

## Online grievance redressal system

### Objective:

Grievance Redressal System is an online platform to receive and act on complaints reported by students of private or public institutions, enabling prompt actions on any issue raised by them and to avail services more effectively.

### Users of the System:

1. Admin
2. Employee(Developer)
3. User

### Functional Requirements:

- Users should be able to create new account, log-in to their existing accounts which will give them the authority to use the services provided by the system.
- Authenticated users should be able to issue complaints, check complaint status.
- Employee can log-in to their accounts as created by administrator.
- Employee can access all the complaints, suggestions from users.
- Give response to complaints with activity reports.
- Admin can Create, and monitor accounts of authorities
- **An employee can manage a maximum of 10 complaints per day.**

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

- Online Surveys.
- Facility to upload photos of the complaint. for eg, garbage problem.
- Email integration for intimating new compliant.
- Multi-factor authentication for the sign-in process

### Output/ Post Condition:

- Queries and responses answered report
- Escalation reports based on responsibility matrix
- Standalone application / Deployed in an app Container
- Monthly Report

### Non-Functional Requirements:

<b>Security</b>	<ul style="list-style-type: none"><li>• App Platform –UserName/Password-Based Credentials</li><li>• Sensitive data has to be categorized and stored in a secure manner</li><li>• Secure connection for transmission of any data</li></ul>
<b>Performance</b>	<ul style="list-style-type: none"><li>• Peak Load Performance</li><li>• Online grievance redressal system &lt; 3 Sec</li><li>• Admin application &lt; 2 Sec</li><li>• Non Peak Load Performance</li><li>• Online grievance redressal system &lt; 2 Sec</li></ul>

	<ul style="list-style-type: none"> <li>• Admin Application &lt; 2 Sec</li> </ul>
<b>Availability</b>	<ul style="list-style-type: none"> <li>• 99.99 % Availability</li> </ul>
<b>Standard Features</b>	<ul style="list-style-type: none"> <li>• Scalability</li> <li>• Maintainability</li> <li>• Usability</li> <li>• Availability</li> <li>• Failover</li> </ul>
<b>Logging &amp; Auditing</b>	<ul style="list-style-type: none"> <li>• The system should support logging(app/web/DB) &amp; auditing at all levels</li> </ul>
<b>Monitoring</b>	<ul style="list-style-type: none"> <li>• Should be able to monitor via as-is enterprise monitoring tools</li> </ul>
<b>Cloud</b>	<ul style="list-style-type: none"> <li>• The Solution should be made Cloud-ready and should have a minimum impact when moving away to Cloud infrastructure</li> </ul>
<b>Browser Compatible</b>	<ul style="list-style-type: none"> <li>• IE 7+</li> <li>• Mozilla Firefox Latest – 15</li> <li>• Google Chrome Latest – 20</li> <li>• Mobile Ready</li> </ul>

### 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 AuthGuard 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
Signup	/signup	POST	True/false
Add Compliant	/add Compliant	POST	Compliant added
List logged in user Compliant	/compliant/{id}	GET	Array of Compliant
Update Compliant	/compliant/{id}	PUT	Compliant Updated.
Update Status	/status/{id}	PUT	Status Updated.
ADMIN			
Action	URL	Method	Response
Get All Compliant	/admin	GET	Array of Compliant
Add Employee	/admin/addEmployee	POST	Employee added
Update Employee	/admin/updateEmployee /{id}	PUT	Employee Updated
Delete Developer	/admin/deleteEmployee /{id}	DELETE	Delete Successful
Map Compliant	/admin/mapCompliant /{issueId}	POST	Save the Changes
Update Compliant	/admin/updateCompliant /{id}	PUT	Update Success

Get All Opened Status	/admin/openStatus	GET	Array of Status
Get All Closed Status	/admin/closedStatus	GET	Array of Status

### **Frontend:**

### **Customer:**

### **Login:**

Output Screenshot:

**Grievance Redressal**

**Login**

Enter email

Enter Password

Login

New User? [Sign Up](#)

### **Signup:**

Output Screenshot:

# Grievance Redressal

## Sign Up

Enter email

Enter Username

Enter Mobilenumber

Password

Confirm Password

Submit

Already a user? [Login](#)

Home:

Output Screenshot:

Grievance Redressal					Home	+	ADD	Logout
Active   Solved								
#202103114	Issue Damage Product	Created On 18-03-2021	Developer Mr XYZ	Status Active	<div>User1</div> <div>Total Issue 5</div> <div>Active Issue 3</div> <div>Solved Issue 2</div>			
#202103102	Issue Wrong Product	Created On 17-03-2021	Developer Mr BEN	Status Active				
#20210301	Issue Product Damage	Created On 11-03-2021	Developer Mr TOM	Status Active				

Add Issue:

Output Screenshot:

The screenshot shows the 'Add Issue' form in the Grievance Redressal system. The form is centered on a light gray background. It has a dark blue header bar with the text 'Grievance Redressal' on the left and 'Home + ADD Logout' on the right. The form itself is a white rounded rectangle with the title 'Add Issue' at the top. It contains three input fields: 'Name of issue', 'Description', and 'Image Url'. Below these fields is a gray rectangular area labeled 'image preview'. At the bottom of the form is a blue 'Submit' button.

**Developer:**

**Home:**

Output Screenshot:

The screenshot shows the developer's home page in the Grievance Redressal system. The page has a dark blue header bar with 'Grievance Redressal' on the left and 'Home Logout' on the right. Below the header is a light gray bar with 'Active | Solved' tabs. The main content area is divided into three sections. On the left is a table with the following data:

#	Issue	Created On	Developer	Status	
#202103114	Damage Product	18-03-2021	Mr XYZ	Active	

In the center is a form for updating the issue. It has a dropdown menu labeled 'Select the status' with a downward arrow, an input field labeled 'Issue Description', and a blue 'Update' button.

On the right is a white rounded rectangle showing the developer's profile 'Mr XYZ' and a summary of issues:

- Total Issue 104
- Active Issue 1
- Solved Issue 103

**Admin:**

**Home:**

### Output Screenshot:

**Grievance Redressal**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

### Manage User:

#### Output Screenshot:

**Grievance Redressal**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

### Backend:

### Class and Method description:

### Model Layer:

1. UserModel: This class stores the user type (Admin or the Employee or the User) 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. CompliantModel: This class stores the details of the Issue.

a. Attributes:

- i. compliantId: String
- ii. compliantName: String
- iii. createdOn: Date
- iv. createdBy: UserModel
- v. resolvedBy: UserModel
- vi. status: StatusModel

b. Methods: -

4. StatusModel: This is hold the Status of all the Issues.

a. Attributes:

- i. statusId: String
- ii. status: String
- iii. statusDesc: Desc

b. Methods: -



## **Controller Layer:**

1. SignupController: This class control the user signup
  - a. Attributes: -
  - b. Methods:
    - i. saveUser(UserModel user): This method helps the user to create account in the database and return true or false based on the database transaction
2. 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.
3. 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
4. CompliantModel: This class controls the add/edit/update/view Issue.
  - a. Attributes: -
  - b. Methods:
    - i. List<CompliantModel > getIssue(): This method helps the admin to fetch all Compliant from the database.
    - ii. List<CompliantModel> getHomeIssue(): This method helps to retrieve all the Compliant from the database.
    - iii. CompliantModel IssueEditData(String id): This method helps to retrieve a Compliant from the database based on the Compliant Id.
    - iv. compliantEditSave(CompliantModel data): This method helps to edit a Compliant and save it to the database.

- v. `compliantSave(CompliantModel data)`: This method helps to add a new Compliant to the database.
- vi. `compliantDelete (String id)`: This method helps to delete a Compliant from the database.

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

a. Attributes: -

b. Methods:

- i. `mapCompliant(String compliantId)`: 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.