Table of contains

- **User Acceptance Testing (UAT)**: Steps taken to ensure the system meets user requirements.

Developers documentation

1.Project Overview

Introduction: "Apna Employee" is an advanced employee management system designed to streamline the management of employees within an organization. The system provides a user-friendly interface with separate login options for administrators and employees. It ensures that administrators can effectively manage employee details, categories, and overall organizational data, while employees can securely access their personal information, such as salary, address, and category details. The project is role-based, ensuring that each user has access only to the relevant features based on their role within the organization.

Features:

1. **Role-Based Login**: The system offers two distinct login options—one for admins and another for employees—ensuring that users access the appropriate dashboard based on their role.
2. **Admin Dashboard**: Once logged in, admins can view statistics like the number of employees, total salaries, and the number of administrators. The dashboard also allows admins to manage employee details and categories.
3. **Employee Self-Service**: Employees can log in to view their personal information, including salary details, address, and the category they belong to. This feature empowers employees by giving them direct access to their data.
4. **Category Management**: Admins have the option to add new categories to which employees can be assigned, allowing for better organization and management of employee roles within the company.
5. **Secure Logout**: The system includes a secure logout feature, ensuring that users can safely end their sessions without leaving their accounts vulnerable to unauthorized access.

2. System Requirements

2.1 Hardware Requirements

- **Server:**
  - **Processor**: Intel Core i5 or higher
  - **RAM**: 8 GB or higher
  - **Storage**: 100 GB SSD or higher
  - **Network**: 1 Gbps Ethernet

Client:

- **Processor**: Intel Core i3 or higher
- **RAM**: 4 GB or higher
- **Storage**: 10 GB HDD or SSD
- **Network**: Stable internet connection

2.2 Software Requirements

- **Server:**
  - **Operating System**: Ubuntu 20.04 LTS or Windows Server 2019
  - **Web Server**: Nginx or Apache HTTP Server
  - **Database**: MySQL 8.0 or PostgreSQL 13
  - **Backend Framework**: Node.js 14.x or higher
  - **Frontend Framework**: React.js 17.x or higher
  - **Runtime Environment**: Node.js with npm/yarn package manager
  - **Additional Libraries**: Multer (for file uploads), Express.js (for server-side operations)
- **Client:**
  - **Browser**: Google Chrome, Firefox, or Microsoft Edge (latest versions)
  - **Operating System**: Windows 10, macOS 10.15, or Ubuntu 20.04
  - **Dependencies**:
    - React.js for front-end rendering
    - Axios for API requests

2.3 Development Tools

- **IDE**: Visual Studio Code
- **Build Tools**: Vite for React.js .
- **Testing Tools**: Postman for API testing

3. System Architecture

3.1 Overview of the Architecture

The "Apna Employee" system follows a **three-tier architecture** comprising the Presentation Layer, Business Logic Layer, and Data Access Layer. This architecture ensures a separation of concerns, making the system more modular, scalable, and maintainable.

1. **Presentation Layer (Frontend):**
    - **Technology Used:** React.js
    - **Description:** The frontend is responsible for the user interface (UI) that interacts with the users. It includes components like the login page, dashboard, employee details, and category management. The UI communicates with the backend via RESTful APIs.
2. **Business Logic Layer (Backend):**
    - **Technology Used:** Node.js with Express.js framework
    - **Description:** The backend handles the application logic, processes user requests, and performs operations like authentication, CRUD operations on employee data, and category management. It also ensures that data validation and security checks are enforced.
3. **Data Access Layer (Database):**
    - **Technology Used:** MySQL
    - **Description:** The database layer is where all the data is stored and managed. This includes tables for storing employee information, categories, admin credentials, and other relevant data. The backend interacts with the database to fetch, update, or delete records as requested by the user.

3.2 Flow of Data

1. **User Interaction:**

- The user (either admin or employee) interacts with the system through the UI. For example, the admin logs in by entering credentials, which are sent to the backend via an API call.

2. **Request Handling:**
   - The backend receives the request, processes it, and performs necessary operations such as authentication or database queries.

3. **Data Processing:**
   - Once the backend processes the request, it might fetch data from the database, perform calculations, or execute business logic.
4. **Response Generation:**
   - After processing, the backend sends the relevant data back to the frontend, which updates the UI accordingly. For instance, after a successful login, the admin is redirected to the dashboard.

3.3 Technologies and Tools

- **Frontend:**
  - **React.js:** A JavaScript library for building user interfaces, particularly for single-page applications.
  - **Axios:** For making HTTP requests to the backend API.
- **Backend:**
  - **Node.js:** A JavaScript runtime built on Chrome's V8 engine for server-side scripting.
  - **Express.js:** A minimal and flexible Node.js web application framework.
  - **Multer:** For handling file uploads (e.g., employee profile images).
- **Database:**
  - **MySQL/PostgreSQL:** Relational database management systems used for storing and managing data.

4.Database Design

4.1 Overview of the Database

The database is a critical component of the "Apna Employee" system, storing all the necessary data related to employees, categories, and admin

credentials. The database design follows a relational model, ensuring data integrity, consistency, and efficient querying.

4.2 Key Tables and Structures

1. **Admin Table:**

Column Name	Data Type	Description
id	INT	Primary key, auto-incremented
email	VARCHAR	Admin's email, unique
password	VARCHAR	Hashed password

2. **Description:** This table stores the credentials and metadata for all admin users who can access the admin panel.

3. **Employee Table:**

Column Name	Data Type	Description
id	INT	Primary key, auto-incremented
name	VARCHAR	Employee's full name
email	VARCHAR	Employee's email, unique
password	VARCHAR	Hashed password
salary	DECIMAL	Employee's salary
address	VARCHAR	Employee's address
category_id	INT	Foreign key referencing the Category ID
image	VARCHAR	Path to the employee's profile image

4. **Description:** This table holds detailed information about all employees, including their personal details, category affiliation, and profile image.
5. **Category Table:**

Column Name	Data Type	Description
id	INT	Primary key, auto-incremented
name	VARCHAR	Name of the category (e.g., HR, IT)

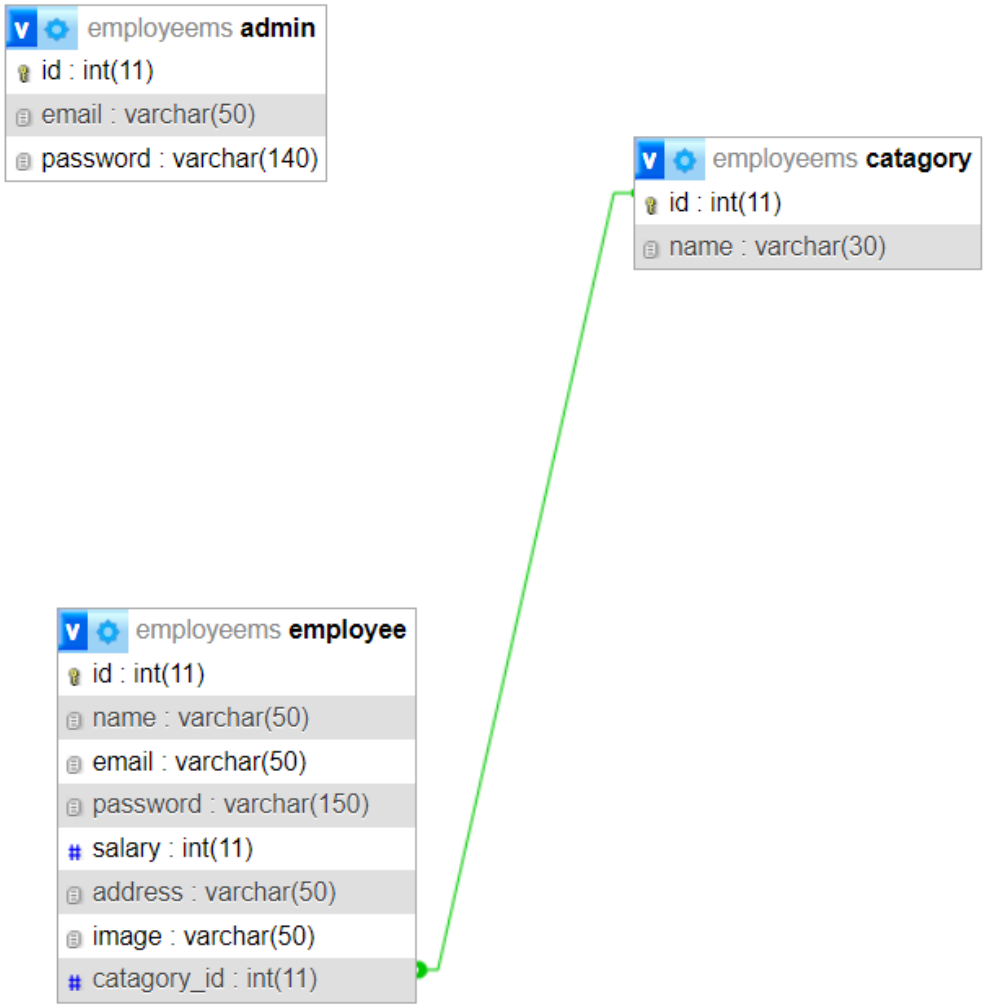
Column Name	Data Type	Description
description	TEXT	Detailed description of the category

6. **Description:** This table stores the different categories available in the system, which are used to classify employees based on their roles or departments.

4.3 Relationships Between Tables

- **Admin and Employee:** There is no direct relationship between the Admin and Employee tables. Admins manage the employee data but are not stored in the Employee table.
- **Employee and Category:** Employees are assigned to categories via a foreign key in the employee_id column, which references the category_id in the Category table. If multiple categories are allowed, the Employee_Category table is used to manage this relationship.
- **Employee and Employee_Category:** This table handles the many-to-many relationship between the Employee and Category tables, where each employee can belong to one or more categories, and each category can include multiple employees.

4.4 Entity-Relationship Diagram (ERD)



This diagram will visually represent the entities (tables) within the database and their relationships, providing a clear overview of how data is structured and interlinked in the "Apna Employee" system.

5. Backend Development

5.1 Technologies Used

The backend of the "Apna Employee" system is built using the following technologies:

- **Node.js**: A JavaScript runtime built on Chrome's V8 JavaScript engine, used for building fast and scalable server-side applications.

- **Express.js**: A web application framework for Node.js, providing a robust set of features for web and mobile applications.
- **MySql**: A sql database to storing system's data, it offers flexablity to to data storage
- **Multer**: A Node.js middleware for handling multipart/form-data, which is primarily used for uploading files (e.g., profile images).

5.2 Backend Architecture

The backend architecture follows a standard Model-View-Controller (MVC) pattern:

- **Model**: Represents the data structure of the application, with Mongoose schemas defining the structure of the data in MongoDB.
- **View**: Not applicable in this context as the backend handles data and logic, not user interfaces.
- **Controller**: Contains the logic for handling incoming requests, interacting with the models, and sending the appropriate responses.

5.3 API Design

The backend exposes a set of RESTful APIs that allow the frontend to interact with the system. Below are some of the key API endpoints:

1. **Admin Authentication**
    - **POST /api/admin/login**
        - **Description**: Authenticates an admin user and generates a JWT token for session management.
        - **Request Body**: { "email": "admin@example.com", "password": "adminpassword" }
        - **Response**: { "token": "jwt-token", "message": "Login successful" }
2. **Employee Management**
    - **GET /api/employees**
        - **Description**: Retrieves a list of all employees.
        - **Response**: [{ "id": 1, "name": "John Doe", "email": "john@example.com", ... }]
    - **POST /api/employees**
        - **Description**: Adds a new employee to the system.

        - **Request Body**: { "name": "Jane Doe", "email": "jane@example.com", "salary": 50000, ... }
        - **Response**: { "message": "Employee added successfully" }
    - **GET /api/employees/**
        - **Description**: Retrieves details of a specific employee.
        - **Response**: { "employee_id": 1, "name": "John Doe", "email": "john@example.com", ... }
    - **PUT /api/employees/**
        - **Description**: Updates details of a specific employee.
        - **Request Body**: { "name": "John Smith", "salary": 55000, ... }
        - **Response**: { "message": "Employee updated successfully" }
    - **DELETE /api/employees/**
        - **Description**: Deletes a specific employee from the system.
        - **Response**: { "message": "Employee deleted successfully" }
3. **Category Management**
    - **GET /api/categories**
        - **Description**: Retrieves a list of all categories.
        - **Response**: [{ "id": 1, "name": "HR",... }]
    - **POST /api/categories**
        - **Description**: Adds a new category to the system.
        - **Request Body**: { "name": "Finance", }
        - **Response**: { "message": "Category added successfully" }
    - **PUT /api/categories/**
        - **Description**: Updates a specific category's details.
        - **Request Body**: { "name": "IT", "description": "Information Technology" }
        - **Response**: { "message": "Category updated successfully" }
    - **DELETE /api/categories/**
        - **Description**: Deletes a specific category from the system.
        - **Response**: { "message": "Category deleted successfully" }

5.4 Core Functionalities Implementation

1. **Authentication**
    - Admin and Employee authentication is handled using JWT (JSON Web Tokens). Upon successful login, a JWT token is generated

and returned to the client, which must be included in the headers of subsequent requests to protected endpoints.

2. **Role-Based Access Control (RBAC)**
    - The system enforces role-based access control, ensuring that admins can manage employees and categories, while employees can only view their own details.
3. **Data Validation and Error Handling**
    - The system implements robust data validation to ensure the integrity of the data being processed. Any invalid data or unauthorized access attempts trigger appropriate error responses.
4. **File Uploads**
    - The Multer middleware is used to handle file uploads, such as profile images for employees. These files are stored in the designated public/Images directory with a custom naming convention.

5.5 Security Considerations

- **Password Hashing**: Passwords are hashed using bcrypt before being stored in the database, ensuring that even if the database is compromised, the passwords remain secure.
- **Input Sanitization**: All inputs are sanitized to prevent SQL injection and other common security vulnerabilities.
- **HTTPS**: The system is configured to run over HTTPS to ensure encrypted communication between the client and server.

6. Frontend Development

6.1 Technologies Used

The frontend of the "Apna Employee" system is built using the following technologies:

- **React.js**: A JavaScript library for building user interfaces, particularly single-page applications (SPAs).
- **Vite**: A fast development build tool that provides a smooth development experience with hot module replacement (HMR) and other modern features.

- **CSS3**: Cascading Style Sheets for styling the user interface, ensuring a responsive and visually appealing design.
- **Bootstrap**: A front-end framework that provides pre-designed UI components and responsive grid systems.

6.2 Frontend Architecture

The frontend of the "Apna Employee" system follows a component-based architecture, which is a hallmark of React.js development. Each part of the user interface is broken down into reusable components.

- **Components**: Reusable pieces of UI, such as buttons, forms, and tables. Examples include LoginForm, Dashboard, EmployeeList, and CategoryForm.
- **Routing**: Handled by react-router-dom, enabling navigation between different views, such as login pages, dashboards, and employee management screens.

7. Testing

7.1 Types of Testing

1. **Unit Testing**:
    - **Purpose**: To verify the functionality of individual components or modules, such as controllers, models, and utility functions.
    - **Tools Used**: Jest, Mocha, and Chai.
    - **Scope**: Test cases are written for each function to ensure they return the expected outputs given a set of inputs.
2. **Integration Testing**:
    - **Purpose**: To ensure that different modules or services work together as expected.
    - **Tools Used**: Supertest (for HTTP assertions), Mocha, and Chai.
    - **Scope**: Tests focus on the interaction between the frontend and backend, such as API endpoint responses and database operations.
3. **End-to-End (E2E) Testing**:
    - **Purpose**: To simulate real-world scenarios and test the application from start to finish, covering the entire flow from the user's perspective.

- **Tools Used**: Cypress and Selenium.
- **Scope**: Tests include logging in as an admin or employee, performing CRUD operations, and verifying the correct display of dashboard statistics.

4. **Security Testing**:
   - **Purpose**: To identify potential vulnerabilities and ensure the application is secure against threats such as SQL injection, XSS, and unauthorized access.
   - **Tools Used**: OWASP ZAP, Burp Suite.
   - **Scope**: The system is tested against various attack vectors, including penetration testing and vulnerability scanning.

8. Conclusion

The "Apna Employee" system represents a comprehensive and efficient solution for managing employee and administrative operations within an organization. Through its user-friendly interface, the system streamlines the management of employee data, role-based access, and category-specific information, ensuring that both admins and employees can interact with the platform with ease.

This project highlights the importance of modern web technologies, integrating a robust backend with a responsive frontend to deliver a seamless user experience. The system's flexibility allows for future scalability, making it adaptable to the evolving needs of any organization.

Furthermore, the thorough testing procedures implemented throughout the development process ensure that the system is reliable, secure, and performs well under various conditions. The use of automated testing and continuous integration tools has also facilitated rapid development cycles and ensured high code quality.

9. References

1. **Express.js Documentation**
   - Description: Official documentation for Express.js, the web application framework used in the project.
   - URL: https://expressjs.com/
2. **React.js Documentation**

- Description: Official documentation for React.js, the JavaScript library used for building the frontend.
- URL: https://reactjs.org/

3. **Node.js Documentation**
   - Description: Official documentation for Node.js, the JavaScript runtime used for the backend server.
   - URL: https://nodejs.org/
4. **Multer Documentation**
   - Description: Official documentation for Multer, the middleware used for handling file uploads in the project.
   - URL: https://github.com/expressjs/multer

5. **MySQL Documentation**

   Description: Official documentation for MySQL, the database management system used for storing project data.

   URL: https://dev.mysql.com/doc/

10.Source code:

Git hub repository: https://github.com/swaroop897/apna_employee.git