PROJECT REPORT ON

EMPLOYEE MANAGEMENT SYSTEM

SUBMITTED BY

AMISH MANJUL
(12022002016015)

ADARSH PRASAD
(12022002016040)

UNDER THE SUPERVISION AND GUIDANCE OF

DR. DEEPSHUBRA GUHA ROY

ASSOCIATE PROFESSOR, DEPARTMENT OF CSE(AIML),
INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA.

PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE FIFTH SEMESTER

ACADEMIC YEAR: 2022-26

SUBMITTED TO

INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING), INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA

1. CERTIFICATE OF RECOMMENDATION

We hereby recommend that the project prepared under our supervision by **AMISH MANJUL**, **ADARSH PRASAD** entitled <EMPLOYEE MANAGEMENT SYSTEM> be accepted in partial fulfilment of the requirements for the degree of partial fulfilment of the fifth semester.

Head of the Department
CSE(AIML),IEM, Kolkata

Project Supervisor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING), INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA

DECLARATION

We AMISH MANJUL, ADARSH PRASAD, hereby declare that this project report entitled “Online Action Platform” which is being submitted to Institute of Engineering and Management (IEM), Kolkata , in partial fulfilment of the requirements for the fifth semester is an authentic record of our genuine work done under the guidance of **Dr.Deepsubhra Guha Roy**, Associate Professor, Department of CSE (AIML), Institute of Engineering and Management (IEM), Kolkata, Salt Lake, India, during the academic year 2022-2026.

AMISH MANJUL [12022002016015]
ADARSH PRASAD [12022002016040]

Date:…….

Place:……

ACKNOWLEDGEMENT

We would like to sincerely thank all the people directly or indirectly involved in this project without whose help the completion of this project would have been impossible.

I would like to express my sincere gratitude to my DBMS faculty Dr.Deepsubhra Guha Roy sir who in spite of his busy schedule took keen interest in giving me all the necessary information for developing a good project and for his unlisted encouragement and timely support till my successful completion of the project work.

We thank Deepsubhra sir who has guided us and provided support and genuine replies to our queries. He has not only guided us in the project but also helped us by providing suggestions without which we couldn't improve at work. HIS certain suggestions and interest in helping us complete the project has led me to the successful result of my project and also all the other Lecturers and lab instructors who have been more of a support to us than ever.

We thank whole heartedly to our beloved Prof. Amartya Mukherjee, Head of CSE(AIML) for granting permission to undertake the project. Finally, we thank our parents, who were there by us, financially supported throughout the project and provided the support we needed throughout my life.

CHAPTER 1

1.1 Title of the project

EMPLOYEE MANAGEMENT SYSTEM

1.2 Objective of the project

The objective of an Employee Management System (EMS) is to improve the efficiency of HR processes and enhance organisational productivity. This system centralizes employee data, allowing HR and management to access information quickly, manage payroll and attendance accurately, and streamline performance evaluations. By automating routine tasks, EMS reduces the likelihood of errors, ensures compliance with labor laws, and enhances communication between employees and management. Additionally, it provides tools for performance tracking and feedback, helping to support employee development. Overall, EMS serves as a comprehensive tool to support decision-making and optimize workforce management, fostering a productive and compliant workplace environment.

1.3 PROJECT CATEGORY

Web application

1.4 Programming languages used

JAVASCRIPT(REACT.js), MYSQL

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

1.5.1 HARDWARE REQUIREMENT

- ➢ Processor - intel pentium IV
- ➢ Processor speed - 1.40 gHz
- ➢ RAM - 2GB or above
- ➢ Monitor Resolution - 1000*700 or more

1.5.2 SOFTWARE REQUIREMENT

- ➢ IDE: VSCODE
- ➢ FRONT-END : REACT.JS, NODE.JS, HTML, CSS , JAVASCRIPT
- ➢ BACKEND : MySQL server 5.1.1

1.6 Structure of the program

1.6.1 ANALYSIS

The purpose of this system is to manage employee records efficiently, including their personal details, attendance, payroll, and performance. The EMS must provide functionalities for:

- Adding, updating, and deleting employee information.
- Tracking attendance and payroll.
- Managing performance evaluations and compliance reports.

1.6.2 Module Descriptions

a) Frontend Modules (React)

- **Login and Authentication Module:**

Users, such as HR personnel or managers, log in to access the EMS.
Authentication can be handled using JWT (JSON Web Tokens) or sessions.

- **Dashboard Module:**

Acts as the central hub for all EMS operations, displaying links to all other modules.

- **Employee Management Module:**

Form for adding or updating employee information (e.g., name, contact info, department).
Interface to search, view, and delete employee records.

- **Attendance Module:**

Allows employees to mark attendance or view their attendance records.
Calendar view for managers to see attendance over time.

- **Payroll Management Module:**

Interface to calculate, view, and update payroll for each employee.
Function to generate payroll reports.

- **Performance Review Module:**

Interface for adding, updating, and viewing performance reviews.

- **Reports Module:**

Generates detailed reports (e.g., attendance, payroll, performance) and exports as PDF or Excel.

b) Backend Modules (Express.js + MySQL)

- **Employee Controller:**
  APIs to manage employee records (addEmployee, updateEmployee, getEmployee, deleteEmployee).
- **Attendance Controller:**
  APIs for marking attendance, viewing records, and generating attendance reports.
- **Payroll Controller:**
  APIs to manage payroll calculations and fetch payroll records.
- **Performance Controller:**
  APIs for adding performance evaluations, viewing history, and updating performance data.
- **Report Generator:**
  API for generating and exporting various reports based on filtered data.

\

1.7 LIMITATION

The research papers have privacy issues related to it. So a higher level of security must be ensured to the system.

1.8. FUTURE SCOPE

Future scope for an Employee Management System includes advanced analytics, integrations with payroll and project management tools, mobile app access, an employee self-service portal for tasks like leave requests, and enhanced security measures like data encryption and role-based access control to improve usability, accessibility, and compliance.

CHAPTER 2

SOFTWARE REQUIREMENTS SPECIFICATIONS

2.1 INTRODUCTION

A software requirements specification (SRS) is a description of a software system to be developed, laying out functional and non-functional requirements, and may include a set of use cases that describe interactions the users will have with the software.

A basic purpose of the SRS is to bridge this communication gap between client and the developer so they have a shared vision of the software being built. An SRS establishes the basis for agreement between the client and the supplier on what the software product will do. SRS provides a reference for validation of the final product. A high-quality SRS is a prerequisite to high-quality software and also reduces the development cost.

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references and overview of the SRS. The aim of this document is to gather and analyze and give an in-depth insight of the complete “EMPLOYEE MANAGEMENT” by defining the problem statement in detail. The detailed requirements of “EMPLOYEE MANAGEMENT” are provided in this document.

2.2 PURPOSE:

A Software Requirements Specification (SRS) is a document that describes the software system to be developed, detailing both functional and non-functional requirements, and may include use cases that outline user interactions with the software. The main purpose of an SRS is to bridge the communication gap between the client and developer, ensuring a shared understanding of the software being built. The SRS establishes an agreement between the client and the supplier on the software product’s functionality, serving as a reference for validating the final product. A high-quality SRS is essential for developing high-quality software and helps reduce development costs.

The introduction of the SRS for the Employee Management System provides an overview of the SRS, including its purpose, scope, definitions, acronyms, abbreviations, references, and structure. The aim of this document is to gather, analyze, and provide a comprehensive insight into the “Employee Management System” by defining the problem statement in detail. Detailed requirements for the “Employee Management System” are outlined in this document.

2.2.1 SCOPE

This Employee Management System (EMS) project includes the following features and scope:

- Centralized Online Platform: The system operates online, allowing HR personnel and employees to access it from anywhere, at any time.

- Employee Registration and Authentication: All users must register to access the system. Only authenticated users can view and update their profiles, access payroll information, and manage attendance.

- Profile and Performance Management: Employees can view their profiles, track attendance history, and check performance reviews. Managers can monitor employee performance and update records as needed.

- Notifications and Alerts: Employees receive email notifications for updates on attendance records, payroll status, performance reviews, and other important events.

EMS is designed to streamline HR processes, providing easy access and real-time updates for both employees and managers.

2.2.2 Description, Acronyms and Abbreviation:

- SRS : Software Requirements Specification
- OS : Operating System
- DBMS : Database management system
- SQL : Structured Query Language
- JS : Javascript
- RAM : Random Access Memory
- MB : Mega Bytes

2.2.3 References:

- www.w3school.com
- www.tutorialpoints.com
- www.stackoverflow.com
- An integrated approach to Software Engineering by PANKAJ JALOTE
- www.geeksforgeeks.com

2.2.4 Overview :

The Employee Management System (EMS) is an online platform designed to streamline HR operations by providing efficient access to employee data, attendance, payroll, and performance records. Users can securely log in to view and update profiles, monitor performance, and manage attendance. Managers can track employee progress, generate

reports, and oversee payroll. Automated notifications keep employees informed about updates, enhancing communication and productivity within the organization.

2.3 Overall Description:

The Employee Management System (EMS) is a comprehensive platform that simplifies HR tasks by centralizing employee data, attendance tracking, payroll, and performance management. It enables secure login for employees and managers, allowing access to profiles, performance metrics, and attendance records. Managers can generate detailed reports, oversee payroll, and track employee progress, while automated notifications keep users informed of essential updates. This system improves organizational efficiency, streamlines communication, and enhances productivity**.'**

2.4 Product Perspective :

The Employee Management System (EMS) is a web-based solution designed to streamline HR processes, enhance data accessibility, and improve communication between employees and management.

2.5 PRODUCT FEATURES :

1. Employee Profile Management: Secure storage and easy access to employee details, including personal information, job role, and contact data.

2. Attendance and Leave Tracking: Automated tracking of employee attendance, leave requests, and approvals.

3. Payroll Management: Calculation of salaries, deductions, bonuses, and generation of payslips.

4. Performance Evaluation: Tools for managing and tracking employee performance reviews and feedback.

2.6 USER CLASSES AND CHARACTERISTICS:

In the Employee Management System (EMS), there are two primary user classes: **Admin** and **Employee**.

Admin:

Manages the system, oversees employee data, approves leave requests, tracks attendance, generates payroll, and performs performance evaluations. Admins have full access to all functionalities and data within the system.

Employee:

Can view and update personal information, track attendance, request leave, and view payroll and performance reviews. Employees have restricted access based on their role and data privacy.

2.7 Design and implementation constraints:

The developed Employee Management System (EMS) should run on any platform (Unix, Linux, Mac, Windows, etc.) with a modern web browser that supports React.js, JavaScript, and AJAX.

- Internet connectivity is required for features such as email notifications and system updates.
- The user accessing the system must be authenticated and authorized.
- MySQL database is used as the backend to store and manage data.

The system includes the following tables in the Employee Management Database:

- **Employee**
- **Attendance**
- **Payroll**
- **Performance**
- **User (for authentication)**
- **Leave Requests**
- **Departments**
- **Notifications**

2.8 Assumptions and Dependencies:

Assumptions:

1. Users will have internet access and compatible devices.
2. All users must be registered and authenticated.
3. Role-based access control will be followed.
4. Employees will have restricted access, and admins will have full access.

Dependencies:

1. MySQL for database management.
2. React.js for frontend development.
3. Server-side technology (Node.js or similar).
4. Internet for email notifications and updates.
5. Secure authentication mechanisms (JWT/OAuth).

2.9 Specific Requirements:

2.9.1 External Interface Requirements:

This section outlines the inputs and outputs of the Employee Management System (EMS). The system will accept inputs such as employee data, attendance records, payroll details, and performance feedback. Outputs will include employee profiles, reports, notifications, and data visualizations.

2.9.2 User Interfaces:

The user interface will be designed to be intuitive and user-friendly. The system will use fast-loading fonts and buttons to ensure responsiveness. Pages will be optimized to load quickly, maintaining a clean and efficient design to enhance user experience.

2.10 Hardware Interfaces:

- Operating System: Unix, Linux, Mac, Windows, etc.
- Processor: Intel Pentium or higher.
- RAM: 312MB or higher.
- Monitor: 14” or larger.
- Input Devices: Keyboard and mouse.

2.11 Software Interfaces:

- Development Tools: React.js, JavaScript, AJAX.
- Web Server: Apache or similar.
- Database Server: MySQL.
- IDE: Visual Studio Code, Sublime Text, or similar code editor.

2.12 Communication Interfaces:

Internet connection is required.

2.13 FUNCTIONAL REQUIREMENTS:

Functional Dependencies of the Employee Management System (EMS):

2.13.1. User Authentication & Role Management:

Login → Employee Login (Authenticated users can access the system)
Employee Login → Role-Based Access (Admin, HR, Employee roles define system access)
Admin Login → Full System Access (Admins have access to all modules, including employee data, reports, and settings)
Employee Login → Restricted Access (Employees can only view and manage their own data)

2.13.2. Employee Management:

AddEmployee → Employee (New employee records are created and added to the system)
EditEmployee → Employee Profile (Employee details can be updated based on their ID or unique attributes)
Employee Profile → Attendance, Payroll, Performance Records (Employee profile links to their attendance, payroll, and performance evaluation data)

2.13.3. Attendance Management:

Employee → Attendance (Employees' attendance records are created when they log in and mark their attendance)
Attendance → Payroll Calculation (Attendance records influence salary computation, including deductions for absences or overtime)
Leave Requests → Attendance (Leave requests affect attendance tracking and may update the attendance record with leave status)

2.13.4. Payroll Management:

Employee → Payroll (Employee payroll details are generated based on attendance, deductions, and other financial information)
Payroll → Employee Profile (Payroll is linked to the employee profile for easy access to salary details)
Payroll Calculation → Tax Deductions (Salary calculations may trigger automatic deductions based on applicable tax rules)

2.13.5. Performance Evaluation:

Employee → Performance Review (Employees can be evaluated based on predefined metrics like job performance and work quality)
Performance Review → Employee Profile (Performance reviews are stored as part of the employee's profile, affecting career growth, promotions, and feedback)

2.13.6. Leave Management:

Employee → Leave Requests (Employees can apply for various types of leaves, such as vacation or sick leave)
Leave Requests → Employee Profile (Leave requests are logged under the employee's profile and affect attendance calculations)
Leave Approval → Attendance Records (Managers or HR must approve leave requests, updating attendance records accordingly)

2.13.7. Notification System:

Employee Profile → Notifications (Employees receive notifications for leave approval, payroll updates, and performance reviews)
Admin Profile → System Notifications (Admins are notified about system updates, new registrations, and performance feedback)
Employee Actions → Email Notifications (Actions like leave approvals or performance feedback trigger email notifications to employees)

2.13.8 Reporting and Analytics:

Employee Profile, Attendance, Payroll, and Performance → Reports (Admins and HR can generate comprehensive reports on employees, including attendance summaries, payroll, and performance reviews)
Employee Data → Analytics Dashboard (Data analytics modules provide visual insights into employee performance, attendance trends, and payroll analysis)

2.13.9 Profile Management:
Employee Profile → Personal Information (Employees can view and update their personal details, contact information, and emergency contacts)
Admin → Employee Data (Admins can update or modify employee profiles, including job titles, department assignments, and salary information)

2.13.10 Role-Based Access Control:
User Role → System Access (Based on roles, users are granted access to specific modules, such as admins having access to all features and employees having limited access to their own data)

2.13.11 Employee Directory:
Employee → Directory (The system should allow HR and admin users to search for and view an organized list of all employees, including personal details, job titles, and departments)

2.13.12 System Backup and Data Security:
Employee Data → Encrypted Storage (Employee information, including sensitive data like salary and performance reviews, is encrypted for secure storage)
Backup → Employee Data (Automated backups of the system data ensure that no critical employee information is lost in case of system failure)

CHAPTER 3

SYSTEM DESIGNS

3.1 Introduction

Design is processes through which requirements are translated into a representation of the software .The purpose of the designing phase is to plan a solution for the problem specified by the requirement document i.e. the requirement are translated into software. ‘

The design activities often result in three separate outputs:

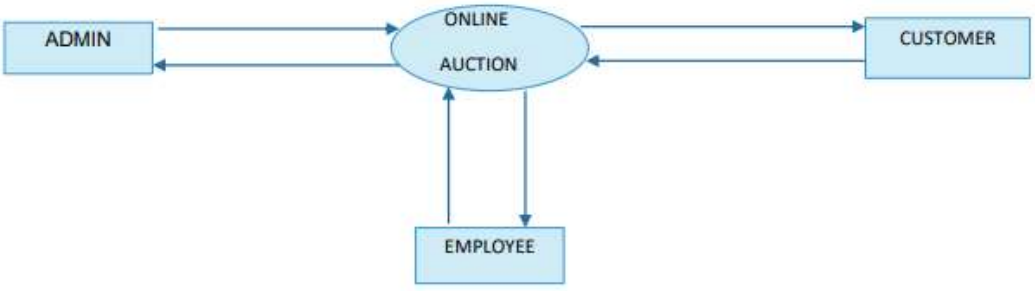
1. Architecture Design
2. High Level Design
3. Detailed Design System

Design is a solution a “How to “approach to the creation of a new system. The importance phase is composed of several steps. It provides the understanding of procedural detail necessary for the system recommended in the feasibility study. Emphasis is on translating the performance requirement into design specification

3.2 System Context Diagram

The System Context Diagram or Context Flow Diagram (CFD) describes the external entities acting on the system. The environment in which the system is used is depicted in the figure

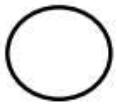
Context Flow Diagram (CFD)



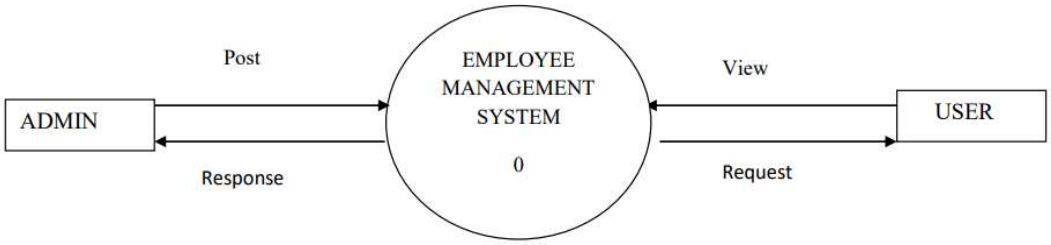
3.3 Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of the data through an information System. A DFD also can be used for the visualization of data processing. It is common practice for design to draw a context level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then "exploded" to show more detail of the system being modelled.

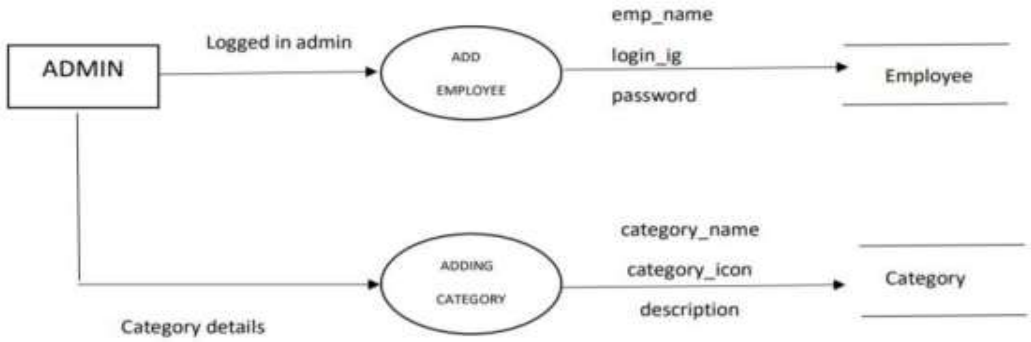
The DFD uses four symbols, and explained below:

A SQUARE 	Which defines the source or destination of system data also called an external entity, is not responsible for any task performed by the System.
An ARROW	Represents data flow. It represents the path over which data travels in the system. A data can move between processes,flow into or out of the stores to and from external entities.It must be given a name the arrow head showing the direction of flow
CIRCLE 	Represents a process that transforms data from one to another by performing some tasks with the data. The process name must be given a general idea of its function
HORIZONLPARALLEL LINES 	Represents data store, a data store is place where data is held temporarily from one transaction to the next or is permanently Data Flow Diagram describes what data flow (logical)rather than how they are processed, so it does not depend on hardware, software, data structure or file organization

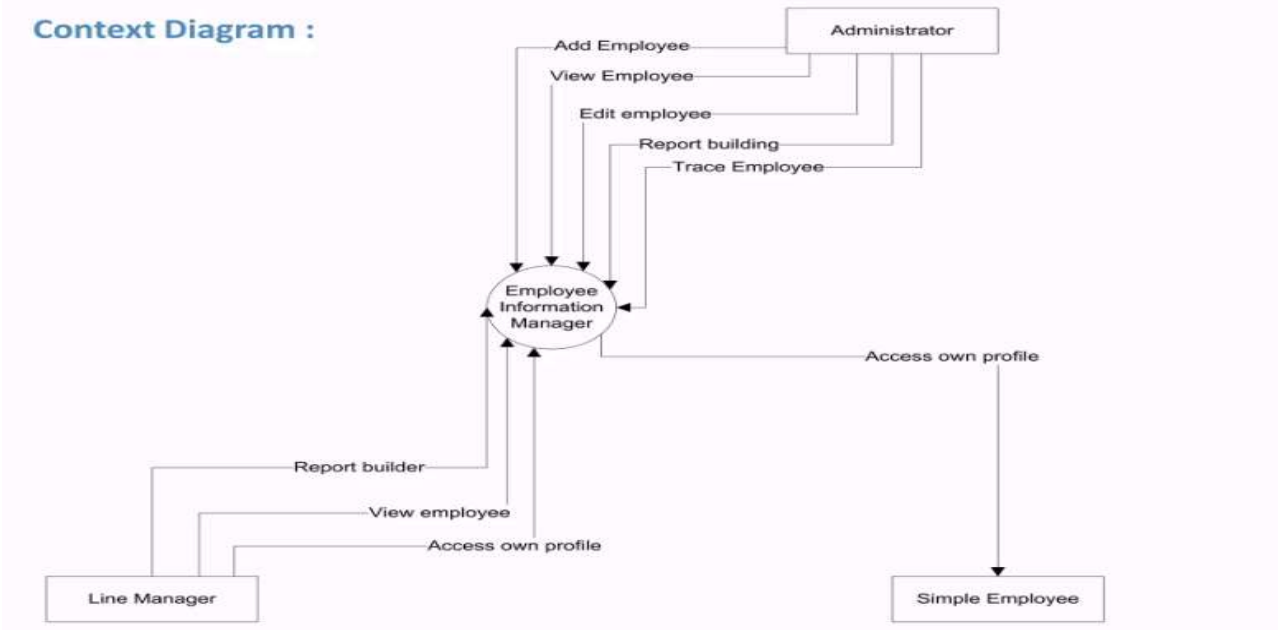
3.3.0 DFD For Level 0 :



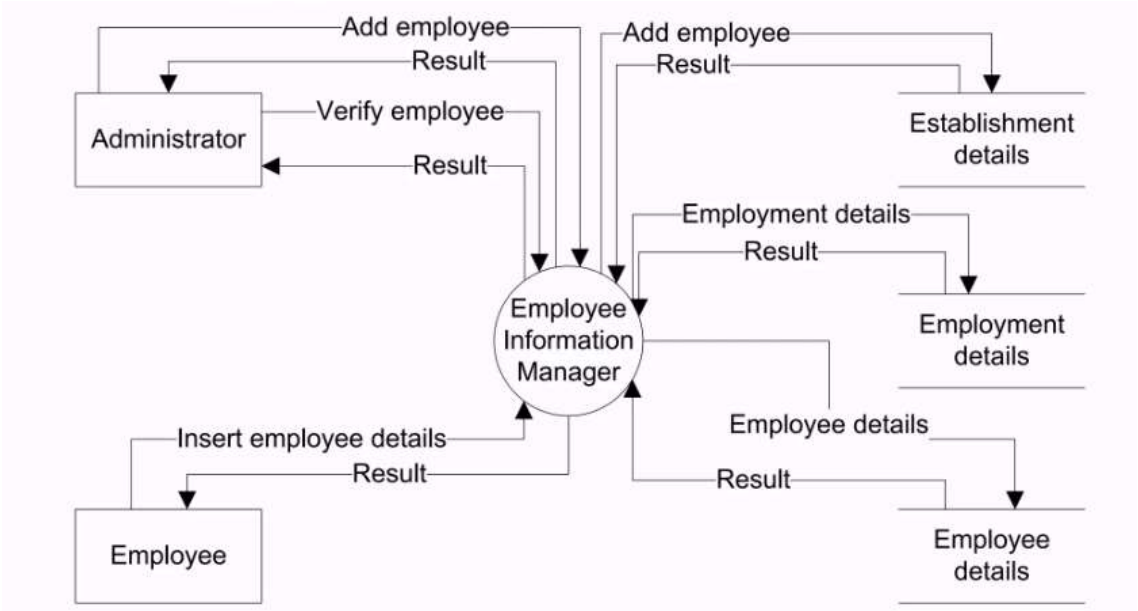
3.3.1 DFD for Level 1 Admin:



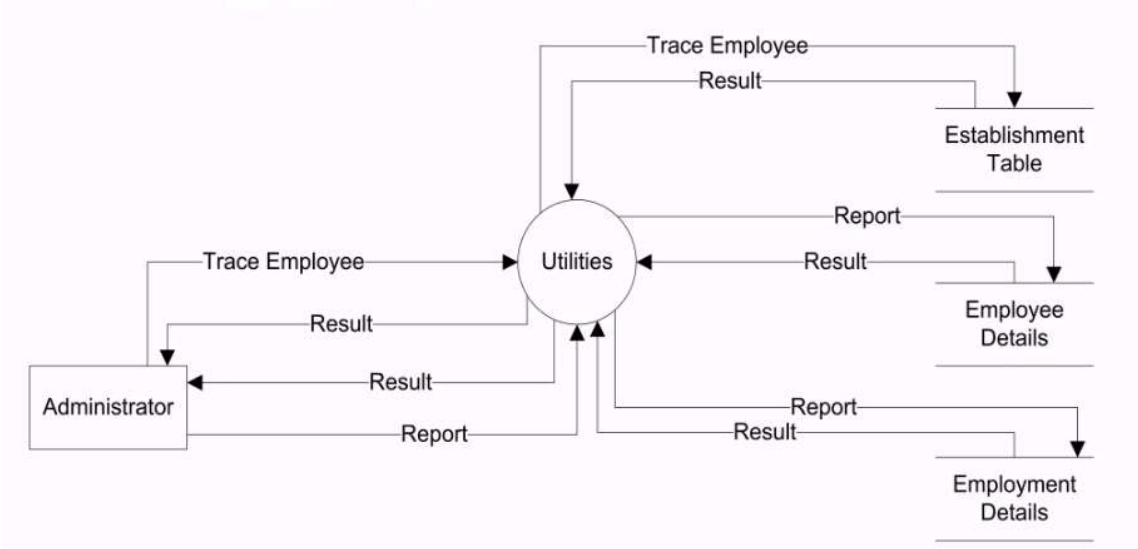
3.3.2 DFD for Level 1 EMPLOYEE:



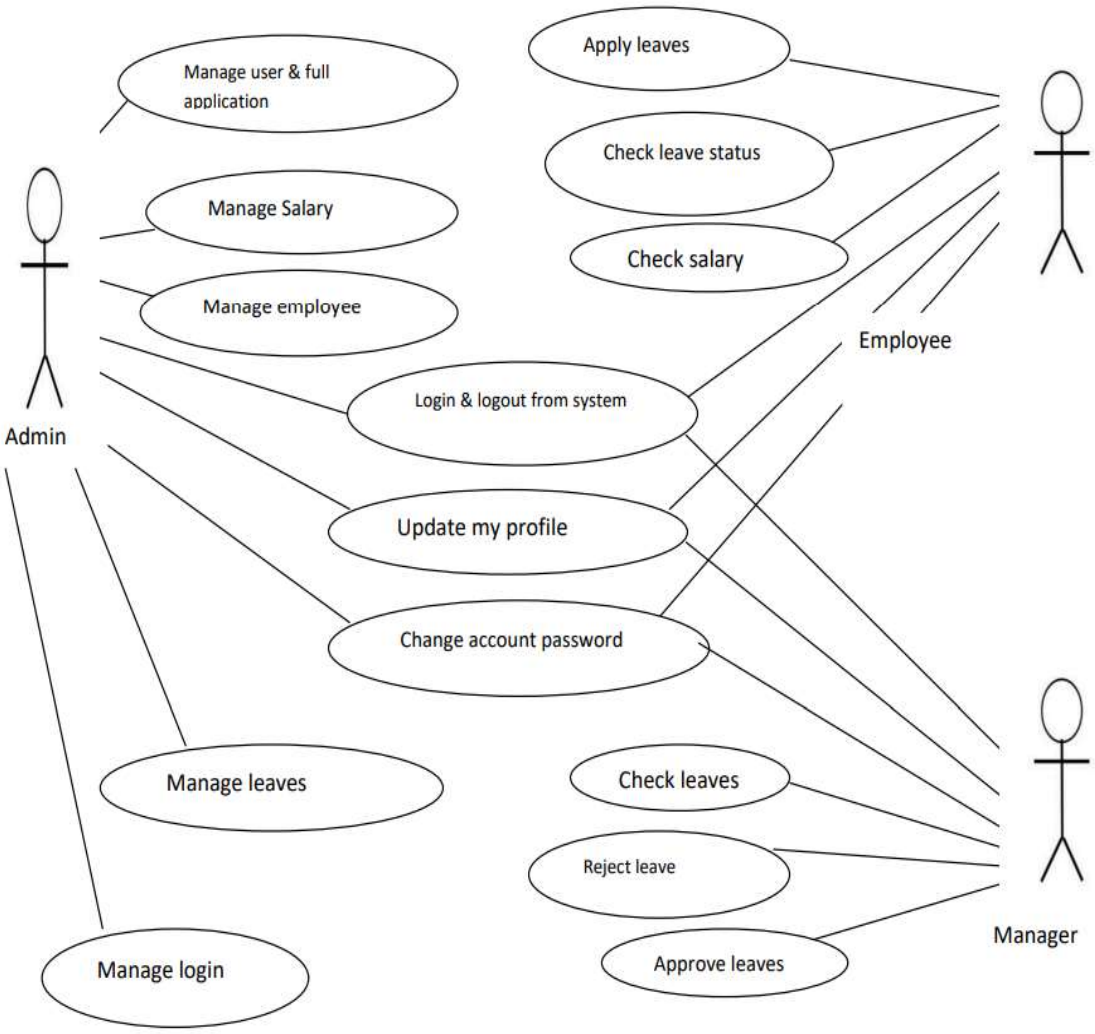
3.3.3 DFD for ADDING EMPLOYEE



3.3.4 DFD for tracing Employee:



3.4 UML DIAGRAM For Employee Management System



CHAPTER 4

DATABASE DESIGN

4.1 INTRODUCTION

The description of the database is called the database scheme. A database scheme is specified during database creation and is not accepted to change frequently. Most data model has certain conversations for diagrammatically

4.1.1. Internal level

displaying scheme. The scheme diagram displays only some aspects of the scheme, such aspects are not specified in the scheme diagram, that is neither the data type of each data item changes frequently.

The data in the database at a particular moment in timer is called a database state. Schema can be defined in the following three levels.

The internal Level has an internal scheme. It describes the physical storage structure of the database. The internal schema uses a particular data model and describes the complete details of the data storage and access paths of the database.

4.1.2. Conceptual level

The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of the physical storage structure and concentrates on describing entities, data types, relationship user operation and constraints. A high-level data model or an implementation data model can be used in this model.

4.1.3. External level

External level or view includes a number of external schema or view. Each internal schema describes the part interested in and hides the rest of the database from the user group. A high level data model or an implementation model can be used at this level.

4.2. WHAT IS DATABASE?

A database is an organized collection of data. It is the collection of schema tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information. Which can be of any size and complexity. By using the concept of a database, we can easily store and retrieve the data. The major purpose of the database is to provide the information, which utilizes data that the system needs according to its own requirements.

4.3. DATABASE DESIGN

Database design is the process of producing a detailed data model of a database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model, these are the tables and views. In an object database, the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).

To retrieve data from the database;

- The application program determines what data is needed and communicates the need to the DBMS.
- The data must be defined in the sub schema.
- The copy of the data is given to the operating system for processing.
- A database must be created before it can be used.

4.4 Table Schemas :

4.4.1 EMPLOYEE TABLE

Column Name	Data Type
employee_id	INT (Primary Key)
first_name	VARCHAR(50)
last_name	VARCHAR(50)
date_of_birth	DATE
email	VARCHAR(100) UNIQUE
phone_number	VARCHAR(15)
address	TEXT
hire_date	DATE
job_title	VARCHAR(100)
department_id	INT (Foreign Key)
salary	DECIMAL(10, 2)

4.4.2 DEPARTMENT/ CATEGORY TABLE

Column Name	Data Type
department_id	INT (Primary Key)
department_name	VARCHAR(100)
description	TEXT
created_at	TIMESTAMP
updated_at	TIMESTAMP

4.4.3 ATTENDANCE TABLE

Column Name	Data Type
attendance_id	INT (Primary Key)
employee_id	INT (Foreign Key)
date	DATE
status	ENUM('present', 'absent', 'leave', 'overtime')
leave_type	VARCHAR(50)
hours_worked	DECIMAL(5, 2)

4.4.5 PAYROLL TABLE

Column Name	Data Type
payroll_id	INT (Primary Key)
employee_id	INT (Foreign Key)
basic_salary	DECIMAL(10, 2)
bonus	DECIMAL(10, 2)
deductions	DECIMAL(10, 2)
net_salary	DECIMAL(10, 2)
salary_month	DATE
created_at	TIMESTAMP

4.4.6 USER TABLE

Column Name	Data Type
user_id	INT (Primary Key)
employee_id	INT (Foreign Key)
username	VARCHAR(50) UNIQUE
password	VARCHAR(255)
role	ENUM('admin', 'hr', 'employee')
created_at	TIMESTAMP

4.4.7 EMPLOYEE DIRECTORY TABLE

Column Name	Data Type
directory_id	INT (Primary Key)
employee_id	INT (Foreign Key)
job_title	VARCHAR(100)
department_name	VARCHAR(100)
contact_info	TEXT
created_at	TIMESTAMP

4.5 ENTITY RELATIONSHIP DIAGRAM

Entity relationship diagram is used in modern database software. Software engineering is to illustrate logical structure of database. It is a relational schema database, modelling method, used to model a system and approach. This approach is commonly used in database design. The diagram created using this are called entity relationship diagram

. The ER diagram depicts the various relationship among entities, considering each object as an entity. Relationship depicts the relationship between data objects. The ERD is the notation that is used to conduct the data modelling activity.

Entity

Entity is a thing, which we want to store information. It is an elementary basic building block of storing information about business process. An entity represents an object desired within the information system abut which u want to store information.

Relationship

A relationship is a named connection, associated between entities, or used to relate two or more entities with some common attributes or meaningful interaction between the object.

Attributes

Attributes are the properties of entities and relationship. Description of the entity. Attributes are elementary pieces of information attached to an entity.

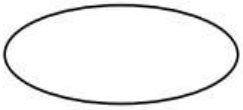
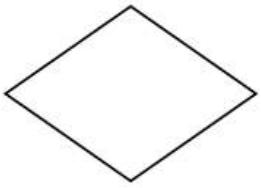
Link

Lines link attributes to entity set and entity sets to relation.

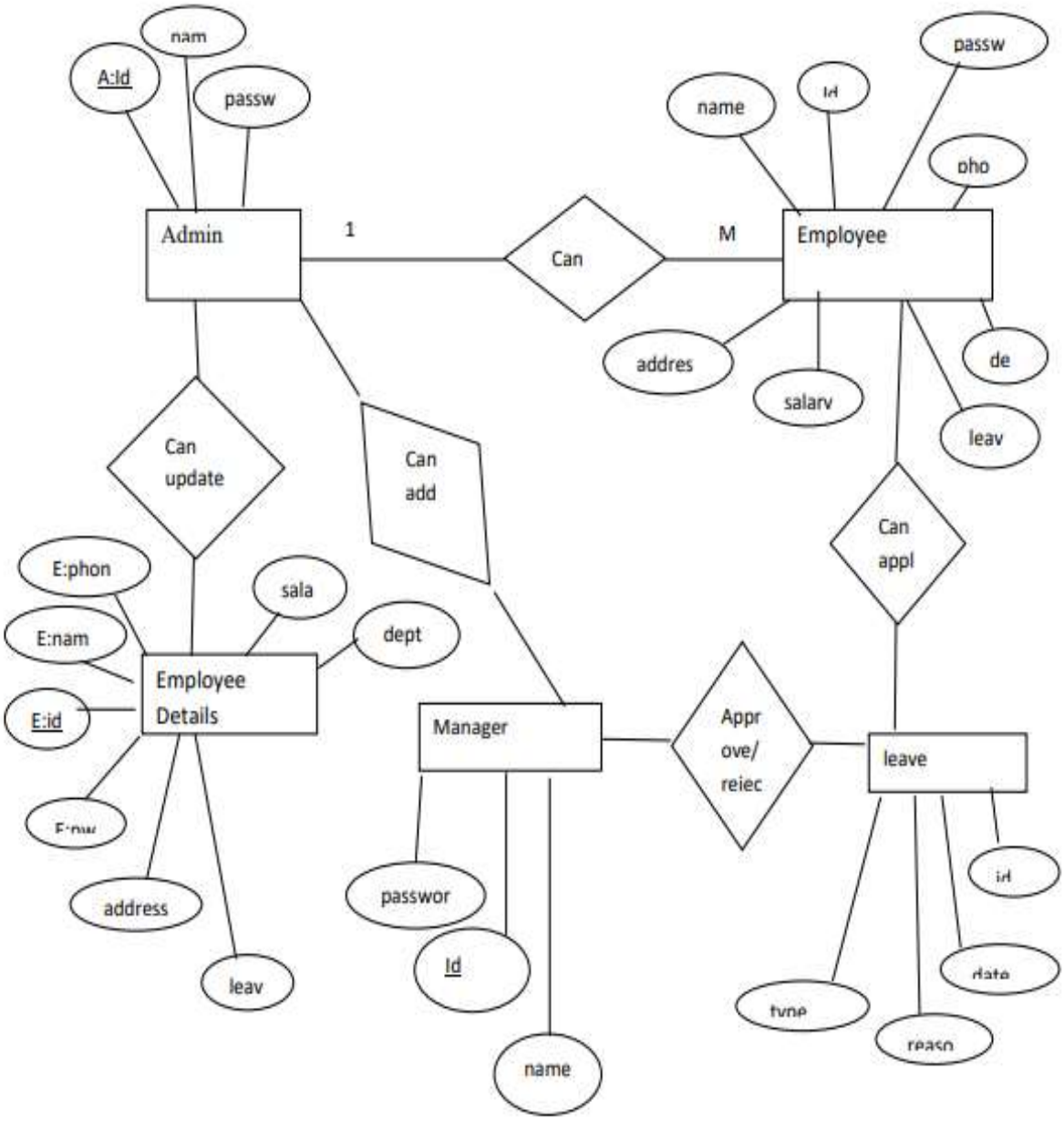
Cardinality Ratio

It specifies the maximum number of relationships instances that an entity can participate in. There are four cardinality ratios.

ER-DIAGRAM SYMBOLS:

Name	Notation	Description
Entity		It may be an object with the physical existence or conceptual existence. It is represented by a rectangle.
Attribute		The properties of the entity. Can be attribute. It is represented by a Ellipse.
Relationship		Whenever an attribute of one entity refers to another entity, some relationship exists. It is represented by a Diamond
Link		Lines link attributes to entity set and entity sets to relation
Cardinality Ratio	1:1 1:N N:1 M:1	It specifies the maximum number of relationships instances that an entity can participate in. There are four cardinality ratios.

ER DIAGRAM FOR EMPLOYEE MANAGEMENT SYSTEM:



CHAPTER 5

CODING

5.1 Introduction:

The goal of coding or programming phase is to translate the system design, produced during the designing phase, into code in a given programming language which can be executed by a computer and that performs the computation specified by the design. During the implementation, it should be kept in mind that the programs should not be constructed so that they are easy to write, but that they are easy to read and understand.

5.2 Programming practices:

5.2.1 Top-down & Bottom-up approaches:

In a top-down implementation, the implementation starts from the top of the hierarchy and proceeds to the lower levels. First the main module is implemented, and their subordinates, and so on. In a bottom-up implementation, the process is the reverse. The development starts with implementing the modules at the bottom of the hierarchy and proceeds through the higher levels until it reaches the top. Top-down and bottom-up implementation should not be confused with top-down and bottom-up design, here the design is being implemented, and if the design is fair detailed and complete, its implementation can proceed in either the top-down manner.

5.2.2 Structured Programming:

The basic objective of the coding activity is to produce programs are easy to understand. A program has a static structure as well as a dynamic structure. The static structure is the structure of the text of the program, which is usually just a line organization of statements of the program. The dynamic structure of the program is the sequences of statements executed during the execution of the program.
In other words both the static structure and the dynamic behavior are sequences of statements; when the sequence representing the static structure is fixed, the sequences of statements it executable can change from execution to execution.

5.2.3 Information Hiding:

A software solution to a problem always contains data structures that are meaningfully represent information in the problem domain. That is, when software is developed results a problem, the software uses some data structures to capture the information in the problem domain. Any software solution to a problem contains data structures that represent information in the problem domain. In the problem domain, in general, only certain operations are performed on some information. That is, a piece of information in the problem domain is used only in a limited number of ways in the problem domain.

5.2.4 Programming style:

Here we will list some general rules that can be applied for writing good code.

Names: Selecting module and variable names is often not considered important novice programmers. Most variable sin a program reflects some entity in the problem domain, and the modules reflect some process. Variable names should be closely related to the entity they represent, and module names should reflect their activity.

Control constraints: As discussed earlier, it is desirable that as much as possible single entry, single exit constructs be used. It is also desirable to use a few standard control constructs rather than using a wide variety of constructs, just because they are available in the language.

Information hiding: As discussed earlier, information hiding should be supported where possible. Only the access functions for the data structures should be made visible while hiding the data structures behind these functions. User-defined types: Modern languages allow users to define data types when such facilities are available, they should be exploited where applicable.

Nesting: The different control constructs, particularly the if-then-else, can be nested. If the nesting becomes too deep, the programs become harder to understand. In case of deeply nested if-then-elses, it is often difficult to determine if statement to which a particular else clause is associated. If possible, deep nesting should be avoided.

5.2.5 Verification:

Verification of the output of the coding phase is primarily interested for detecting errors introduced during this phase. That is, the goal of verification of the code produced is to show that the code is consistent with the design is supposed to implement. It should be avoided out that by verification we mean providing correctness of programs. Program verification method falls into two categoriesstatic and dynamic methods.

In dynamic methods, the program executes some test data and the outputs of the program are examined to determine if there are any errors present. Static techniques, on the other hand, do not involve actual program execution on actual numeric data, though it may involve some form of conceptual execution:

5.2.5.1 Code Reading:

Code reading involves careful reading of the code by the programmer to detect any discrepancies between the design specifications and the actual implementation. It involves determining the abstraction of a module and then comparing it with its specifications.

The process of code reading is best done by reading the code insideout, starting with the innermost structure of the module. First determine its abstract behavior and specify the

abstraction. Then the higher level structure is considered, with the inner structure replaced by its abstraction. This process is continued until we reach the module or program being read.

5.2.5.2 Static analysis:

Analysis of programs by methodically analyzing the program text is called static analysis is usually performed mechanically by the aid of software tools. During static analysis the program itself is not executed, but the program text is the input to the tools. The aim of the static analysis tools is to detect errors or potential errors or to generate information about the structure of the program that can be useful for documentation or understanding of the programs. An advantage is that static analysis sometimes detects the errors themselves, not just the presence of errors as in testing. This saves the effort of tracing the error from the data that reveals the presence of errors. Static analysis can provide insight into the structure of the program.

5.2.5.3 Symbolic execution:

Inputs to the program are not numbers but symbols representing the input data, which can take different values. The execution of the program proceeds like normal execution, except that it deals with values that are not numbers but formulates consisting of the symbolic input values. The outputs are symbolic formulae of input values. These formulae can be checked to see if the program will behave as expected. This approach is called as symbolic execution.

CODES:

LOGIN

```
import React, { useState } from 'react'
import './style.css'
import axios from 'axios'
import { useNavigate } from 'react-router-dom'

function Login() {

    const [values, setValues] = useState({
        email: '',
        password: ''
    })
    const navigate = useNavigate()
    axios.defaults.withCredentials = true;
    const [error, setError] = useState('')

    const handleSubmit = (event) => {
        event.preventDefault();
        axios.post('http://localhost:8081/login', values)
        .then(res => {
            if(res.data.Status === 'Success') {
```

```
            navigate('/');
          } else {
            setError(res.data.Error);
          }
        })
        .catch(err => console.log(err));
    }

    return (
      <div className='d-flex justify-content-center align-items-center vh-100 loginPage'>
        <div className='p-3 rounded w-25 border loginForm'>
          <div className='text-danger'>
            {error && error}
          </div>
          <h2>Login</h2>
          <form onSubmit={handleSubmit}>
            <div className='mb-3'>
              <label htmlFor="email"><strong>Email</strong></label>
              <input type="email" placeholder='Enter Email' name='email'
                onChange={e => setValues({...values, email: e.target.value})}
className='form-control rounded-0' autoComplete='off'/>
            </div>
            <div className='mb-3'>
              <label htmlFor="password"><strong>Password</strong></label>
              <input type="password" placeholder='Enter Password' name='password'
                onChange={e => setValues({...values, password: e.target.value})}
className='form-control rounded-0' />
            </div>
            <button type='submit' className='btn btn-success w-100 rounded-0'> Log
in</button>
            <p>You are agree to aour terms and policies</p>
          </form>
        </div>
      </div>
    )
}

export default Login
```

ADD EMPLOYEE

```
import axios from 'axios';
import React, { useState } from 'react'
```

```
import { useNavigate } from 'react-router-dom';

function AddEmployee() {
	const [data, setData] = useState({
		name: '',
		email: '',
		password: '',
		address: '',
		salary: '',
		image: ''
	})
	const navigate = useNavigate()

	const handleSubmit = (event) => {
		event.preventDefault();
		const formdata = new FormData();
		formdata.append("name", data.name);
		formdata.append("email", data.email);
		formdata.append("password", data.password);
		formdata.append("address", data.address);
		formdata.append("salary", data.salary);
		formdata.append("image", data.image);
		axios.post('http://localhost:8081/create', formdata)
		.then(res => {
			navigate('/employee')
		})
		.catch(err => console.log(err));
	}
	return (
		<div className='d-flex flex-column align-items-center pt-4'>
			<h2>Add Employee</h2>
			<form class="row g-3 w-50" onSubmit={handleSubmit}>
			<div class="col-12">
					<label for="inputName"
class="form-label">Name</label>
					<input type="text" class="form-control"
id="inputName" placeholder='Enter Name' autoComplete='off'
					onChange={e => setData({...data, name:
e.target.value})}/>
				</div>
				<div class="col-12">
					<label for="inputEmail4"
class="form-label">Email</label>
```

```
                                        <input type="email" class="form-control"
id="inputEmail4" placeholder='Enter Email' autoComplete='off'
                                        onChange={e => setData({...data, email:
e.target.value})}/>
                                </div>
                                <div class="col-12">
                                        <label for="inputPassword4"
class="form-label">Password</label>
                                        <input type="password" class="form-control"
id="inputPassword4" placeholder='Enter Password'
                                         onChange={e => setData({...data, password:
e.target.value})}/>
                                </div>
                                <div class="col-12">
                                        <label for="inputSalary"
class="form-label">Salary</label>
                                        <input type="text" class="form-control"
id="inputSalary" placeholder="Enter Salary" autoComplete='off'
                                        onChange={e => setData({...data, salary:
e.target.value})}/>
                                </div>
                                <div class="col-12">
                                        <label for="inputAddress"
class="form-label">Address</label>
                                        <input type="text" class="form-control"
id="inputAddress" placeholder="1234 Main St" autoComplete='off'
                                        onChange={e => setData({...data, address:
e.target.value})}/>
                                </div>
                                <div class="col-12 mb-3">
                                        <label class="form-label"
for="inputGroupFile01">Select Image</label>
                                        <input type="file" class="form-control"
id="inputGroupFile01"
                                        onChange={e => setData({...data, image:
e.target.files[0]})}/>
                                </div>
                                <div class="col-12">
                                        <button type="submit" class="btn
btn-primary">Create</button>
                                </div>
                        </form>
                </div>
```

```
        )
}

export default AddEmployee
```

APP

```
import React from 'react'
import Login from './Login'
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import Dashboard from './Dashboard'
import Employee from './Employee'
import Profile from './Profile'
import Home from './Home'
import AddEmployee from './AddEmployee'
import EditEmployee from './EditEmployee'
import Start from './Start'
import EmployeeDetail from './EmployeeDetail'
import EmployeeLogin from './EmployeeLogin'


function App() {
  return (
    <BrowserRouter>
    <Routes>
      <Route path='/' element={<Dashboard />}>
        <Route path='' element={<Home />}></Route>
        <Route path='/employee' element={<Employee />}></Route>
        <Route path='/profile' element={<Profile />}></Route>
        <Route path='/create' element={<AddEmployee />}></Route>
        <Route path='/employeeEdit/:id' element={<EditEmployee />}></Route>
      </Route>
      <Route path='/login' element={<Login />}></Route>
      <Route path='/start' element={<Start />}></Route>
      <Route path='/employeeLogin' element={<EmployeeLogin />}></Route>
      <Route path='/employeedetail/:id' element={<EmployeeDetail />}></Route>
    </Routes>
    </BrowserRouter>
  )
}

export default App
```

DASHBOARD

```
import React, { useEffect } from 'react'
import 'bootstrap-icons/font/bootstrap-icons.css'
import { Link, Outlet, useNavigate } from 'react-router-dom'
import axios from 'axios'

function Dashboard() {
	const navigate = useNavigate()
	axios.defaults.withCredentials = true;
	useEffect(()=>{
		axios.get('http://localhost:8081/dashboard')
		.then(res => {
			if(res.data.Status === "Success") {
				if(res.data.role === "admin") {
					navigate('/');
				} else {
					const id = res.data.id;
					navigate('/employeedetail/'+id)
				}
			} else {
				navigate('/start')
			}
		})
	}, [])

	const handleLogout = () => {
		axios.get('http://localhost:8081/logout')
		.then(res => {
			navigate('/start')
		}).catch(err => console.log(err));
	}
	return (
		<div className="container-fluid">
			<div className="row flex-nowrap">
				<div className="col-auto col-md-3 col-xl-2 px-sm-2 px-0
bg-dark">
					<div className="d-flex flex-column align-items-center
align-items-sm-start px-3 pt-2 text-white min-vh-100">
						<a href="/" className="d-flex
align-items-center pb-3 mb-md-1 mt-md-3 me-md-auto text-white text-decoration-none">
							<span className="fs-5 fw-bolder
d-none d-sm-inline">Admin Dashboard</span>
						</a>
```

```
                                        <ul className="nav nav-pills flex-column
mb-sm-auto mb-0 align-items-center align-items-sm-start" id="menu">
                                            <li>
                                                <Link to="/"
data-bs-toggle="collapse" className="nav-link text-white px-0 align-middle">
                                                    <i className="fs-4
bi-speedometer2"></i> <span className="ms-1 d-none d-sm-inline">Dashboard</span>
</Link>
                                            </li>
                                            <li>
                                                <Link to="/employee"
className="nav-link px-0 align-middle text-white">
                                                    <i className="fs-4
bi-people"></i> <span className="ms-1 d-none d-sm-inline">Manage Employees</span>
</Link>
                                            </li>
                                            <li>
                                                <Link to="profile"
className="nav-link px-0 align-middle text-white">
                                                    <i className="fs-4
bi-person"></i> <span className="ms-1 d-none d-sm-inline">Profile</span></Link>
                                            </li>
                                            <li onClick={handleLogout}>
                                                <a href="#"
className="nav-link px-0 align-middle text-white">
                                                    <i className="fs-4
bi-power"></i> <span className="ms-1 d-none d-sm-inline">Logout</span></a>
                                            </li>
                                        </ul>
                                    </div>
                                </div>
                                <div class="col p-0 m-0">
                                    <div className='p-2 d-flex justify-content-center
shadow'>
                                        <h4>Employee Management System</h4>

                                    </div>
                                    <Outlet />
                                </div>
                            </div>
                        </div>
                )
}
```

```
export default Dashboard
```

EDIT EMPLOYEE

```
import axios from 'axios';
import React, { useEffect, useState } from 'react'
import { useNavigate, useParams } from 'react-router-dom';

function EditEmployee() {
        const [data, setData] = useState({
                name: '',
                email: '',
                address: '',
                salary: '',
        })
        const navigate = useNavigate()

        const {id} = useParams();

        useEffect(()=> {
                axios.get('http://localhost:8081/get/'+id)
                .then(res => {
                        setData({...data, name: res.data.Result[0].name,
                                email: res.data.Result[0].email,
                                address: res.data.Result[0].address,
                                salary: res.data.Result[0].salary
                        })
                })
                .catch(err =>console.log(err));
        }, [])

        const handleSubmit = (event) => {
                event.preventDefault();
                axios.put('http://localhost:8081/update/'+id, data)
                .then(res => {
                        if(res.data.Status === "Success") {
                                navigate('/employee')
                        }
                })
                .catch(err => console.log(err));
        }
  return (
    <div className='d-flex flex-column align-items-center pt-4'>
```

```
                        <h2>Update Employee</h2>
                        <form class="row g-3 w-50" onSubmit={handleSubmit}>
                        <div class="col-12">
                                <label for="inputName"
class="form-label">Name</label>
                                <input type="text" class="form-control"
id="inputName" placeholder='Enter Name' autoComplete='off'
                                onChange={e => setData({...data, name:
e.target.value})} value={data.name}/>
                        </div>
                        <div class="col-12">
                                <label for="inputEmail4"
class="form-label">Email</label>
                                <input type="email" class="form-control"
id="inputEmail4" placeholder='Enter Email' autoComplete='off'
                                onChange={e => setData({...data, email:
e.target.value})} value={data.email}/>
                        </div>
                        <div class="col-12">
                                <label for="inputSalary"
class="form-label">Salary</label>
                                <input type="text" class="form-control"
id="inputSalary" placeholder="Enter Salary" autoComplete='off'
                                onChange={e => setData({...data, salary:
e.target.value})} value={data.salary}/>
                        </div>
                        <div class="col-12">
                                <label for="inputAddress"
class="form-label">Address</label>
                                <input type="text" class="form-control"
id="inputAddress" placeholder="1234 Main St" autoComplete='off'
                                onChange={e => setData({...data, address:
e.target.value})} value={data.address}/>
                        </div>
                        <div class="col-12">
                                <button type="submit" class="btn
btn-primary">Update</button>
                        </div>
                    </form>
                </div>
  )
}

export default EditEmployee
```

EMPLOYEE

```
import axios from 'axios'
import React, { useEffect, useState } from 'react'
import { Link } from 'react-router-dom'

function Employee() {
  const [data, setData] = useState([])

  useEffect(()=> {
    axios.get('http://localhost:8081/getEmployee')
    .then(res => {
      if(res.data.Status === "Success") {
        setData(res.data.Result);
      } else {
        alert("Error")
      }
    })
    .catch(err => console.log(err));
  }, [])

  const handleDelete = (id) => {
    axios.delete('http://localhost:8081/delete/'+id)
    .then(res => {
      if(res.data.Status === "Success") {
        window.location.reload(true);
      } else {
        alert("Error")
      }
    })
    .catch(err => console.log(err));
  }

  return (
    <div className='px-5 py-3'>
      <div className='d-flex justify-content-center mt-2'>
        <h3>Employee List</h3>
      </div>
      <Link to="/create" className='btn btn-success'>Add Employee</Link>
      <div className='mt-3'>
        <table className='table'>
          <thead>
```

```
          <tr>
            <th>Name</th>
            <th>Image</th>
            <th>Email</th>
            <th>Address</th>
            <th>Salary</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {data.map((employee, index) => {
           return <tr key={index}>
              <td>{employee.name}</td>
              <td>{
                <img src={`http://localhost:8081/images/`+employee.image} alt=""
className='employee_image'/>
                }</td>
              <td>{employee.email}</td>
              <td>{employee.address}</td>
              <td>{employee.salary}</td>
              <td>
                <Link to={`/employeeEdit/`+employee.id} className='btn btn-primary btn-sm
me-2'>edit</Link>
                <button onClick={e => handleDelete(employee.id)} className='btn btn-sm
btn-danger'>delete</button>
              </td>
            </tr>
          })}
        </tbody>
      </table>
    </div>
  </div>
 )
}

export default Employee
```

HOME

```
import axios from 'axios'
import React, { useEffect, useState } from 'react'

function Home() {
```

```
const [adminCount, setAdminCount] = useState()
const [employeeCount, setEmployeeCount] = useState()
const [salary, setSalary] = useState()

useEffect(() => {
  axios.get('http://localhost:8081/adminCount')
                .then(res => {
                        setAdminCount(res.data[0].admin)
                }).catch(err => console.log(err));

  axios.get('http://localhost:8081/employeeCount')
                .then(res => {
                        setEmployeeCount(res.data[0].employee)
                }).catch(err => console.log(err));

  axios.get('http://localhost:8081/salary')
                .then(res => {
                        setSalary(res.data[0].sumOfSalary)
                }).catch(err => console.log(err));

} , [])
return (
  <div>
    <div className='p-3 d-flex justify-content-around mt-3'>
      <div className='px-3 pt-2 pb-3 border shadow-sm w-25'>
        <div className='text-center pb-1'>
          <h4>Admin</h4>
        </div>
        <hr />
        <div className=''>
          <h5>Total: {adminCount}</h5>
        </div>
      </div>
      <div className='px-3 pt-2 pb-3 border shadow-sm w-25'>
        <div className='text-center pb-1'>
          <h4>Employee</h4>
        </div>
        <hr />
        <div className=''>
          <h5>Total: {employeeCount}</h5>
```

```
          </div>
        </div>
        <div className='px-3 pt-2 pb-3 border shadow-sm w-25'>
          <div className='text-center pb-1'>
            <h4>Salary</h4>
          </div>
          <hr />
          <div className=''>
            <h5>Total: {salary}</h5>
          </div>
        </div>
      </div>

      {/* List of admin  */}
      <div className='mt-4 px-5 pt-3'>
        <h3>List of Admins</h3>
        <table className='table'>
          <thead>
            <tr>
              <th>Email</th>
              <th>Action</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Admin</td>
              <td>Admin</td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  )
}

export default Home
```

START

```
import React from 'react'
```

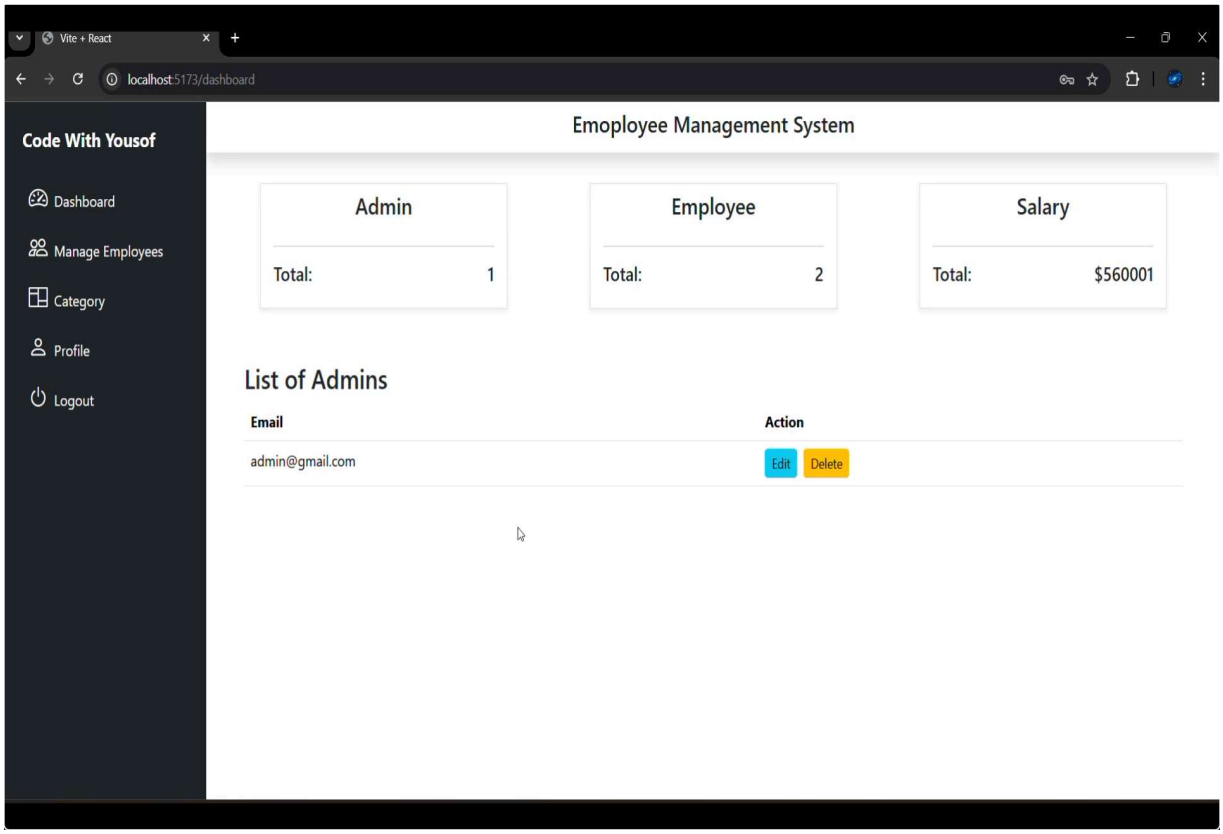
```
import { useNavigate } from 'react-router-dom'

function Start() {
   const navigate = useNavigate()
  return (
    <div className='d-flex justify-content-center align-items-center vh-100 loginPage'>
          <div className='p-3 rounded w-25 border loginForm text-center'>
             <h2>Login As</h2>
             <div className='d-flex justify-content-between mt-5'>
                <button className='btn btn-primary btn-lg' onClick={e =>
navigate('/employeeLogin')}>Employee</button>
                <button className='btn btn-success btn-lg' onClick={e =>
navigate('/login')}>Admin</button>
             </div>
          </div>
       </div>
  )
}

export default Start
```

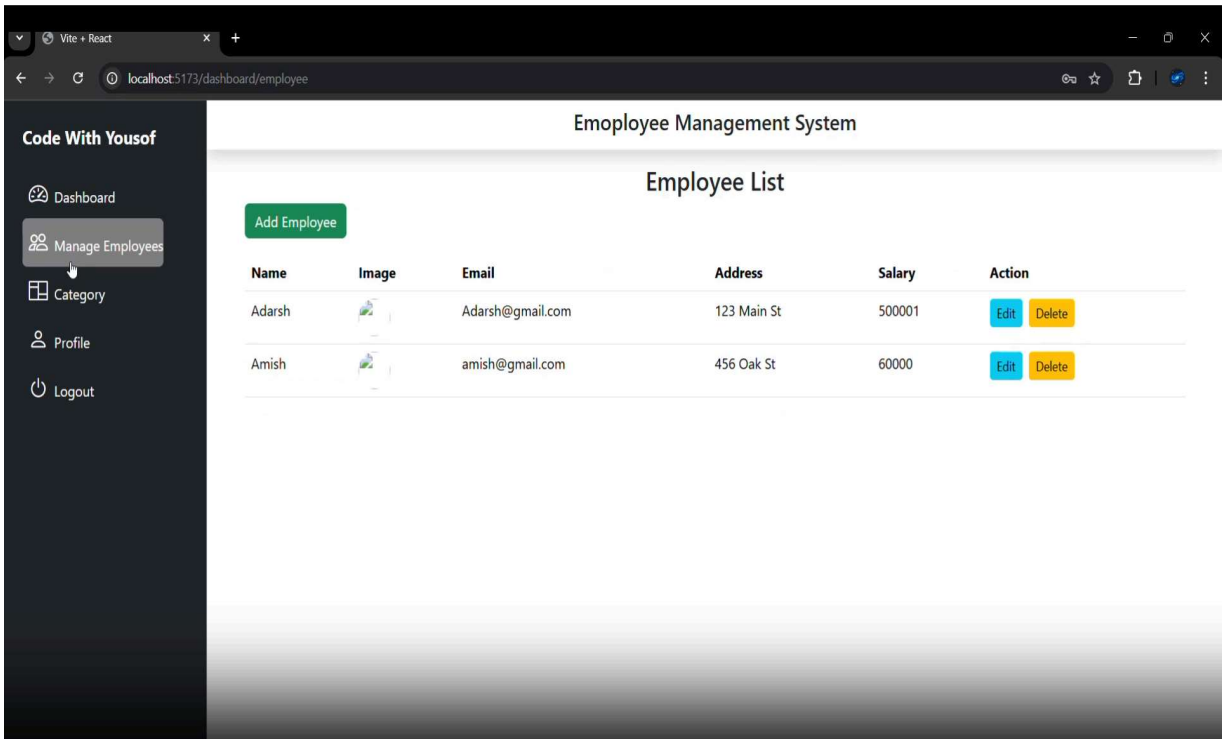
MAIN

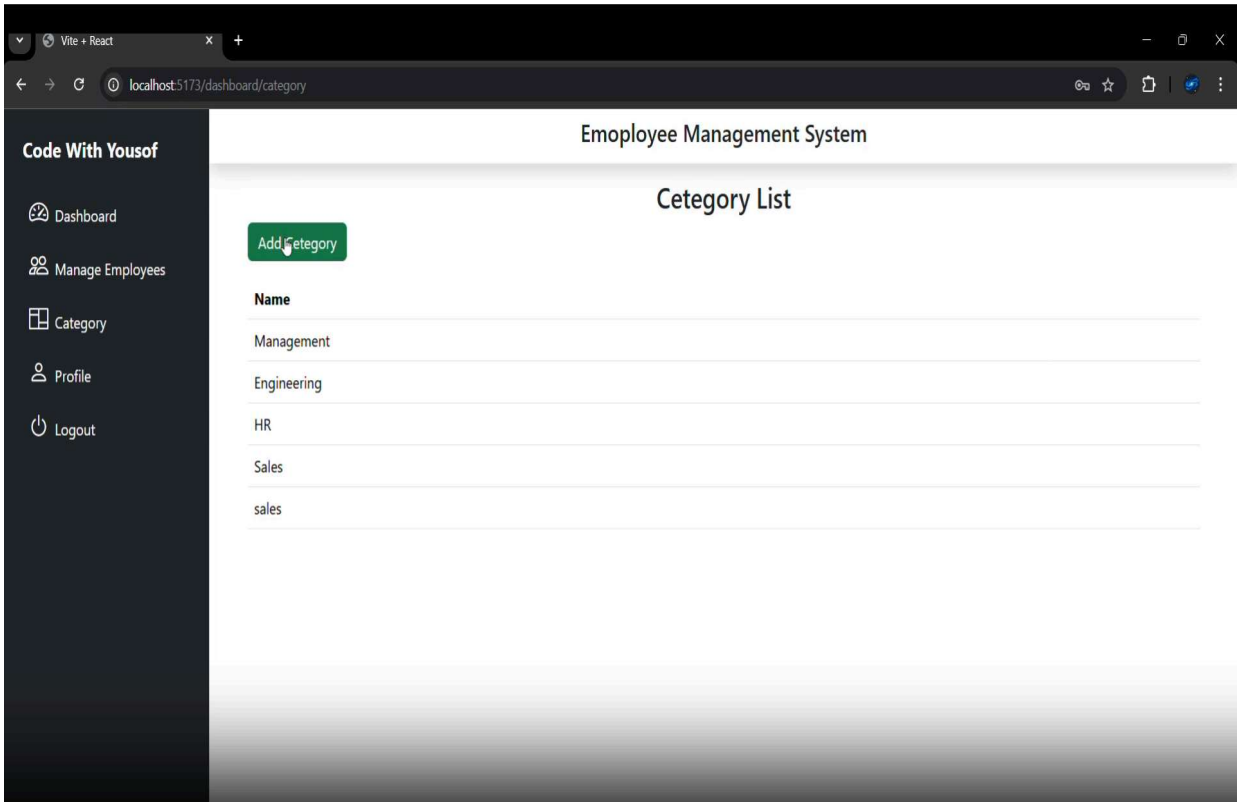
```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import 'bootstrap/dist/css/bootstrap.min.css'


ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

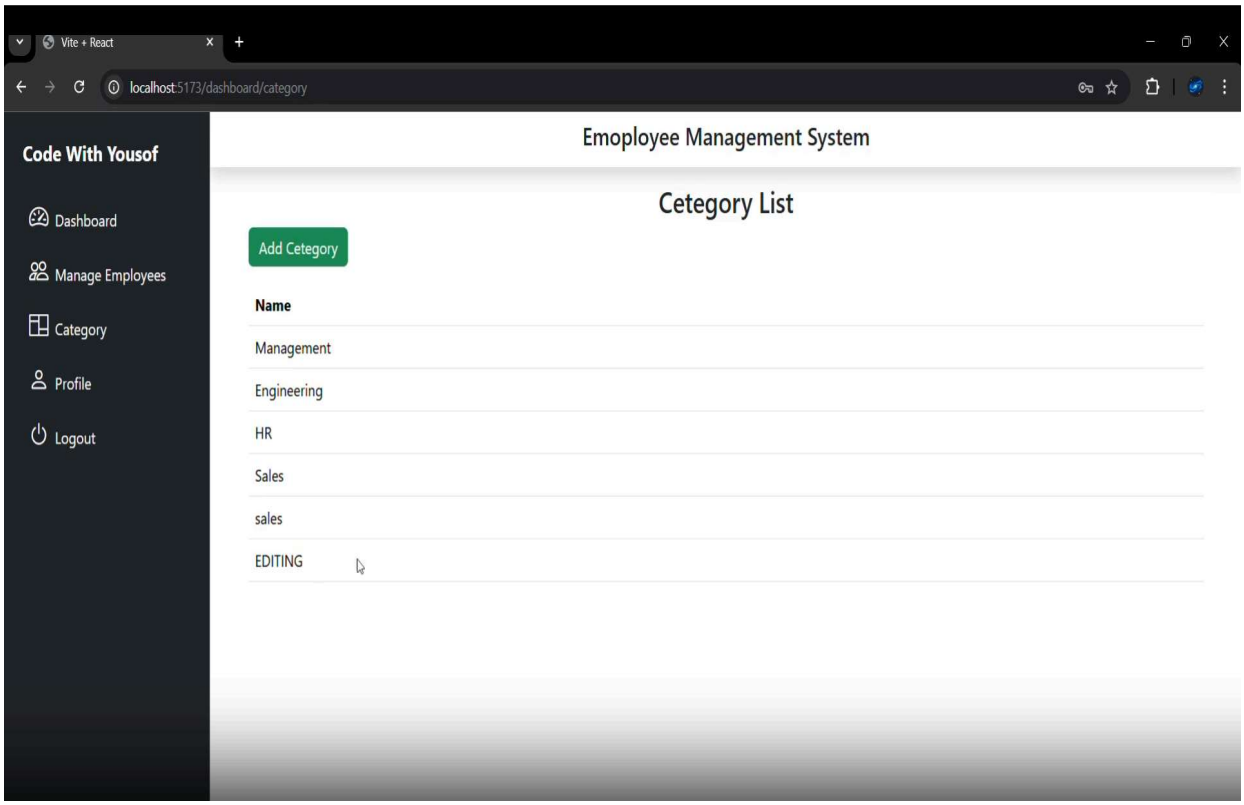
CHAPTER 6

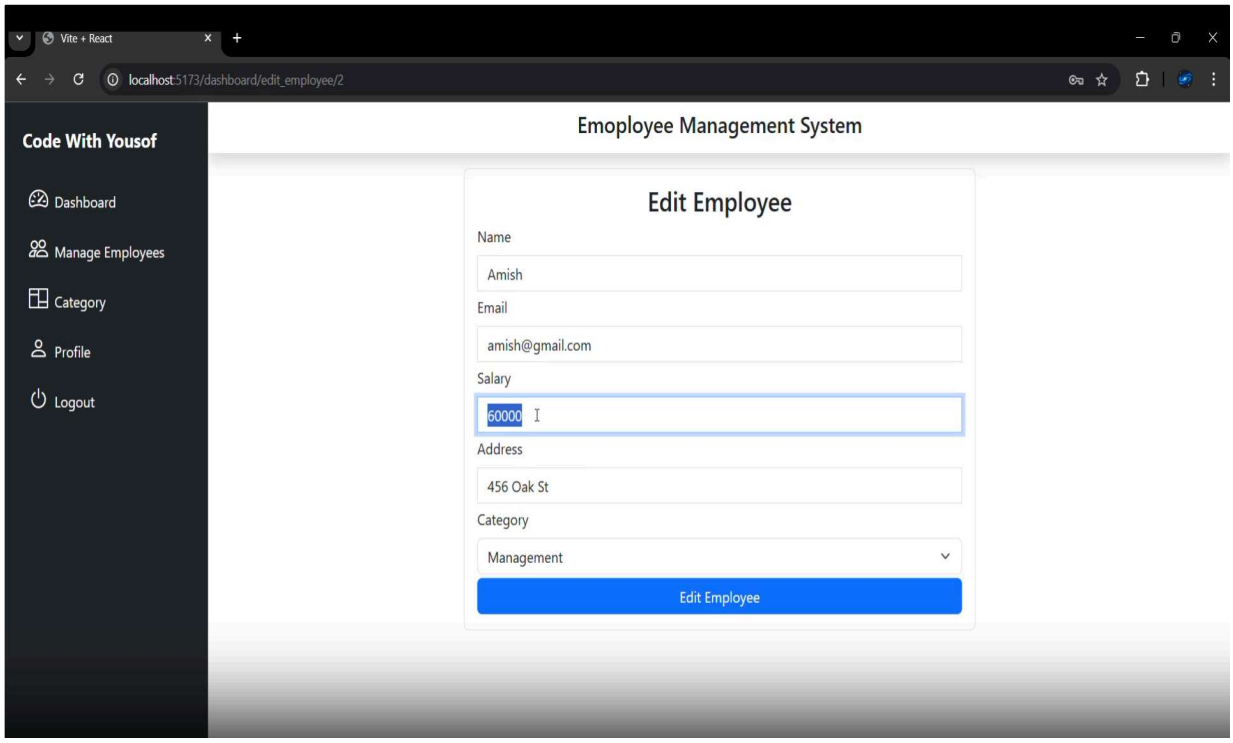
SCREENSHOTS

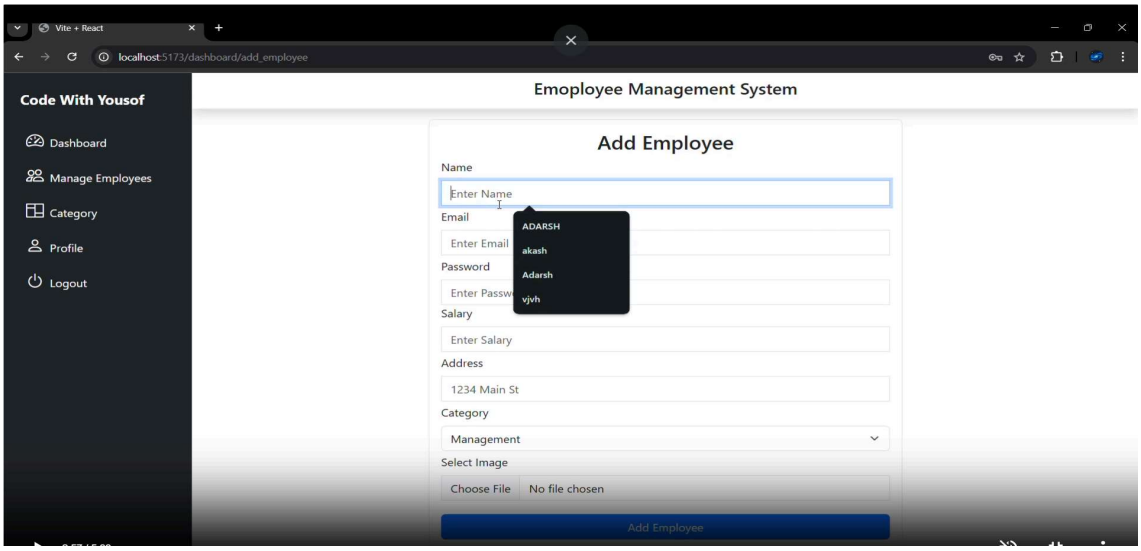


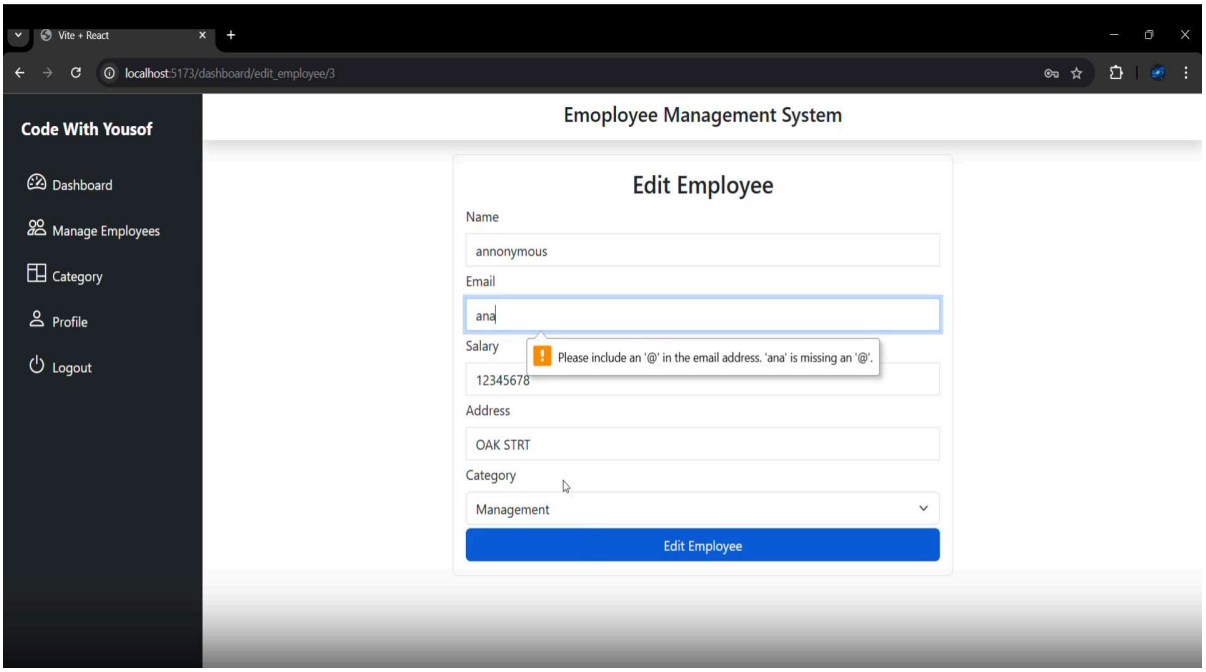

Vite + React
localhost:5173/dashboard/employee
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Employee List
Add Employee
Name
Image
Email
Address
Salary
Action
Adarsh
Adarsh@gmail.com
123 Main St
500001
Edit
Delete
Amish
amish@gmail.com
456 Oak St
60000
Edit
Delete

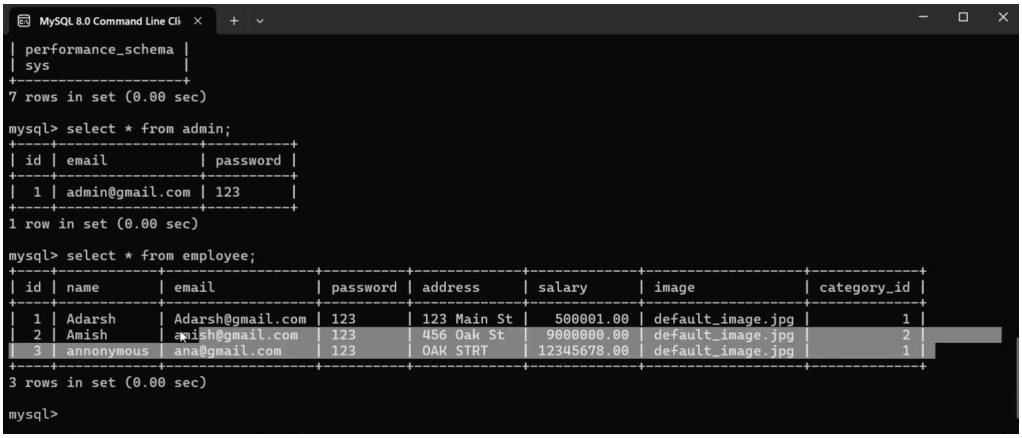

Vite + React
localhost:5173/dashboard/category
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Cetegory List
Add Cetegory
Name
Management
Engineering
HR
Sales
sales


Vite + React
localhost:5173/dashboard/category
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Cetegory List
Add Cetegory
Name
Management
Engineering
HR
Sales
sales
EDITING


Vite + React
localhost:5173/dashboard/edit_employee/2
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Edit Employee
Name
Amish
Email
amish@gmail.com
Salary
60000
Address
456 Oak St
Category
Management
Edit Employee


Vite + React
localhost:5173/dashboard/add_employee
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Add Employee
Name
Enter Name
ADARSH
akash
Adarsh
vjvh
Email
Enter Email
Password
Salary
Enter Salary
Address
1234 Main St
Category
Management
Select Image
Choose File
No file chosen
Add Employee


Vite + React
localhost:5173/dashboard/edit_employee/3
Code With Yousof
Dashboard
Manage Employees
Category
Profile
Logout
Emoployee Management System
Edit Employee
Name
annonymous
Email
ana
Please include an '@' in the email address. 'ana' is missing an '@'.
Salary
12345678
Address
OAK STRT
Category
Management
Edit Employee


MySQL 8.0 Command Line Cli
| performance_schema |
| sys |
7 rows in set (0.00 sec)
mysql> select * from admin;
| id | email | password |
| 1 | admin@gmail.com | 123 |
1 row in set (0.00 sec)
mysql> select * from employee;
id	name	email	password	address	salary	image	category_id
1	Adarsh	Adarsh@gmail.com	123	123 Main St	500001.00	default_image.jpg	1
2	Amish	amish@gmail.com	123	456 Oak St	9000000.00	default_image.jpg	2
3	annonymous	ana@gmail.com	123	OAK STRT	12345678.00	default_image.jpg	1
3 rows in set (0.00 sec)
mysql>

CHAPTER 7

TESTING

7.1 Introduction:

Testing is the major quality control measure used during software development. It is a basic function to detect errors in the software. During the requirement analysis and design the output of the document that is usually textual and non-executable after the coding phase the computer programs are available that can be executed for testing purpose. This implies that testing not only has to uncover errors introduce during the previous phase. The goal of testing is to uncover requirement, design, coding errors in the program. Testing determines whether the system appears to be working according to the specifications. It is the phase where we try to break the system and we test the system with real case scenarios at a point.

7.2 Levels of Testing:

Unit Testing: The unit testing of the source code has to be done for every individual unit of module that was developing part of the system and some errors were found for every turn and rectified. This form of testing was use to check for the behaviour signified the working of the system in different environment as an independent functional unit.

Integration Testing: From the individual parts to the cohesion of each part to make the system as a whole, there is need to test the working between the assembled modules of the system. The modules are integrated to makeup the entire system. The testing process is concerned with finding errors that result from unanticipated interaction between the sub-system and system component. It is also concerned with validating the system meets its functional and non-functional requirement.

System Testing: The requirement specification document that is the entire system is to be tested to see whether it meets the requirement or not.

Test Unit: Admin Component

- **Adding admin:**

Serial No.	Condition To be Tested	Test Data	Expected Output	Remarks
1.	If any field in the form is empty.	Value of form fields.	Alert the user to enter all the fields and then proceed.	SUCCESSFUL
2.	If admin name contains other than Character values.	Admin name	Alert the user to enter only Characters and return to same page.	SUCCESSFUL
3.	If login ID is invalid.	Login id	Alert the user to enter a valid login ID and return to the same page.	SUCCESSFUL
4.	If password length is not between 8 to 16 characters.	password	Alert the user to enter a password of length between 8 to 16 characters.	SUCCESSFUL
5.	If confirm password and password does not match.	cpassword, password	Alert the user to enter matching password and confirm password.	SUCCESSFUL

- **Admin login:**

Serial No.	Condition To be Tested	Test Data	Expected Output	Remarks
1.	If any field in the form is empty.	Value of form fields.	Alert the user to enter all the fields and then proceed.	SUCCESSFUL
2.	If the E-mail ID and Password does not match.	Email id, password	Alert user that E-mail ID and password are not matching and stay in the same page.	SUCCESSFUL

- **Admin Change Password:**

Serial No.	Condition To be Tested	Test Data	Expected Output	Remarks
1.	If field in the form is empty.	Value of form fields.	Alert the user to enter the fields and then proceed.	SUCCESSFUL
2.	If the E-mail ID is invalid.	Login id	Alert user to enter a valid E-mail ID and stay in the same page.	SUCCESSFUL
3.	If the old password is not between the given range of 8 to 16 characters.	Old password	Alert the user to enter a password of length between 8 to 16 characters.	SUCCESSFUL
4.	If the new password is not between the given range of 8 to 16 characters.	New password	Alert the user to enter a password of length between 8 to 16 characters.	SUCCESSFUL
5.	If the confirm password and password does not match.	cpassword, password	Alert user that confirm password and password are not matching and stay in the same page.	SUCCESSFUL

- **Customer Change Password:**

Serial No.	Condition To be Tested	Test Data	Expected Output	Remarks
1.	If field in the form is empty.	Value of form fields.	Alert the user to enter the fields and then proceed.	SUCCESSFUL
2.	If the E-mail ID is invalid.	Login id	Alert user to enter a valid E-mail ID and stay in the same page.	SUCCESSFUL
3.	If the old password is not between the given range of 8 to 16 characters.	Old password	Alert the user to enter a password of length between 8 to 16 characters.	SUCCESSFUL
4.	If the new password is not between the given range of 8 to 16 characters.	New password	Alert the user to enter a password of length between 8 to 16 characters.	SUCCESSFUL
5.	If the confirm password and password does not match.	c password, password	Alert user that confirm password and password are not matching and stay in the same page.	SUCCESSFUL

- **Worker Login:**

Serial No.	Condition To be Tested	Test Data	Expected Output	Remarks
1.	If any field in the form is empty.	Value of form fields.	Alert the user to enter all the fields and then proceed.	SUCCESSFUL
2.	If the E-mail ID and Password does not match.	Email id, password	Alert user that E-mail ID and password are not matching and stay in the same page.	SUCCESSFUL

CHAPTER 8

USER MANUAL

User Manual for Employee Management System (EMS)

8. 1. Introduction

This User Manual provides instructions for users of the Employee Management System (EMS). The EMS allows the management of employee data, attendance, payroll, leave requests, performance reviews, and more. The system provides role-based access with different privileges for employees, HR, and admins.

8.2. System Requirements

Browser: Google Chrome, Mozilla Firefox, or Safari (latest versions)
Operating System: Windows, macOS, Linux
Required Tools: React, MySQL Database, Node.js, and an active internet connection

8.3. User Roles & Permissions

There are three main user roles in the system:

1. Admin: Full access to the system, can manage all employees, create reports, and modify system settings.
2. HR: Manages employee data, processes payroll, handles leave requests, and evaluates performance.
3. Employee: Views and updates their own personal information, requests leave, and views their payroll and performance.

8.4. Getting Started

8. 4.1 Logging In

- Open the application in a browser.
- Enter your username and password in the login form.
- Click Login.

8. 4.2 Resetting Password

- On the login screen, click on Forgot Password.
- Enter your email address to receive a reset link.
- Follow the instructions in the email to reset your password.

8.5. Features for Employees

8.5.1 Viewing Profile

- After logging in, click on Profile in the navigation menu.

View or update your personal details, contact information, and emergency contacts.

8.5.2 Attendance
- Click on Attendance to view your attendance records.
- You can see the days you were present, absent, or on leave.

8.5.3 Leave Management
- To request leave, go to the Leave section.
- Click Apply Leave, select leave type (sick, casual, etc.), and choose the start and end dates.
- Submit the request for approval.

8.5.4 Payroll
- Access your Payroll details by clicking the Payroll section.
- View your salary breakdown, including basic salary, bonuses, and deductions.

8.5.5 Performance Review
- Go to Performance to view past reviews.
- Your HR or manager will input feedback and ratings on your performance.

8.6. Features for HR

8.6.1 Managing Employee Profiles
- From the Employee Management section, HR can view, edit, and update employee details, such as job title, department, and salary.
- HR can add new employees and deactivate or terminate profiles.

8.6.2 Payroll Management
- HR can generate monthly payroll for employees, including calculating deductions, bonuses, and net salary.
- Payroll reports are available for HR to track salary payments.

8. 6.3 Leave Management
- HR can approve or reject leave requests submitted by employees.
- View the leave history and manage leave balances.

8.6.4 Performance Evaluation
- HR can add performance reviews for employees, including ratings and feedback.
- Generate reports on employee performance for analysis.

8.7. Features for Admin

8.7.1 Employee Directory
- Admin has access to the **Employee Directory**, where all employee details are available.
- Admin can search for employees by name, department, or job title.

8.7.2 System Reports
- Admin can generate reports on employee data, payroll summaries, attendance, and performance.

8.7.3 Role Management
- Admin can assign roles to users (admin, HR, employee) and manage access to various system features.

8.7.4 Notifications
- Admin receives notifications about system updates, new registrations, and leave requests.

8.8. Reporting and Analytics

8.8.1 Employee Analytics
- Admin and HR can access visual reports on employee performance, attendance trends, and payroll analysis via the Analytics Dashboard.

8.8.2 Report Generation
- Admin and HR can generate reports on specific employee data, performance, payroll, and attendance.

8.9. Notifications
- The system sends notifications to employees when their leave requests are approved or rejected.
- Admin and HR are notified about new employee registrations, leave requests, or performance reviews.

8.10. System Security
- Data is encrypted to ensure employee information remains secure.
- The system uses role-based access control to restrict unauthorized access to sensitive information.

8.11. Troubleshooting
- Login Issues: If you cannot log in, ensure you are entering the correct username and password. If forgotten, reset your password.
- Slow Performance: Check your internet connection and refresh the page. If the problem persists, contact your system administrator.

CHAPTER 9

CONCLUSION

The Employee Management System (EMS) is designed to streamline and automate the management of employee data, attendance, payroll, performance reviews, leave requests, and more, enabling organizations to efficiently manage their workforce. By providing a comprehensive platform with role-based access, the EMS ensures that employees, HR personnel, and administrators have the tools they need to perform their tasks effectively.

Employees can easily manage and update their personal information, request leave, view attendance records, and access payroll details, all through an intuitive interface. HR and administrators are empowered to oversee employee data, process payroll, approve leave requests, and evaluate performance. The system also supports the generation of detailed reports and analytics, which help HR and management track trends in performance, attendance, and payroll.

With the implementation of role-based access control, data security is prioritized, ensuring that sensitive employee information is protected while allowing each user role the appropriate level of access to system features. The EMS is a powerful solution that improves administrative efficiency, ensures compliance, and enhances communication within the organization.

Overall, the Employee Management System offers a robust platform to manage critical employee-related processes, contributing to smoother operations and better decision-making, while also providing a scalable solution for future growth and improvements.