Internal Job Portal

Objective:

Internal Job Portal is an online application to be built as a product that can provide a one-stop point for all employees to apply/view job postings, comment on postings, post job requirements etc..

Users of the System:

1. Admin
2. HR
3. Employees

Functional Requirements:

- The System should enable Project Managers to submit their job requirements to the HR for posting.
- System should incorporate an approval cycle where the HR validates the submitted job posting before posting to the portal
- System should enable employees to view and apply different jobs, allow discussions about jobs etc..
- System should enable Project Managers to View their Job Postings, Applicant details and their profiles for a particular Job Posting.
- **An Employee can apply maximum 2 jobs.**

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

- Filters for products like Low to High or showcasing products based on the customer's price range, specific brands etc.
- Email integration for intimating new personalized offers to customers.
- Multi-factor authentication for the sign-in process
- Payment Gateway

Output/ Post Condition:

- Records Persisted in Success & Failure Collections
- Standalone application / Deployed in an app Container

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 • Job portal -< 3 Sec • Admin application < 2 Sec
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:

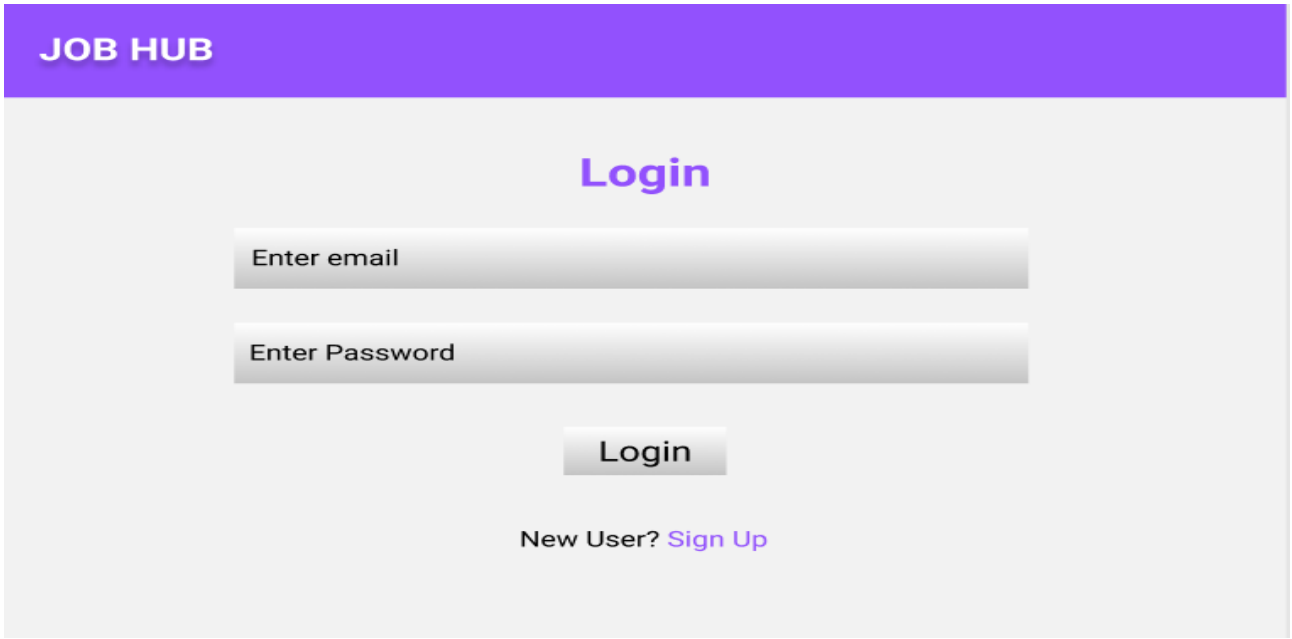
USER			
Action	URL	Method	Response
Login	/login	POST	true/false
Get All Job Post – Home	/home	GET	Array of Job Post
Add to Applied Job	/home/{id}	POST	Job aApplied
All Applied Jobs	/appliedJobs/{id}	GET	Array of AppliedJobs
Delete Applied Jobs	/ appliedJobs /delete	Delete	Applied Job Deleted
HR			
Action	URL	Method	Response
Get All Jobs	/hr	GET	Array of Jobs
Add Job	/hr/addJob	POST	Job added
Delete Job	/hr/delete/{id}	GET	Job deleted
Job Edit	/hr/jobEdit/{id}	GET	Get All details of Particular id
Job Edit	/hr/jobEdit/{id}	PUT	Save the Changes
Get All AppliedJobs	/hr/allAppliedJobs	GET	Array of AppliedJobs
ADMIN			
Add HR	/admin/add	POST	HR added
Update HR	/admin/update/{id}	PUT	HR Updated
Delete HR	/admin/delete/{id}	DELETE	HR deleted
Get All HR	/admin/	GET	Return array of HR

Frontend:

Employee:

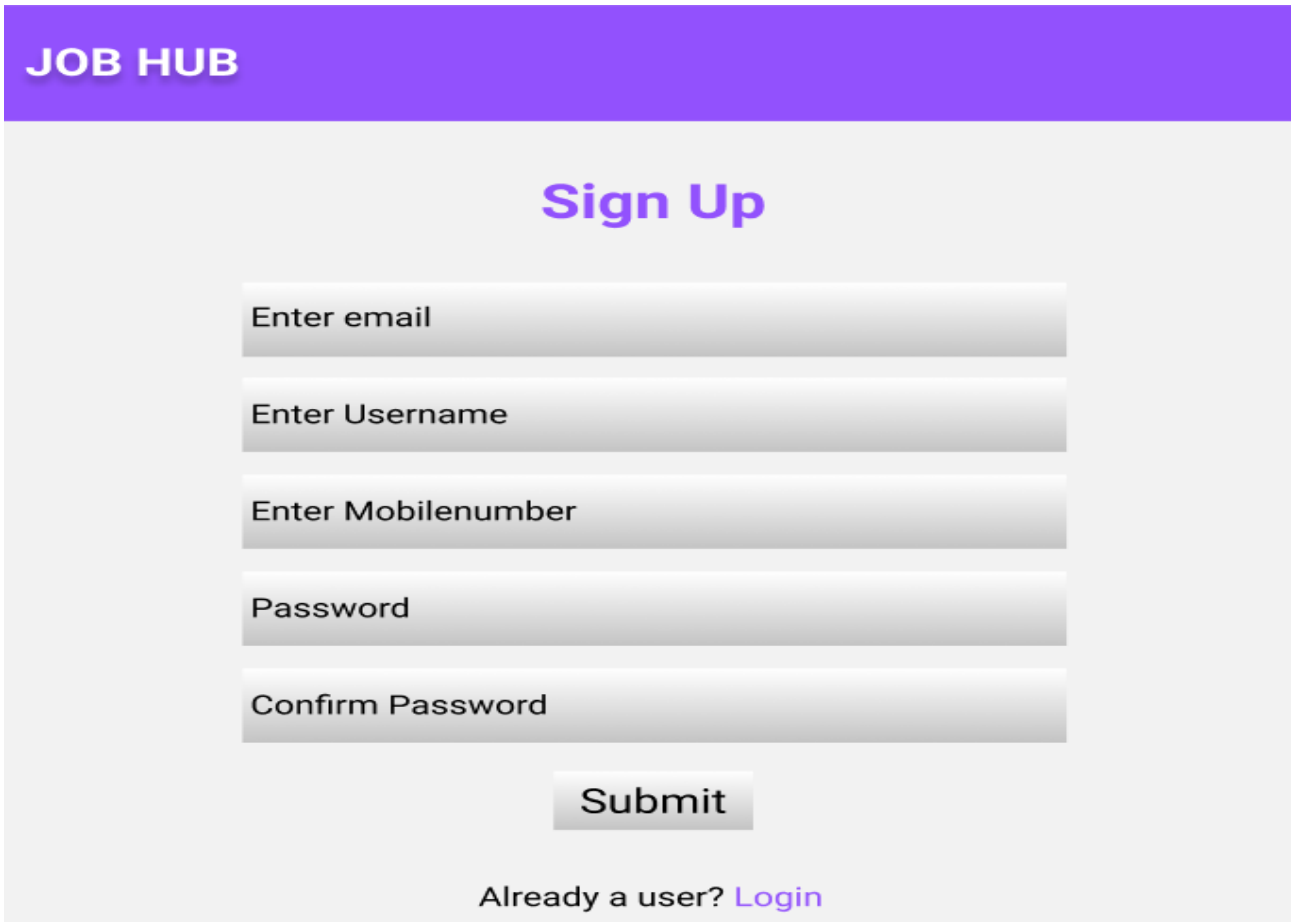
Login:

Output Screenshot:



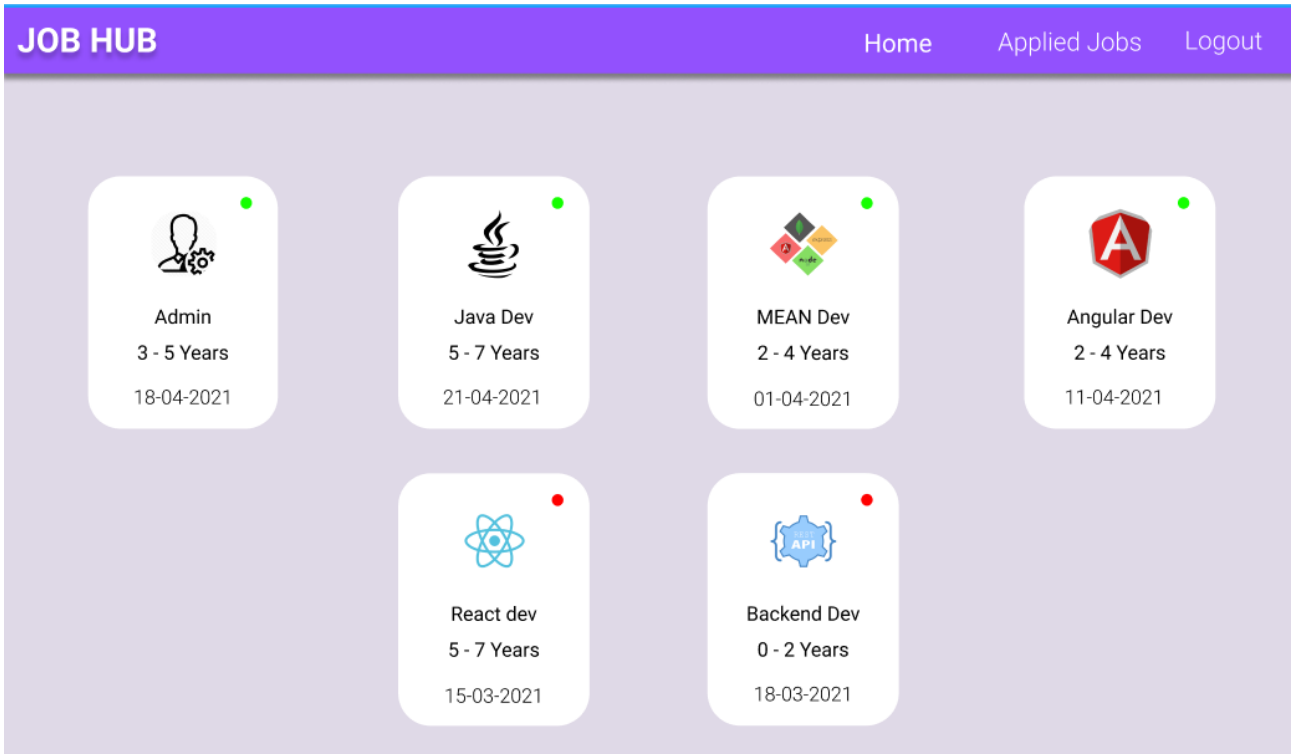
Signup:

Output Screenshot:



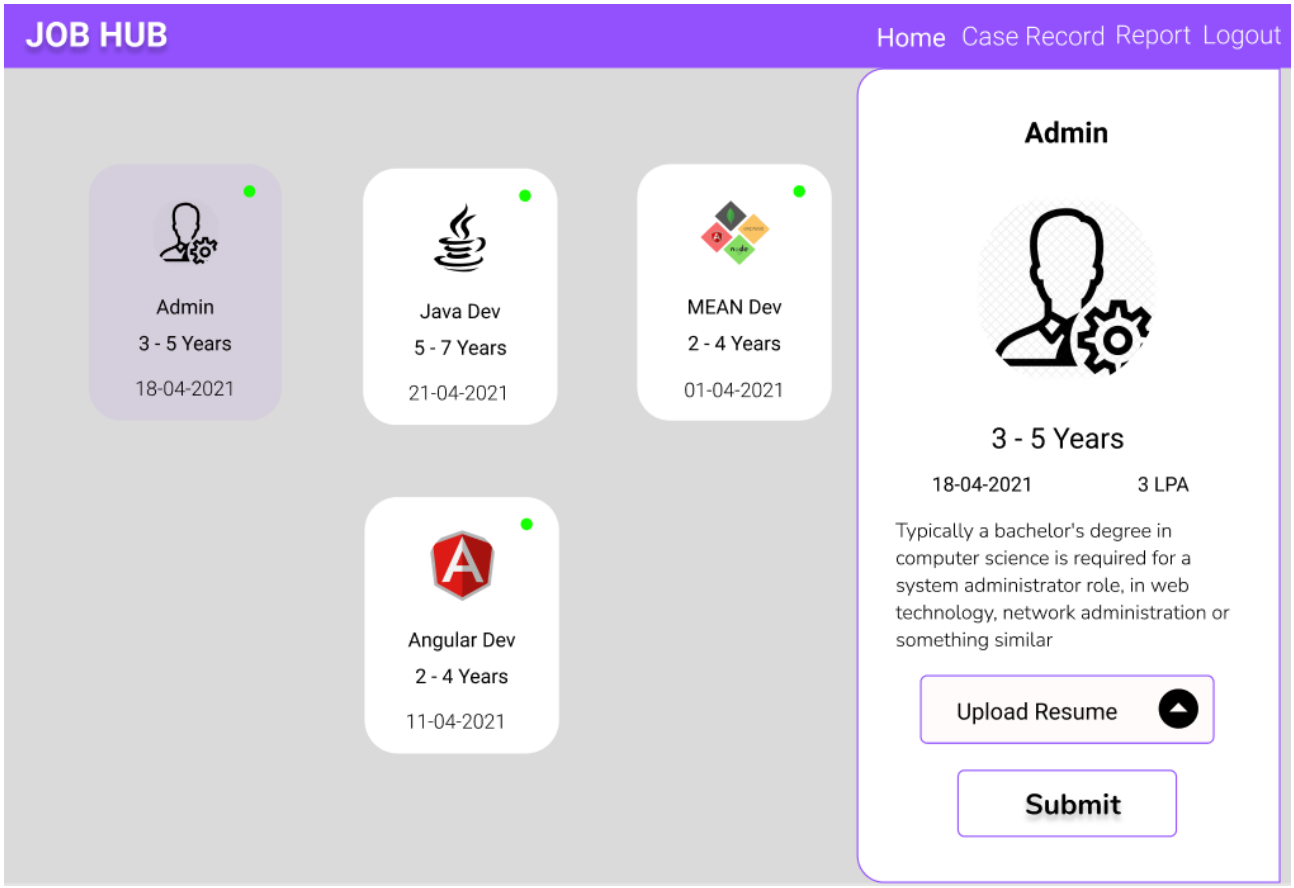
Home:

Output Screenshot:



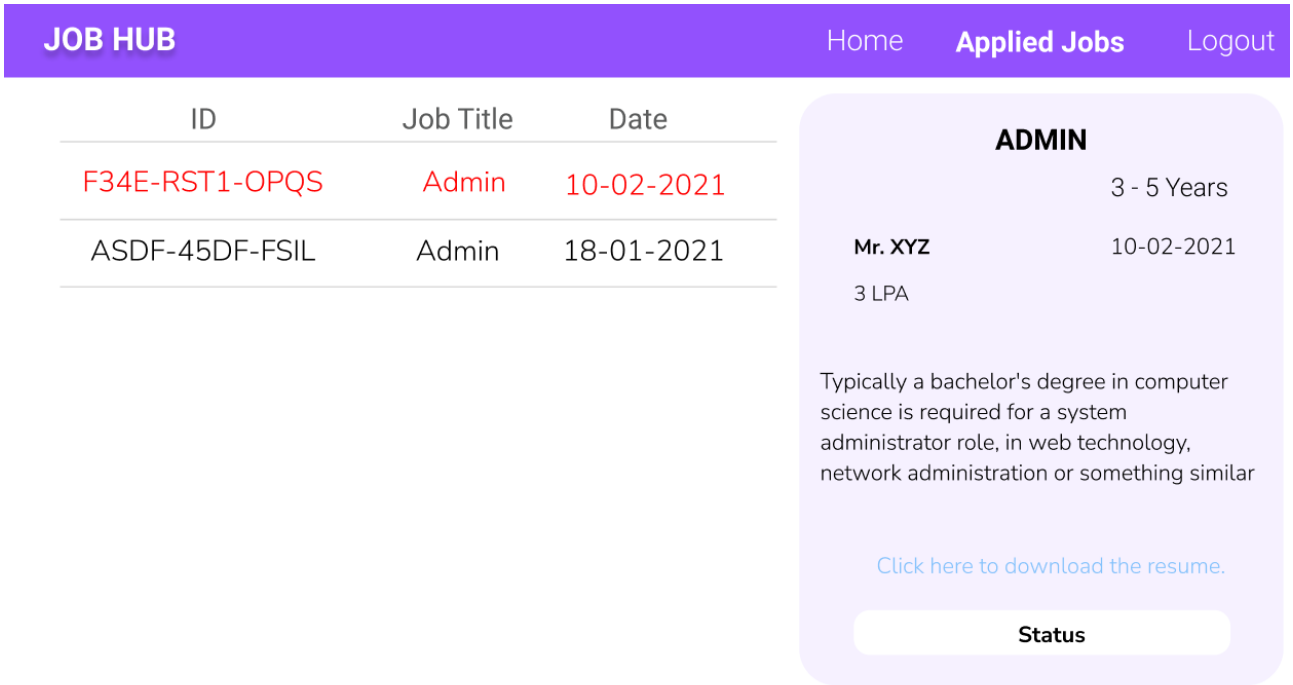
Apply:

Output Screenshot:



Applied Jobs:

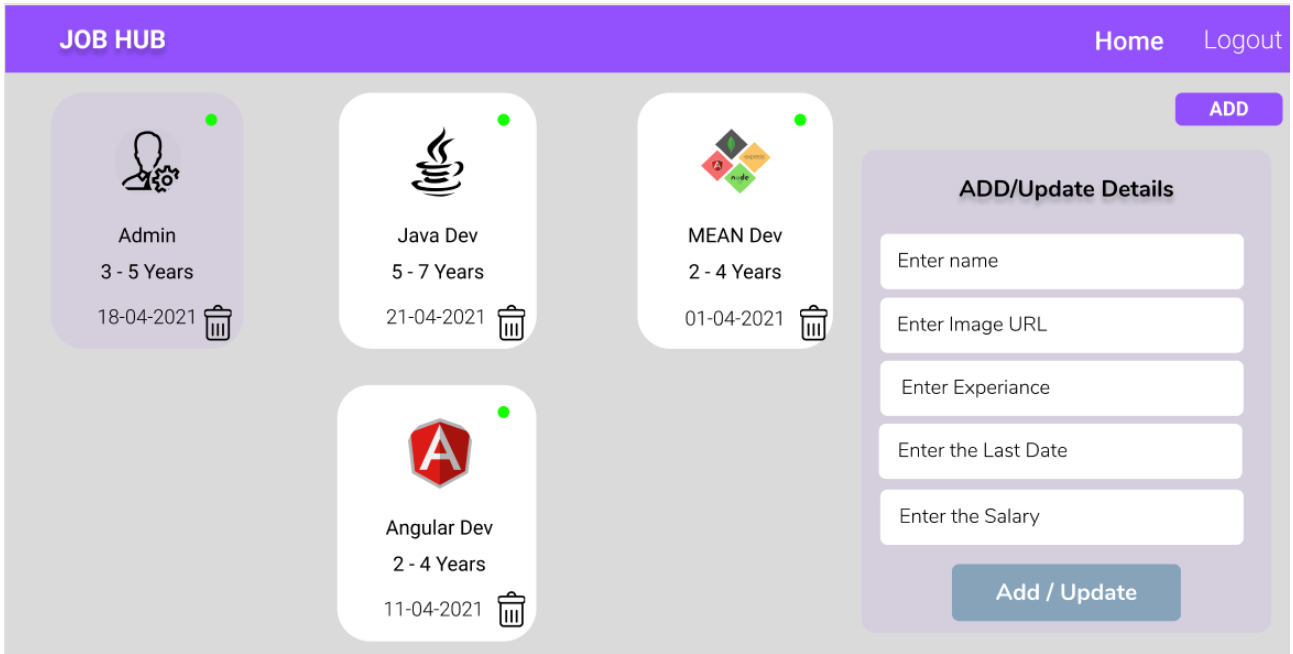
Output Screenshot:



<u>HR:</u>

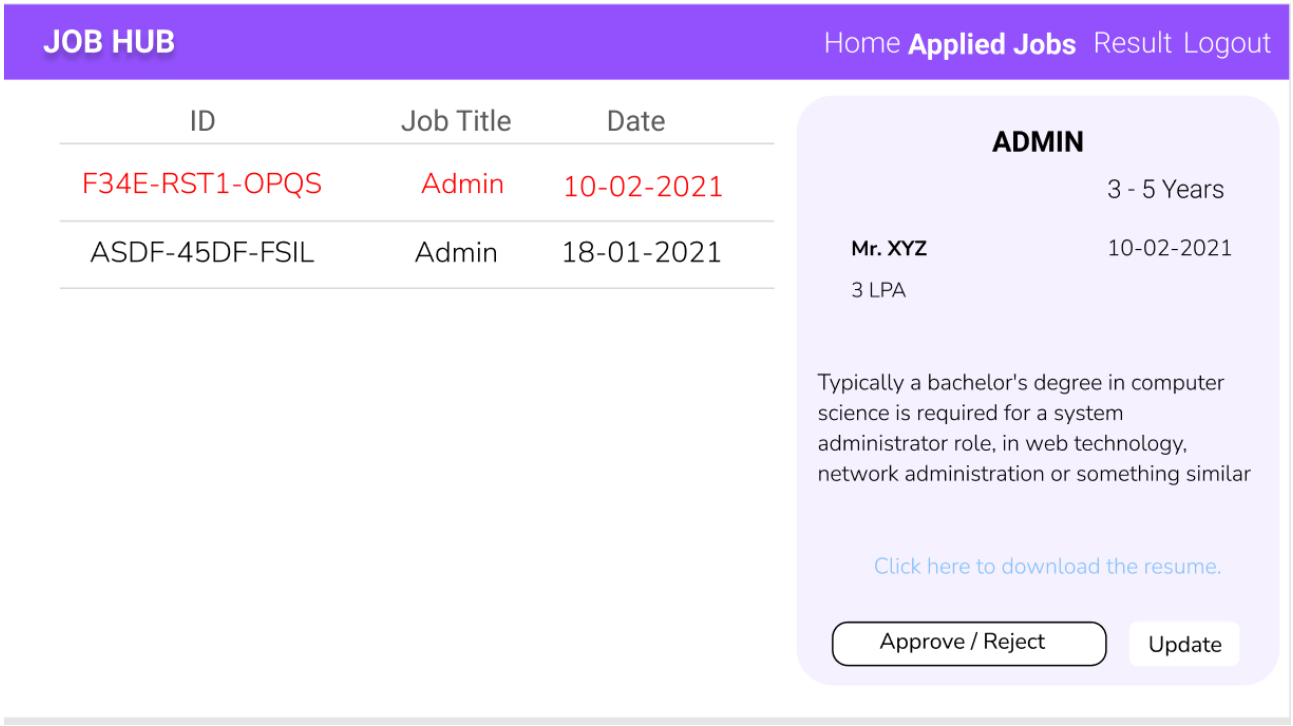
Home:

Output Screenshot:



Applied Jobs:

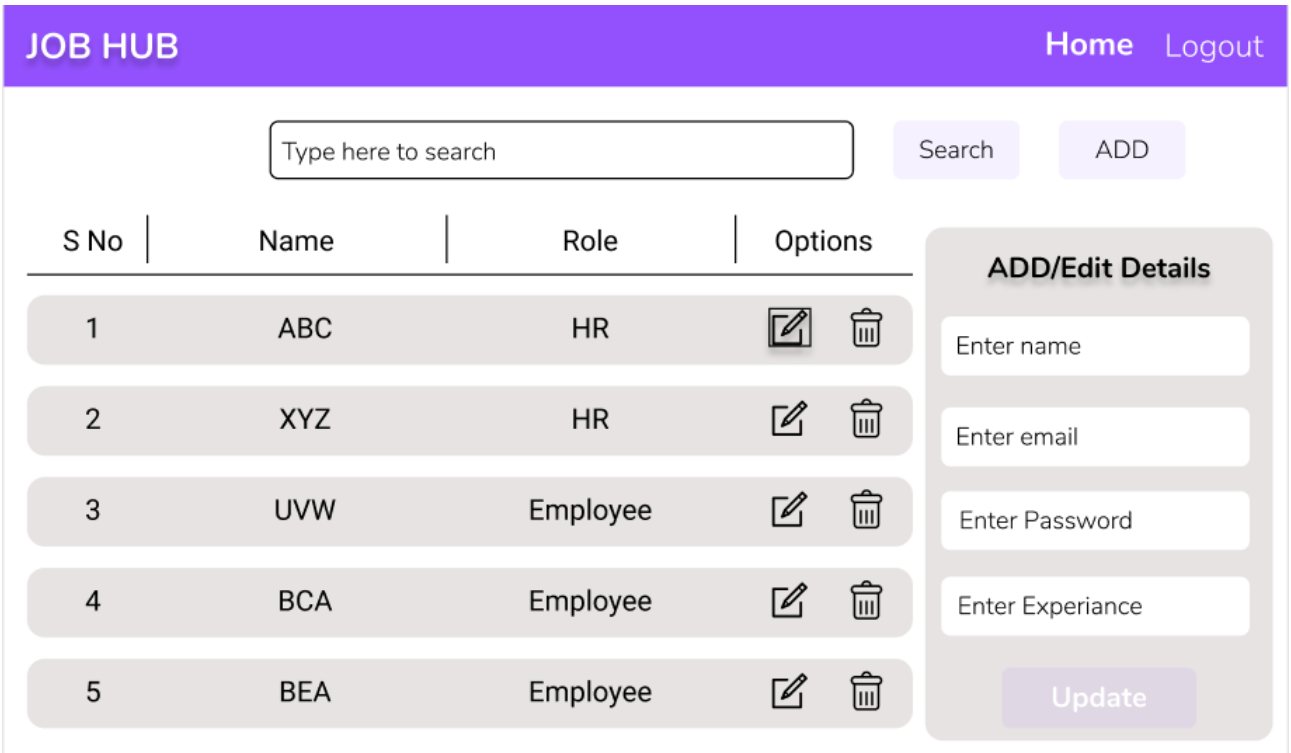
Output Screenshot:



Admin:

Home:

Output Screenshot:



Backend:

Class and Method description:

Model Layer:

1. EmployeeModel: This class stores the user type (HR or the Employee) and all user information.
    a. Attributes:
        i. email: String
        ii. password: String
        iii. empId: String
        iv. mobileNumber: String
        v. department: Boolean
        vi. role: String
    b. Methods: -
2. LoginModel: This class contains the email and password of the user.
    a. Attributes:
        i. empid: String
        ii. password: String
    b. Methods: -
3. JobModel: This class stores the details of the Jo.
    a. Attributes:
        i. jobTitle: String
        ii. jobLocation: String
        iii. jobType: String
        iv. jobDesc: String
        v. salary: String
        vi. : String
    b. Methods: -
4. AppliedJobModel: This class stores the applied jobs details.
    a. Attributes:
        i. jobId: String
        ii. employeeId: String

iii. appliedDate: Date

b. Methods: -

Controller Layer:

1. EmployeeController: This class controls the add/edit/update/view Employees.
    a. Attributes: -
    b. Methods:
        i. List<EmployeModel> getEmployee(): This method helps the admin / HR to fetch all employees from the database.
        ii. EmployeModel getEmployeeById(String id): This method helps to retrieve a Employee from the database based on the employee id.
        iii. EmployeeModel editEmployee(EmployeeModel data): This method helps to edit a employee and save it to the database.
        iv. saveEmployee(EmployeeModel data): This method helps to add a new employee to the database.
        v. deleteEmployee(String id): This method helps to delete a Employee 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. JobController: This class controls the add, edit, update, view Jobs.
    a. Attributes: -
    b. Methods:
        i. List<JobModel> getJobs(): This method helps the HR to fetch all jobs from the database.
        ii. List< JobModel > getHomeJobs(): This method helps to retrieve all the jobs from the database.
        iii. JobModel jobEditData(String id): This method helps to retrieve a job from the database based on the jobId.
        iv. jobsEditSave(JobModel data): This method helps to the HR to edit a jobs and save it to the database.
        v. jobSave(JobModel data): This method helps the HR to add a new job to the database.
        vi. jobtDelete (String id): This method helps the HR to delete a jobs from the database.
4. AppliedJobController: This class helps to mapping the employees with the jobs.

a. Attributes: -

b. Methods:

   i. jobMapping(String jobId, String employee Id): This method helps to mapping the job

   ii. List<AppliedJobModel> showJobs(String id): This method helps to view all the bobs by specific user.

   iii. deleteMapping(String jobId, String employee Id): This method helps to delete a mapping between the employee and jobs applied.