“SuppleMate” App Maintenance Manual

Team:

- Gregy Thomas Kokkaparampil – Project Manager
- Shuo Li– Assistant Project Manager/ Software Engineer
- Siddharth Santhanakrishnan – Software Engineer
- Wone Eui (Ella) Jung – Software Engineer

Table of Contents

1. File Structures

The project’s file structure is managed using IntelliJ for local and Github for version control repository. All of our files are saved under the Project Documentation root directory “src”.

Source code is specifically saved under Github, where each module has its own directory where relevant code and updates can be stored. The Google Drive and Github files are accessible to all team members, where each member can upload, download, and view files. Each newest update is saved in its own folder with the date and brief description.

<u>Controller package:</u>

SuppleMateController.java:

Most of the main REST APIs go here.

- **"/login" -** Login instruction "Please enter your user name and password to login."
- **“/home” -** "Welcome to SuppleMate Home Page."
- **“/addSupplementTypes/{customerId}” -** Add supplement types futures based on customer ID
- **“/dayLogs/{customerId}” -** Add day logs based on customer ID
- **“/notes/{customerId}/{date}” -** Add note based on customer ID and date
- **“/dates/{customerId}" -** Get all log dates for a user
- **“/supplementTypes/{customerId}” -** Get all supplement types for a customer
- **“/supplementEntries/{customerId}” -** Get all supplement entries for a user
- **“/supplementEntries/{customerId}/{date}” -** Get all entries for a user by date
- **“/updateLog/{customerId}” -** Get updated log based on customer id
- **“/thank-you” -** Displays thank you in the page

AdminController. java:

Admin Role specific page contains the following API.

- **"/admin" -** Returns string admin (currently we do not have front-end)

RegistrationController.java:
All customers sign up, account creation, login authentication related information including password encryption protocol logic goes here.

- **“/signup” -** For customers signing up or registering with our application.
- **"/addCustomer" -** Create new customer account (ID, name, email, creation time, time zone) to the service based on details entered by the user
- Password encryption and there are set criterias for password

ExceptionHandlerController.java:
Whenever there is an exception the corresponding function for each case code in the exceptions package’s code responds and exceptions are being handled using the exception handler in this controller.

Contentcontroller.java:
This controller is used to handle content displayed to customer as well as some basic CRUD operations as mentioned below.

- **“/content” -** When any content that needs to be generated to fetch data from a database for a particular customer and if the customer account is not valid then returns showing that there is some incorrect data. This handles the content of the user.
- **“/addSupplement” -** handles content displayed to customer when adding a supplement
- **“/removeSupplement” -** handle content displayed to customer when removing a supplement

- **"/deleteSupplementType" -** handle content displayed to customer when a customer is deleting a supplement type from their catalog
- **"/deleteConfirmation"** - When a customer is trying to get supplement related data from a database and trying to delete that information then check if delete is correct.

<u>Exceptions Package:</u>
This package contains all java class files related to type of exceptions thrown at runtime such as follows and performs simple exception throwing operations exactly as file name suggests.
InvalidEmailExeption.java
InvalidEntryException.java
InvalidPasswordException.java
InvalidSupplementTypeException.java
InvalidUsernameException.java

<u>Model Package:</u>
Set up basic parameters and creation of each object including basic getters and setters for customer related object and members for customer, supplement entries(supplement dosage values, type, id, entry id, etc), supplement type(customer id, supplement type id, supplement name, scale, and unit), day log(getting day log detail with day log id and customer id, log timestamp, and note).

Customer.java:
Database related object classes such as customer information (Customer ID, first name, last name, email, username, password, create timestamp and customer time zone), user role(admin or customer). Get information and set and put into the database.

NewEntryInfo.java:
This class gives the ability for a user to enter new supplement related information.

UpdateEntryInfo:
This class gives the ability for a user to update existing supplement entry information.

Data Package:

This package contains all interfaces and classes related to the data layer. SQL querying and JDBC operations are done here such as INSERT Customer into the database using JDBC template across all Model Objects wherever implemented. Contains data access object files, database connection where our application connects to the database and database fetching activities such as updating database with latest information from customer using JDBC connection. Some sample api operations are POST, GET, PUT, DELETE customer, supplement entry, supplement type, day log, information, etc. All classes in this package use the @Repository annotation from Spring Framework to represent them as data classes.

CustomerDaoImpl.java:
Database operations are performed based on inputs from user, and are covered through customer data access objects.

DayLogDaoImpl.java:
Various database operations are performed for daylog, and are covered through daylog data access objects

RoleDaoImpl.java:

Various database operations are performed for roles, and are covered through role data access objects

SupplementEntryDaoImpl.java:
Various database operations are performed for Supplement entries made, and are covered through supplement data access objects

SupplementTypeDaoImpl.java:
Various database operations are performed for supplement types, and are covered through supplement-type data access objects

<u>Service Package:</u>
This package holds interfaces and implementation classes for every service. The actual logic is implemented here for all the listed services - add and delete service, lookup service, update service, validation service, user detail service, all work hand-in-hand, synchronously.

AddServiceImpi.java:
Contains all creation operation logic of every model and controller

DeleteService.java:
Delete supplement entry, log, supplement type, customer account

LookupServiceImpl.java:
Fetch data from the database, filter out exactly what the customer is looking for. Based on the given filer fetch the data and sort, collect and share with the user. By ID, date, etc.

UpdateServiceImlp.java:
Get a particular type of data to update. Populate supplement types with customers and update supplement type and day log the customer takes.

UserDetailsServiceImpl.java:
Grant authorities for the user based on the role, if the user is admin then has more authority then customer.

ValidateService.java:
Validate username, password, any account setting. When a user enters a username then check if the username already exists in the database or not so that we can inform the user that the username has been taken and not available to use. Validate password with what to add and what not to add including security protocols and password encoder.

__**Security Package:**__
This file has a strictly customized HTTP firewall and firewall response, password encoder and security web application initializer.
CorsConfig.java
CustomStrictHttpFirewall.java
FirewalledResponse.java
PasswordEncoder.java
SecurityWebApplicationInitializer.java

SuppleMateWebSecurityConfig.java:
Web security config that allows only certain http methods as mentioned in the code such as delete, get, post, put, etc.

__**Test Files:**__

There are three main test files corresponding to the service package.

- **AddServiceTest.java**
- **DeleteServiceTest.java**
- **LookupServiceTest.java**

Dependencies and Config files:

- **pom.xml -** This file contains all dependencies related to java spring framework, maven, external API integration dependency files.
- **application.properties -** This file contains all the application related properties such as the server to which we are connecting(currently /localhost:8080 port) and all database related properties which are being used.
- **README.txt -** In a world where an increasing number of people rely on supplements to meet their specific health and wellness needs, the SuppleMate app emerges as a comprehensive solution. This project aims to address the diverse supplement requirements of individuals, offering a user-friendly platform to monitor, access, and optimize their supplement intake.

2. Instructions

The application can be run using on of the following options listed below:

- Local Server
- Emulating it on a Windows Computer

To run the application through any of the above methods:

- Use a laptop or desktop computer that is connected to the internet
- Use Google Chrome to access or emulate the application.

To access functionalities, user can log in with this account:

- Email: User1@gmail.com
- Password: 1234

Run the application through a URL generated and hosted by local host, the following steps must be taken:

- 1. Use a laptop or desktop computer that is connected to the internet.
- 2. Actively run Google Chrome (other browsers will not work).
- 3. Type the URL: https://supple-mate.web.app/index.html into the address bar.
- 4. Press enter

3. Good Implementation

Positive implementations of the application:

- The application has a Figma prototype, which shows the application is pleasant in appearance.
- The app has a simple layout and clear logic on each page.
- The application contains error handling that should prevent unexpected crashes/undesired output.
- The application has functional code for customer registration, log in, daily log for users to add notes, security protocols, setted database and use bigquery for easy data sharing through Google API.

4. Needs Improvements

The following are weak points of the current implementation:

- The application does not have a user interface to work on a website yet.
- The application is not mobile compatible.
- Initially mySQL was planned to use, however due to data sharing issues we used bigquery.

5. Overall Recommendations

If a future development team were to work on our application, we would recommend:

- Develop a functional user interface with websites and mobile devices.
- Making the database functional with FDA data to be useful for dosage recommendation.
- Making the application compatible on different browsers and mobile devices.