Issue Tracker

Objective:

Issue Tracker is an online application to be built as a product that provide Issue tracking for projects.

Users of the System:

1. Admin
2. Developers
3. Guest

Functional Requirements:

- Individual accounts for Developers.
- Ticket creation and updation.
- Assigning or UN-assigning a ticket to Developer by higher authorities or by themselves.
- Uploading patch files or any other required files after solving the issue and update the ticket status
- Export a ticket in different formats like doc and pdf.
- **An Developer can manage a maximum of 5 complaints per day.**

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

- ➢ Watch service for subscribing a ticket.
- ➢ Vote for ticket.
- ➢ Share ticket through mail

Output/ Post Condition:

- ➢ Daily Tickets Reports
- ➢ Daily Solved tickets Reports
- ➢ Monthly Tickets Reports

Non-Functional Requirements:

Security	• App Platform –UserName/Password-Based Credentials • Sensitive data has to be categorized and stored in a secure manner • Secure connection for transmission of any data
Performance	• Peak Load Performance • Issue Tracker -< 3 Sec • Admin application < 2 Sec • Non Peak Load Performance
Availability	• 99.99 % Availability
Standard Features	• Scalability • Maintainability • Usability

	• Availability • Failover
Logging & Auditing	• The system should support logging(app/web/DB) & auditing at all levels
Monitoring	• Should be able to monitor via as-is enterprise monitoring tools
Cloud	• The Solution should be made Cloud-ready and should have a minimum impact when moving away to Cloud infrastructure
Browser Compatible	• IE 7+ • Mozilla Firefox Latest – 15 • Google Chrome Latest – 20 • Mobile Ready

Technology Stack

Front End	React Google Material Design Bootstrap / Bulma
Server Side	Spring Boot Spring Web (Rest Controller) Spring Security Spring AOP Spring Hibernate
Core Platform	OpenJDK 11
Database	MySQL or H2

Platform Pre-requisites (Do's and Don'ts):

1. The React app should run in port 8081. Do not run the React app in the port: 3000.
2. Spring boot app should run in port 8080.

Key points to remember:

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints given below.

Application assumptions:

1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by using AuthGaurd by implementing the canActivate interface. For example, if the user enters as http://localhost:3000/signup or http://localhost:3000/home the page should not navigate to the corresponding page instead it should redirect to the login page.
3. Unless logged into the system, the user cannot navigate to any other pages.
4. Logging out must again redirect to the login page.
5. To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
6. Use admin/admin as the username and password to navigate to the admin dashboard.

Validations:

1. Basic email validation should be performed.
2. Basic mobile validation should be performed.

Project Tasks:

API Endpoints:

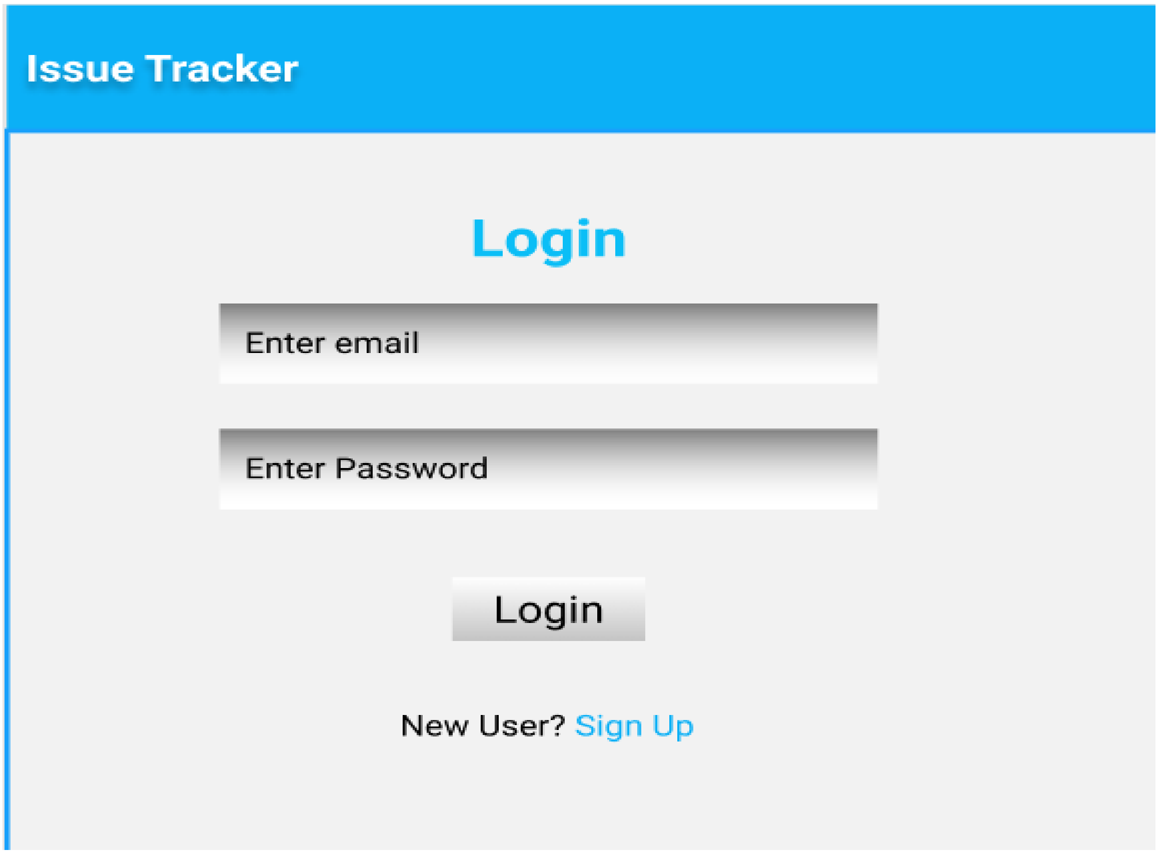
USER			
Action	URL	Method	Response
Login	/login	POST	true/false
Add Issue	/addIssue	POST	Issue added
List logged in user issue	/issue/{id}	GET	Array of Issue
Update Issue	/issue/{id}	PUT	Issue Updated.
Update Status	/status/{id}	PUT	Status Updated.
ADMIN			
Action	URL	Method	Response
Get All Issue	/admin	GET	Array of Issue
Add Developers	/admin/addDevelopers	POST	Developer added
Update Developer	/admin/updateDeveloper/{id}	PUT	Developer Updated
Delete Developer	/admin/deleteDeveloper/{id}	DELETE	Delete Successful
Map Issue	/admin/mapIssue/{issueId}	POST	Save the Changes
Get All Opened Status	/admin/openStatus	GET	Array of Status
Get All Closed Status	/admin/closedStatus	GET	Array of Status

Frontend:

User:

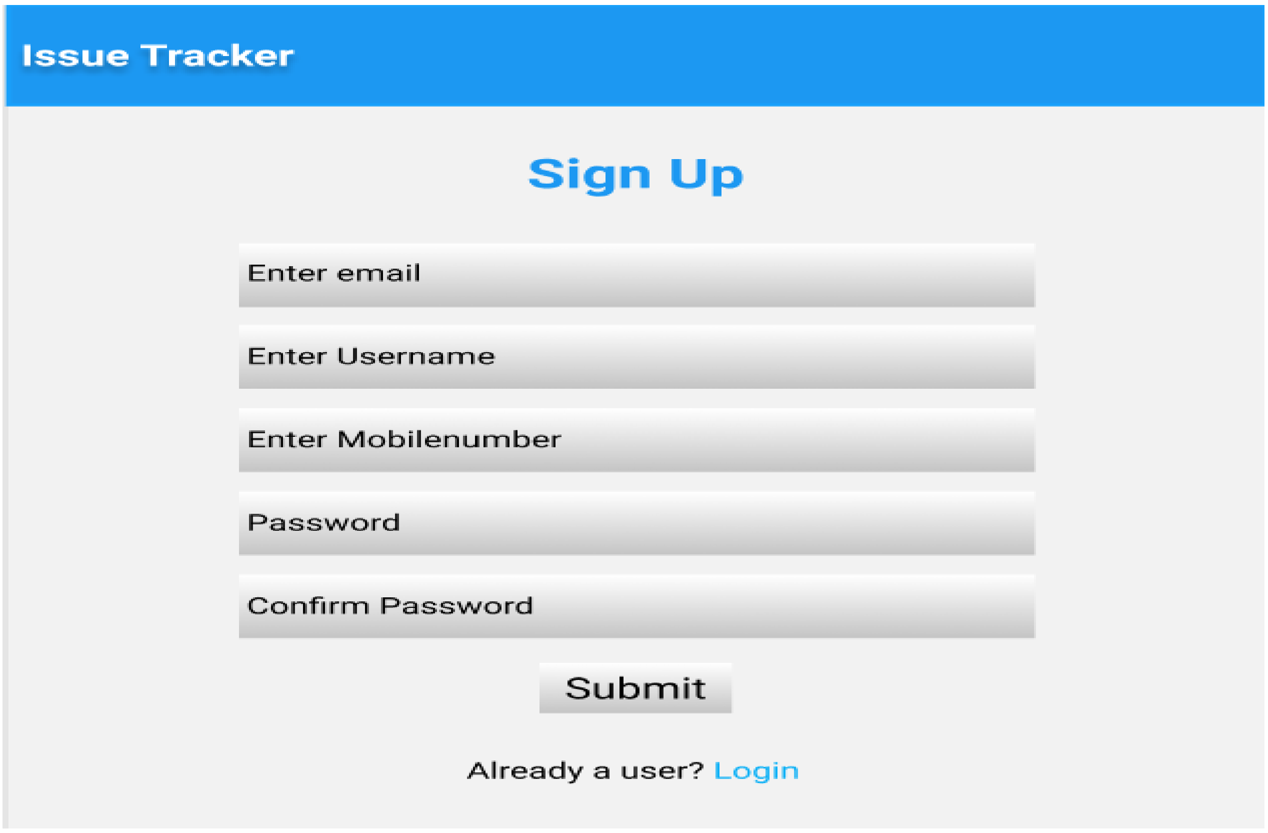
Login:

Output Screenshot:



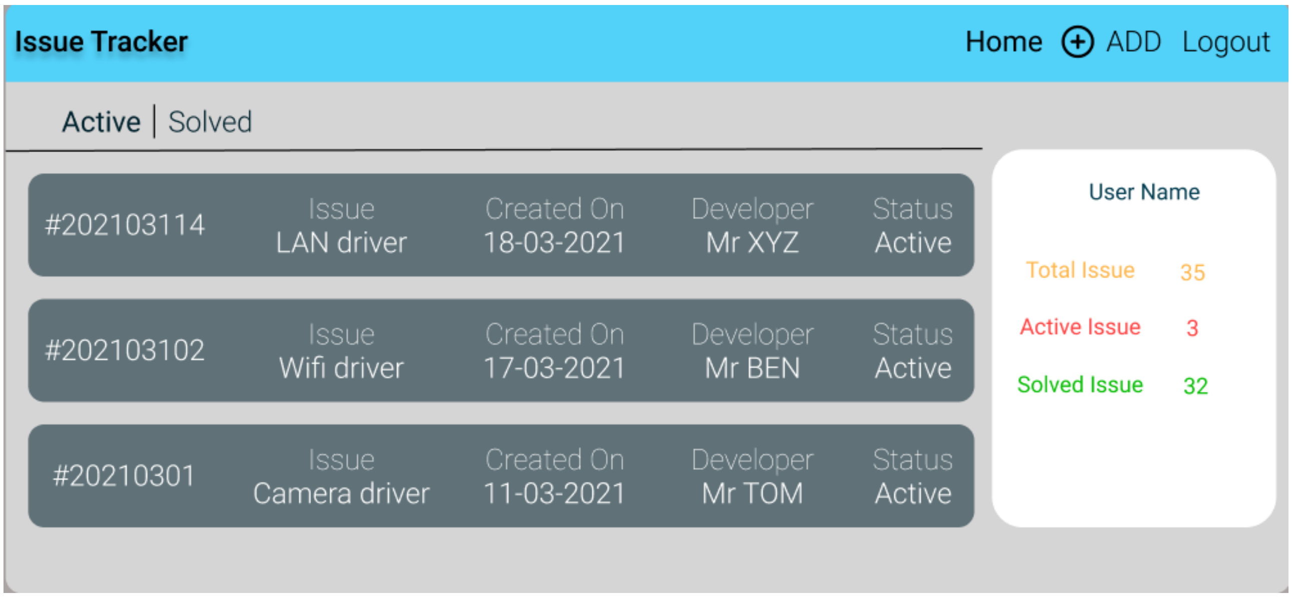
Signup:

Output Screenshot:



Home:

Output Screenshot:



Issue Tracker

Home ⊕ ADD Logout

Active | Solved

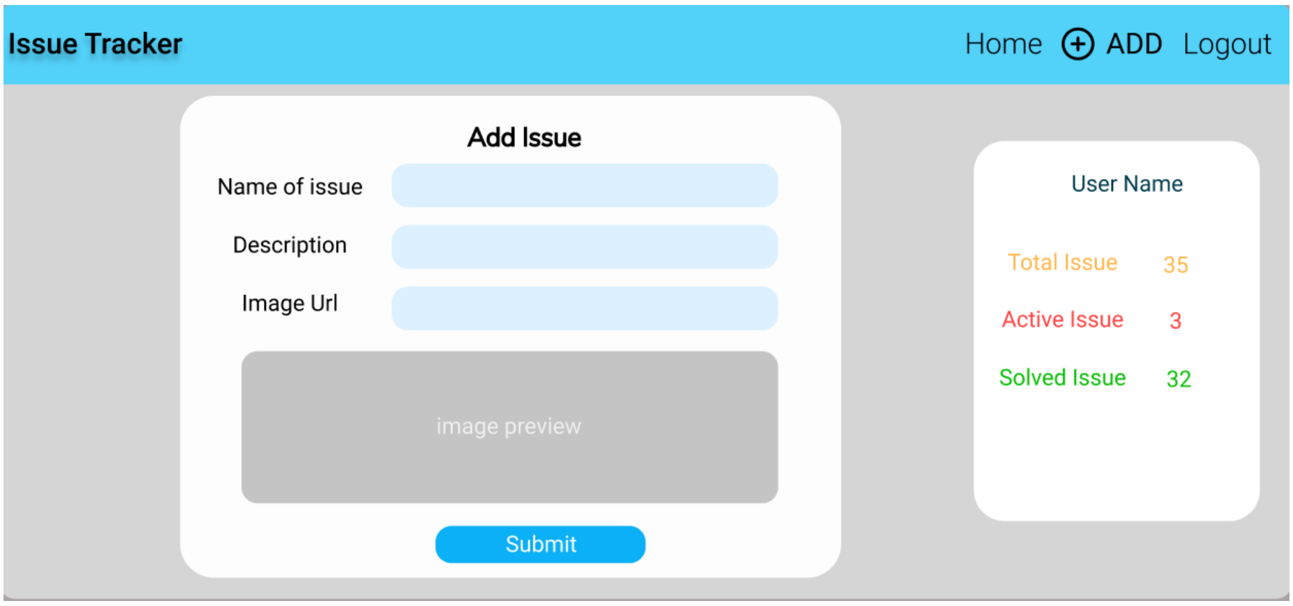
#202103114	Issue LAN driver	Created On 18-03-2021	Developer Mr XYZ	Status Active
#202103102	Issue Wifi driver	Created On 17-03-2021	Developer Mr BEN	Status Active
#20210301	Issue Camera driver	Created On 11-03-2021	Developer Mr TOM	Status Active

User Name

Total Issue	35
Active Issue	3
Solved Issue	32

Add Issue:

Output Screenshot:



Issue Tracker

Home ⊕ ADD Logout

Add Issue

Name of issue

Description

Image Url

image preview

Submit

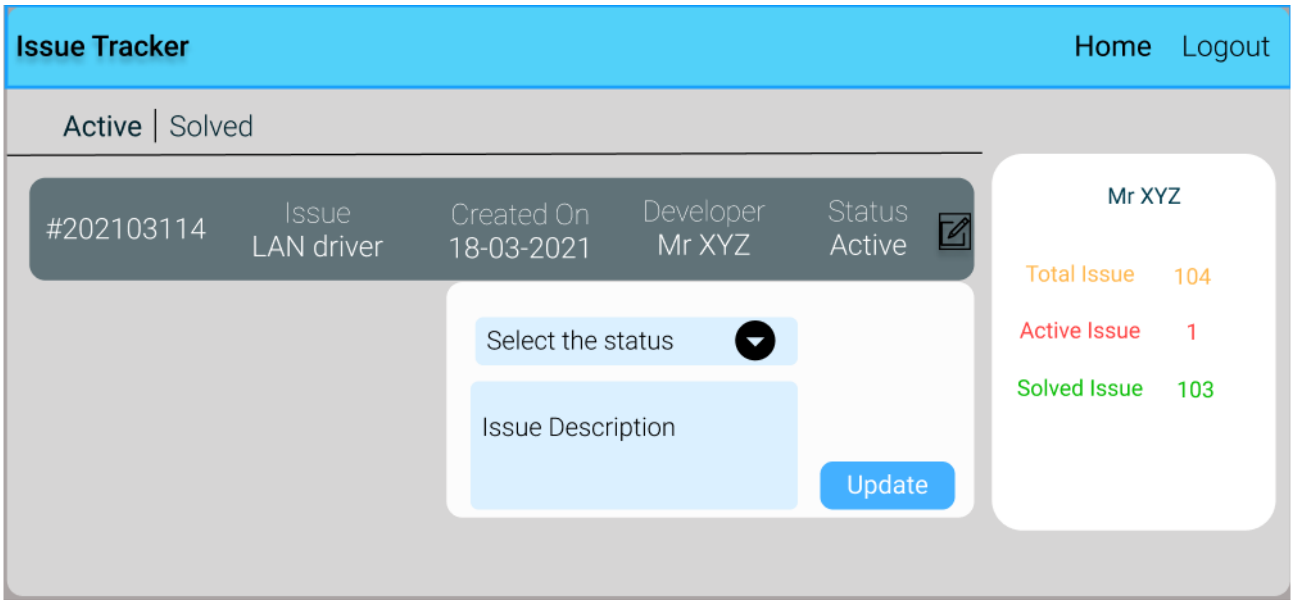
User Name

Total Issue	35
Active Issue	3
Solved Issue	32

Developer:

Home:

Output Screenshot:


Issue Tracker

Home Logout

Active | Solved

#202103114 Issue LAN driver Created On 18-03-2021 Developer Mr XYZ Status Active

Select the status

Issue Description

Update

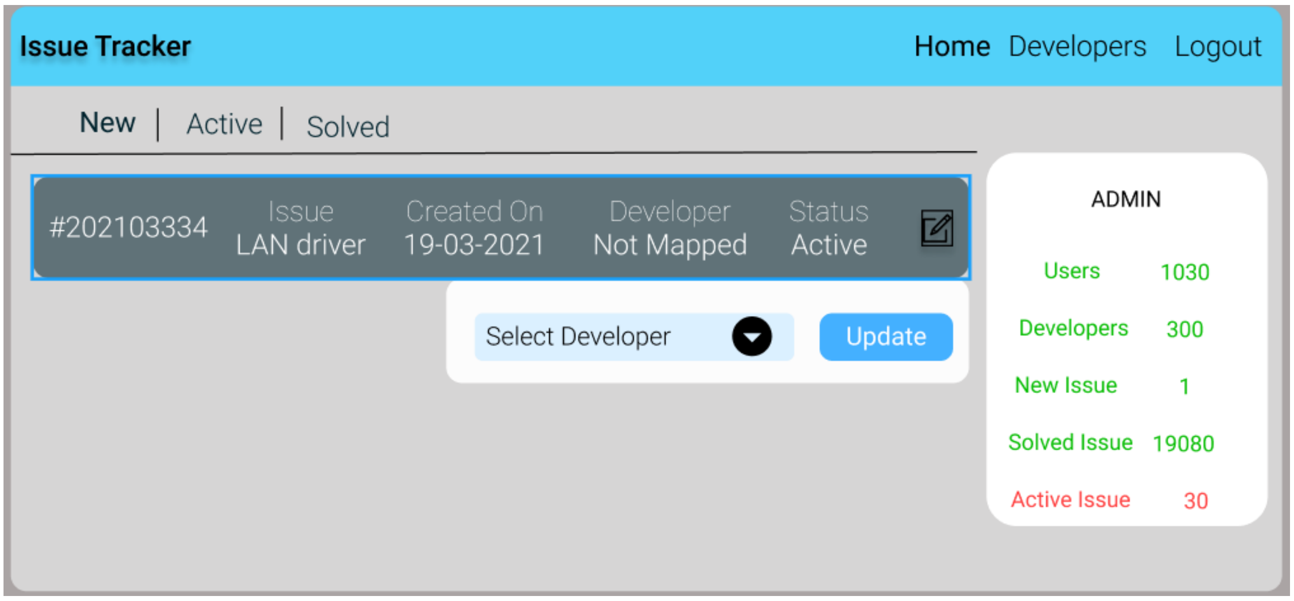
Mr XYZ

Total Issue	104
Active Issue	1
Solved Issue	103

Admin:

Home:

Output Screenshot:


Issue Tracker

Home Developers Logout

New | Active | Solved

#202103334 Issue LAN driver Created On 19-03-2021 Developer Not Mapped Status Active

Select Developer

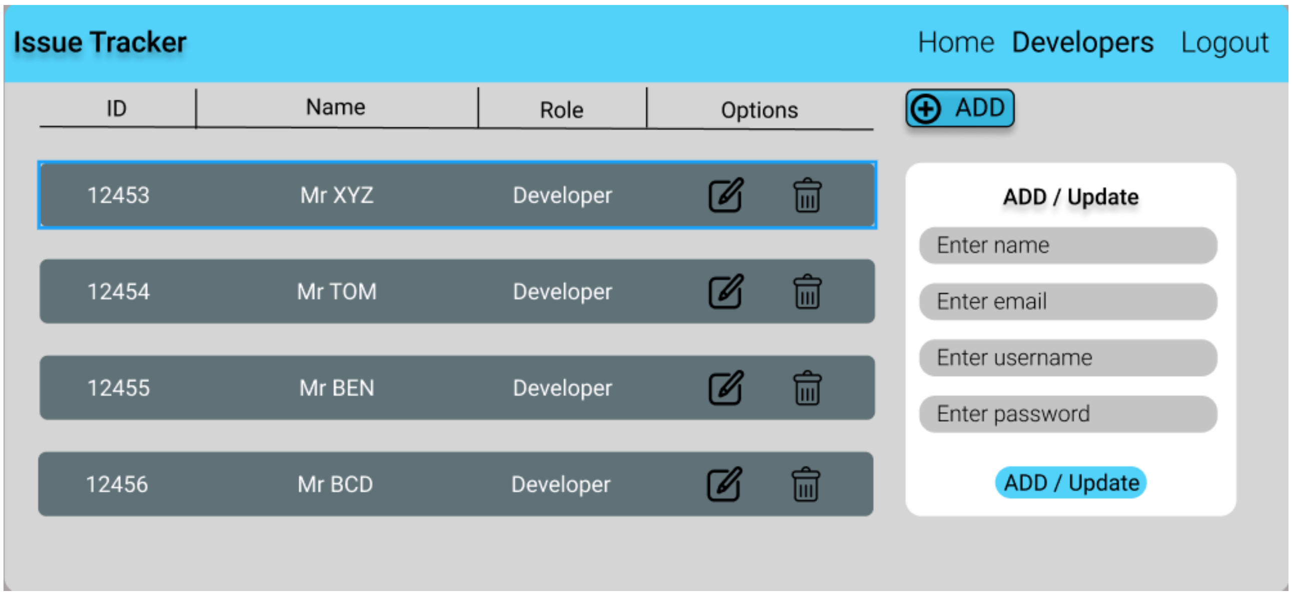
Update

ADMIN

Users	1030
Developers	300
New Issue	1
Solved Issue	19080
Active Issue	30

Admin Developers:

Output Screenshot


Issue Tracker

Home Developers Logout

ID	Name	Role	Options
12453	Mr XYZ	Developer	
12454	Mr TOM	Developer	
12455	Mr BEN	Developer	
12456	Mr BCD	Developer	

ADD

ADD / Update

Enter name

Enter email

Enter username

Enter password

ADD / Update

<u>Backend:</u>

<u>Class and Method description:</u>

<u>Model Layer:</u>

1. UserModel: This class stores the user type (admin or the customer) and all user information.
    a. Attributes:
        i. email: String
        ii. password: String
        iii. username: String
        iv. mobileNumber: String
        v. active: Boolean
        vi. role: String
    b. Methods: -

2. LoginModel: This class contains the email and password of the user.
    a. Attributes:
        i. email: String
        ii. password: String
    b. Methods: -

3. IssueModel: This class stores the details of the Issue.
    a. Attributes:
        i. issueId: String
        ii. imageUrl: String
        iii. issueName: String
        iv. issueDesc: String
        v. createdOn: Date
        vi. createdBy: UserModel
        vii. connectedBy: UserModel
        viii. status: String
    b. Methods: -
4. StatusModel: This is hold the Status of all the Issues.
    a. Attributes:
        i. issueId: String
        ii. statusId: String
        iii. status: String
        iv. statusDesc: Desc
    b. Methods: -

Controller Layer:

1. UserController: This calss controls the add/edit/update/view the users.
    a. Attributes: -
    b. Methods:
        i. List<userModel> getUsers(): This method helps the admin to fetch all users from the database.
        ii. UserModel userDataById(String id): This method helps the admin to retrieve a user from the database based on the user id.
        iii. userEditSave(UserModel data): This method helps the admin to edit a user and save it to the database.
        iv. userSave(UserModel data): This method helps the admin to add a new user to the database.
        v. UserDelete(UserDelete String id): This method helps the admin to delete a user from the database.
2. LoginController: This class controls the user login.
    a. Attributes: -

b. Methods:

    i. checkUser(LoginModel data): This method helps the user to sign up for the application and must return true or false

3. IssueController: This class controls the add/edit/update/view Issue.

    a. Attributes: -

    b. Methods:

        i. List<IssueModel> getIssue(): This method helps the admin to fetch all Issue from the database.

        ii. List<IssueModel> getHomeIssue(): This method helps to retrieve all the Issue from the database.

        iii. IssueModel IssueEditData(String id): This method helps to retrieve a Issue from the database based on the Issue Id.

        iv. IssueEditSave(IssueModel data): This method helps to edit a Issue and save it to the database.

        v. IssueSave(IssueModel data): This method helps to add a new Issue to the database.

        vi. IssueDelete (String id): This method helps to delete a Issue from the database.

4. StatusController: This class helps to manage the open / closed issues.

    a. Attributes: -

    b. Methods:

        i. mapIssue(String issueId, String StatusId): This method helps the map the issue with status.

        ii. List<StatusModel> showOpenStaus(): This method helps to view the all opened status

        iii. List<StatusModel> showClosedStaus(): This method helps to view the all Closed status.

        iv. updateStatus(String id): This method helps to update the status of the