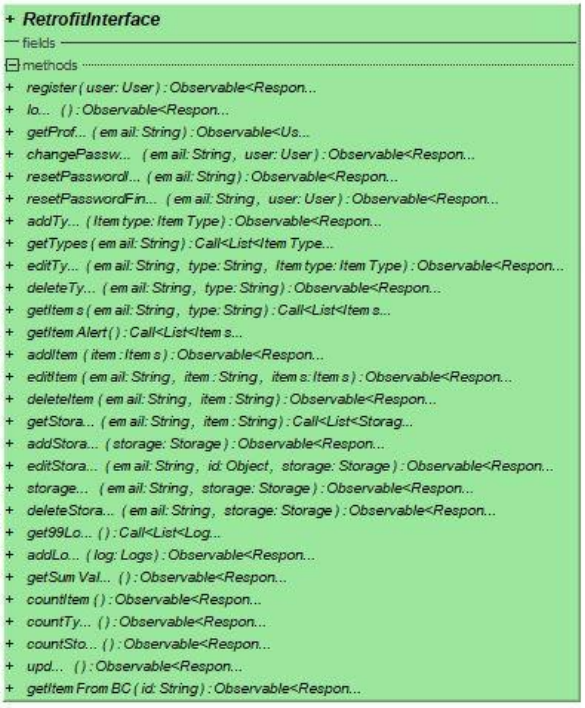

Figure 11. RetrofitInterface class diagram

“RetrofitInterface.class” contains the RESTful endpoints that will be called for subsequent operation. Every method returns a RxJava Observable, which can be subscribed to by an Observer.

3.5 Sequence Diagram

The Sequence diagram of the application is given in Figure 12.

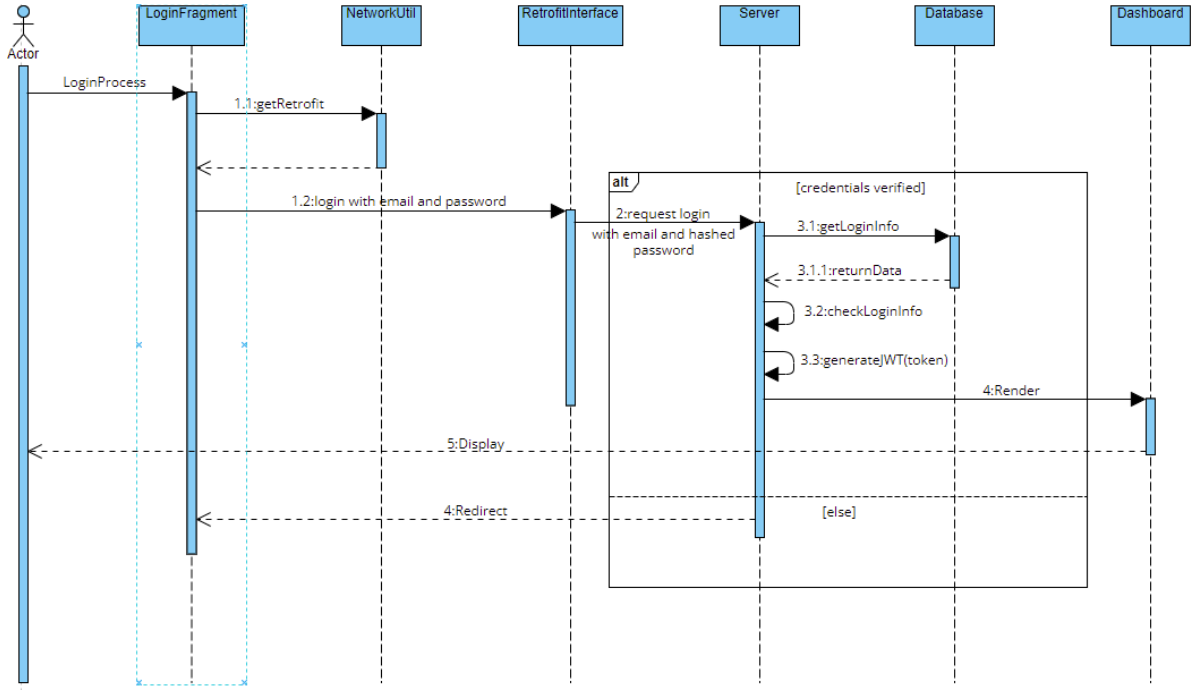

Figure 12. Sequence diagram

The sequence diagram in Figure 12 shows the interaction of different objects with each other in the process of using the application. At first, the user logs in into the application using email and password. The login parameters will be combined, encoded at the NetworkUtil class and then sent to the Restful endpoints at the RetrofitInterface class. The HTTP request is sent to the server for authentication. If the credentials are verified in the server, the client will receive login information including a JSON web token for handling the session. If the successful login leads to a redirect of application, the user can access the application features. If the verification process returns fail, the client will receive a notification and starts the login process from beginning.

3.6 Component Diagram

The Component diagram is given in Figure 13.

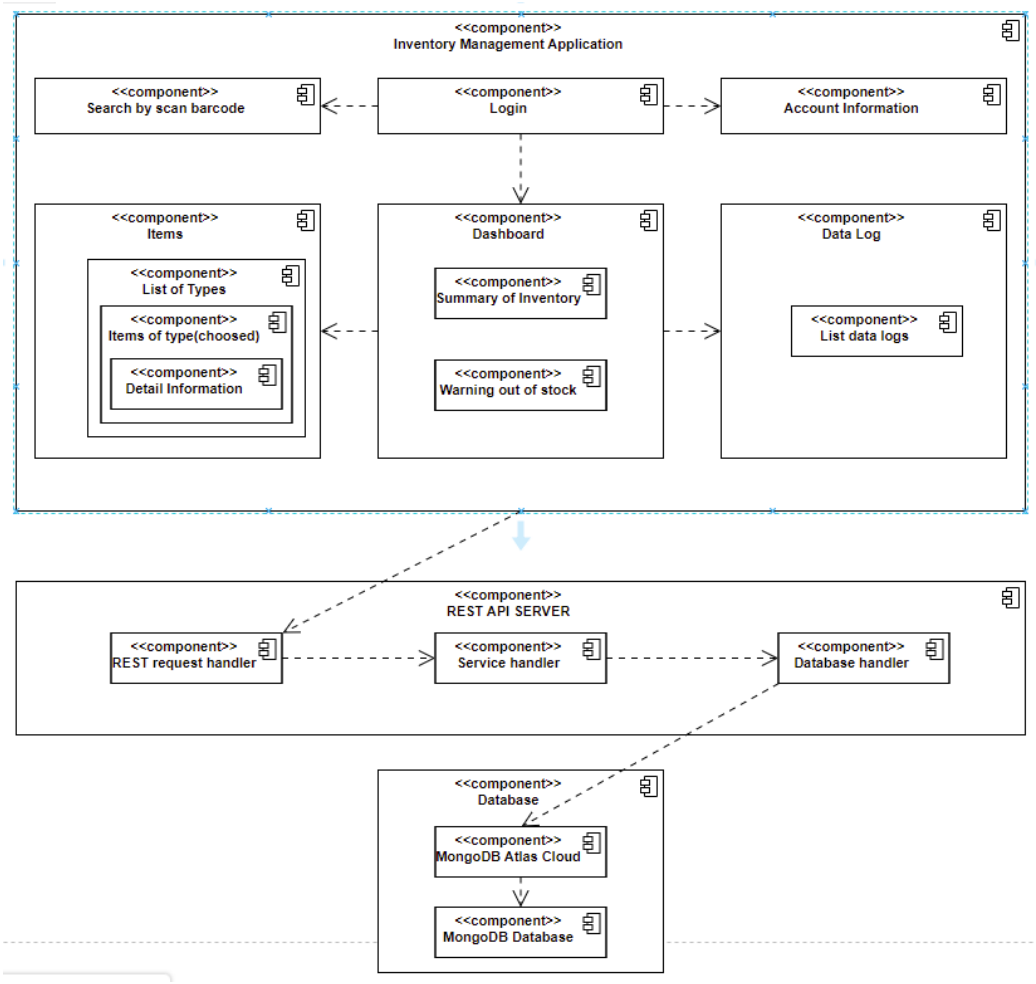

Figure 13. Component diagram

The component diagram in Figure 13 shows how components in this application are connected. This project includes three main parts: client Android application, REST API server, Data access.

The User Interface is an Android application. The REST API server handles requests sent by the client, communicates with the database, processes data and returns responses to the Android application.

3.7 Deployment Diagram

The Deployment of project is given in Figure 14.

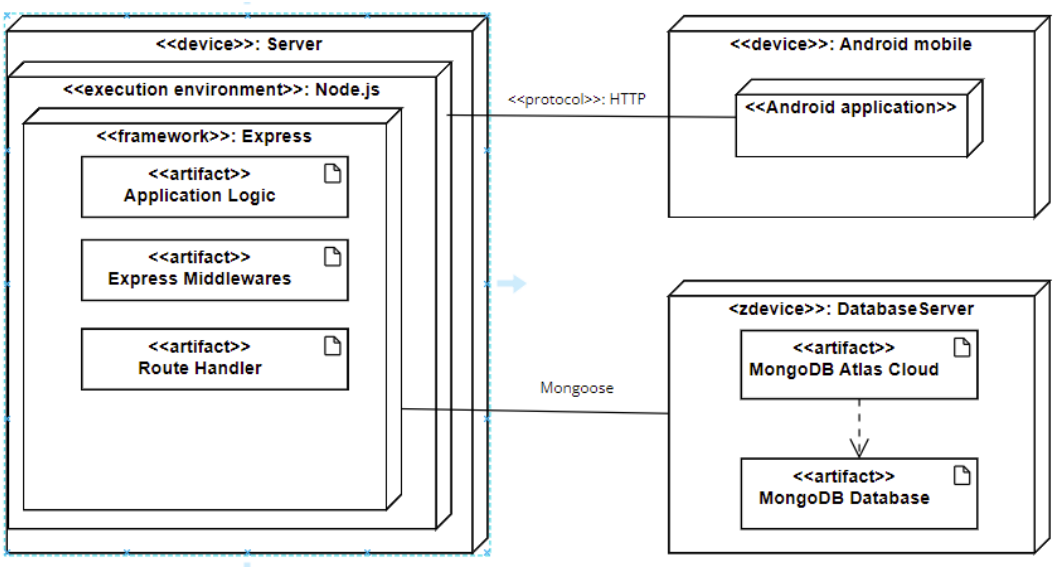

Figure 14. Deployment diagram

The deployment diagram in Figure 14 shows the execution architecture including hardware, execution environments, and the middleware connecting between them.

4 DATABASE AND GUI DESIGN

In this chapter, the design of database and GUI is introduced.

4.1 Database Design

This project stores and retrieves all data information from the MongoDB database. The server is written using Node.js and connects to MongoDB by using Mongoose. The database is a NoSQL database, it stores data records as documents which gathered together in collections. These documents are stored in the JSON (JavaScript Object Notation) format. Five data collections are used in this project including User, Type, Item, Storage and Logs collections. These data models are described in Figure 15 to Figure 19.

```
const userSchema = mongoose.Schema({
name                  : String,
email                 : String,
hashed_password       : String,
created_at            : String,
temp_password  : String,
temp_password_time: String,
type                  : Number
                   });
```

Code snippet 1. Snippet of User data model

This User model includes the user's login information.

```
const typeSchema = mongoose.Schema({
   type              : String,
   description       : String,
   maxStock          : Number,
   sumStorage        : Number
       });
```

Code snippet 2. Type data model

The Type model stores the type information.

```
const itemSchema = mongoose.Schema({
   item              : String,
   description       : String,
   barcode                 :  String,
   type          : String,
   maxStock          : Number,
   sumStorage        :Number
       });
```

Code snippet 3. Item data model

The Item model stores the item information.

```
const storageSchema = mongoose.Schema({
   item              : String,
   inAmount        : Number,
   outAmount       : Number,
```

```
    dateInStock       : String,
    dateExpiry        : String,
    price             : Number
});
```

Code snippet 4. Storage data model

The Storage model stores the detailed information of each time stock-in or stock-out.

```
const logsSchema = mongoose.Schema({
    user              : String,
    action                : String,
    time              : String,
    note          : String
});
```

Code snippet 5. Logs data model

The Logs model stores the information of each time user made change to database.

4.2 User interface design

The aim of this chapter is to introduce the user interface of the application.

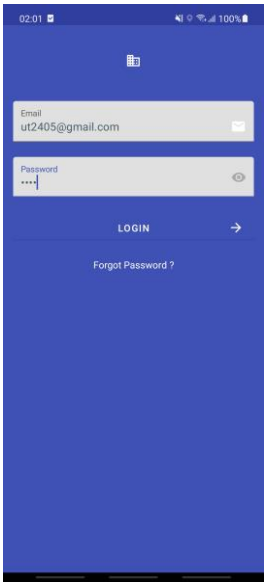

Figure 15. Login UI

Figure 15 shows the login page. The user can log in by email and password. After user logs in successfully, the user can access to the application features and their account information setting will be available on ‘My account’ page. The User can request for the reset of the password by clicking ‘Forgot password’.

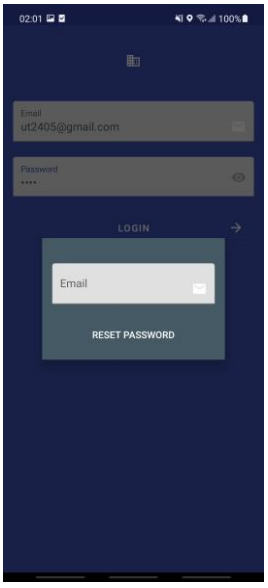

Figure 16. Reset Password UI

Figure 16 shows the reset password page. After the user sends a request to reset the password, the user will receive a token through email, and the user needs to confirm the token and the new password in order to complete the reset password process.

Figures 22, 23, 24 shows the account information page and setting functions.

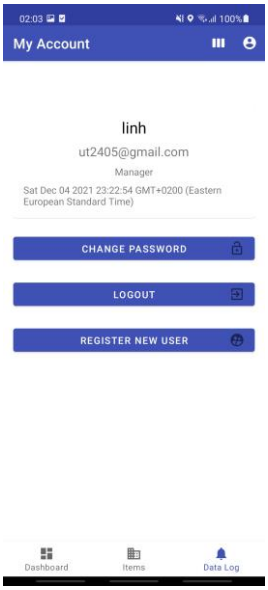

Figure 17. Account setting UI

My account page includes user information and three functions: change password, logout and register new user. Only the user account with the manager role has the "Register new user" function.

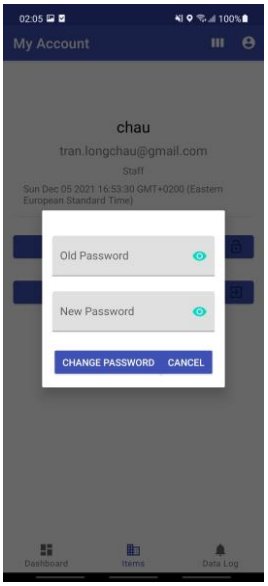

Figure 18. Change password UI

The user can request to change the password from the account page.

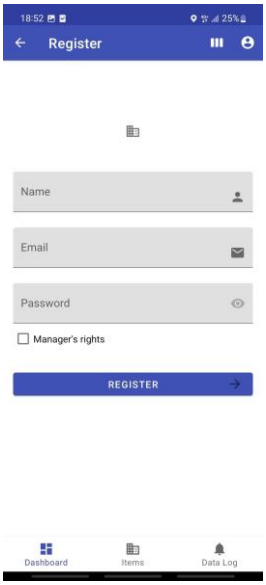

Figure 19. Register new user UI

When registering a new user, one checkbox needs to be selected to grant the account-role as manager. Otherwise, the new account will be granted as staff.

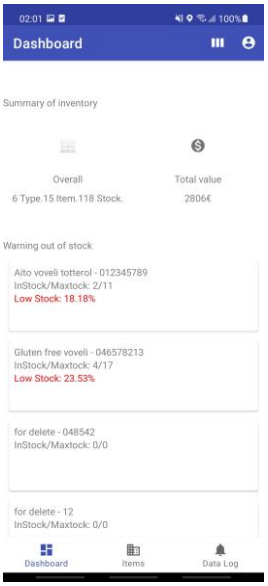

Figure 20. Dashboard UI

Figure 20 shows the dashboard page. The Dashboard page includes the overall information of inventory and a list of warning items.

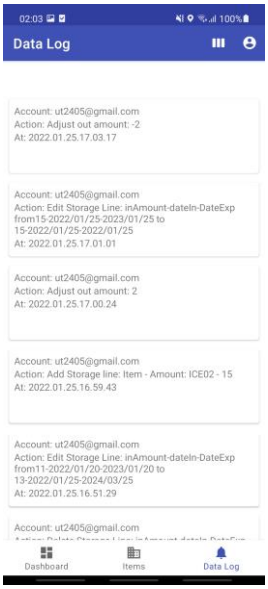

Figure 21. Data logs UI

Figure 21 shows the Data Log page. This page presents a list of histories data including user information, user action and date time of action. The list is sorted in the descending order by time.

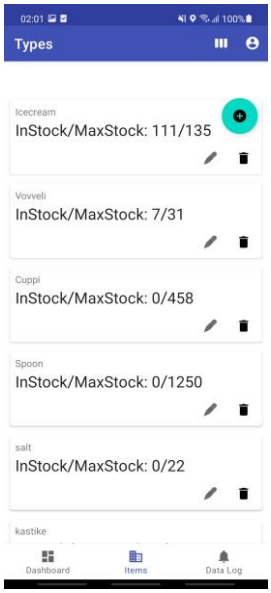

Figure 22. Type UI

Figure 22 show the Types page. This page work as a list of folders, each folder is a type of items. One type can have many items of that type inside. Each type in the list can be deleted and edited. The user also can add a new type. The index “Instock/Maxstock” presents the current quantity item of a type over maximum capacity of that type.

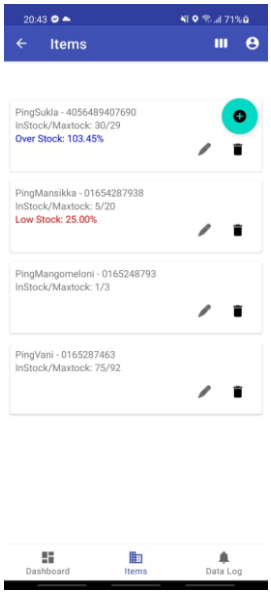

Figure 23. Item UI

Figure 23 shows the Item page. This page includes a list items of a type selected. Each item information can be edited and deleted. Each item has its own maxstock number. When the quantity of an item biger than its own maxstock, there will be a blue warning “Over Stock”. When the quantity of an item equal or smaller than quarter of maxStock, there will be a red warning “Low Stock”.

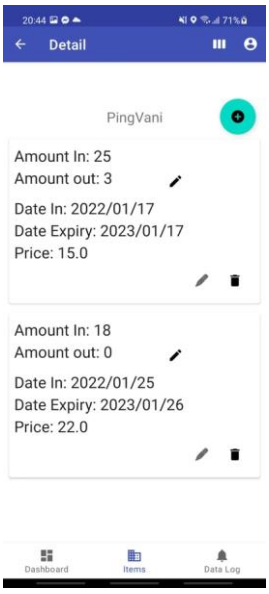

Figure 24. Storage UI

Figure 24 shows the storage page. This page presents the list of in and out stock. The list is sorted in the ascending order by the date of stock in(First in first out). When ‘amount out’ equals ‘amount in’, the tag will disapear.

Figure 25. Read - add new barcode UI

Figure 25 shows how the barcode is read. When the user adds new items, they can add a barcode by using the scanning function.

Figure 26. Search item by barcode UI

Figure 26 shows how to search detailed information of an item through scanning the barcode. The barcode will be detected and the application will redirect to the detail page of that item.

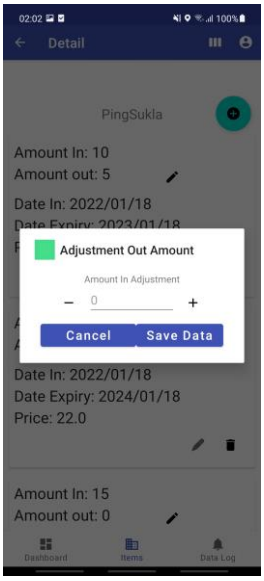

Figure 27. Adjustment amount UI

Figure 27 shows how the user can manage the amount when the stock goes.

5 IMPLEMENTATION

In this chapter, detailed implementation of this project is introduced. This chapter has two parts, front-end part and back-end part. Each part will explain its own structure, primary classes and methods.

5.1 Front-end

5.1.1 Structure

This chapter gives a general description about the front-end structure.

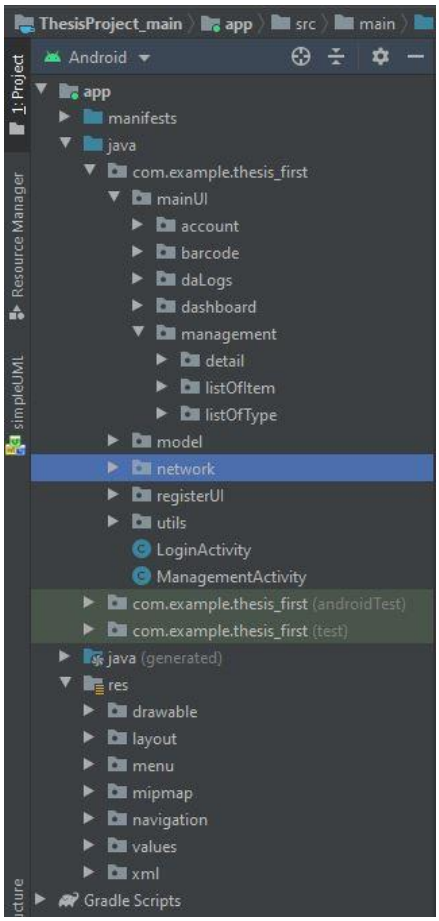

Figure 28. Front-end structure

As shown in the front-end structure tree in Figure 28, “ThesisProject_main” is the project directory, with “manifests”, “java”, “java(generated)”, “res” and “Gradle Scripts” in it.

5.1.2 Packages and Primary Classes

The main packages and special classes will be introduced respectively in this chapter.

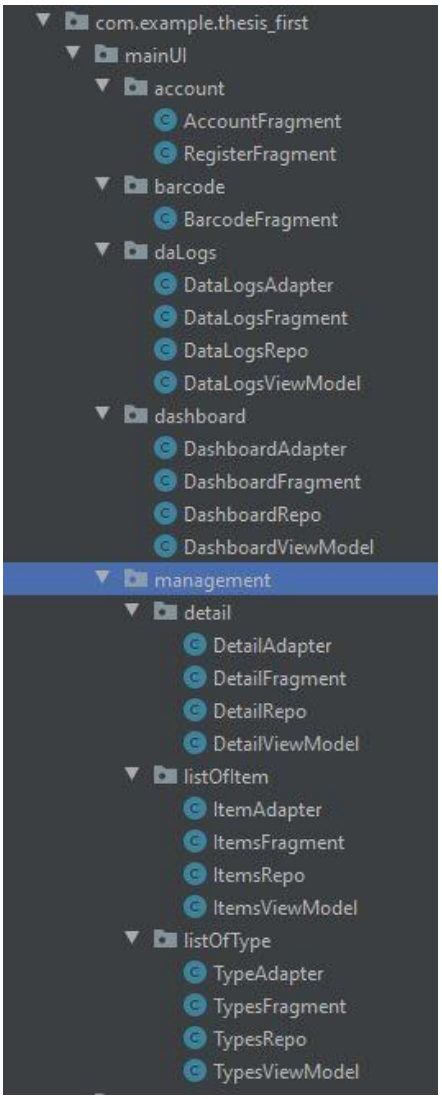

Figure 29. “mainUI” package

The package “mainUI” is a Java package that contains core Java classes, which are serve for client presentation logic and UI logic.

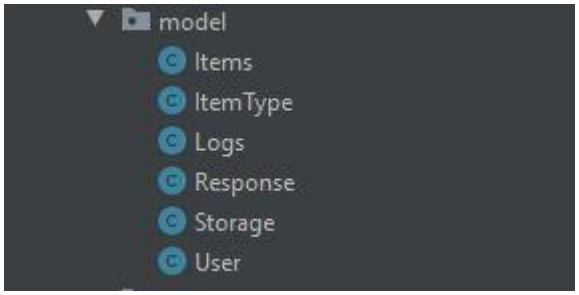

Figure 30. Model package

The “model” package includes six model classes for storing data. When the user ask for an information, the view goes to View Model, View Model notifies the model. Then the model gives that information to View Model and then notifies View about that information so the user can see that information.

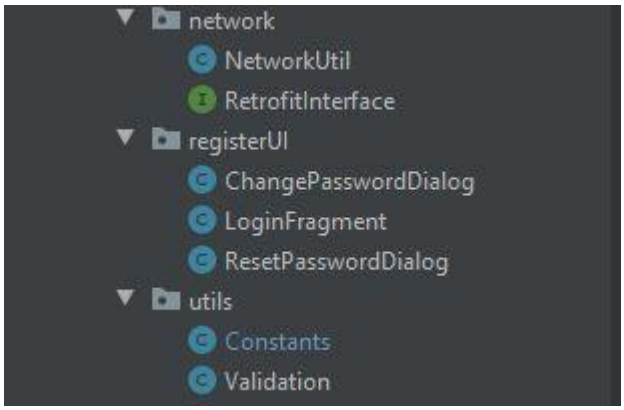

Figure 31. Network - utils - register packages

The Nestwork package handles communications between the Android application and the RESTful API server.

5.2 Back-end

The Structure of back-end and some of special functions will be introduced here.

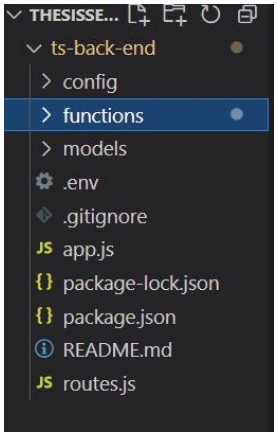

Figure 32. Back-end structure

Figure 32 shows the server structure of this project. “ts-back-end” is the server directory, with three main directories: “config”, “function”, “model”. “.env” defines the database connection string.

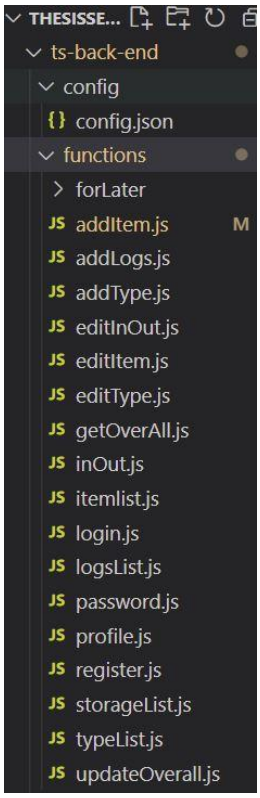

Figure 33. Functions directory

The Functions directory (Figure 33) stores the controllers of this server.

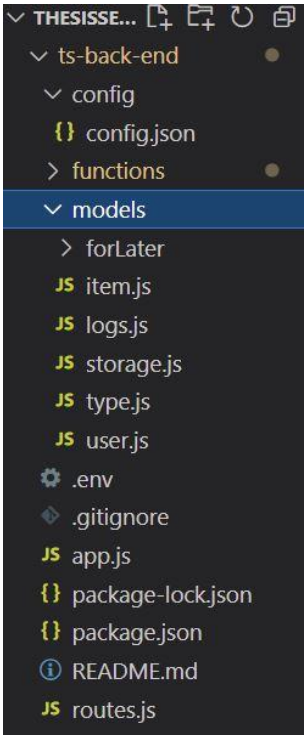

Figure 34. Model directory

The Model directory (Figure 34) stores five data models which are connected to the MongoDB database.

```
function checkToken(req) {
        const token = req.headers['x-access-token'];
        if (token) {
            try {
                var    decoded    =    jwt.verify(token,
config.secret);
                return decoded.message === req.params.id;
            } catch(err) {                return false; }
        } else {
            return false;
        }    }
```

Code snippet 6. Snippet of “checkToken function”

After a successful login, the user will receive a JSON web token (JWT). When the user sends a HTTP request data to the server, the HTTP request will be sent with a

JWT in it. The “checkToken” function is used to determine whether the HTTP request originated from a trusted client.

```
const transporter = nodemailer.createTransport(
        {
                host: 'smtp.office365.com',
                port: 587,
                auth: {
                    user: `${config.email}`,
                    pass: `${config.password}`
                }
            }
                );
            const mailOptions = {
                from:                   `"${config.name}"
<${config.email}>`,
                to: email,
                subject: 'Reset Password Request ',
                html: `Hello ${user.name},
                     Your reset password token is
<b>${random}</b>.
                     The token is valid for only 2
minutes.
                Thanks,
                Linh.`
            };
            return transporter.sendMail(mailOptions);
```

Code snippet 7. Snippet of “send email from server”

“Nodemailer” is a module for Node.js applications to allow email sending. This server using Office365 host: “smtp.office365.com”.

```
'use strict';
const express    = require('express');
const app        = express();
const bodyParser = require('body-parser');
const logger      = require('morgan');
const router      = express.Router();
const port        = process.env.PORT || 8080;
app.use(bodyParser.json());
app.use(logger('dev'));
require('./routes')(router);
app.use('/api/v1', router);
app.listen(port);
console.log(`App Runs on ${port}`);
```

Code snippet 8. Snippet of “app.js” file

“app.js” is the main file to start Node.js server. The router is set to “express” and every API endpoint begin with “api/v1”. The server will run on port 8080.

```
router.get('/items/:id/:type', (req,res) => {
        if (checkToken(req)) {
        itemsList.getItems(req.params.type)
              .then(result => res.json(result))
              .catch(err                                    =>
res.status(err.status).json({ message: err.message }));
        } else {
                res.status(401).json({ message: 'Invalid
              Token !' });
        }
  });
```

Code snippet 9. Snippet of an endpoint defined in “routes.js” file

Each RESTful endpoint is defined in the “routes.js” file. The snippet code in ‘Code snippet 9’ is an example of an endpoint and is used for getting all items which have

the “type” equal parameter type. The x-access-token will also be present in the header for this request, “checktoken” functions will be called and checked. If “checkToken” return “true”, the controller will be called to process the request, the response will be returned in the json format.

6 TESTING

This project had two testing phases. The first phase was conducted when the REST API was ready, the second phase when the whole project was ready. The first testing phase used Postman to test each API. The second testing phase used black box testing technique. Each part of the application was tested and corrected until all the requirements were met.

The controllers of the project were tested using correct parameters and incorrect parameters. The graphical user interface was tested mainly using Samsung Galaxy A7 and an Android emulator virtual device.

The sample testing cases for the second phase were:

- Login

  Expected Result: Check validation fields, successful sends an HTTP request, gets response from the server and returns the correct action.

  Result: Pass
- Dashboard

  Expected Result: If the user's role is the manager, sends an HTTP request to the server, gets response and returns a correct display information (summary information, the alert list).

  Result: Pass
- Account

  Expected Result: Returns correct user information, "change password" function and "log-out" function work correctly, "add new user" function only available if the user role is the manager.

  Result: Pass

- Search item by scanning barcode

Expected Result: Barcode is detected correctly by the mobile phone camera, sends an HTTP request for checking item information, gets response and displays a correct item that was searched.

Result: Pass

- Data log history

  Expected Result: Successful sends a request for a list data log, gets response and displays the data list correctly.

  Result: Pass

- Manage Item

  Expected Result: Works correctly according to the requirements.

  Result: Pass

7 CONCLUSIONS

With the aim to develop an inventory management application, the project was well implemented. The application was targeted for mobile devices running on the Android operating system. It allows users to login and access all features inside it. The database is stored on the MongoDB cloud, the server was deployed successful on “Heroku”, the application was converted to an “apk” format and can be installed on the Android enabled devices. The project was developed using the Java language and the JavaScript language. The application met 21 out of 22 requirements and works perfectly well.

The most challenging part of this project was keeping the code clean. Many parts work together and clean code is easier to maintain. Keeping the code clean saves a lot of time when returning to the previously written code and understanding what it does. Maintaining clean code has many advantages; however, it was the toughest part since it was challenging to maintain a high standard of code quality while juggling other aspects of life.

The application was first built with basic inventory management requirements but later can be upgraded with more advance requirements. In future, the application can be improved by adding additional functionalities, some of the future works are listed below:

- Re-order list using data analysis. For example, during the winter-time or summer-time the re-order list would be different, holidays and normal days the re-order list would be different, too.
- Manages gift-card information management.
- Exports reports and the manager’s mail every week, every month.
- Inventory management across multiple locations.

REFERENCES

MDN. 2022. JavaScript. Accessed 30.04.2022 https://developer.mozilla.org/en-US/docs/Web/JavaScript.

Kopecky, Christina. 2020. JvaScript versions. Accessed 14.1.2022 https://www.educative.io/blog/javascript-versions-history.

Node.js. 2022. Introduction to Node.js. Accessed 14.1.2022. https://nodejs.dev/learn/.

JWT. 2022. Introduction to JSON Web Tokens. Accessed 15.1.2022. https://jwt.io/introduction.

Arias, Dan. 2021. Understanding bcrypt. Accessed 15.1.2022. https://auth0.com/blog/hashing-in-action-understanding-bcrypt/.

MongoDB. Introduction to MongoDB Accessed 20.1.2022. https://www.mongodb.com/docs/manual/introduction/.

Postman. Introduction to Postman. Accessed 20.12.2021. https://learning.postman.com/docs/getting-started/introduction/.

Android Developers. Meet Android studio. Accessed 17.3.2022. https://developer.android.com/studio/intro.

Wikipedia. 2022. Java (programming language). Accessed 18.3.2022. https://en.wikipedia.org/wiki/Java_(programming_language)

Geeks for Geeks. 2021. MVVM (Model View ViewModel) Architecture Pattern in Android. Accessed 22.1.2022 https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/.

Manning. 2022. Building a data model with MongoDB and Mongoose. Accessed 10.5.2022. https://livebook.manning.com/book/getting-mean-with-mongo-express-angular-and-node-second-edition/chapter-5/15.

Patrzyk, Rycerz, Bubak. 2015. Towards a novel environment for simulation of quantum computing. Accessed 10.5.2022. https://www.researchgate.net/publication/275258051_Towards_A_Novel_Environment_For_Simulation_Of_Quantum_Computing

APPENDIX 1

```
public static RetrofitInterface getRetrofit(){
RxJavaCallAdapterFactory rxAdapter =
RxJavaCallAdapterFactory.createWithScheduler(Schedulers.
io());
return new Retrofit.Builder()
.baseUrl(Constants.BASE_URL)
.addCallAdapterFactory(rxAdapter)
.addConverterFactory(GsonConverterFactory.create())
.build().create(RetrofitInterface.class);
}
public static RetrofitInterface getRetrofit(String
email, String password) {
String credentials = email + ":" + password;
String basic = "Basic " +
Base64.encodeToString(credentials.getBytes(),Base64.NO_W
RAP);
OkHttpClient.Builder httpClient = new
OkHttpClient.Builder();
httpClient.addInterceptor(chain -> {
Request original = chain.request();
Request.Builder builder = original.newBuilder()
.addHeader("Authorization", basic)
.method(original.method(),original.body());
return  chain.proceed(builder.build());
});
RxJavaCallAdapterFactory rxAdapter =
RxJavaCallAdapterFactory.createWithScheduler(Schedulers.
io());
return new Retrofit.Builder()
.baseUrl(Constants.BASE_URL)
.client(httpClient.build())
.addCallAdapterFactory(rxAdapter)
.addConverterFactory(GsonConverterFactory.create())
.build().create(RetrofitInterface.class);
}
```

```
public static RetrofitInterface getRetrofit(String
token) {
OkHttpClient.Builder httpClient = new
OkHttpClient.Builder();
httpClient.addInterceptor(chain -> {
Request original = chain.request();
Request.Builder builder = original.newBuilder()
.addHeader("x-access-token", token)
.method(original.method(),original.body());
return  chain.proceed(builder.build());
});
```

Code snippet 10. Snippet of “RetrofitInterface methods”

NetworkUtil class includes three main methods to send out network request to an API.

APPENDIX 2

```
public interface RetrofitInterface {
    @POST("users")
    Observable<Response>       register(@Body       User
user);//register new user

    @POST("authenticate")
    Observable<Response> login();//login

    @GET("users/{email}")
    Observable<User>   getProfile(@Path("email")   String
email);//return usr info

    @PUT("users/{email}")
    Observable<Response>    changePassword(@Path("email")
String email, @Body User user);//change password

    @POST("users/{email}/password")
    Observable<Response>
resetPasswordInit(@Path("email")  String  email);//reset
password, sent passcode through email
    @POST("users/{email}/password")
    Observable<Response>
resetPasswordFinish(@Path("email")  String  email,  @Body
User user);//
    @POST("types")
    Observable<Response>      addType(@Body      ItemType
Itemtype);//add new type
    @GET("types/all/{email}")
    Call<List<ItemType>>  getTypes(@Path("email")  String
email);//return all of type list
    @PUT("types/{email}/edit/{type}")
    Observable<Response>  editType(@Path("email")  String
email,  @Path("type")  String  type,  @Body  ItemType
Itemtype);//save edit type

    @DELETE("types/{email}/del/{type}")
```

```
    Observable<Response> deleteType(@Path("email") String email, @Path("type") String type);//delete type condiction: no item in this type

    @GET("items/{email}/{type}")
    Call<List<Items>> getItems(@Path("email") String email, @Path("type") String type);//get item list of type parameter

    @GET("item/alert")
    Call<List<Items>> getItemAlert();//get item list those with low stock: less than 30% of maxStock

    @POST("items")
    Observable<Response> addItem(@Body Items item);//add new item

    @PUT("items/{email}/edit/{item}")
    Observable<Response> editItem(@Path("email") String email, @Path("item") String item, @Body Items items);//edit item

    @DELETE("items/{email}/del/{item}")
    Observable<Response> deleteItem(@Path("email") String email, @Path("item") String item);//delete item

    @GET("storages/{email}/byItem/{item}")
    Call<List<Storage>> getStorage(@Path("email") String email, @Path("item") String item);//return storage lines of item parameter

    @POST("storages")
    Observable<Response> addStorage(@Body Storage storage);//add new storage

    @PUT("storages/{email}/edit/{id}")
```

```
    Observable<Response>      editStorage(@Path("email")
String email, @Path("id") Object id, @Body Storage
storage);//edit storage line

    @PUT("storages/{email}/adjustment")
    Observable<Response> storageAdj(@Path("email") String
email, @Body Storage storage);//adjustment the amount of
out stock
    @HTTP(method      =      "DELETE",      path      =
"storages/{email}/del", hasBody = true)
    Observable<Response>    deleteStorage(@Path("email")
String email, @Body Storage storage);//delete stock line
    @GET("logs/90f")
    Call<List<Logs>> get99Logs();//return list of data
logs, just 90line nearest
    @HTTP(method = "POST", path = "logs", hasBody = true)
    Observable<Response> addLogs(@Body Logs log);//add
data log line, when add/update/delete
    @GET("overAll")
    Observable<Response>  getSumValue();//return  total
value of current stock
    @GET("countItem")
    Observable<Response> countItem();//count Item
    @GET("countType")
    Observable<Response> countType();//count type
    @GET("countStock")
    Observable<Response> countStock();//sum off current
stock{sum(stockIn-stockOut)}
    @GET("update")
    Observable<Response> update();//count type
    @GET("getItemOfBC/{id}")
    Observable<Items>  getItemFromBC(@Path("id")  String
id);//count type
```

Code snippet 11. Snippet of “Interface defines endpoints”

The “Interface defines endpoints” defines each endpoint that specifies an annotation of the HTTP method (GET, POST, PUT, DELETE).

APPENDIX 3

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 29
    defaultConfig {
        applicationId "com.example.thesis_first"
        minSdkVersion 28
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildTypes {
        release {            minifyEnabled false
            proguardFiles
getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    buildFeatures{
        dataBinding true
        viewBinding true
    }
}
dependencies {
    implementation   fileTree(dir:   "libs",   include:
["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation
'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation
'com.squareup.retrofit2:retrofit:2.6.0'
```

```
    implementation 'com.google.code.gson:gson:2.6.1'
    implementation    'com.squareup.retrofit2:converter-
gson:2.6.0'
    implementation    'com.squareup.retrofit2:converter-
scalars:2.6.0'
    implementation      'com.squareup.retrofit2:adapter-
rxjava:2.1.0'
    implementation         'com.android.support:support-
annotations:28.0.0'
    implementation 'io.reactivex:rxjava:1.2.0'
    implementation 'io.reactivex:rxandroid:1.2.1'
    implementation        'androidx.lifecycle:lifecycle-
extensions:2.2.0'
    implementation      'androidx.navigation:navigation-
runtime:2.3.3'
    implementation        'com.android.support:cardview-
v7:28.0.0'
    testImplementation 'junit:junit:4.12'
    implementation
'com.google.android.material:material:1.3.0-alpha01'
    implementation 'androidx.core:core-ktx:1.2.0'
    androidTestImplementation
'androidx.test.ext:junit:1.1.2'
    androidTestImplementation
'androidx.test.espresso:espresso-core:3.3.0'
    implementation      'androidx.navigation:navigation-
ui:2.3.0'
    implementation      "androidx.navigation:navigation-
fragment:2.3.0"
    implementation
"androidx.recyclerview:recyclerview:1.1.0"
    implementation         'com.google.android.gms:play-
services-vision:11.0.2'
}
```

Code snippet 12. Snippet of “build.gradle”

Build.gradle file defines dependencies that apply to all modules in this project.

APPENDIX 4

```
{
  "name": "thesis",
  "version": "0.1.0",
  "main": "app.js",
  "dependencies": {
    "basic-auth": "^2.0.1",
    "bcryptjs": "^2.3.0",
    "body-parser": "^1.15.2",
    "dotenv": "^8.2.0",
    "express": "^4.17.3",
    "git-init": "^1.0.0",
    "jsonwebtoken": "^8.5.1",
    "mongodb": "^3.6.5",
    "mongoose": "^6.3.1",
    "morgan": "^1.7.0",
    "nodemailer": "^6.7.2",
    "nodemailer-smtp-transport": "^2.7.4",
    "nodemon": "^2.0.7",
    "randomstring": "^1.1.5"
  },
  "description": "This is my server site!",
  "devDependencies": {},
  "scripts": {
    "start": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
```

```
    "thesis"
  ],
```

Code snippet 13. Snippet of “package.json”

The “package.json” defines metadata relevant to the project and it is used for managing the project's dependencies, scripts, version.