Pill Reminder Application

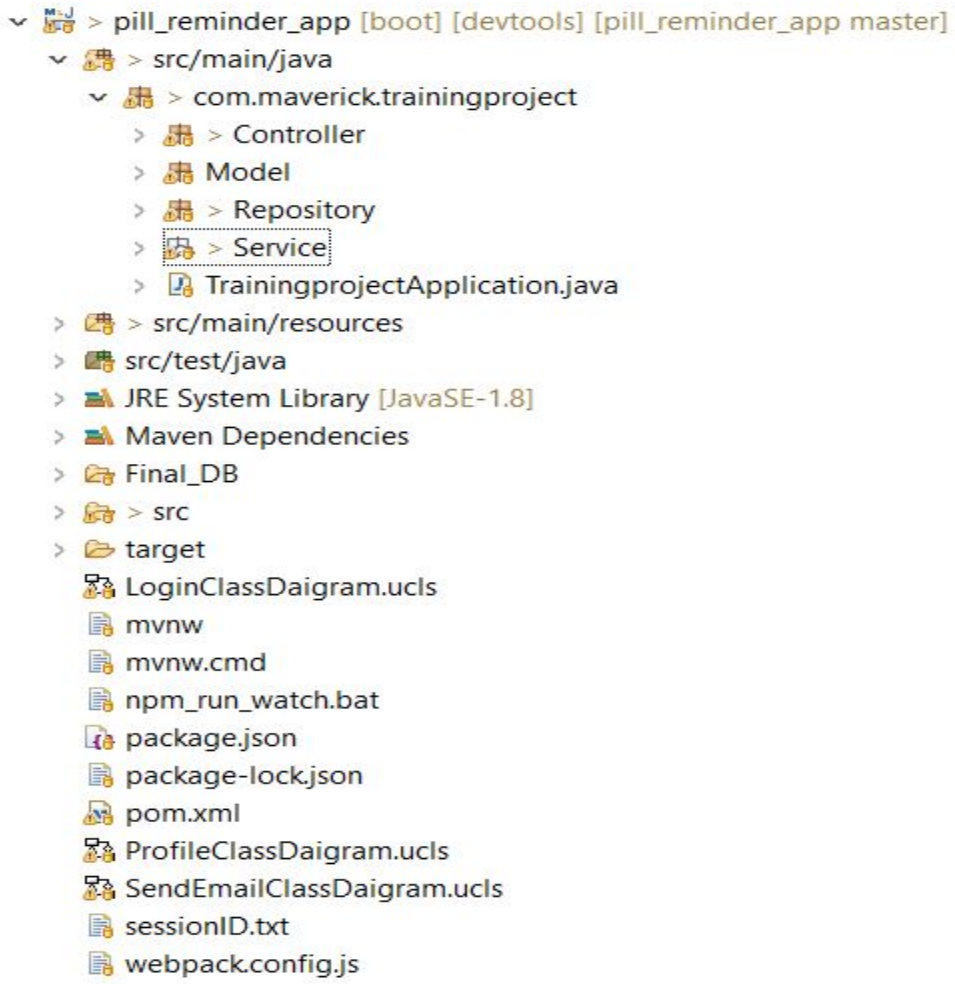
Tree Structure of the Project.

Server-Side:

1. **Controller class :**
    a. Controller class will deal with Rest Api.
    b. It will receive the request from the Client-side.
    c. Pass the request to Service class and receive a response from Service Class.
    d. And then pass the Server response to the Client-side.
2. **Service class:**
    a. The Service class will contain the Business Logic.
    b. It will send and receive data to the Repository class.
3. **Repository class :**
    a. The Repository class will contain the Business logic related to deal with DataBase.
    b. It will take the request of the Service class and give the response of the Database.
4. **Model class :**
    a. The Model class will represent the entities required in the project.

Client-Side:

1. **src/main/resources/static :**
    a. It will contain the static HTML pages (login.html).
2. **src/main/js :**
    a. It will contain dynamic js pages, which send the Rest Api request and receive the response from The Server-side.



Modules

1. Login User, Registration and Forgot Password:

a. User Login :

i. Login Request will receive by LoginController.java class, Login user data will be stored in UserRegistrationInformationModel.java class and then it will send to LoginService.java class

ii. The LoginService.java class will encrypt the password and send data to LoginRepository.java class.

iii. The LoginRepository.java class will trigger the SQL query to check the user exit in the database.

iv. Database collection will be from DaseBaseConnection.java class.

v. The response will then be generated in LoginService.java class, and LoginController.java class will send the response to the Client-side.

b. User Registration :

i. The registration request will be received by LoginController.java class, then the registration data will be stored in UserRegistrationInformationModel.java class, and then it will send to UserRegistrationService.java class.
ii. The UserRegistrationService.java class will encrypt the password and do the server validation in UserDataValidation.java class, and then the data will be sent to LoginRepository.java class.
iii. The LoginRepository.java class will trigger the SQL query to insert the user data into the database.
iv. Database collection will be from DaseBaseConnection.java class.
v. The response will then be generated in UserRegistrationService.java class, and LoginController.java class will send the response to the Client-side.

c. Forgot Password :

i. The forgot password request will be received by LoginController.java class.
ii. The new password will be encrypted in PasswordEncrytion.java class, Then the forgot password data would be stored in UserRegistrationInformationModel.java class, and then it will send to ForgotPasswordService.java class.
iii. The ForgotPasswordService.java class will send the data to LoginRepository.java class.
iv. The LoginRepository.java class will trigger the SQL query to update the user data into the database.
v. Database collection will be from DaseBaseConnection.java class.
vi. The response will then be generated in ForgotPasswordService.java class, and LoginController.java class will send the response to the Client-side.

2. Home and Menu :

a. Menu sidebar:

i. A side navigation bar provides ease of access to the user to navigate between pages of a web application. The static side navigation bar contains hyperlinks to 3 pages of this web application, ‘Home’, ‘Profile’ and ‘Medical History’.
ii. Upon clicking, the router in the onload.js will check for the url, the hyperlink wants to access and from the frontend itself, a related page will be rendered on the user’s screen dynamically, the sidebar being static.
iii. This sidebar is also collapsible.

b. Home Page:

i. Upon loading, an initial call will be made to the api to render the user profile picture. If not available, a default avatar will be displayed to the user.
ii. Along with a profile picture, User's name, personal information is rendered on the homepage.
iii. The home page consists of cards which displays the pills of a user and a dependent (if added). Only those pills which are scheduled for the current date are displayed on the homepage.
iv. The cards also have a functionality of marking the pill as 'taken' using a checkbox. It will show whether a pill has been consumed or not depending upon the user updation timings.
v. A logout button along with a profile picture is rendered on the top right section. A secure logout button enables the user to logout of the app and exit his existence from the session.

3. Profile :

a. View Profile Data, Update Profile Data (User , Dependent):

i. The profile request will be received by UserProfileController.java class.
ii. The data would be stored in UserProfileModel.java class, and then it will be sent to UserProfileService.java class.
iii. The UserProfileService.java class will send the data to UserProfileRepositoryImplementation.java class.
iv. The UserProfileRepositoryImplementation.java class will trigger the SQL query to select, update the user data or user-dependent data into the database.
v. The methods to select or update data are declared in the UserProfileRepository.java interface.
vi. Database collection will be from DaseBaseConnection.java class.
vii. The response will then be generated in UserProfileService.java class, and the UserProfileController.java class will send the response to the Client-side.

4. Medical History :

a. Display Medical History

1. On clicking of medical history the user medical history will be displayed using data provided by the display data Api call .
2. The MedicalHistoryController.java class receives the request for the medical data.

3. Then using the user Email data related to user is fetched from the database in the medicalHistoryRepository.java class returned to the client-side in the JSON array form.
4. The data is sorted according to enddate and if incase enddate are same then, inside the same enddate date we are sorting it by medicine name .
5. Medical data is displayed in table format on the web app .

b. Add Medical History Data (User and Dependent)

1. User can add medical history data for himself and his dependent using add Data service.
2. User will add data in the row which is validated on both client and server side .
3. The data is then sent to server which is collected and inserted into the database. Through (MedicalHistoryController.java, MedicalHistoryAddService.java, and MedicalHistoryRepository.java) classes.
4. Once data inserted successfully then, the display Api is called to re-render the table displaying the information on the client side.

c. Update Medical History Data(User and Dependent)

1. For updating the existing table , user needs to double click on the row cell which he needs to update.
2. Once he has made the changes , all the cells are validated and then an API call is made .
3. At server-side data is updated using the unique ID for that particular medical History in the database after validation at the server-side.
4. Once the data updation is done successfully then, the data in table is re-rendered at the client side using display Api call. Thus it is Providing real time user experience.
5. Using (MedicalHistoryController.java, MedicalHistoryUpdateService.java, and MedicalHistoryRepository.java) classes.

5. Jwt Token :

a. Generating Json web token

i. When the login API is called using the POST request, the request is first checked for an ‘Authorization’ header.

ii. This authorization header contains a token containing information about the type of token, its hashing algorithm, and the payload data, i.e, the data it is containing
iii. If the request already has a token, it will be checked for its validity. Else a token generation function will be called, which will check whether the user is present in the database or not.
iv. If the user is valid, a token will be generated using a secret key stored at the server, and it will be passed to the user, which will be saved at the client's local storage.

b. Validating the Json web token

i. This token will then be sent along with every api request and it will be checked for its validity as well as authenticity using the server side secret key. If true then and then only the request will be processed and relevant data will be displayed as stated in other parts of this documentation.
ii. A server sided secret key is useful to identify key tampering and avoid unauthorized access to restricted content.

c. Spring Security

i. Spring security is set to allow all api calls outside the web application including but not limited to the login page, register and forgot password page.
ii. All the CSS, JS and Image files are also excluded from the spring security.

6. Send Email :

a. Sending email to users on registration.

i. The registration request will be received by LoginController.java class. Then the registration data will be stored in UserRegistrationInformationModel.java class.
ii. Here we will call the SendmailService.java class to send a registration email to the user.
iii. And then, the registration data will be sent to UserRegistrationService.java class for further use.

b. Sending email to remind user's or dependent's to take their pills.

i. The SendEmailRepository.java class will trigger the SQL select query to fetch the user's or dependent's data from the database.
ii. The database connection will be from DataBaseConnection.java class.
iii. The fetch data will be stored in SendEmailModel.java class.

iv. Then the SendEmailService.java class is called to send email to users and dependents by javaMailSender interface.