# Employee Portal

# Project Report

**By**

**Namrata Deshmukh – AF04957420**

**Rohit Deshmukh – AF04957234**

# Acknowledgement

The project **”Employee Portal”**

is the Project work carried out by

| Name | Enrolment no. |
|---|---|
| **Namrata Deshmukh** | **AF04957420** |
| **Rohit Deshmukh** | **AF04957234** |

Under the Guidance.

We are thankful to my project guide **Rajshri Mam** for guiding me to complete the Project.

Her suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

# Abstract

The **Employee Portal System (EPS)** is a full-stack web application designed to streamline employee data management within an organization.

It provides features for adding, updating, deleting, and viewing employee details, managing roles, departments, and payroll data. Built using ReactJS (frontend), Node.js with Express (backend), and MySQL (database), the system ensures efficient handling of employee information with a responsive interface and secure authentication.

The system enables HR teams and administrators to manage employee details including personal information, roles, departments, and payroll-related records.

It is implemented using ReactJS for the frontend, Node.js with Express for the backend, and MySQL as the database. EMS ensures role-based access, secure authentication with JWT, and encrypted password management through bcrypt. This project improves efficiency, reduces human errors, and provides a scalable foundation for integrating advanced HR modules.

# Introduction

Managing employees effectively is one of the most crucial tasks for any organization.

Traditional methods of managing employee data involve paperwork, manual registers, or spreadsheets, which are not only time-consuming but also prone to errors and data loss.

The Employee Management System (EMS) aims to resolve these issues by offering a centralized, secure, and user-friendly platform for storing and retrieving employee data.

The system provides fast access to employee records and simplifies day-to-day HR tasks, thereby increasing productivity and operational efficiency.

# Objectives

- To design and implement a centralized employee database system.
- To provide secure and role-based access for administrators and HR staff.
- To reduce redundancy and errors in maintaining employee information.
- To ensure scalability for integrating future modules like payroll, leave, and performance management.

- Develop a centralized system for managing employee records.
- Ensure secure and role-based access.
- Provide scalability for future modules like payroll and leave management.

# System Analysis

The analysis of the system covers the shortcomings of the existing system and how the proposed system resolves those issues.

**Existing System:**
- Manual record keeping or use of spreadsheets.
- Highly prone to errors and data loss.
- Limited accessibility and lack of centralized storage.

**Proposed System:**
- A centralized, web-based solution accessible securely by authorized staff.
- Reduces manual workload and minimizes redundancy.
- Offers scalability for future enhancements.

**Feasibility Study:**
- Technical: Implemented using open-source technologies like React, Node.js, and MySQL.
- Economic: Low development cost due to free technology stack.
- Operational: User-friendly interface requiring minimal training.
- Schedule: Can be developed in modular phases for better management.

**Software Requirement Specification (SRS):**

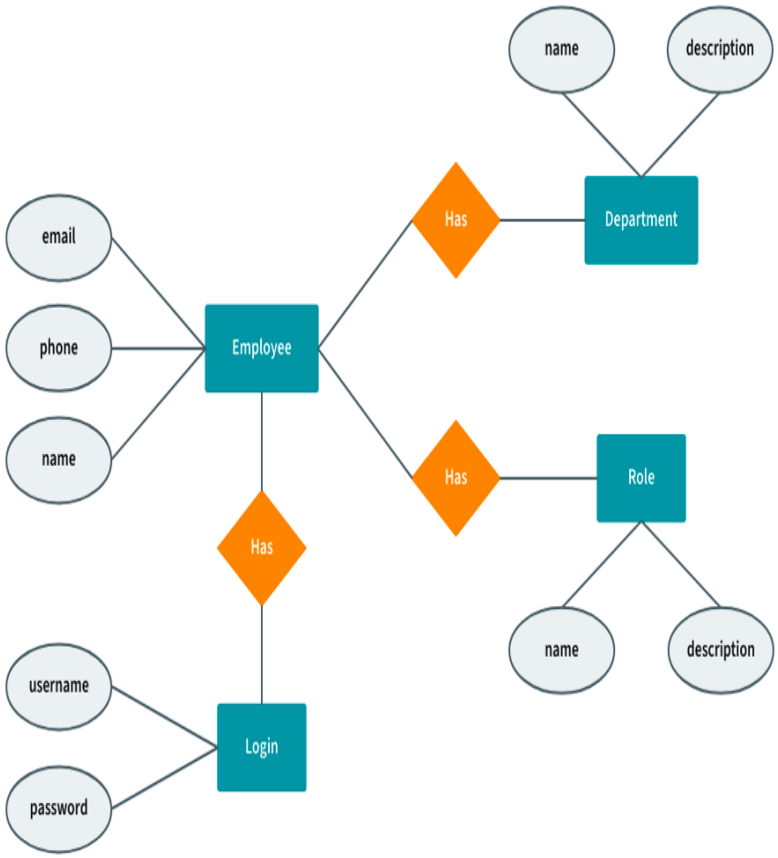
**a.Functional Requirements:**
- Employee CRUD operations.
- Department and Role management.
- Authentication and Authorization.

**b.Non-Functional Requirements:**

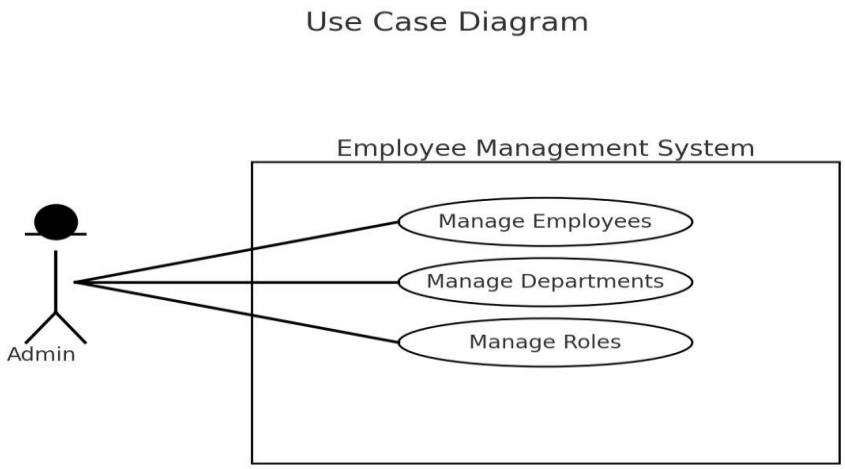
- Usability: Simple UI.
- Security: JWT, bcrypt.
- Reliability and scalability.

# System Design

**System Architecture:** The system follows a 3-tier architecture (Frontend, Backend, Database).



**Use Case Diagram:** Actors include Admin and System. Use cases include Manage Employees, Manage Departments, and Manage Roles.




**DFD:**

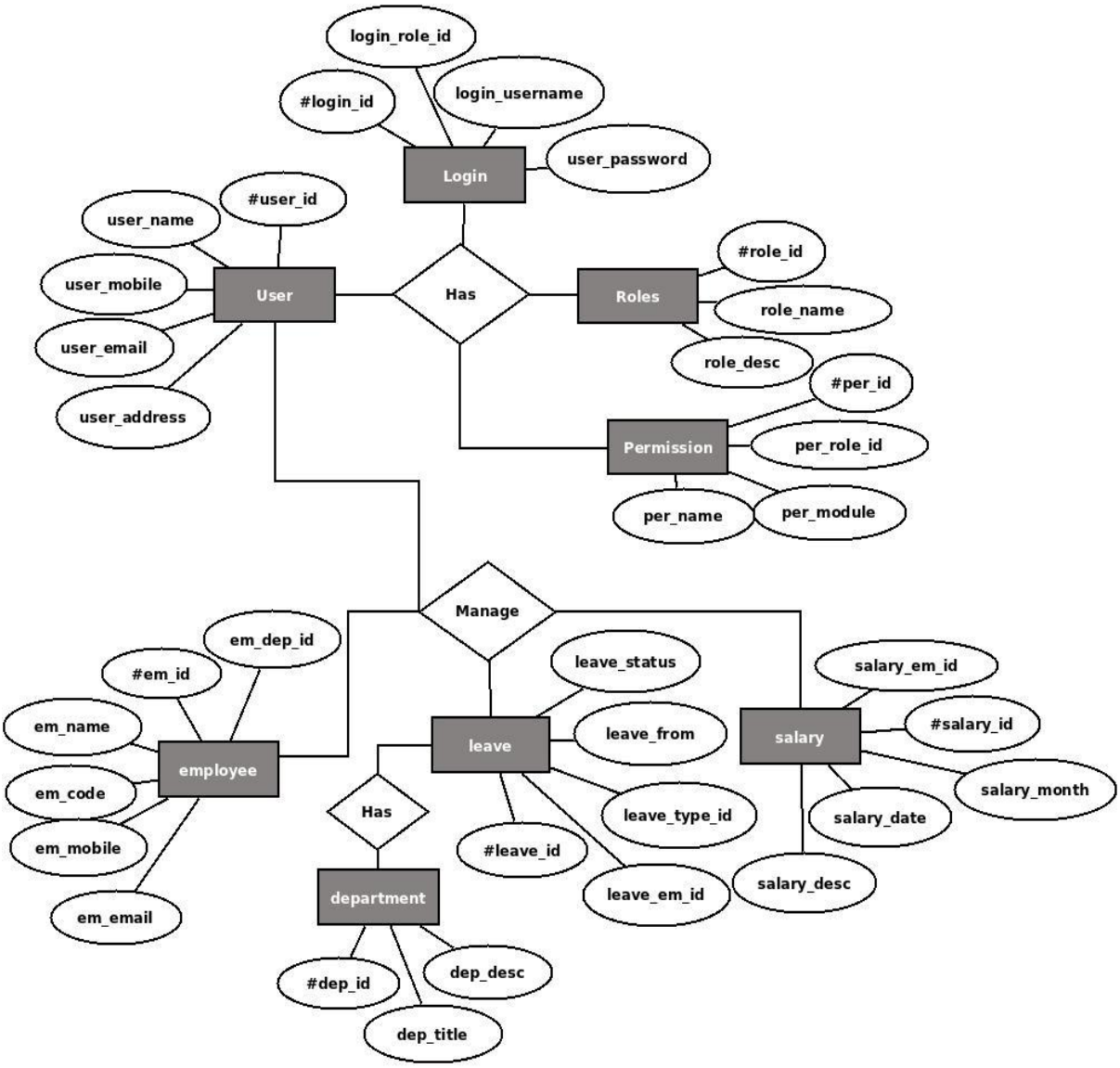
- Level 0: Admin interacts with EMS → Manage Employees, Departments, Roles.


**Level Zero DFD**

- Level 1: Employee CRUD operations + Department/Role assignment.

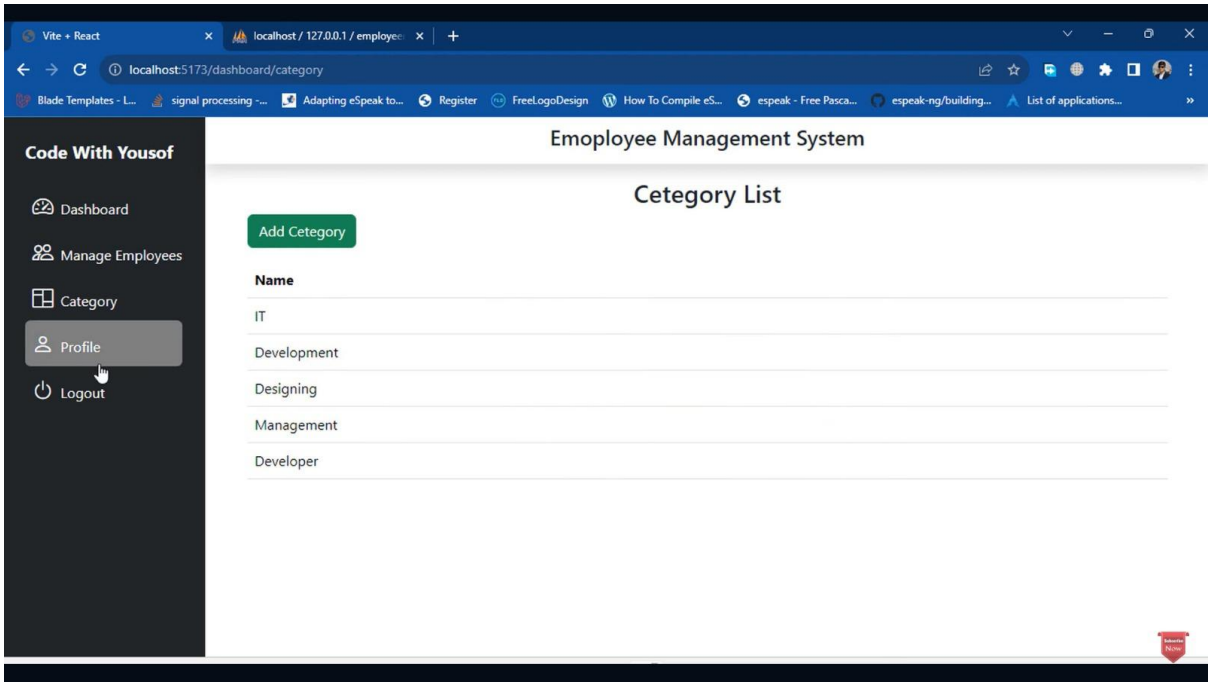
**ER Diagram Entities:**

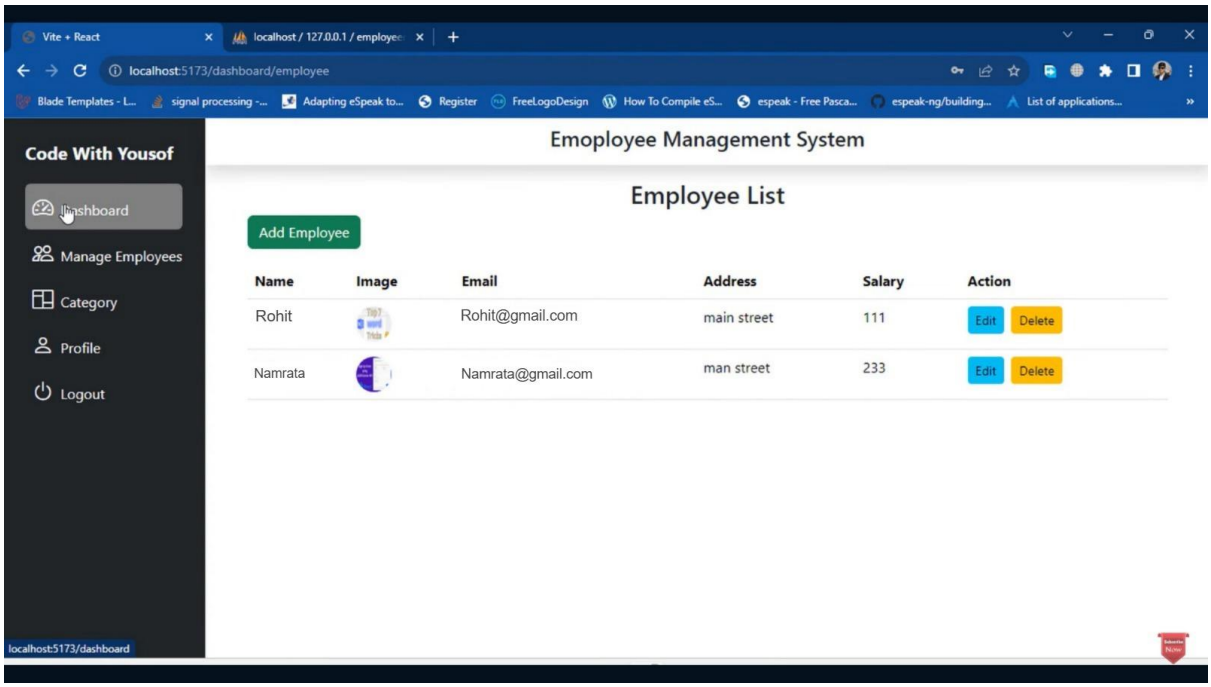
- Employee (id, name, email, phone, department_id, role_id, salary)
- Department (id, name)
- Role (id, title, description)
- Admin (id, username, password).




ER Diagram Employee Management System

# Screenshots



Vite + React
localhost / 127.0.0.1 / employee
localhost:5173/dashboard/category
Code With Yousof
Emoployee Management System
Dashboard
Manage Employees
Category
Profile
Logout
Cetegory List
Add Cetegory
Name
IT
Development
Designing
Management
Developer


Vite + React
localhost / 127.0.0.1 / employee
localhost:5173/dashboard/employee
Code With Yousof
Emoployee Management System
Dashboard
Manage Employees
Category
Profile
Logout
Employee List
Add Employee
Name
Image
Email
Address
Salary
Action
Rohit
Rohit@gmail.com
main street
111
Edit
Delete
Namrata
Namrata@gmail.com
man street
233
Edit
Delete
localhost:5173/dashboard

# Coding

## About html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

## AdminRoute.js

```
import express from "express";
import con from "../utils/db.js";
import jwt from "jsonwebtoken";
import bcrypt from 'bcrypt'
import multer from "multer";
import path from "path";


const router = express.Router();
```

```
router.post("/adminlogin/", (req, res) => {
  const sql = "SELECT * from admin Where email = ? and password = ?";
  con.query(sql, [req.body.email, req.body.password], (err, result) => {
    if (err) return res.json({ loginStatus: false, Error: "Query error" });
    if (result.length > 0) {
      const email = result[0].email;
      const token = jwt.sign(
        { role: "admin", email: email, id: result[0].id },
        "jwt_secret_key",
        { expiresIn: "1d" }
      );
      res.cookie('token', token)
      return res.json({ loginStatus: true });
    } else {
        return res.json({ loginStatus: false, Error:"wrong email or password" });
    }
  });
});

router.get('/category', (req, res) => {
    const sql = "SELECT * FROM category";
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"})
        return res.json({Status: true, Result: result})
    })
})

router.post('/add_category', (req, res) => {
    const sql = "INSERT INTO category (`name`) VALUES (?)"
```

```
    con.query(sql, [req.body.category], (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"})
        return res.json({Status: true})
    })
})

// image upload
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'Public/Images')
    },
    filename: (req, file, cb) => {
        cb(null, file.fieldname + "_" + Date.now() + path.extname(file.originalname))
    }
})
const upload = multer({
    storage: storage
})
// end imag eupload

router.post('/add_employee',upload.single('image'), (req, res) => {
    const sql = `INSERT INTO employee
    (name,email,password, address, salary,image, category_id)
    VALUES (?)`;
    bcrypt.hash(req.body.password, 10, (err, hash) => {
        if(err) return res.json({Status: false, Error: "Query Error"})
        const values = [
            req.body.name,
            req.body.email,
```

```
            hash,
            req.body.address,
            req.body.salary,
            req.file.filename,
            req.body.category_id
        ]
        con.query(sql, [values], (err, result) => {
            if(err) return res.json({Status: false, Error: err})
            return res.json({Status: true})
        })
    })
})

router.get('/employee', (req, res) => {
    const sql = "SELECT * FROM employee";
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"})
        return res.json({Status: true, Result: result})
    })
})

router.get('/employee/:id', (req, res) => {
    const id = req.params.id;
    const sql = "SELECT * FROM employee WHERE id = ?";
    con.query(sql,[id], (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"})
        return res.json({Status: true, Result: result})
    })
})
```

```
router.put('/edit_employee/:id', (req, res) => {
    const id = req.params.id;
    const sql = `UPDATE employee
        set name = ?, email = ?, salary = ?, address = ?, category_id = ?
        Where id = ?`
    const values = [
        req.body.name,
        req.body.email,
        req.body.salary,
        req.body.address,
        req.body.category_id
    ]
    con.query(sql,[...values, id], (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
})

router.delete('/delete_employee/:id', (req, res) => {
    const id = req.params.id;
    const sql = "delete from employee where id = ?"
    con.query(sql,[id], (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
})

router.get('/admin_count', (req, res) => {
```

```
    const sql = "select count(id) as admin from admin";
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
})

router.get('/employee_count', (req, res) => {
    const sql = "select count(id) as employee from employee";
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
})

router.get('/salary_count', (req, res) => {
    const sql = "select sum(salary) as salaryOFEmp from employee";
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
})

router.get('/admin_records', (req, res) => {
    const sql = "select * from admin"
    con.query(sql, (err, result) => {
        if(err) return res.json({Status: false, Error: "Query Error"+err})
        return res.json({Status: true, Result: result})
    })
```

```
})

router.get('/logout', (req, res) => {
   res.clearCookie('token')
   return res.json({Status: true})
})
```

# Testing

The testing strategy includes unit testing for individual modules, integration testing for combined modules, and system testing to verify overall functionality. Security testing was performed on the authentication module.

**Sample Test Cases:**

1. Login with valid credentials → Success.
2. Login with invalid credentials → Failure.
3. Add employee with valid details → Employee added successfully.
4. Update employee record → Database updated.
5. Delete employee record → Record deleted successfully.

# Report / Results

The EPS achieved centralized employee management, reducing redundancy and manual effort. It allows quick access to employee information, improving HR operations.

The EPS achieved the goal of centralizing employee management, significantly reducing manual effort and redundancy. The system ensures quick and secure access to employee details, improving the efficiency of HR operations. Limitations include the absence of payroll automation and advanced analytics, which can be integrated in the future.

Limitations:
- Payroll automation not included.
- No advanced analytics.

The project sets the foundation for further expansion into a complete HR management suite.

# Future Scope

- Integration of payroll management.
- Leave and attendance tracking system.
- Performance appraisal and review management.
- Employee self-service portal for accessing personal details.
- Advanced analytics and reporting features for HR decision-making.

# Conclusion

The Employee Portal System provides a secure, scalable, and efficient solution for managing employee data. By replacing manual systems with a digital platform, it improves accuracy, accessibility, and productivity. The system has successfully simplified HR tasks and laid a strong foundation for further automation.

The Employee Portal System simplifies HR tasks by providing a secure, scalable, and user-friendly solution.

# Bibliography

1. ExpressJS Documentation
2. ReactJS Documentation
3. MySQL Docs
4. StackOverflow
5. MDN Web Docs

# References


1. ExpressJS Documentation – https://expressjs.com
2. ReactJS Documentation – https://react.dev
3. MySQL Official Documentation – https://dev.mysql.com/doc
4. MDN Web Docs (JavaScript & Web Development) – https://developer.mozilla.org
5. Stack Overflow – https://stackoverflow.com
6. GeeksforGeeks – Employee Management System Project Articles – https://www.geeksforgeeks.org
7. TutorialsPoint – Node.js, ExpressJS, and MySQL Tutorials – https://www.tutorialspoint.com
8. W3Schools – React, JavaScript, SQL Tutorials – https://www.w3schools.com
9. Open Source JWT Documentation – https://jwt.io
10. Bcrypt Documentation (Password Hashing) – https://www.npmjs.com/package/bcrypt