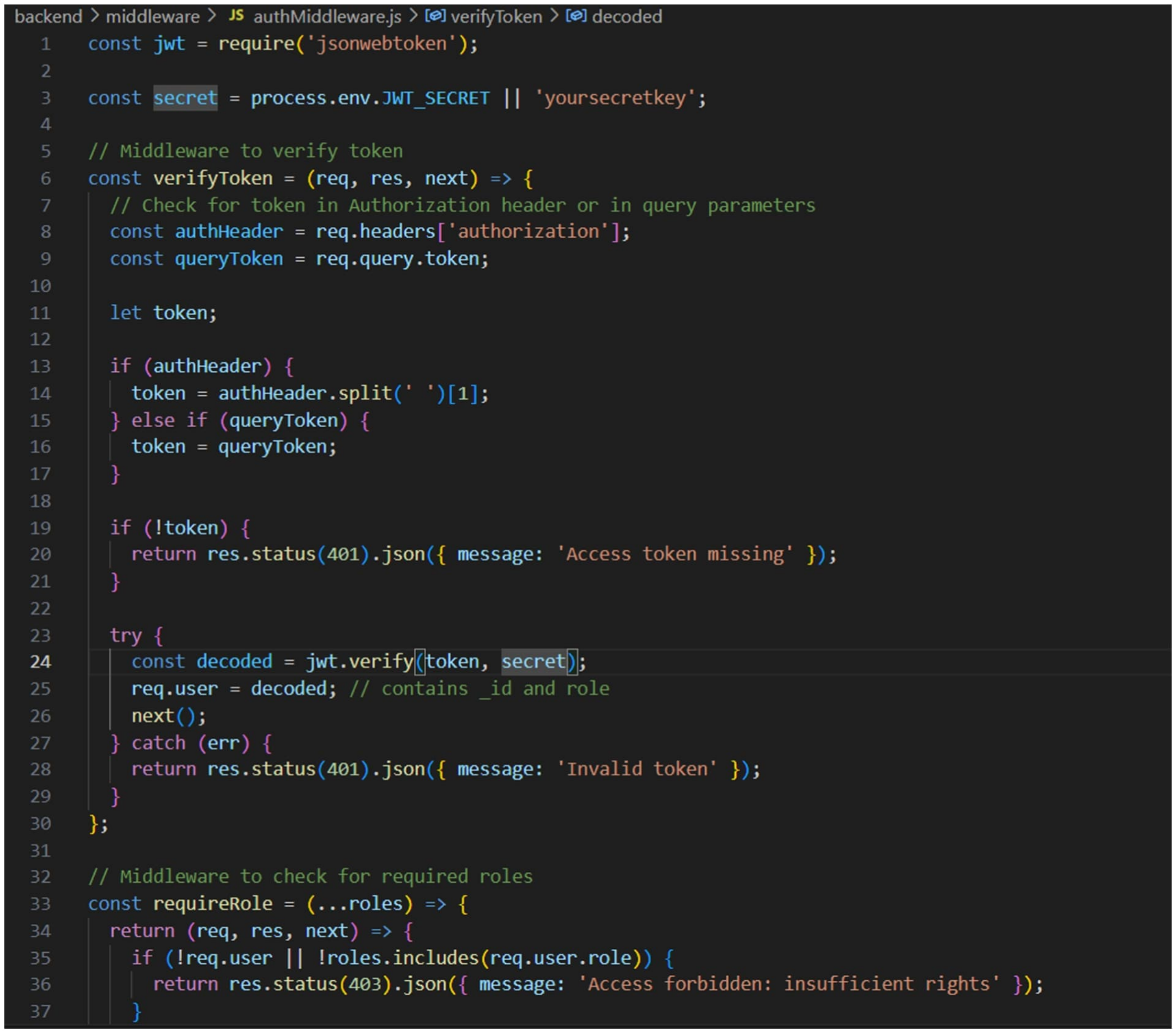
EXPERIMENT 6

Screenshots

authMiddelware.js

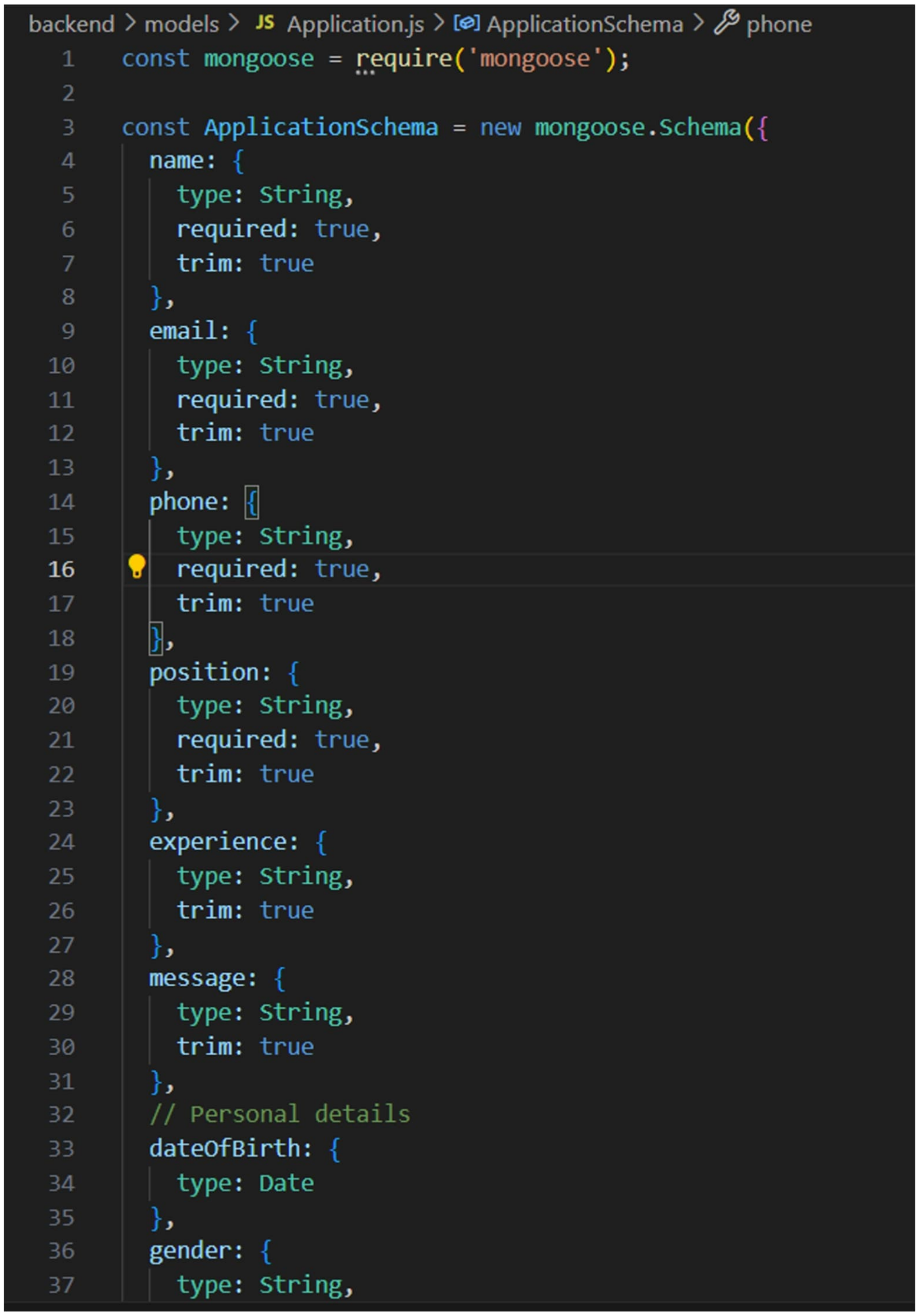


backend > middleware > JS authMiddleware.js > verifyToken > decoded

```javascript
const jwt = require('jsonwebtoken');

const secret = process.env.JWT_SECRET || 'yoursecretkey';

// Middleware to verify token
const verifyToken = (req, res, next) => {
  // Check for token in Authorization header or in query parameters
  const authHeader = req.headers['authorization'];
  const queryToken = req.query.token;

  let token;

  if (authHeader) {
    token = authHeader.split(' ')[1];
  } else if (queryToken) {
    token = queryToken;
  }

  if (!token) {
    return res.status(401).json({ message: 'Access token missing' });
  }

  try {
    const decoded = jwt.verify(token, secret);
    req.user = decoded; // contains _id and role
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
};

// Middleware to check for required roles
const requireRole = (...roles) => {
  return (req, res, next) => {
    if (!req.user || !roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Access forbidden: insufficient rights' });
    }
```

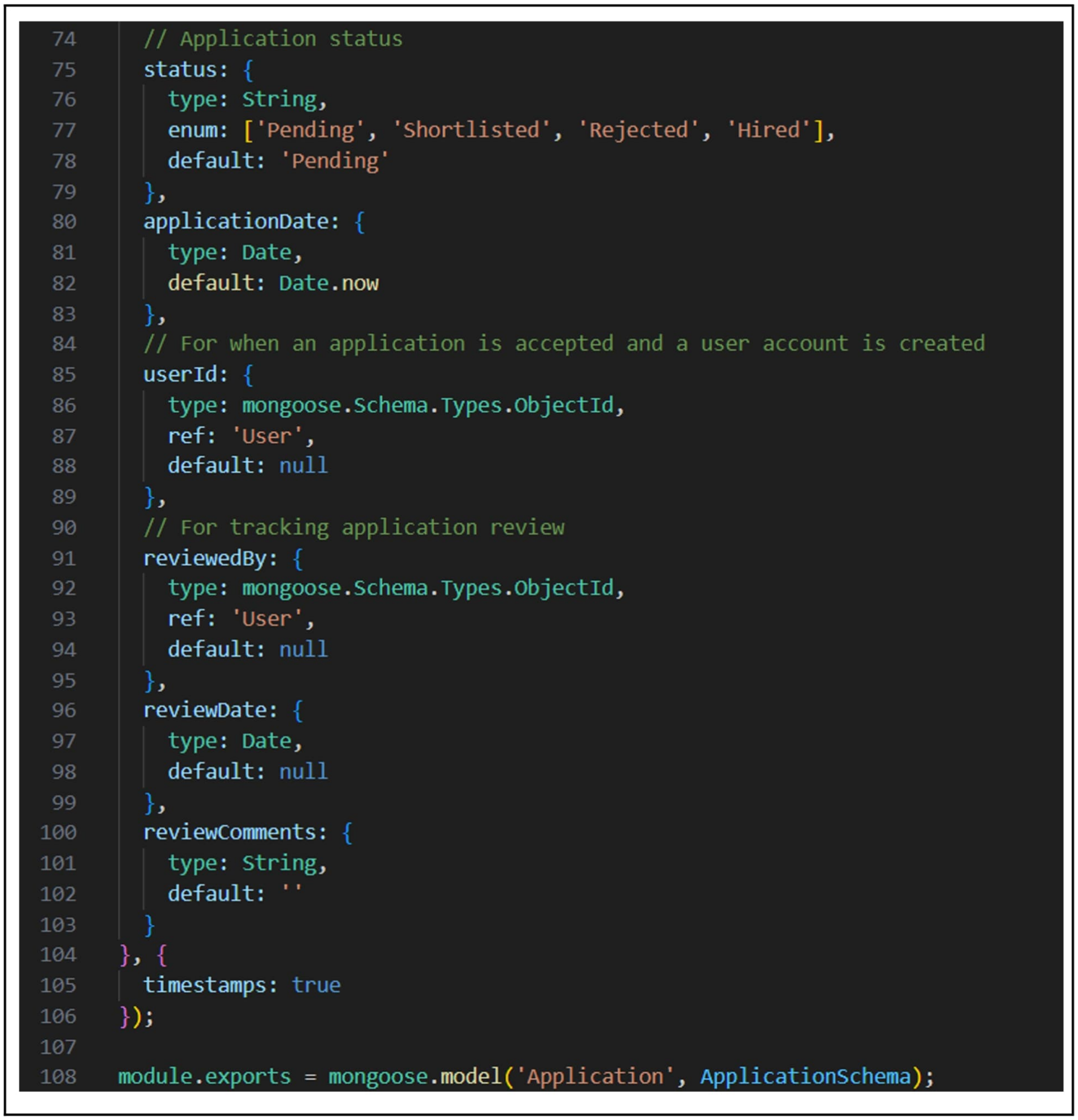
Fig 4.1.1

Application.js


backend > models > JS Application.js > ApplicationSchema > phone

```javascript
const mongoose = require('mongoose');

const ApplicationSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  email: {
    type: String,
    required: true,
    trim: true
  },
  phone: {
    type: String,
    required: true,
    trim: true
  },
  position: {
    type: String,
    required: true,
    trim: true
  },
  experience: {
    type: String,
    trim: true
  },
  message: {
    type: String,
    trim: true
  },
  // Personal details
  dateOfBirth: {
    type: Date
  },
  gender: {
    type: String,
```

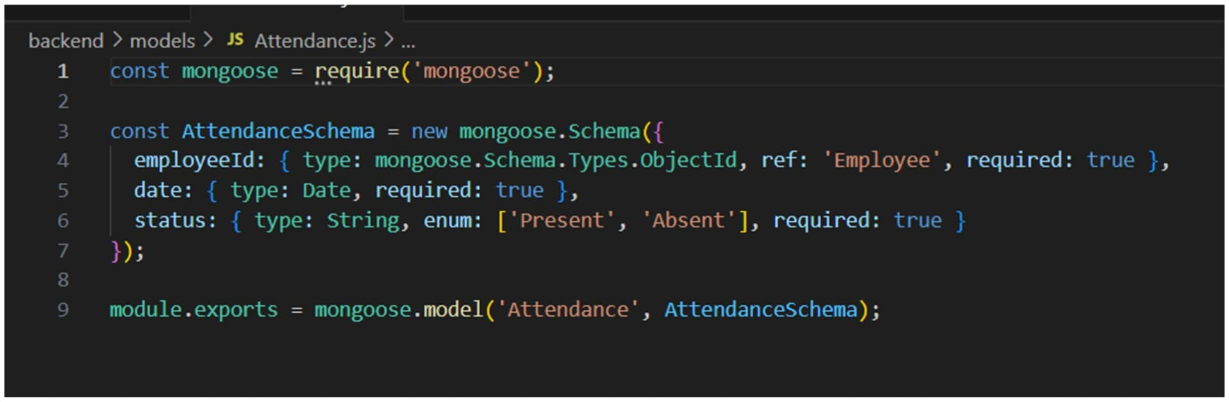
Fig 4.2.1



```
  // Application status
  status: {
    type: String,
    enum: ['Pending', 'Shortlisted', 'Rejected', 'Hired'],
    default: 'Pending'
  },
  applicationDate: {
    type: Date,
    default: Date.now
  },
  // For when an application is accepted and a user account is created
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    default: null
  },
  // For tracking application review
  reviewedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    default: null
  },
  reviewDate: {
    type: Date,
    default: null
  },
  reviewComments: {
    type: String,
    default: ''
  }
}, {
  timestamps: true
});

module.exports = mongoose.model('Application', ApplicationSchema);
```

Fig 4.2.2

Attendance.js

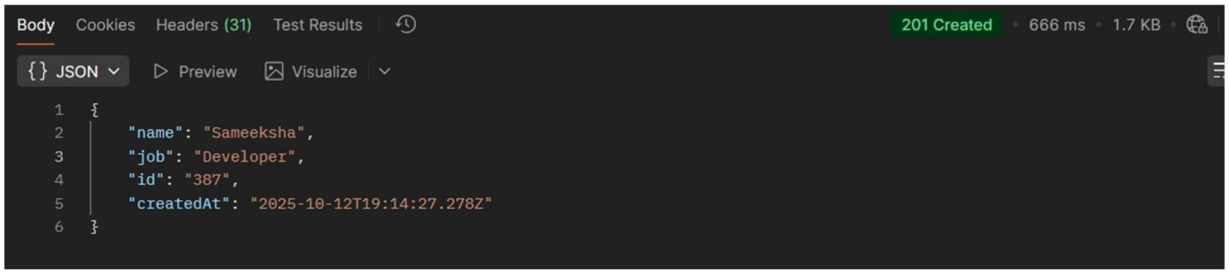


backend > models > JS Attendance.js > ...

```javascript
const mongoose = require('mongoose');

const AttendanceSchema = new mongoose.Schema({
  employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required: true },
  date: { type: Date, required: true },
  status: { type: String, enum: ['Present', 'Absent'], required: true }
});

module.exports = mongoose.model('Attendance', AttendanceSchema);
```

Fig 4.3.1

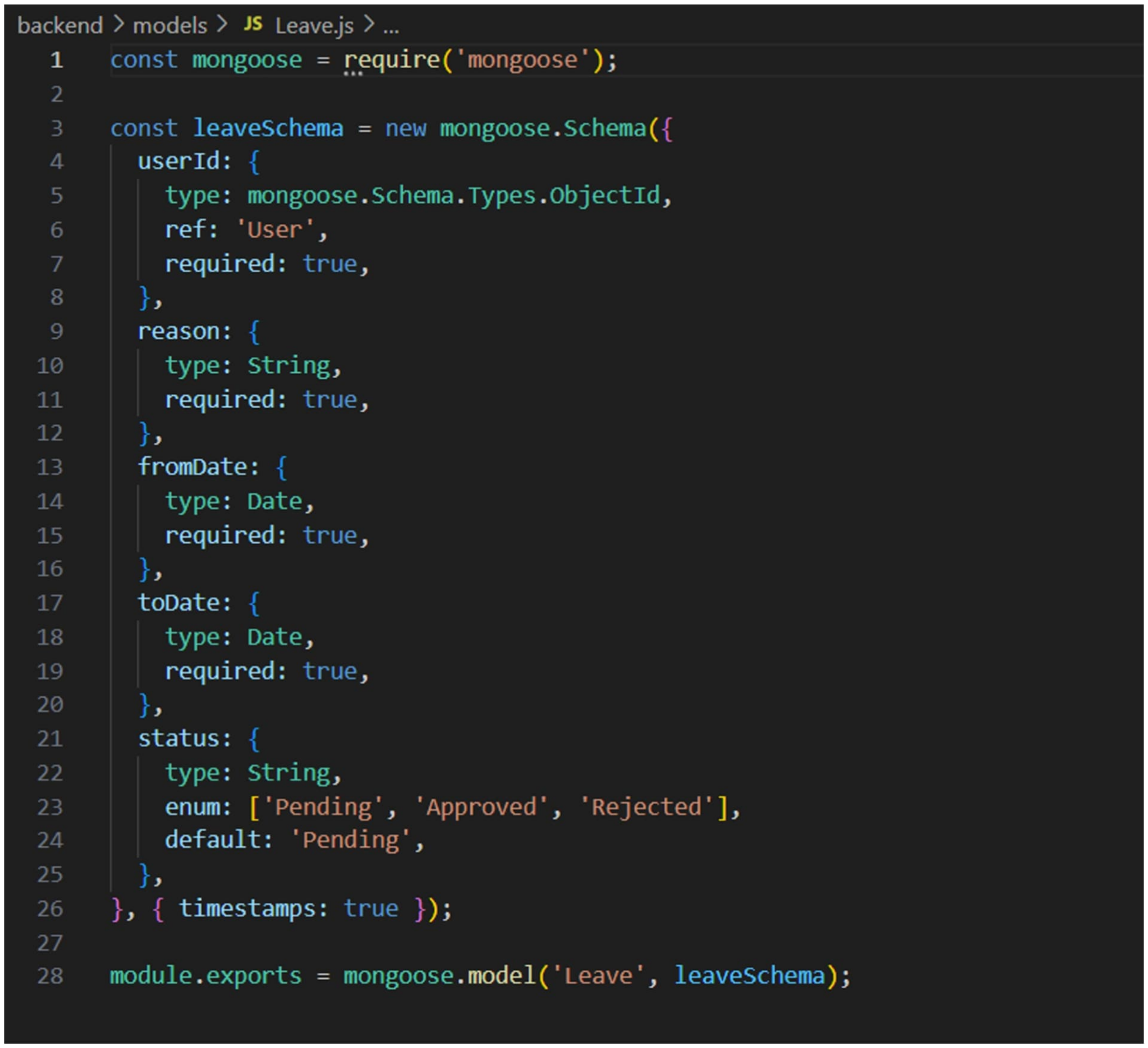
Employee.js



```json
{
    "name": "Sameeksha",
    "job": "Developer",
    "id": "387",
    "createdAt": "2025-10-12T19:14:27.278Z"
}
```

Fig 4.4.1

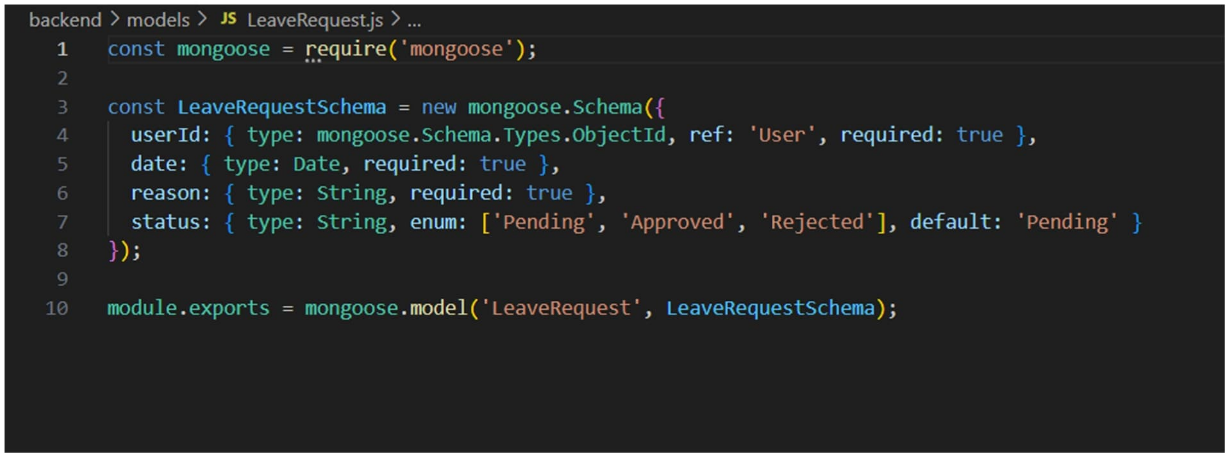
Leave.js


backend > models > JS Leave.js > ...

```javascript
const mongoose = require('mongoose');

const leaveSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  reason: {
    type: String,
    required: true,
  },
  fromDate: {
    type: Date,
    required: true,
  },
  toDate: {
    type: Date,
    required: true,
  },
  status: {
    type: String,
    enum: ['Pending', 'Approved', 'Rejected'],
    default: 'Pending',
  },
}, { timestamps: true });

module.exports = mongoose.model('Leave', leaveSchema);
```

Fig 4.5.1

LeaveRequest.js

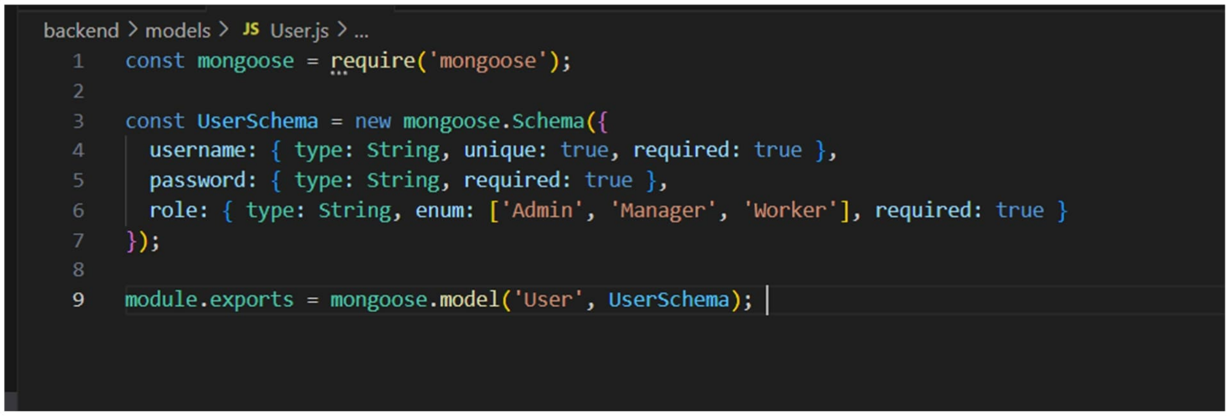


backend > models > JS LeaveRequest.js > ...

```javascript
const mongoose = require('mongoose');

const LeaveRequestSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  date: { type: Date, required: true },
  reason: { type: String, required: true },
  status: { type: String, enum: ['Pending', 'Approved', 'Rejected'], default: 'Pending' }
});

module.exports = mongoose.model('LeaveRequest', LeaveRequestSchema);
```

Fig 4.6.1

User.js

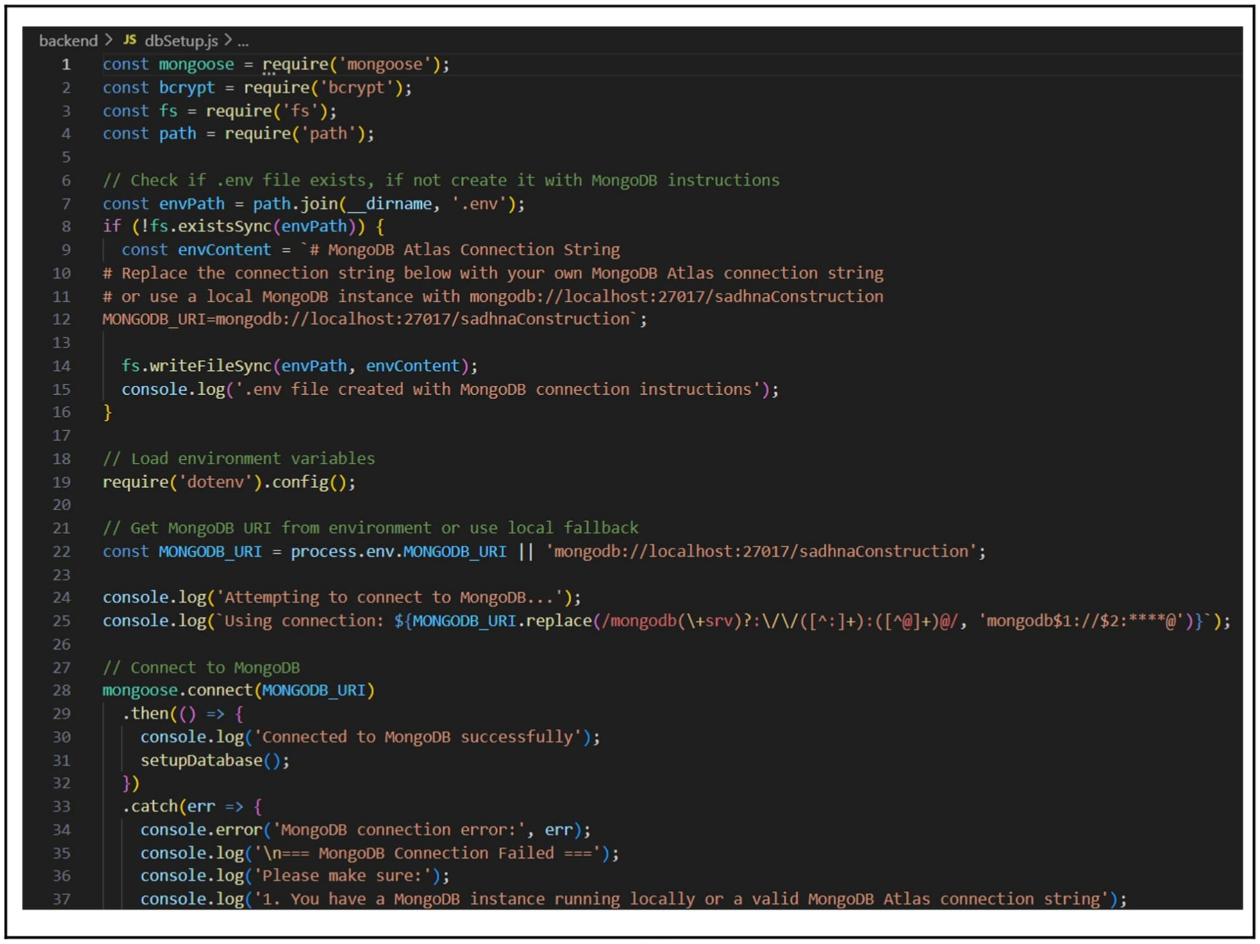


backend > models > JS User.js > ...

```javascript
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  username: { type: String, unique: true, required: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['Admin', 'Manager', 'Worker'], required: true }
});

module.exports = mongoose.model('User', UserSchema);
```

Fig 4.7.1

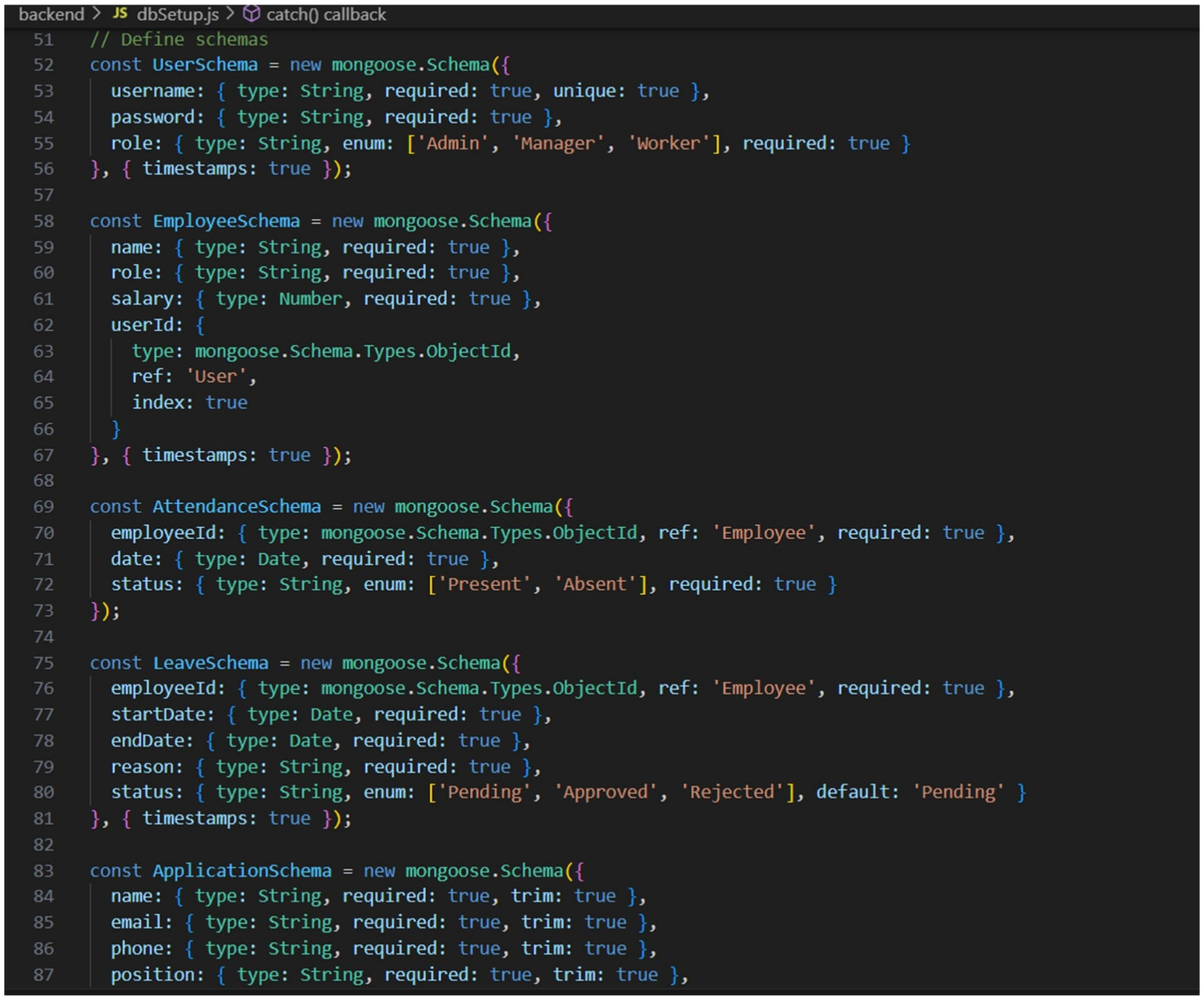
dbSetup.js



backend > JS dbSetup.js > ...

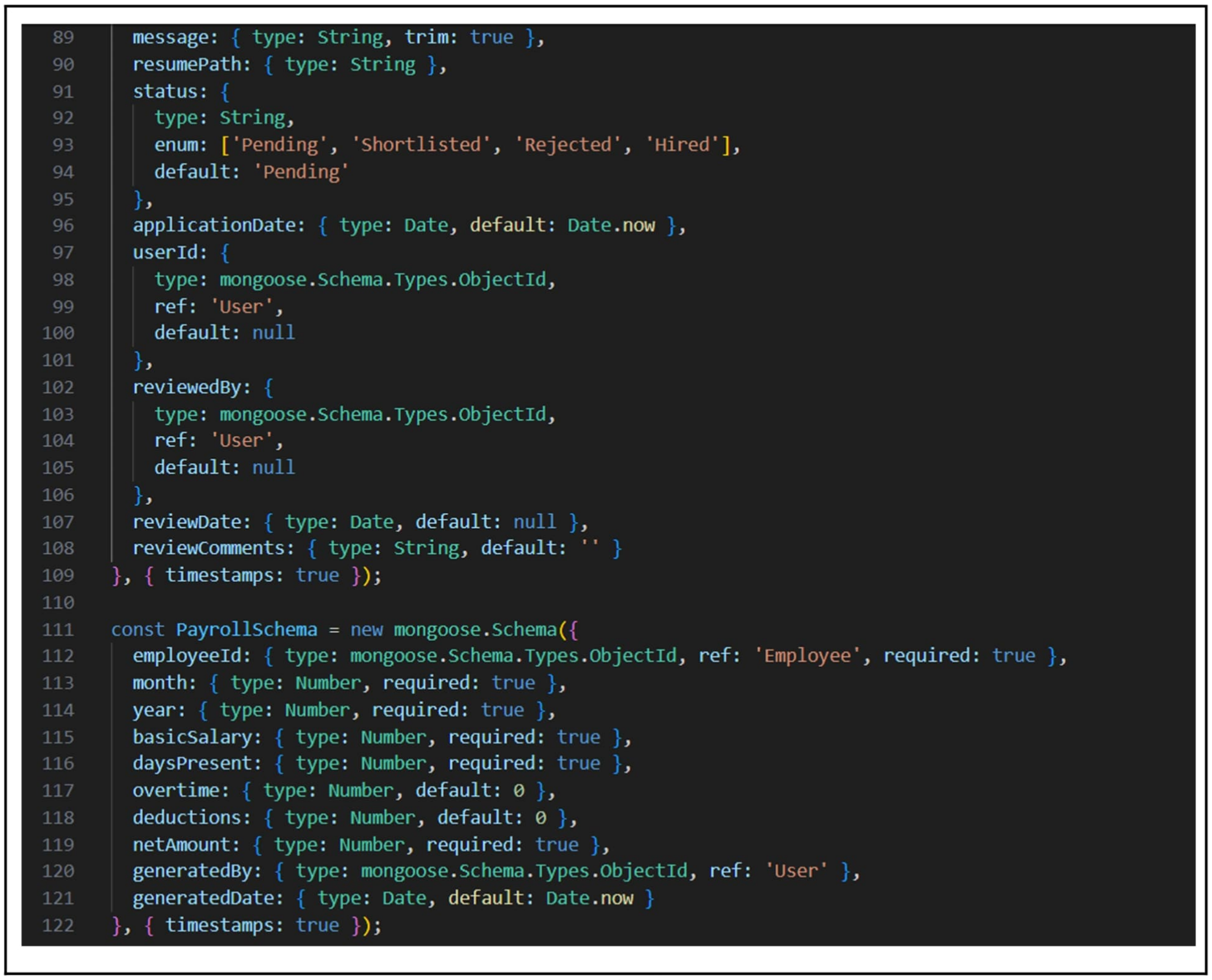
```javascript
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const fs = require('fs');
const path = require('path');

// Check if .env file exists, if not create it with MongoDB instructions
const envPath = path.join(__dirname, '.env');
if (!fs.existsSync(envPath)) {
  const envContent = `# MongoDB Atlas Connection String
# Replace the connection string below with your own MongoDB Atlas connection string
# or use a local MongoDB instance with mongodb://localhost:27017/sadhnaConstruction
MONGODB_URI=mongodb://localhost:27017/sadhnaConstruction`;

  fs.writeFileSync(envPath, envContent);
  console.log('.env file created with MongoDB connection instructions');
}

// Load environment variables
require('dotenv').config();

// Get MongoDB URI from environment or use local fallback
const MONGODB_URI = process.env.MONGODB_URI || 'mongodb://localhost:27017/sadhnaConstruction';

console.log('Attempting to connect to MongoDB...');
console.log(`Using connection: ${MONGODB_URI.replace(/mongodb(\+srv)?:\/\/([^:]+):([^@]+)@/, 'mongodb$1://$2:****@')}`);

// Connect to MongoDB
mongoose.connect(MONGODB_URI)
  .then(() => {
    console.log('Connected to MongoDB successfully');
    setupDatabase();
  })
  .catch(err => {
    console.error('MongoDB connection error:', err);
    console.log('\n=== MongoDB Connection Failed ===');
    console.log('Please make sure:');
    console.log('1. You have a MongoDB instance running locally or a valid MongoDB Atlas connection string');
```

Fig 4.8.1


backend > JS dbSetup.js > catch() callback

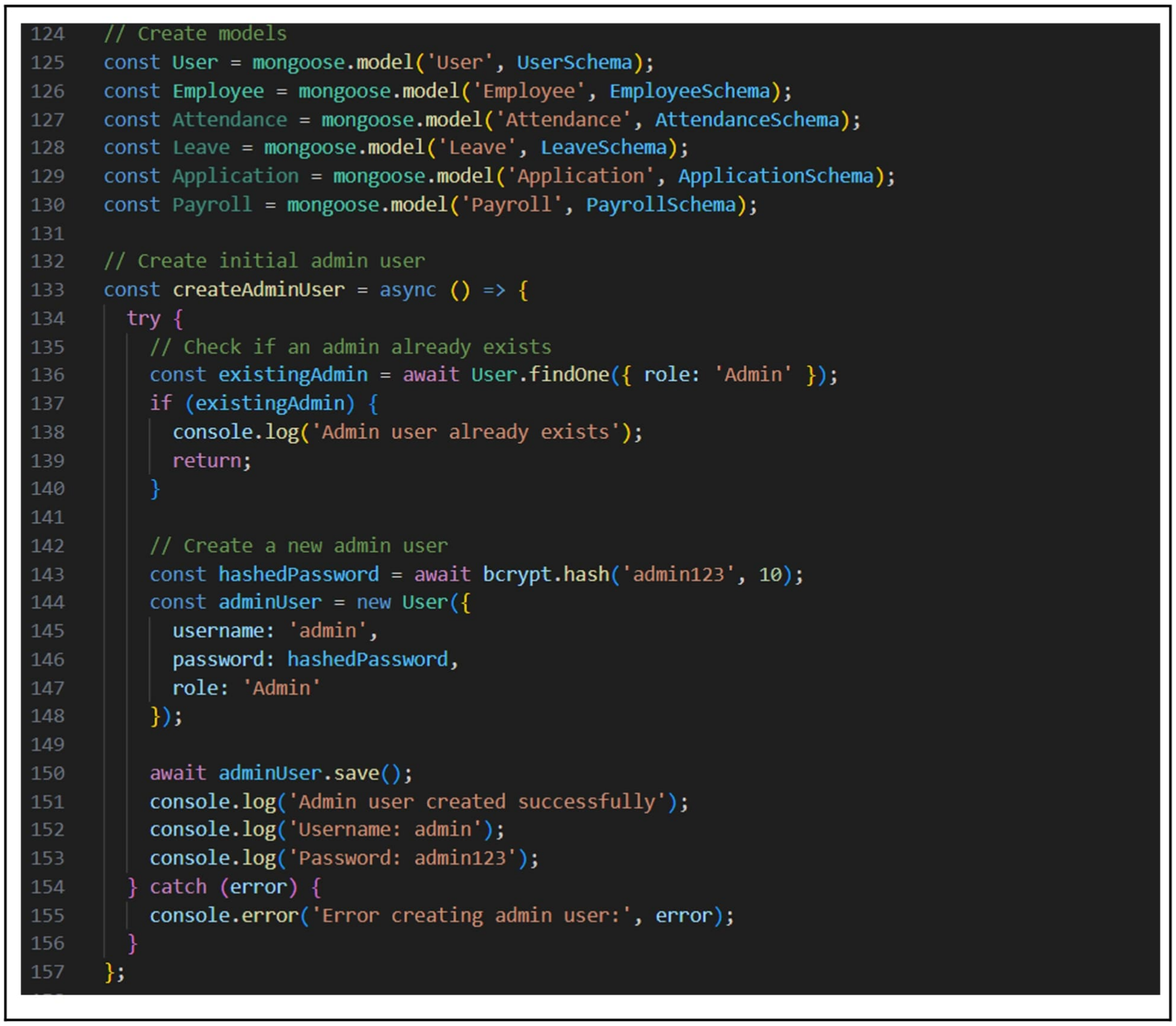
```javascript
// Define schemas
const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['Admin', 'Manager', 'Worker'], required: true }
}, { timestamps: true });

const EmployeeSchema = new mongoose.Schema({
  name: { type: String, required: true },
  role: { type: String, required: true },
  salary: { type: Number, required: true },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    index: true
  }
}, { timestamps: true });

const AttendanceSchema = new mongoose.Schema({
  employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required: true },
  date: { type: Date, required: true },
  status: { type: String, enum: ['Present', 'Absent'], required: true }
});

const LeaveSchema = new mongoose.Schema({
  employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required: true },
  startDate: { type: Date, required: true },
  endDate: { type: Date, required: true },
  reason: { type: String, required: true },
  status: { type: String, enum: ['Pending', 'Approved', 'Rejected'], default: 'Pending' }
}, { timestamps: true });

const ApplicationSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  email: { type: String, required: true, trim: true },
  phone: { type: String, required: true, trim: true },
  position: { type: String, required: true, trim: true },
```

Fig 4.8.2



```
  message: { type: String, trim: true },
  resumePath: { type: String },
  status: {
    type: String,
    enum: ['Pending', 'Shortlisted', 'Rejected', 'Hired'],
    default: 'Pending'
  },
  applicationDate: { type: Date, default: Date.now },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    default: null
  },
  reviewedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    default: null
  },
  reviewDate: { type: Date, default: null },
  reviewComments: { type: String, default: '' }
}, { timestamps: true });

const PayrollSchema = new mongoose.Schema({
  employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required: true },
  month: { type: Number, required: true },
  year: { type: Number, required: true },
  basicSalary: { type: Number, required: true },
  daysPresent: { type: Number, required: true },
  overtime: { type: Number, default: 0 },
  deductions: { type: Number, default: 0 },
  netAmount: { type: Number, required: true },
  generatedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  generatedDate: { type: Date, default: Date.now }
}, { timestamps: true });
```

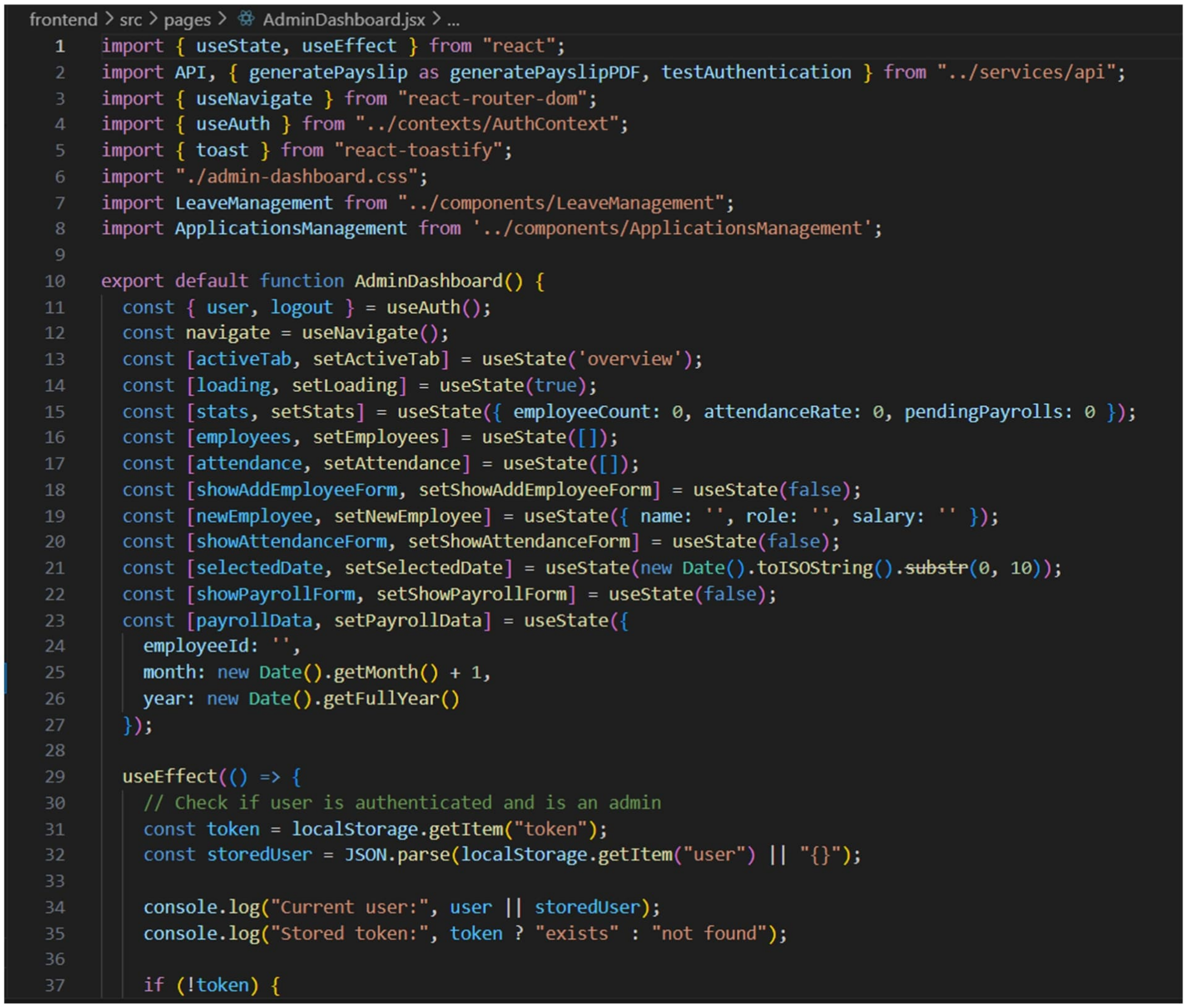
Fig 4.8.3



```javascript
// Create models
const User = mongoose.model('User', UserSchema);
const Employee = mongoose.model('Employee', EmployeeSchema);
const Attendance = mongoose.model('Attendance', AttendanceSchema);
const Leave = mongoose.model('Leave', LeaveSchema);
const Application = mongoose.model('Application', ApplicationSchema);
const Payroll = mongoose.model('Payroll', PayrollSchema);

// Create initial admin user
const createAdminUser = async () => {
  try {
    // Check if an admin already exists
    const existingAdmin = await User.findOne({ role: 'Admin' });
    if (existingAdmin) {
      console.log('Admin user already exists');
      return;
    }

    // Create a new admin user
    const hashedPassword = await bcrypt.hash('admin123', 10);
    const adminUser = new User({
      username: 'admin',
      password: hashedPassword,
      role: 'Admin'
    });

    await adminUser.save();
    console.log('Admin user created successfully');
    console.log('Username: admin');
    console.log('Password: admin123');
  } catch (error) {
    console.error('Error creating admin user:', error);
  }
};
```

Fig 4.8.4

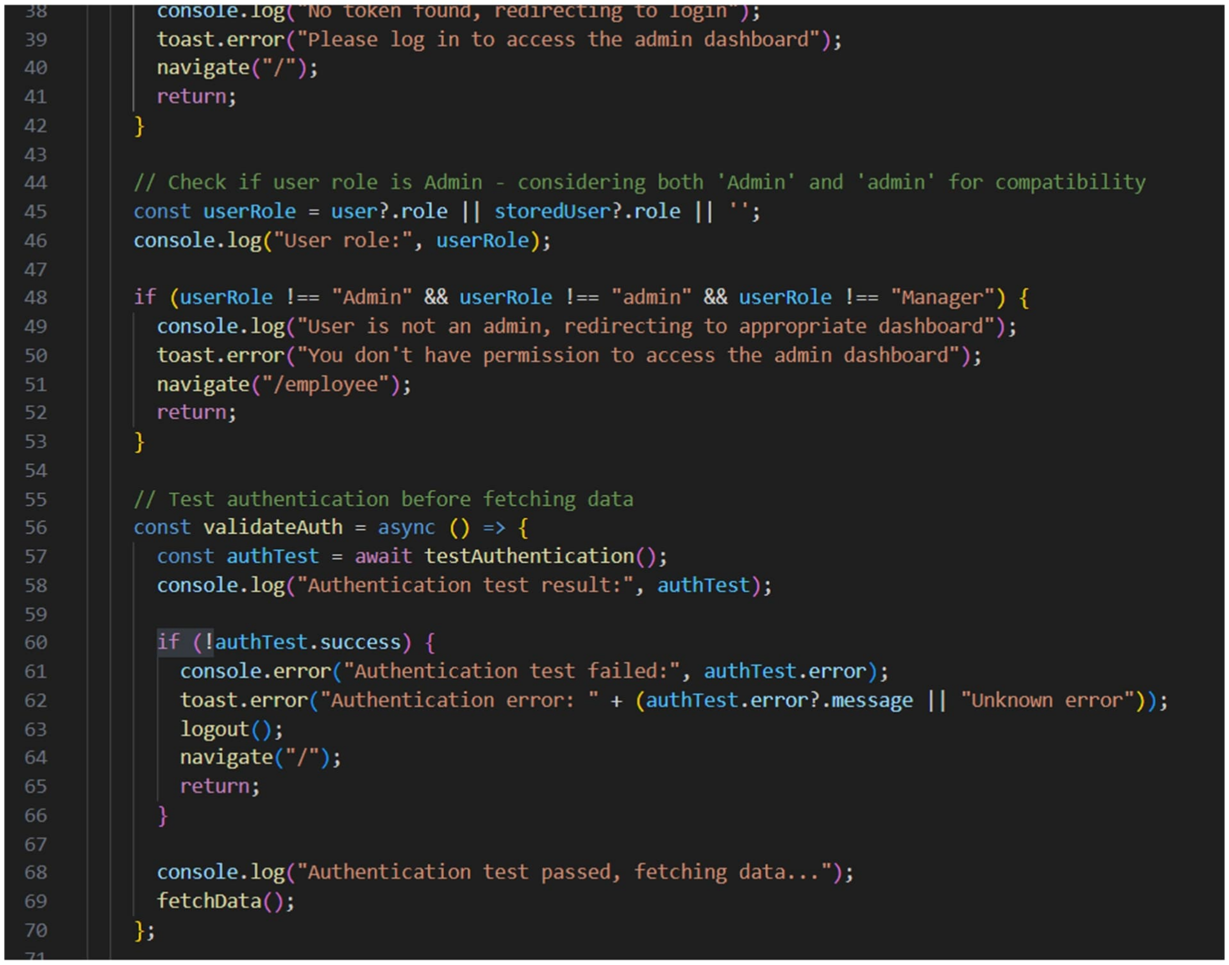
AdminDashboard.jsx



frontend > src > pages > AdminDashboard.jsx > ...

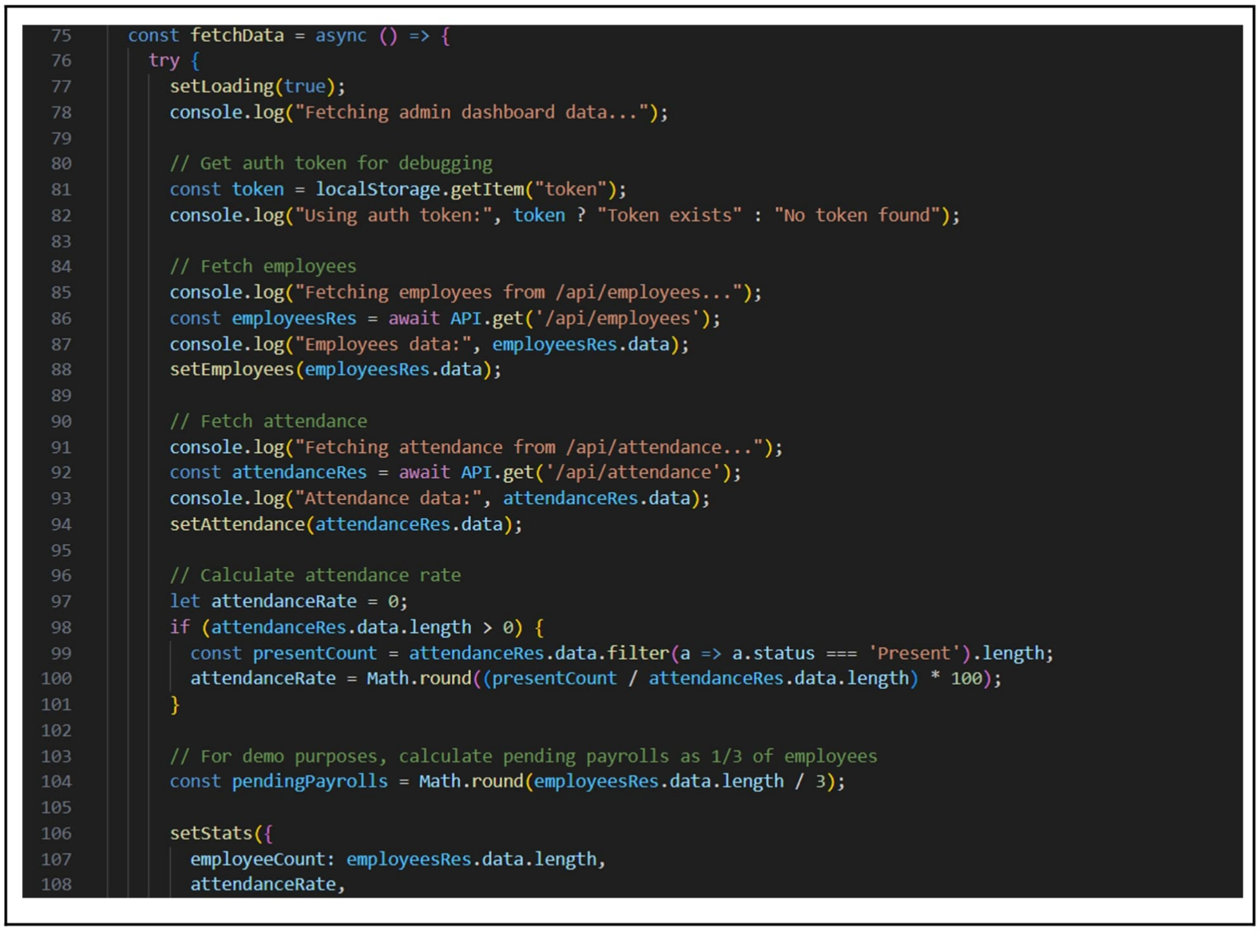
```jsx
import { useState, useEffect } from "react";
import API, { generatePayslip as generatePayslipPDF, testAuthentication } from "../services/api";
import { useNavigate } from "react-router-dom";
import { useAuth } from "../contexts/AuthContext";
import { toast } from "react-toastify";
import "./admin-dashboard.css";
import LeaveManagement from "../components/LeaveManagement";
import ApplicationsManagement from '../components/ApplicationsManagement';

export default function AdminDashboard() {
  const { user, logout } = useAuth();
  const navigate = useNavigate();
  const [activeTab, setActiveTab] = useState('overview');
  const [loading, setLoading] = useState(true);
  const [stats, setStats] = useState({ employeeCount: 0, attendanceRate: 0, pendingPayrolls: 0 });
  const [employees, setEmployees] = useState([]);
  const [attendance, setAttendance] = useState([]);
  const [showAddEmployeeForm, setShowAddEmployeeForm] = useState(false);
  const [newEmployee, setNewEmployee] = useState({ name: '', role: '', salary: '' });
  const [showAttendanceForm, setShowAttendanceForm] = useState(false);
  const [selectedDate, setSelectedDate] = useState(new Date().toISOString().~~substr~~(0, 10));
  const [showPayrollForm, setShowPayrollForm] = useState(false);
  const [payrollData, setPayrollData] = useState({
    employeeId: '',
    month: new Date().getMonth() + 1,
    year: new Date().getFullYear()
  });

  useEffect(() => {
    // Check if user is authenticated and is an admin
    const token = localStorage.getItem("token");
    const storedUser = JSON.parse(localStorage.getItem("user") || "{}");

    console.log("Current user:", user || storedUser);
    console.log("Stored token:", token ? "exists" : "not found");

    if (!token) {
```

Fig 4.9.1



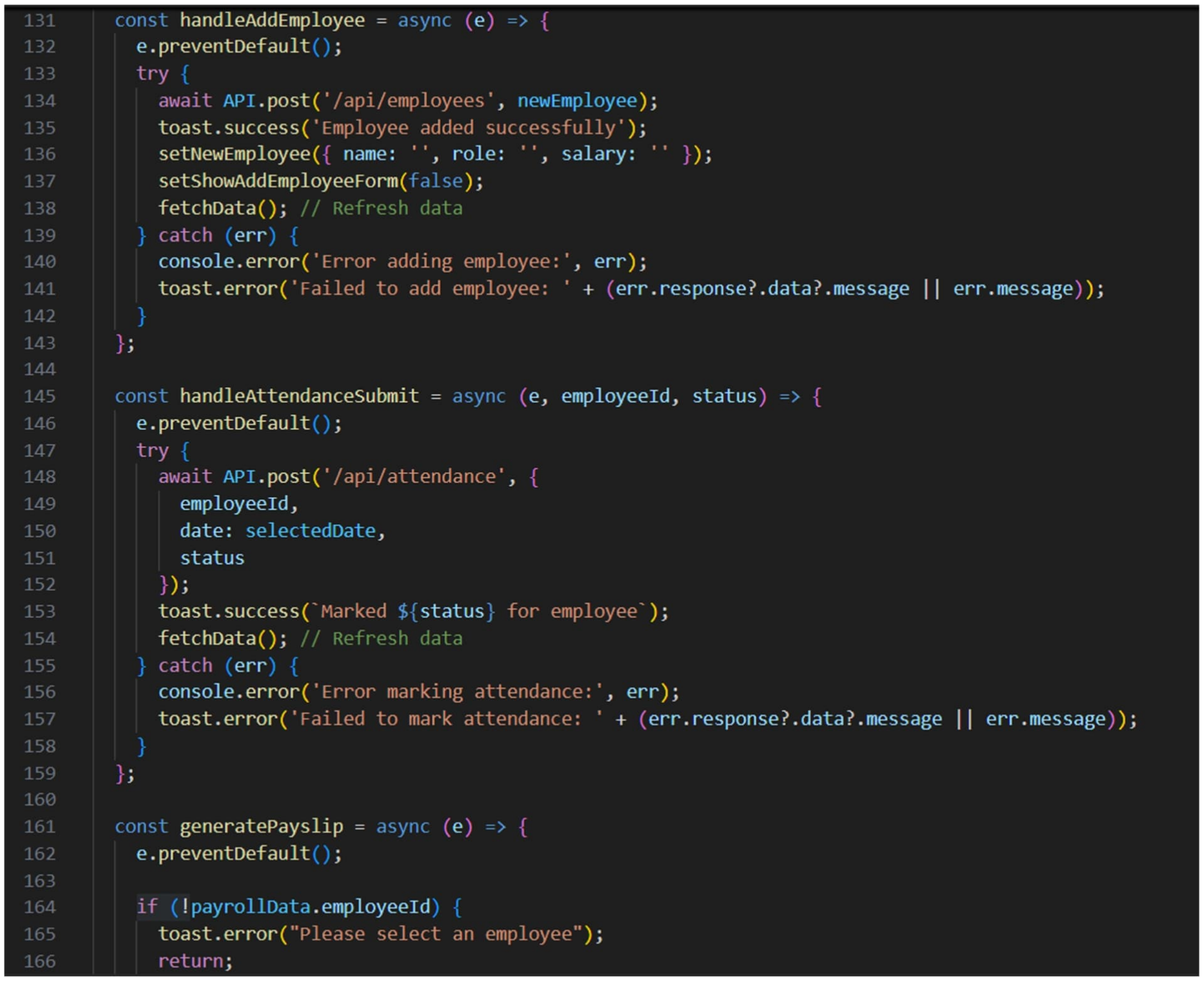
```javascript
      console.log("No token found, redirecting to login");
      toast.error("Please log in to access the admin dashboard");
      navigate("/");
      return;
    }

    // Check if user role is Admin - considering both 'Admin' and 'admin' for compatibility
    const userRole = user?.role || storedUser?.role || '';
    console.log("User role:", userRole);

    if (userRole !== "Admin" && userRole !== "admin" && userRole !== "Manager") {
      console.log("User is not an admin, redirecting to appropriate dashboard");
      toast.error("You don't have permission to access the admin dashboard");
      navigate("/employee");
      return;
    }

    // Test authentication before fetching data
    const validateAuth = async () => {
      const authTest = await testAuthentication();
      console.log("Authentication test result:", authTest);

      if (!authTest.success) {
        console.error("Authentication test failed:", authTest.error);
        toast.error("Authentication error: " + (authTest.error?.message || "Unknown error"));
        logout();
        navigate("/");
        return;
      }

      console.log("Authentication test passed, fetching data...");
      fetchData();
    };
```

Fig 4.9.2



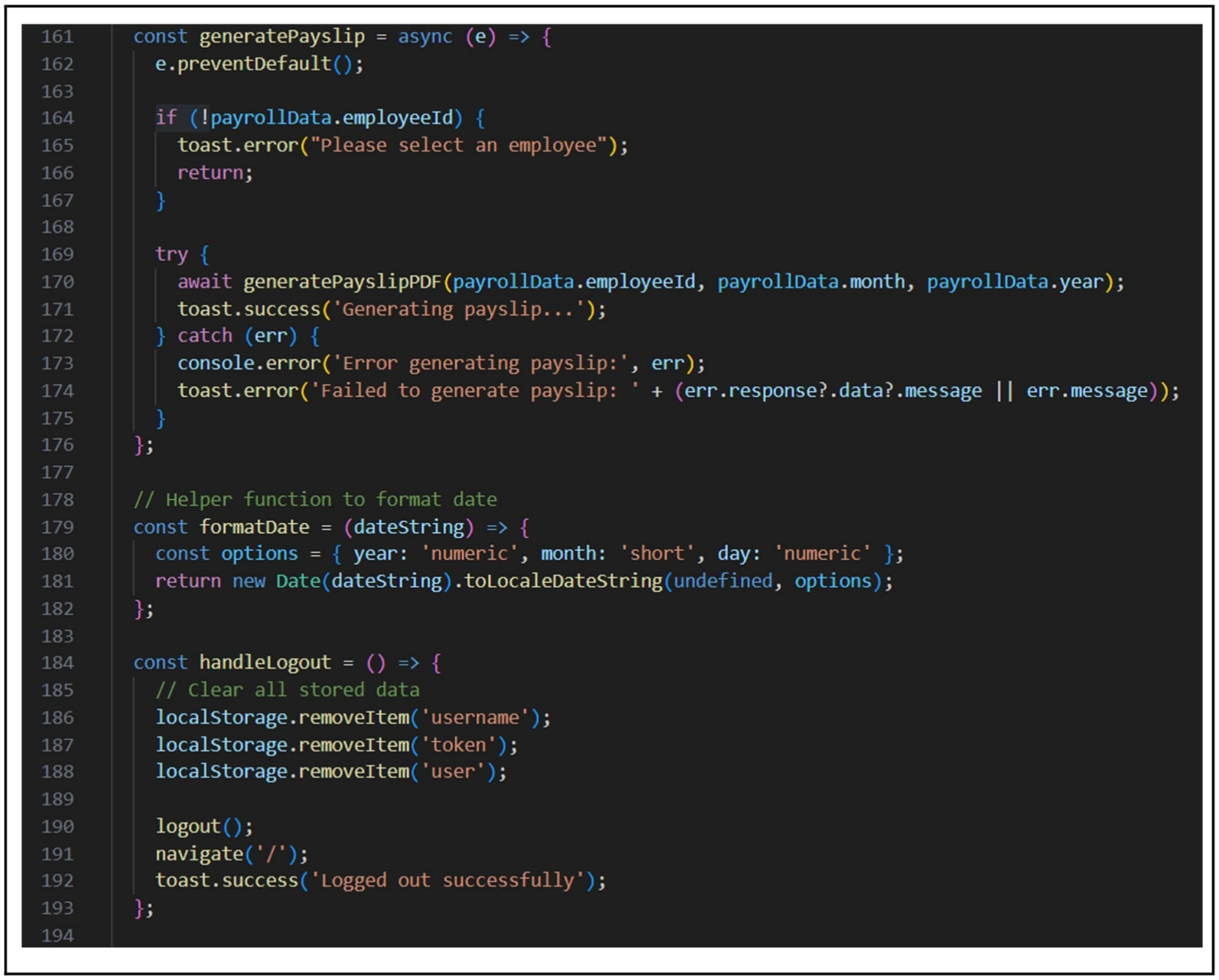
```javascript
const fetchData = async () => {
  try {
    setLoading(true);
    console.log("Fetching admin dashboard data...");

    // Get auth token for debugging
    const token = localStorage.getItem("token");
    console.log("Using auth token:", token ? "Token exists" : "No token found");

    // Fetch employees
    console.log("Fetching employees from /api/employees...");
    const employeesRes = await API.get('/api/employees');
    console.log("Employees data:", employeesRes.data);
    setEmployees(employeesRes.data);

    // Fetch attendance
    console.log("Fetching attendance from /api/attendance...");
    const attendanceRes = await API.get('/api/attendance');
    console.log("Attendance data:", attendanceRes.data);
    setAttendance(attendanceRes.data);

    // Calculate attendance rate
    let attendanceRate = 0;
    if (attendanceRes.data.length > 0) {
      const presentCount = attendanceRes.data.filter(a => a.status === 'Present').length;
      attendanceRate = Math.round((presentCount / attendanceRes.data.length) * 100);
    }

    // For demo purposes, calculate pending payrolls as 1/3 of employees
    const pendingPayrolls = Math.round(employeesRes.data.length / 3);

    setStats({
      employeeCount: employeesRes.data.length,
      attendanceRate,
```

Fig 4.9.3



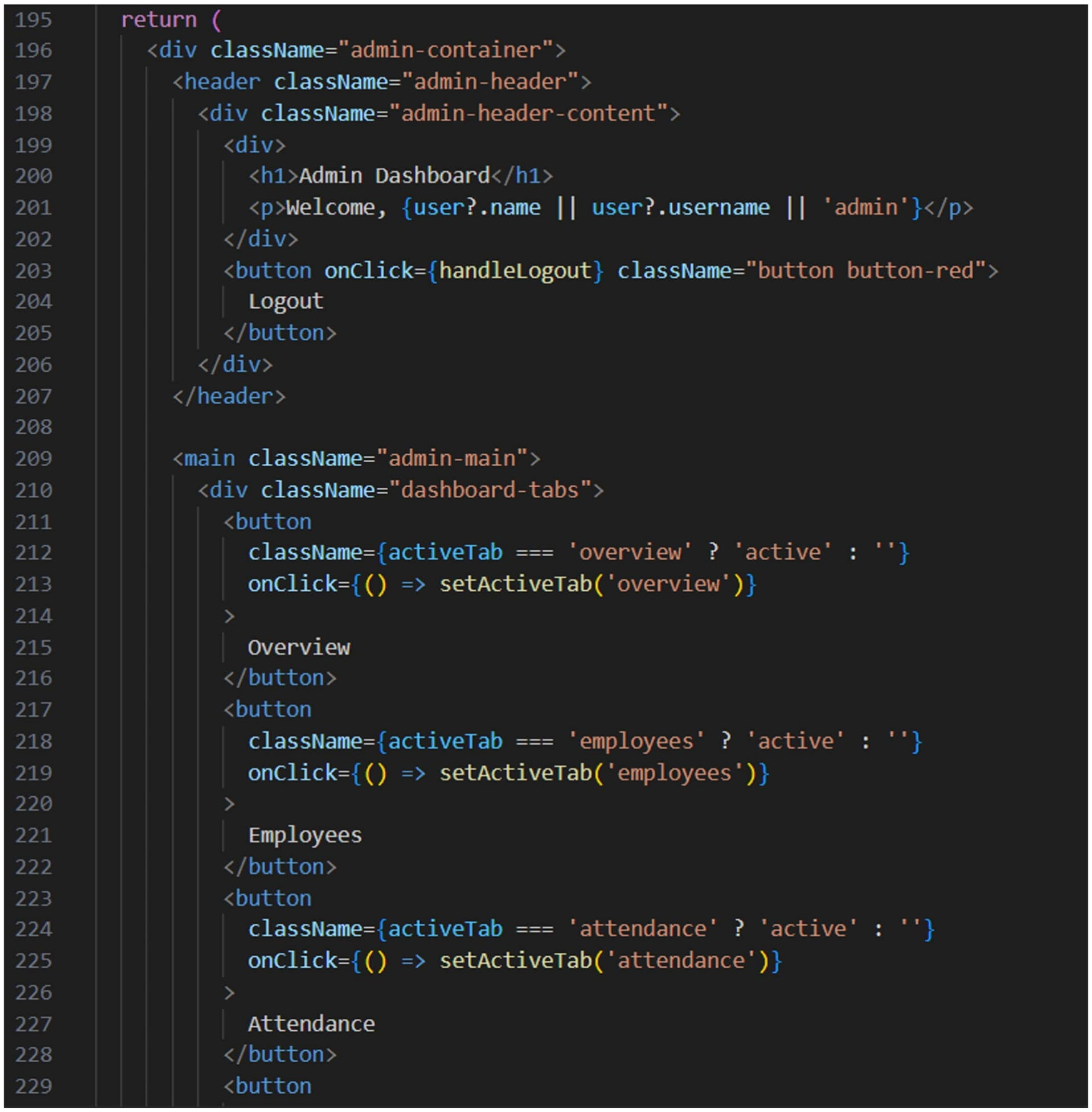
```
const handleAddEmployee = async (e) => {
  e.preventDefault();
  try {
    await API.post('/api/employees', newEmployee);
    toast.success('Employee added successfully');
    setNewEmployee({ name: '', role: '', salary: '' });
    setShowAddEmployeeForm(false);
    fetchData(); // Refresh data
  } catch (err) {
    console.error('Error adding employee:', err);
    toast.error('Failed to add employee: ' + (err.response?.data?.message || err.message));
  }
};

const handleAttendanceSubmit = async (e, employeeId, status) => {
  e.preventDefault();
  try {
    await API.post('/api/attendance', {
      employeeId,
      date: selectedDate,
      status
    });
    toast.success(`Marked ${status} for employee`);
    fetchData(); // Refresh data
  } catch (err) {
    console.error('Error marking attendance:', err);
    toast.error('Failed to mark attendance: ' + (err.response?.data?.message || err.message));
  }
};

const generatePayslip = async (e) => {
  e.preventDefault();

  if (!payrollData.employeeId) {
    toast.error("Please select an employee");
    return;
```

Fig 4.9.4



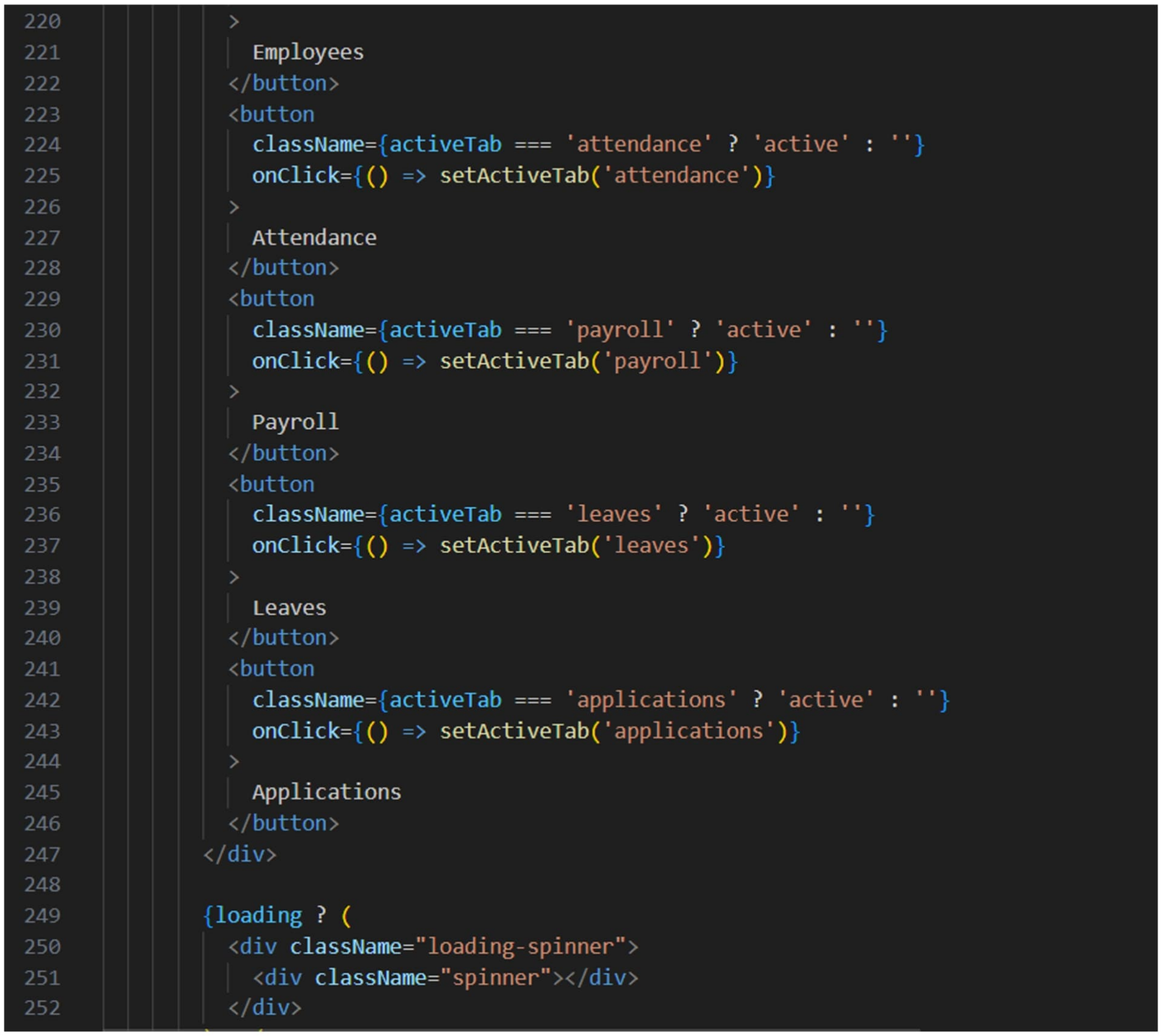
```
const generatePayslip = async (e) => {
  e.preventDefault();

  if (!payrollData.employeeId) {
    toast.error("Please select an employee");
    return;
  }

  try {
    await generatePayslipPDF(payrollData.employeeId, payrollData.month, payrollData.year);
    toast.success('Generating payslip...');
  } catch (err) {
    console.error('Error generating payslip:', err);
    toast.error('Failed to generate payslip: ' + (err.response?.data?.message || err.message));
  }
};

// Helper function to format date
const formatDate = (dateString) => {
  const options = { year: 'numeric', month: 'short', day: 'numeric' };
  return new Date(dateString).toLocaleDateString(undefined, options);
};

const handleLogout = () => {
  // Clear all stored data
  localStorage.removeItem('username');
  localStorage.removeItem('token');
  localStorage.removeItem('user');

  logout();
  navigate('/');
  toast.success('Logged out successfully');
};

```

Fig 4.9.5



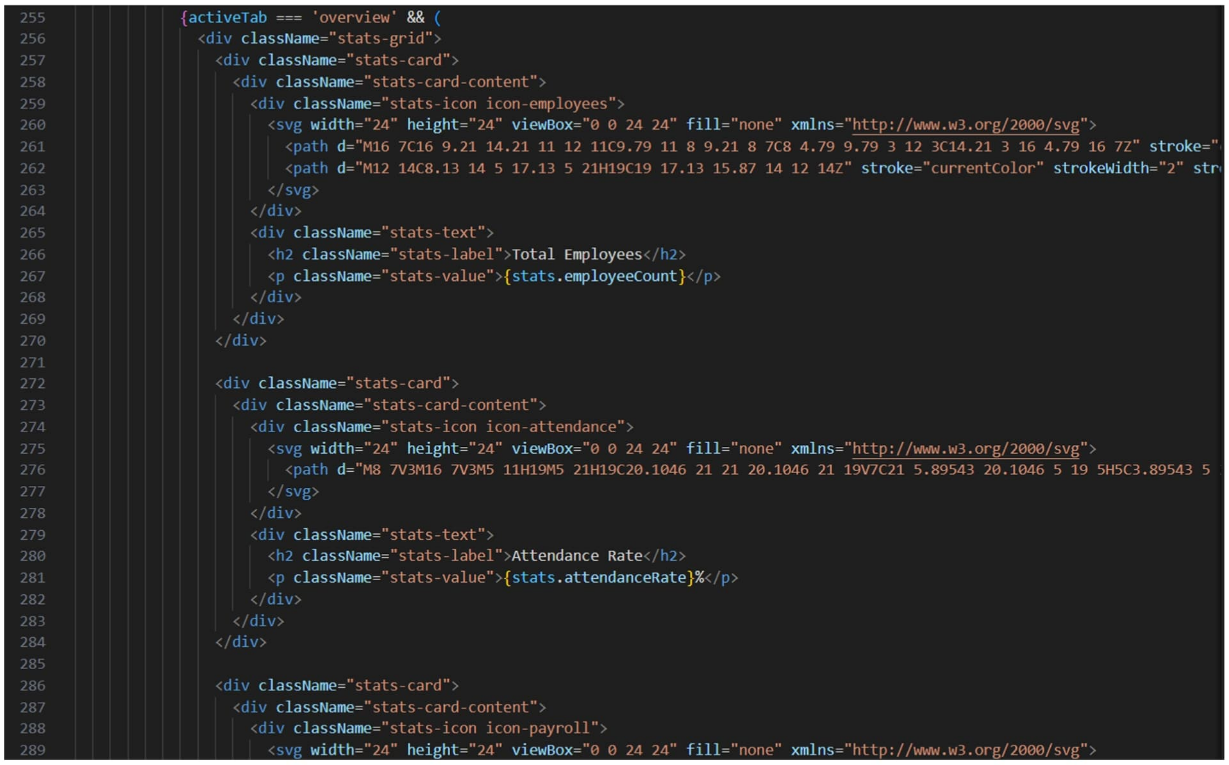
```
  return (
    <div className="admin-container">
      <header className="admin-header">
        <div className="admin-header-content">
          <div>
            <h1>Admin Dashboard</h1>
            <p>Welcome, {user?.name || user?.username || 'admin'}</p>
          </div>
          <button onClick={handleLogout} className="button button-red">
            Logout
          </button>
        </div>
      </header>

      <main className="admin-main">
        <div className="dashboard-tabs">
          <button
            className={activeTab === 'overview' ? 'active' : ''}
            onClick={() => setActiveTab('overview')}
          >
            Overview
          </button>
          <button
            className={activeTab === 'employees' ? 'active' : ''}
            onClick={() => setActiveTab('employees')}
          >
            Employees
          </button>
          <button
            className={activeTab === 'attendance' ? 'active' : ''}
            onClick={() => setActiveTab('attendance')}
          >
            Attendance
          </button>
          <button
```

Fig 4.9.6



```
          >
            Employees
          </button>
          <button
            className={activeTab === 'attendance' ? 'active' : ''}
            onClick={() => setActiveTab('attendance')}
          >
            Attendance
          </button>
          <button
            className={activeTab === 'payroll' ? 'active' : ''}
            onClick={() => setActiveTab('payroll')}
          >
            Payroll
          </button>
          <button
            className={activeTab === 'leaves' ? 'active' : ''}
            onClick={() => setActiveTab('leaves')}
          >
            Leaves
          </button>
          <button
            className={activeTab === 'applications' ? 'active' : ''}
            onClick={() => setActiveTab('applications')}
          >
            Applications
          </button>
        </div>

        {loading ? (
          <div className="loading-spinner">
            <div className="spinner"></div>
          </div>
```

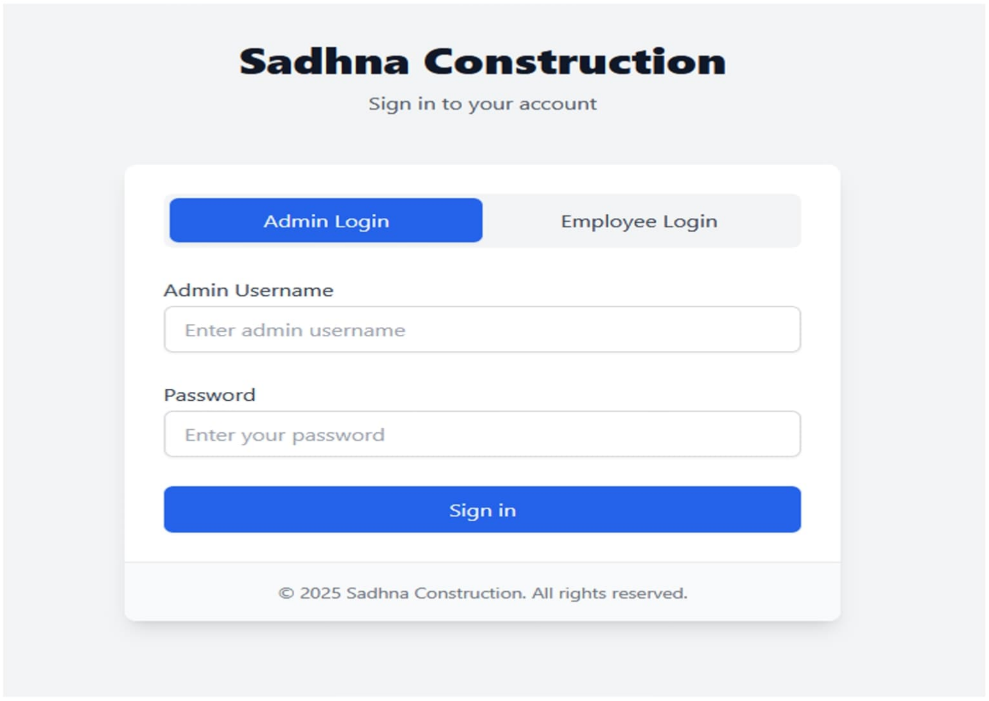
Fig 4.9.7



```
{activeTab === 'overview' && (
  <div className="stats-grid">
    <div className="stats-card">
      <div className="stats-card-content">
        <div className="stats-icon icon-employees">
          <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
            <path d="M16 7C16 9.21 14.21 11 12 11C9.79 11 8 9.21 8 7C8 4.79 9.79 3 12 3C14.21 3 16 4.79 16 7Z" stroke="
            <path d="M12 14C8.13 14 5 17.13 5 21H19C19 17.13 15.87 14 12 14Z" stroke="currentColor" strokeWidth="2" str
          </svg>
        </div>
        <div className="stats-text">
          <h2 className="stats-label">Total Employees</h2>
          <p className="stats-value">{stats.employeeCount}</p>
        </div>
      </div>
    </div>

    <div className="stats-card">
      <div className="stats-card-content">
        <div className="stats-icon icon-attendance">
          <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
            <path d="M8 7V3M16 7V3M5 11H19M5 21H19C20.1046 21 21 20.1046 21 19V7C21 5.89543 20.1046 5 19 5H5C3.89543 5
          </svg>
        </div>
        <div className="stats-text">
          <h2 className="stats-label">Attendance Rate</h2>
          <p className="stats-value">{stats.attendanceRate}%</p>
        </div>
      </div>
    </div>

    <div className="stats-card">
      <div className="stats-card-content">
        <div className="stats-icon icon-payroll">
          <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
```

Fig 4.9.8

5. OUTPUT



Sadhna Construction

Sign in to your account

Admin Login | Employee Login

Admin Username

Enter admin username

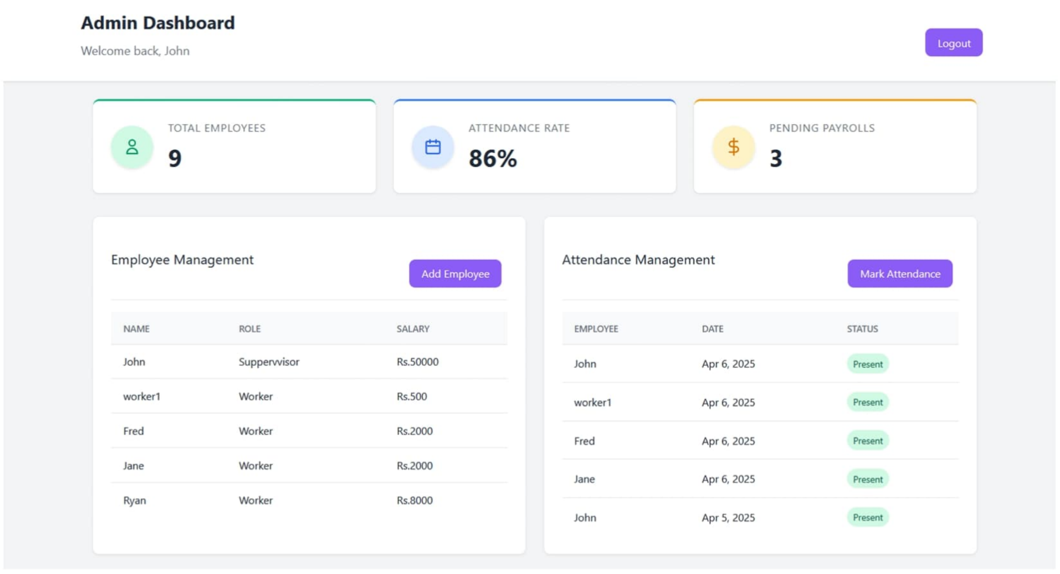
Password

Enter your password

Sign in

© 2025 Sadhna Construction. All rights reserved.

Figure 5.1



Admin Dashboard

Welcome back, John

Logout

TOTAL EMPLOYEES
9

ATTENDANCE RATE
86%

PENDING PAYROLLS
3

Employee Management

Add Employee

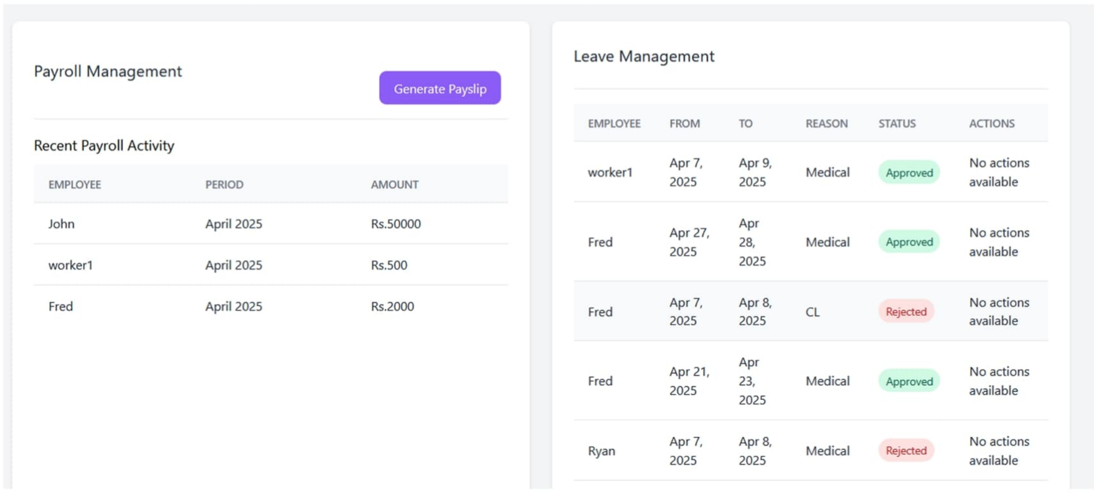
NAME	ROLE	SALARY
John	Suppervisor	Rs.50000
worker1	Worker	Rs.500
Fred	Worker	Rs.2000
Jane	Worker	Rs.2000
Ryan	Worker	Rs.8000

Attendance Management

Mark Attendance

EMPLOYEE	DATE	STATUS
John	Apr 6, 2025	Present
worker1	Apr 6, 2025	Present
Fred	Apr 6, 2025	Present
Jane	Apr 6, 2025	Present
John	Apr 5, 2025	Present

Figure 5.2



Payroll Management

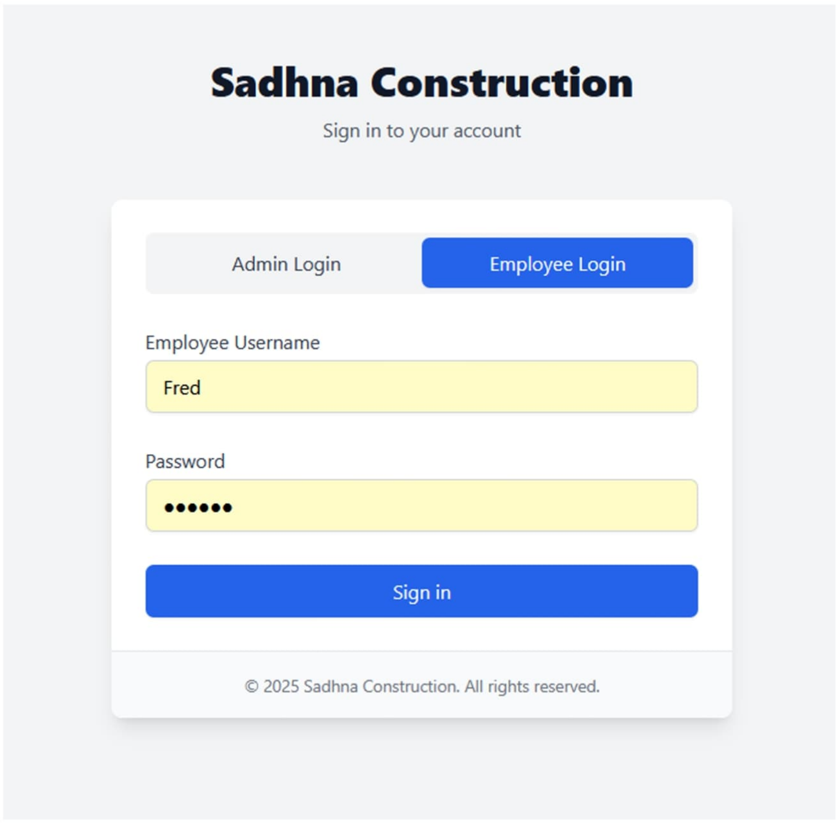
Recent Payroll Activity

EMPLOYEE	PERIOD	AMOUNT
John	April 2025	Rs.50000
worker1	April 2025	Rs.500
Fred	April 2025	Rs.2000

Leave Management

EMPLOYEE	FROM	TO	REASON	STATUS	ACTIONS
worker1	Apr 7, 2025	Apr 9, 2025	Medical	Approved	No actions available
Fred	Apr 27, 2025	Apr 28, 2025	Medical	Approved	No actions available
Fred	Apr 7, 2025	Apr 8, 2025	CL	Rejected	No actions available
Fred	Apr 21, 2025	Apr 23, 2025	Medical	Approved	No actions available
Ryan	Apr 7, 2025	Apr 8, 2025	Medical	Rejected	No actions available

Figure 5.3



Sadhna Construction

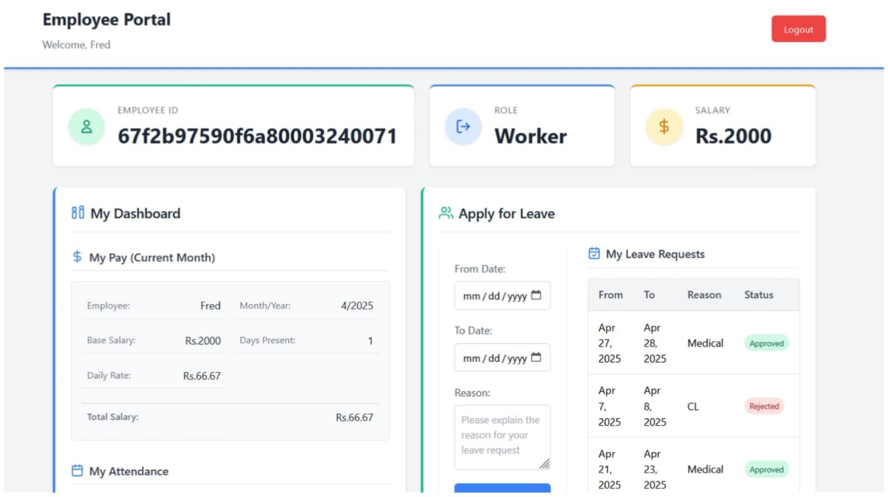
Sign in to your account

Employee Username: Fred

Password: ••••••

© 2025 Sadhna Construction. All rights reserved.

Figure 5.4

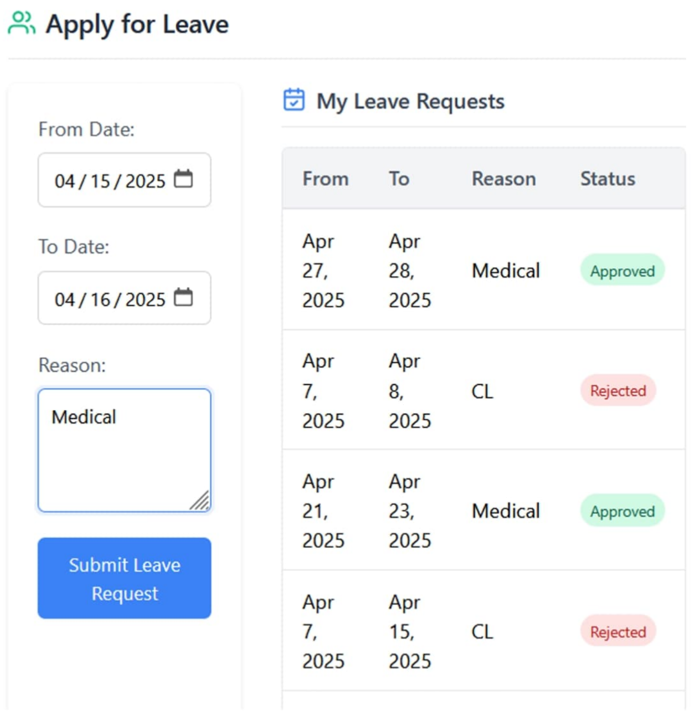


EMPLOYEE ID	67f2b97590f6a80003240071
ROLE	Worker
SALARY	Rs.2000

Employee:	Fred	Month/Year:	4/2025
Base Salary:	Rs.2000	Days Present:	1
Daily Rate:	Rs.66.67		
Total Salary:			Rs.66.67

From	To	Reason	Status
Apr 27, 2025	Apr 28, 2025	Medical	Approved
Apr 7, 2025	Apr 8, 2025	CL	Rejected
Apr 21, 2025	Apr 23, 2025	Medical	Approved

Figure 5.5



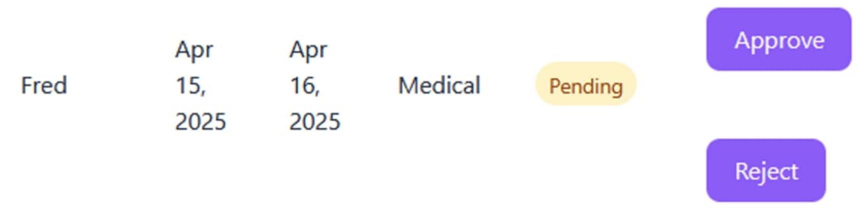
From Date: 04 / 15 / 2025

To Date: 04 / 16 / 2025

Reason: Medical

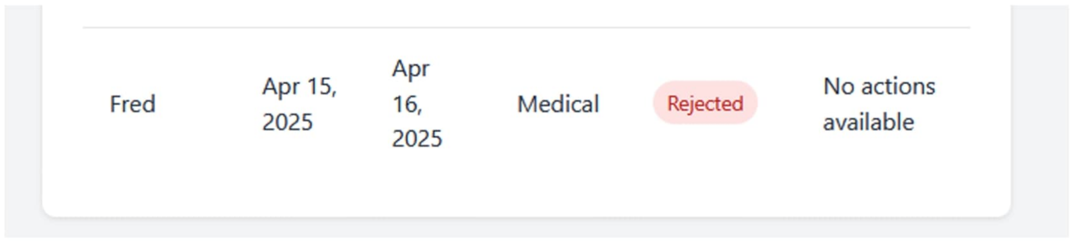
From	To	Reason	Status
Apr 27, 2025	Apr 28, 2025	Medical	Approved
Apr 7, 2025	Apr 8, 2025	CL	Rejected
Apr 21, 2025	Apr 23, 2025	Medical	Approved
Apr 7, 2025	Apr 15, 2025	CL	Rejected

Figure 5.6



Fred	Apr 15, 2025	Apr 16, 2025	Medical	Pending	Approve Reject

Figure 5.7



Fred	Apr 15, 2025	Apr 16, 2025	Medical	Rejected	No actions available

Figure 5.8

30% EXTRA CONTRIBUTION

A. Role-Based Dynamic Dashboards

Beyond simple route access, dashboards dynamically **change content and controls based on user role**:

- **Admin:** Full site overview, user management, task assignment, reports.
- **Manager:** Site-specific task monitoring and team overview.
- **Worker:** Only personal assigned tasks and progress tracking.

Improves usability by **reducing clutter** and enforcing security at the UI level.

B. Activity Audit Trail

Each critical action in the system (login, task assignment, task update, site status change) is **logged with timestamp and user role**. Admins can generate reports showing historical activity for auditing and accountability. This ensures **compliance and traceability**, important in large-scale construction projects.

C. Notification System

Implemented a **role-aware notification system**:

- Managers get notified when tasks are completed or delayed.
- Workers receive notifications for new assignments.
- Admins are alerted to critical site issues.

Notifications are **real-time** and can be viewed in the dashboard or sent via email/SMS for urgent events.