

## **Employee Expense management System**

### **Objective:**

An expense management system is software that simplifies the employee expense reimbursement process by automating much of it. The software reduces the need for paper, lowers the amount of time spent handling expenses and minimizes errors.

### **Users of the System:**

1. Admin
2. Manager
3. Employee

### **Functional Requirements:**

- Voucher Entry – Screen for entering expense vouchers for any reimbursable expenses borne by the employee.
- A voucher should have one header and multiple lines providing detailed information of expenses incurred along with amounts.
- Accounts View – Accounts department users should be able to view approved vouchers of all employees and mark vouchers as paid. This step completes the lifecycle of the voucher and the associated process instance.
- **Maximum limit 5000 per month.**

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

- Email integration for intimating new person signup.
- Multi-factor authentication for the sign-in process

### **Output/ Post Condition:**

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

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>• Expenditure Management &lt; 3 Sec</li><li>• Admin application &lt; 2 Sec</li><li>• Non Peak Load Performance</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.

- 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.
- Unless logged into the system, the user cannot navigate to any other pages.
- Logging out must again redirect to the login page.
- To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
- Use admin/admin as the username and password to navigate to the admin dashboard.

### **Validations:**

- Basic email validation should be performed.
- Basic mobile validation should be performed.

### **Project Tasks:**

### **API Endpoints:**

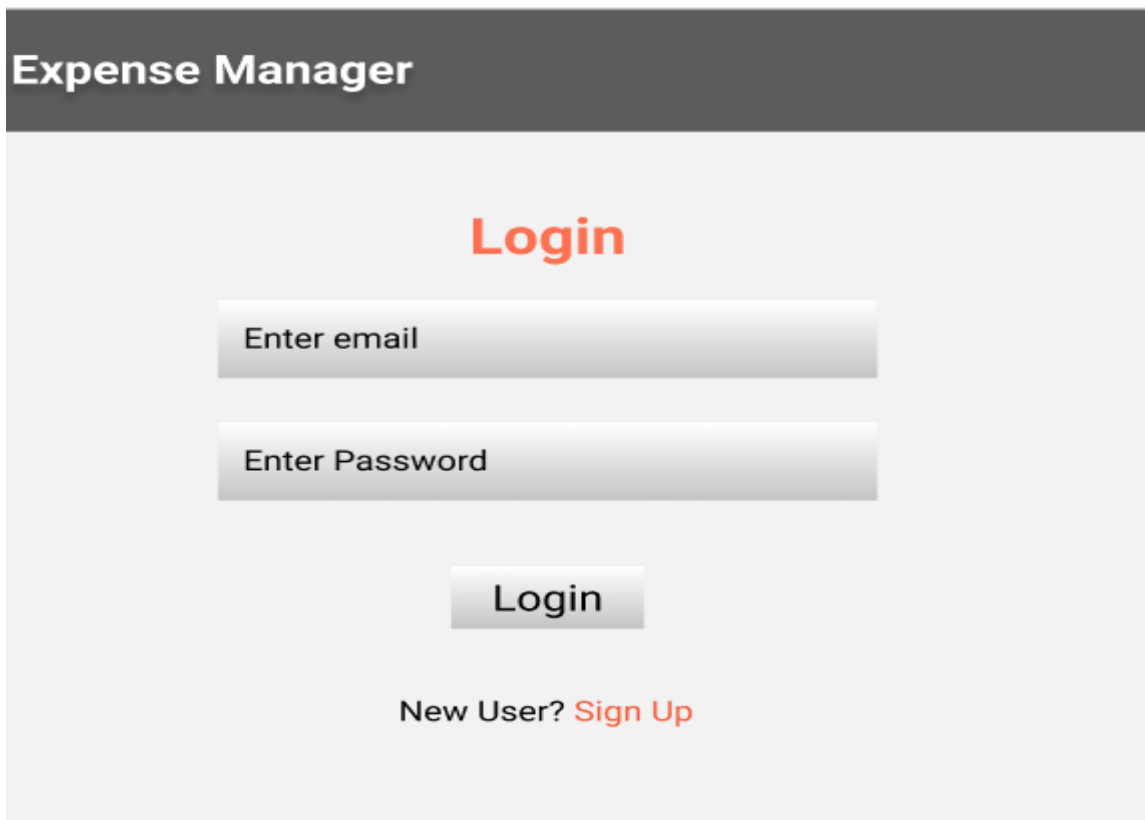
USER			
Action	URL	Method	Response
Login	/login	POST	true/false
Signup	/signup	POST	true/false
All Expense	/expense	GET	Array of expense
Expense Details	/expense/{id}	GET	Expense Detail by Id
Add Expense	/expense	POST	Expense Added
Update Expense	/expense/{id}	PUT	Expense Updated
MANAGER			
Action	URL	Method	Response
Get All Expense	/manager	GET	Array of Expense
Update Expense	/manager/expense/{id}	PUT	Updated
Delete Expense	/manager/expense/{id}	DELETE	Expense deleted
Get Expense	/manager/expense/{id}	GET	Get All details of Particular id
ADMIN			
Get All User	/admin	GET	Array of Expense
Get User	/admin/{id}	GET	User Details
Update User	/admin/user/{id}	PUT	Updated
Delete User	/admin/user/{id}	DELETE	Expense deleted

### **Frontend:**

### **User:**

## Login:

Output Screenshot:



The screenshot shows the login page of an application titled "Expense Manager". The page has a dark gray header with the title in white. Below the header, the word "Login" is displayed in a large, bold, orange font. There are two input fields: "Enter email" and "Enter Password", both with light gray backgrounds and rounded corners. Below these fields is a "Login" button with a light gray background and rounded corners. At the bottom, there is a link that says "New User? Sign Up", where "Sign Up" is in orange.

**Expense Manager**

**Login**

Enter email

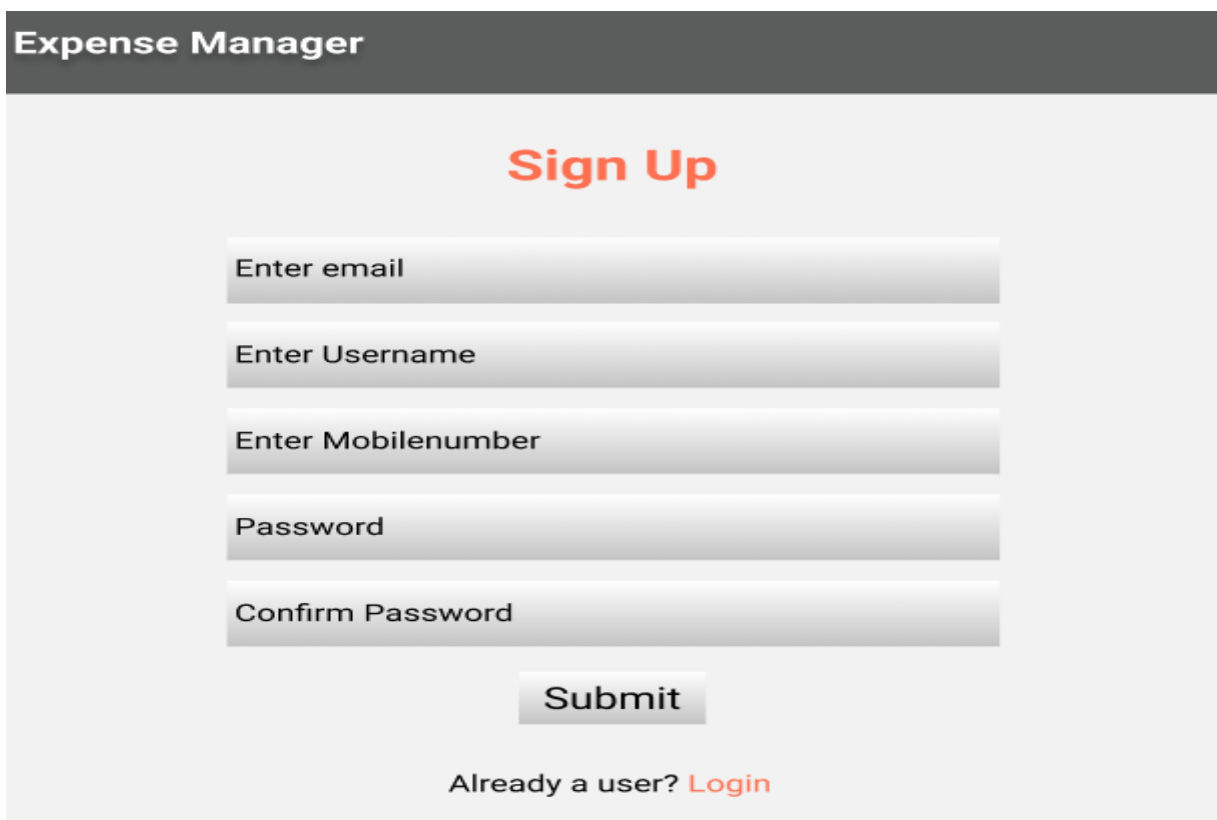
Enter Password

Login

New User? [Sign Up](#)

## Signup:

Output Screenshot:



The screenshot shows the signup page of an application titled "Expense Manager". The page has a dark gray header with the title in white. Below the header, the words "Sign Up" are displayed in a large, bold, orange font. There are five input fields: "Enter email", "Enter Username", "Enter Mobilenumber", "Password", and "Confirm Password", all with light gray backgrounds and rounded corners. Below these fields is a "Submit" button with a light gray background and rounded corners. At the bottom, there is a link that says "Already a user? Login", where "Login" is in orange.

**Expense Manager**

**Sign Up**

Enter email

Enter Username

Enter Mobilenumber

Password

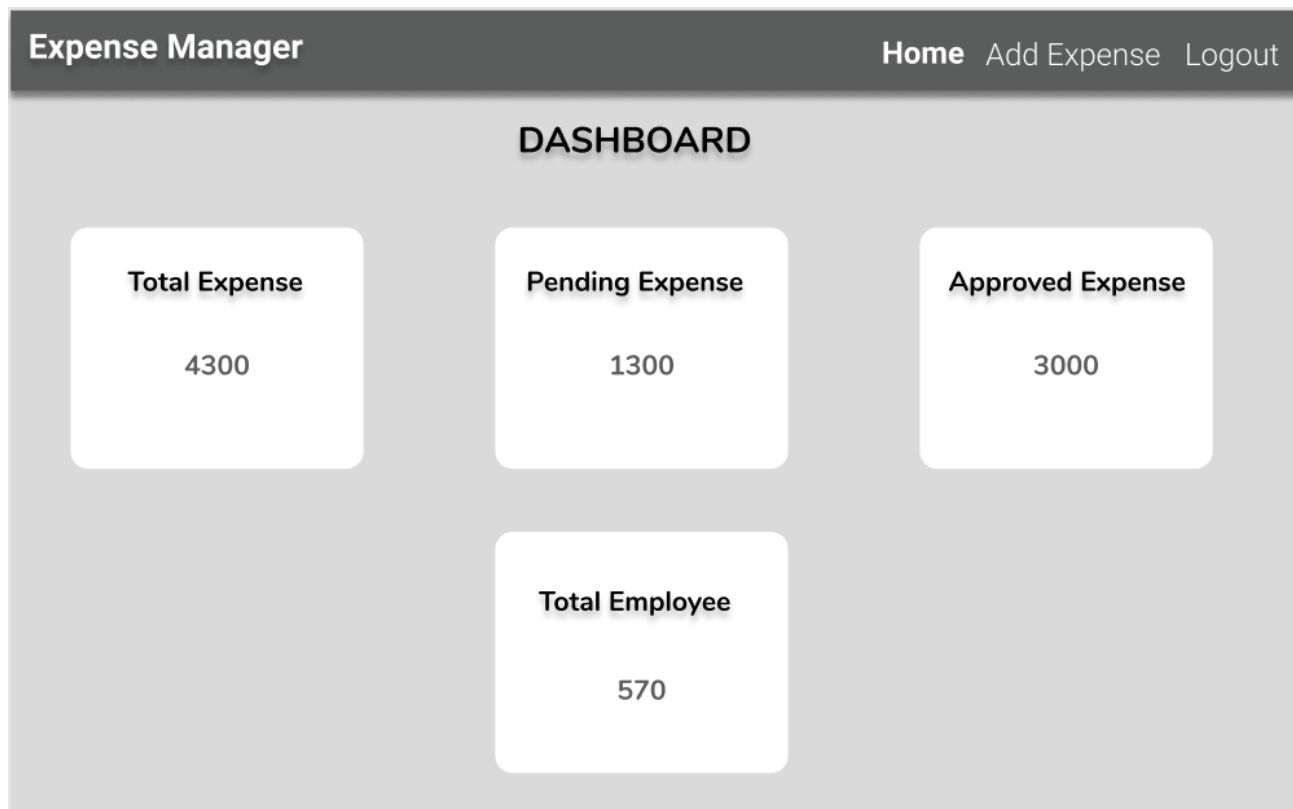
Confirm Password

Submit

Already a user? [Login](#)

## Home:

Output Screenshot:



## ADD Expense:

Output Screenshot:

The screenshot shows the 'Expense Manager' 'Add Expense' form. The header includes 'Expense Manager' on the left and navigation links 'Home', 'Add Expense', and 'Logout' on the right. The main content area is divided into two sections. On the left, there are three user cards, each with a blue circular profile icon, a name, a date, and an amount, followed by a small edit icon. On the right, there is a large white rounded rectangle titled 'Add Expense' containing a form with a blue circular profile icon, a 'Select Date' dropdown, an 'Emp ID' text input, an 'Upload Receipt' button with an upload icon, an 'Amount' text input, a 'Description' text input, and a 'Submit' button.

User	Date	Amount
User 1	18-04-2021	Rs. 550
User 2	18-03-2021	Rs. 110
User 3	11-02-2021	Rs. 800

**Add Expense Form Fields:**


- Select Date (dropdown)
- Emp ID (text input)
- Upload Receipt (button with upload icon)
- Amount (text input)
- Description (text input)
- Submit (button)

## Manager:

### Home:



Expense Manager


Logout



User 1



18-04-2021


 Rs. 550 



User 2



18-03-2021

 Rs. 110 





User 3

11-02-2021


 Rs. 800 

Add Expense



Select Date 

Emp ID

Upload Receipt 

Amount

Description

Submit

## ADMIN:

### HOME:











Expense Manager

Home Logout

Type here to search

Search

ADD

S No	Name	Role	Options
1	ABC	abc@gmail.com	 
2	XYZ	xyz@gmail.com	 
3	UVW	uvw@gmail.com	 
4	BCA	bca@gmail.com	 
5	BEA	bea@gmail.com	 

ADD/Edit Details

Enter name

Enter email

Enter Password

Enter Experiance

Enter Specialist

Update

## **Backend:**

### **Class and Method description:**

#### **Model Layer:**

1. UserModel: This class stores the user type (admin or the Manager or the Employee) 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. ExpenseModel: This class stores the details of the product.
  - a. Attributes:
    - i. expenceld: String
    - ii. billNumber: Int
    - iii. billImage: Blob
    - iv. billCost: int
    - v. datedOn: Date
    - vi. status: String
    - vii. remark: String
    - viii. claimedBy: UserModel
  - b. Methods: -

#### **Controller Layer:**

4. SignupController: This class control the user signup

- a. Attributes: -
  - b. Methods:
    - i. saveUser(UserModel user): This method helps to store users in the database and return true or false based on the database transaction.
5. 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
6. ExpenController: This class controls the add/edit/update/view Expense.
- a. Attributes: -
  - b. Methods:
    - i. List<ExpenseModel> getExpense(): This method helps the admin to fetch all Expense from the database.
    - ii. ExpenseModel expenseEditData(String id): This method helps to retrieve a Expense details from the database based on the Expense id.
    - iii. expenseEditSave(ExpenseModel data): This method helps to edit a Expense details and save it to the database.
    - iv. expenseSave(ExpenseModel data): This method helps to add a new product to the database.
    - v. expenseDelete (String id): This method helps to delete a Expense details from the database.
7. MailController: This class helps in sending mail to the User.
- a. Attributes: -
  - b. Methods:
    - i. sendMail(String id): This method helps to send the mail based on the status updated by the admin/ HR.