PROJECT REPORT

HR Management System

Anish Thakkar
CE082
22CEUON124

Vaibhav Mackwana
CE024
22CEUBG138

Contents

List of Figures

1 Introduction

1.1 Overview

This document details the requirements and specifications of the HR Management System, including its functional and non-functional requirements, constraints, and system design. The system aims to enhance productivity by automating repetitive HR tasks and providing a centralized platform for managing employee-related data.

1.2 Project Objectives

The primary objectives of the HR Management System are:

- Centralized employee information with secure access
- Automated attendance tracking via Excel
- Optimized payroll using stored procedures
- Role-specific functionalities
- Intuitive interfaces
- Use scalable and performance centric architecture

1.3 System Overview

The HR Management System is built using modern technologies including **.NET 8 Web API** for the backend, **Angular** for the frontend, and **Microsoft SQL Server** for database management. The system architecture follows **industry best practices** with a focus on performance optimization.

We have followed a layered architecture with organized folders, including **Controllers**, **DTOs**, **Repositories**, and **Models**. The concept of **Seeders** is used for faster development. Separate APIs for **pagination-based fetching** and a filter middleware for **authentication** are implemented.

Stored procedures and the **ExcelDataReader** NuGet package were major innovations in this project. The stored procedures significantly reduced processing time compared to traditional ORM-based approaches, while ExcelDataReader enabled the

bulk addition of employee attendance efficiently.

The system is deployed on **Azure**, with both backend, fronend and database deployed on azure for enhanced accessibility and reliability.

1.4 Definitions, Acronyms, and Abbreviations

- **HR:** Human Resources
- **enployee:** Our product name
- **.NET Core:** A cross-platform framework for building modern, cloud-based applications
- **CRUD:** Create, Read, Update, Delete

1.5 Link

- Hosted Website: `hrms-client-eyejb3adcth0f7ae.centralindia-01.azurewebsites.net`
- Hosted APIs: `hrms-app-service-g0frh0djd3bug6gh.centralindia-01.azurewebsites.net/swagger`
- Github Repo: `https://github.com/Vaibhav31mak/HRMS`

2 Requirement Specification and Analysis

2.1 Functional Requirements

Based on our research these are the feature requirements a company's HR may have:

2.1.1 Employee Management

- Complete CRUD (Create, Read, Update, Delete) operations for employee data
- Role-based access controls

2.1.2 Payroll Management

- Bulk salary updates for efficiency
- Automated payroll calculations considering weekdays, overtime, and days off
- Integration with attendance data for accurate salary processing

2.1.3 Attendance Management

- Excel-based attendance data upload mechanism
- Real-time integration with payroll systems

2.1.4 Role and Authorization Management

- Role assignment to each used by Superadmin
- Default system roles (Superadmin and Admin)
- Fine-grained control over CRUD operations through claims
- Role-based dashboard views and accessibility

2.1.5 Department Management

- CRUD of Departments

2.2 Non-Functional Requirements

2.2.1 Usage

The system is designed to ensure continuous availability, providing reliable access at all times.

2.2.2 Performance

In a busy office environment, the system offers significantly faster performance compared to traditional pen-and-paper methods. Manual tasks like attendance management are automated for completion within seconds, ensuring efficient operations across all functionalities.

2.2.3 Scalability

The project is built with scalability in mind, capable of accommodating growth and supporting an iterative development approach for future enhancements.

3 Design

3.1 ER Diagram

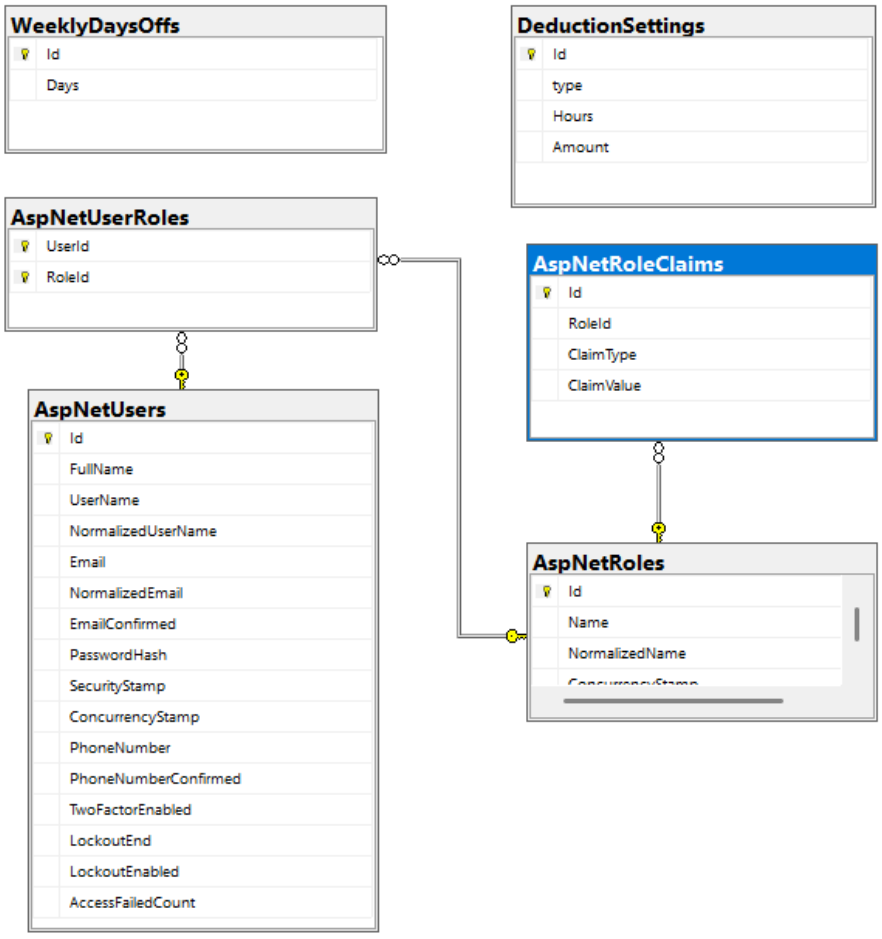


Figure 3.1: Entity Relationship Diagram - 1

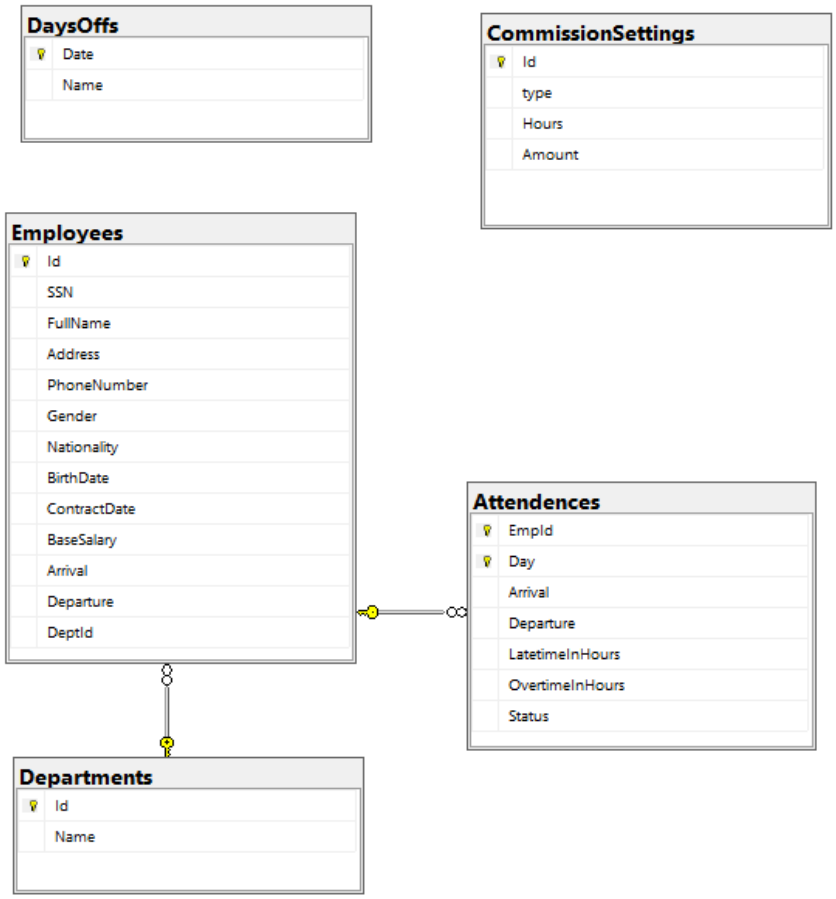


Figure 3.2: Entity Relationship Diagram - 2

3.2 Class Diagram

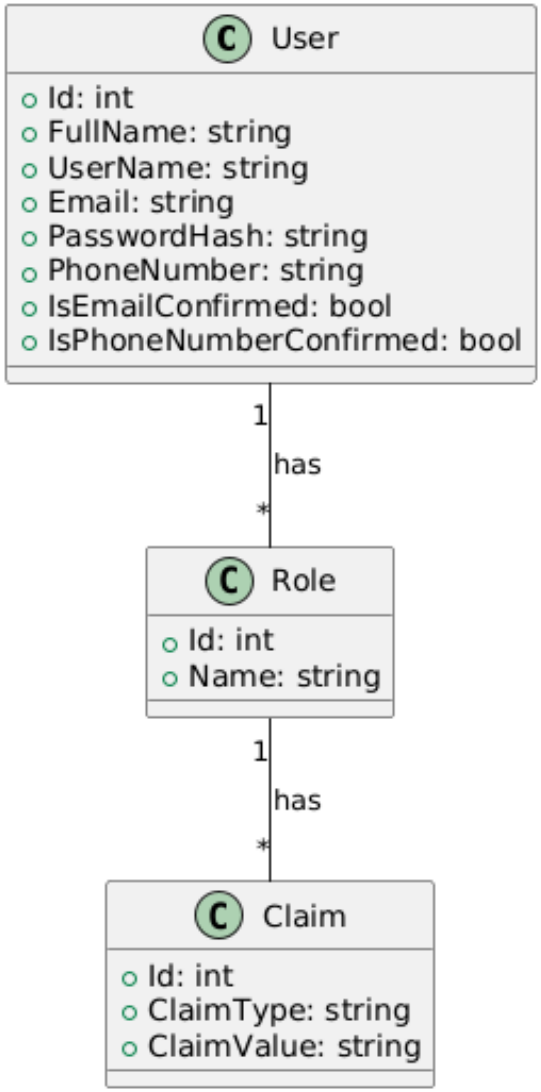


Figure 3.3: Class Diagram - 1

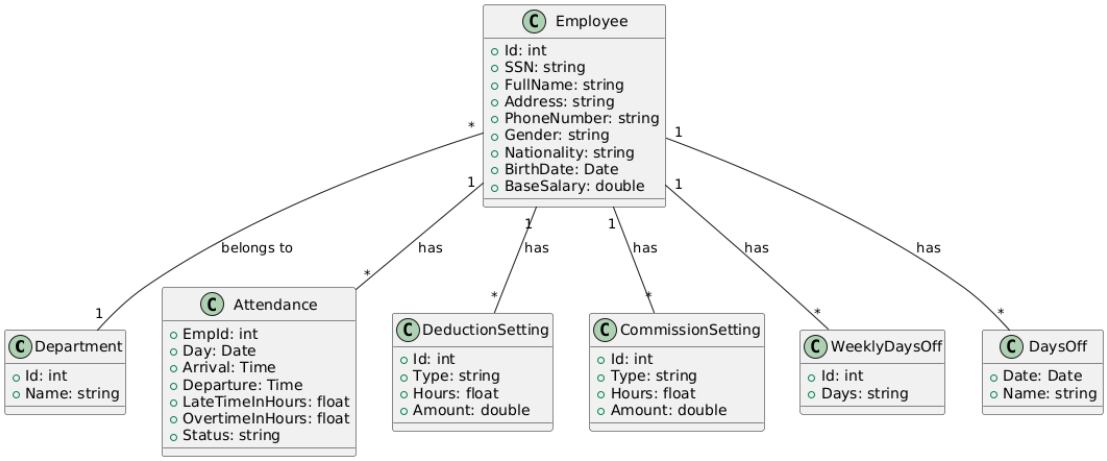


Figure 3.4: Class Diagram - 2

3.2.1 Attendance

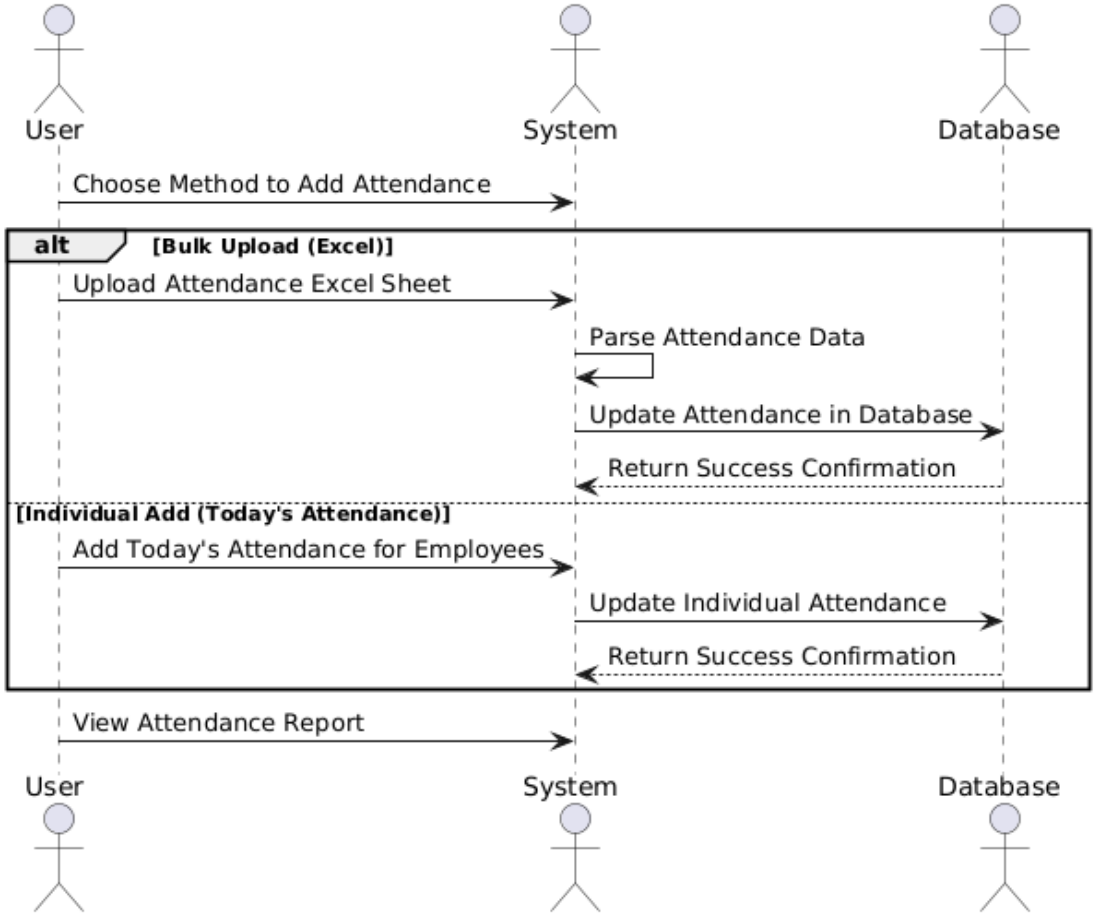


Figure 3.5: Sequence Diagram - Attendance

3.2.2 Payroll

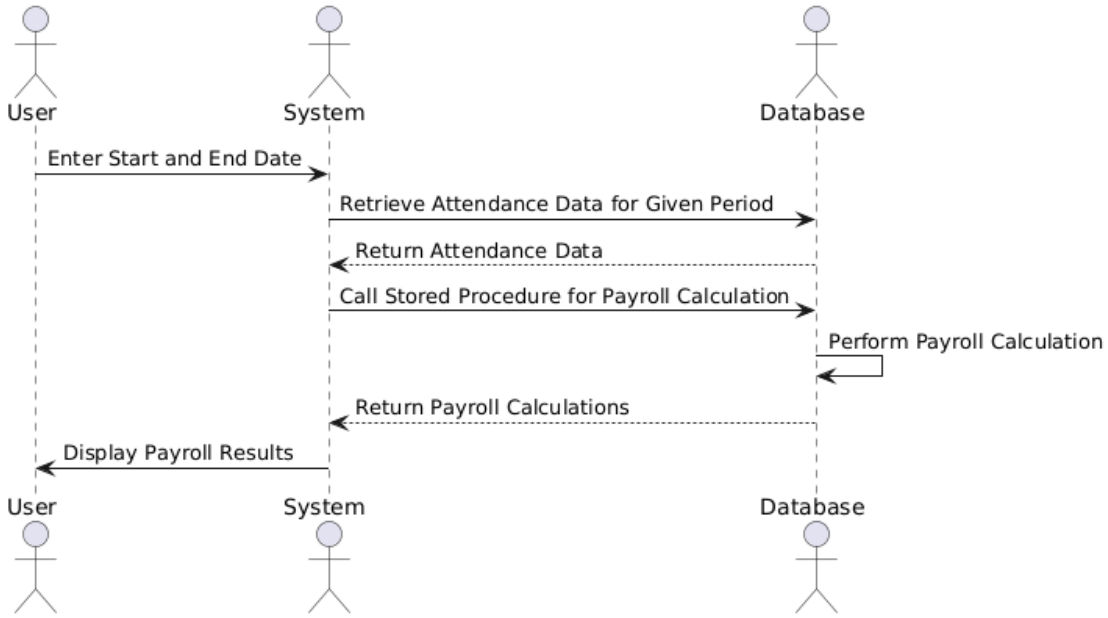


Figure 3.6: Sequence Diagram - Payroll

4 Implementation details

4.1 Technologies Used

4.1.1 Frontend Technologies

- Angular
- Special feature of type checking came very useful
- deployed on Azure

4.1.2 Backend Technologies

- .NET Framework
- WEB API architecture
- Deployment using Azure

4.1.3 Database

- SQL server management Studio
- Special feature of database diagram creation is used
- Deployed using Azure

4.1.4 Development Environment

- Visual Studio 2022
- Visual Studio Code
- SQL Server Management tool
- Azure site

4.2 Screenshots

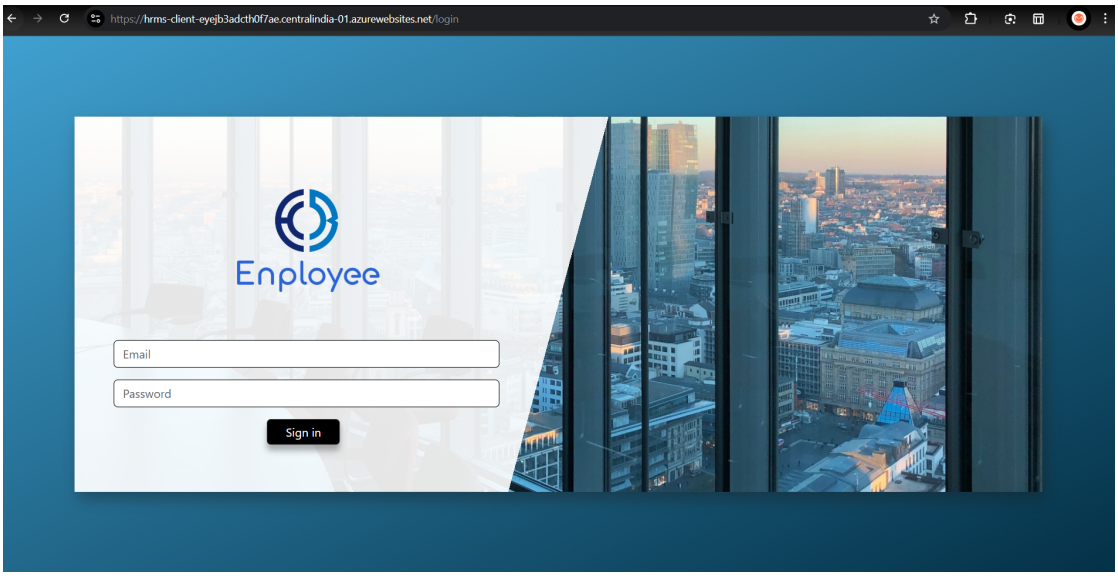


Figure 4.1: login page

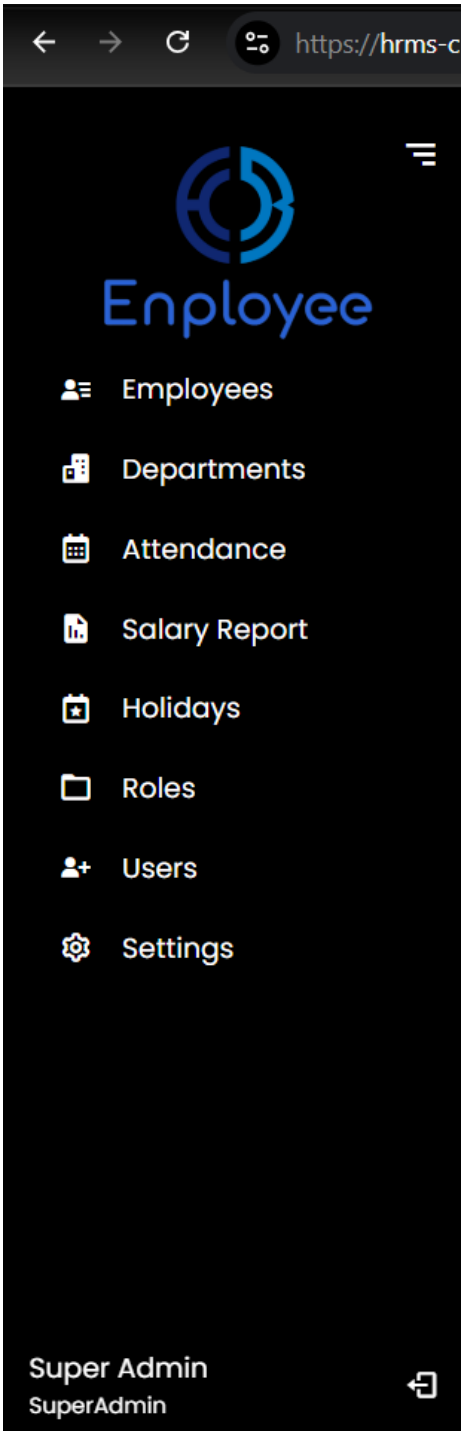


Figure 4.2: navigation bar

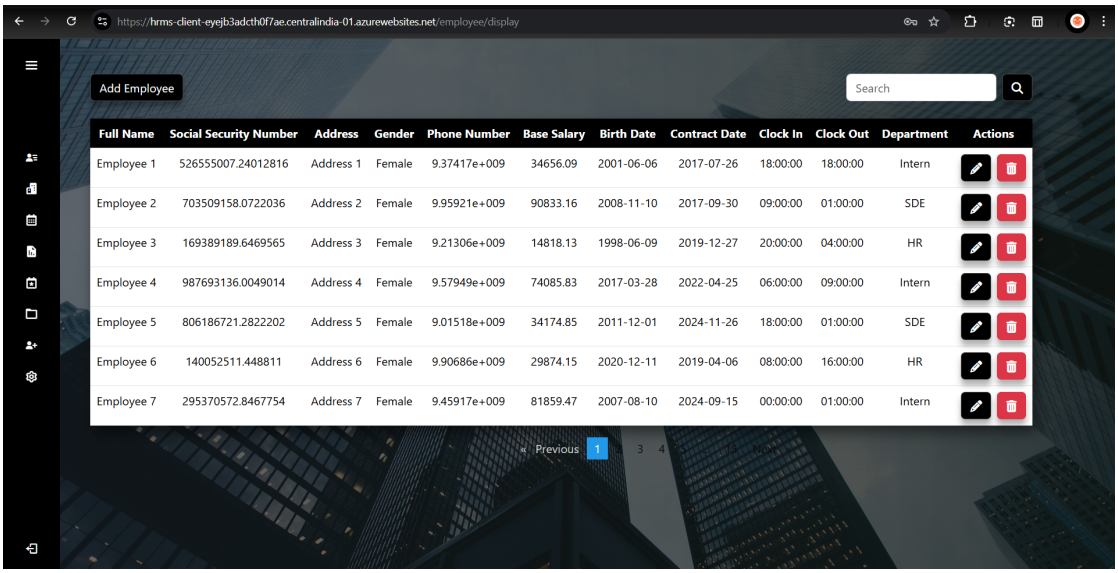


Figure 4.3: employee list

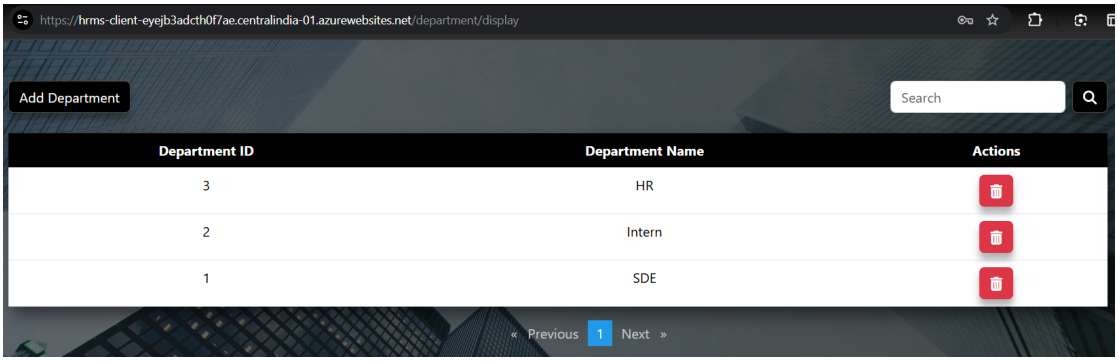


Figure 4.4: departments

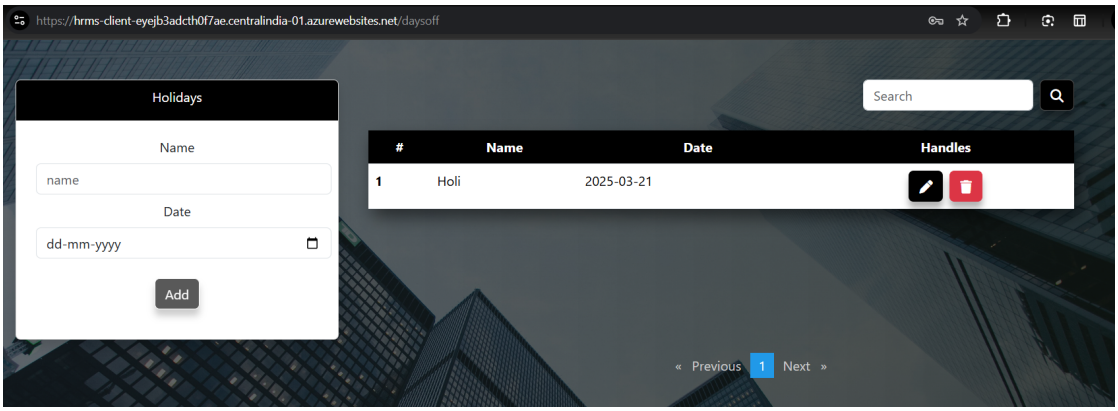


Figure 4.5: holidays

	A	B	C	D	E
1	**EmpId**	**Day**	**Arrival**	**Departure**	**Status**
2	1	2025-02-01	09:00:00	17:00:00	Present
3	1	2025-02-02			Absent
4	1	2025-02-03	09:00:00	17:00:00	Present
5	1	2025-02-04			Absent
6	1	2025-02-05	09:00:00	17:00:00	Present
7	1	2025-02-06			Absent
8	1	2025-02-07	09:00:00	17:00:00	Present
9	2	2025-02-01	09:00:00	17:00:00	Present
10	2	2025-02-02			Absent
11	2	2025-02-03	09:00:00	17:00:00	Present
12	2	2025-02-04			Absent
13	2	2025-02-05	09:00:00	17:00:00	Present
14	2	2025-02-06			Absent
15	2	2025-02-07	09:00:00	17:00:00	Present
16	3	2025-02-01	09:00:00	17:00:00	Present
17	3	2025-02-02			Absent
18	3	2025-02-03	09:00:00	17:00:00	Present
19	3	2025-02-04			Absent
20	3	2025-02-05	09:00:00	17:00:00	Present
21	3	2025-02-06			Absent
22	3	2025-02-07	09:00:00	17:00:00	Present

Figure 4.6: attendance sheet format

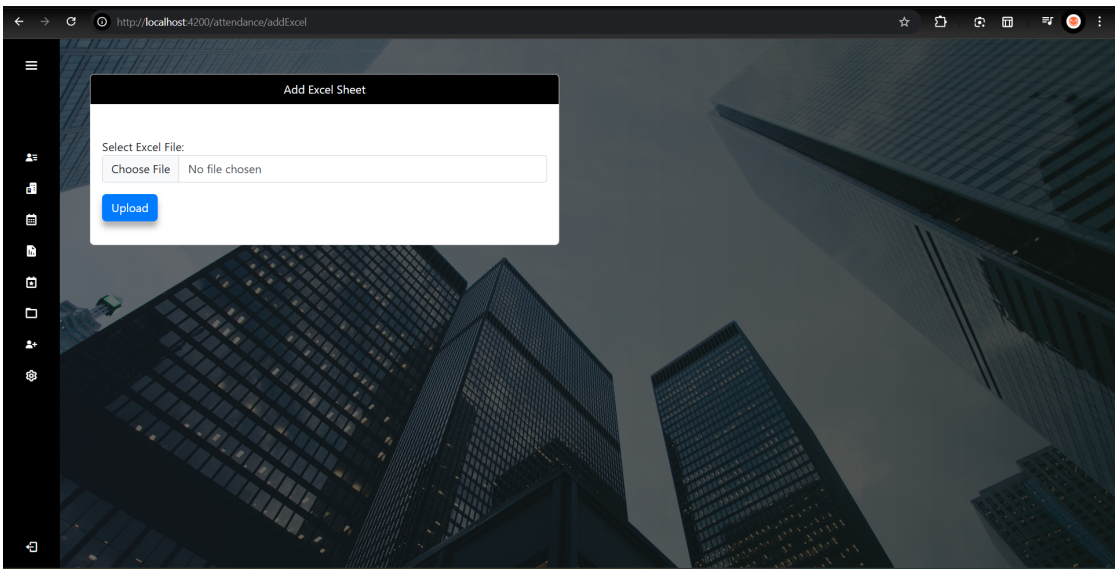


Figure 4.7: bulk attendance addition form

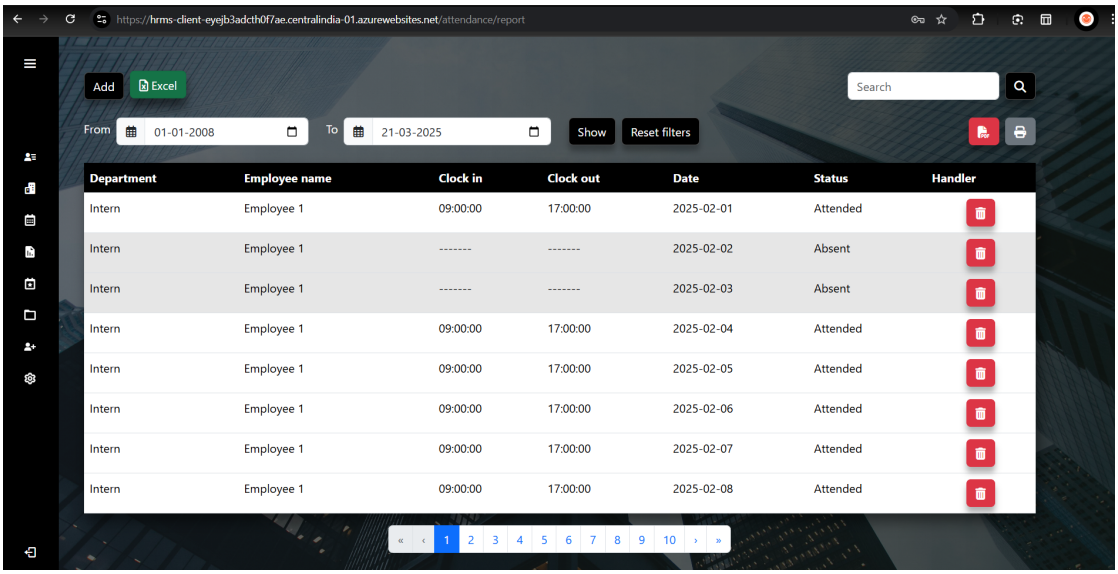


Figure 4.8: attendance list

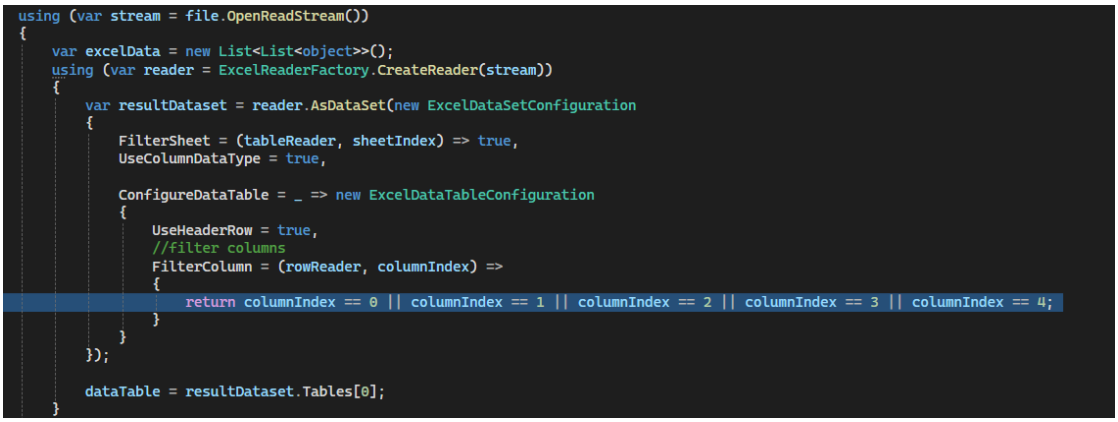


Figure 4.9: code for excel attendance addition

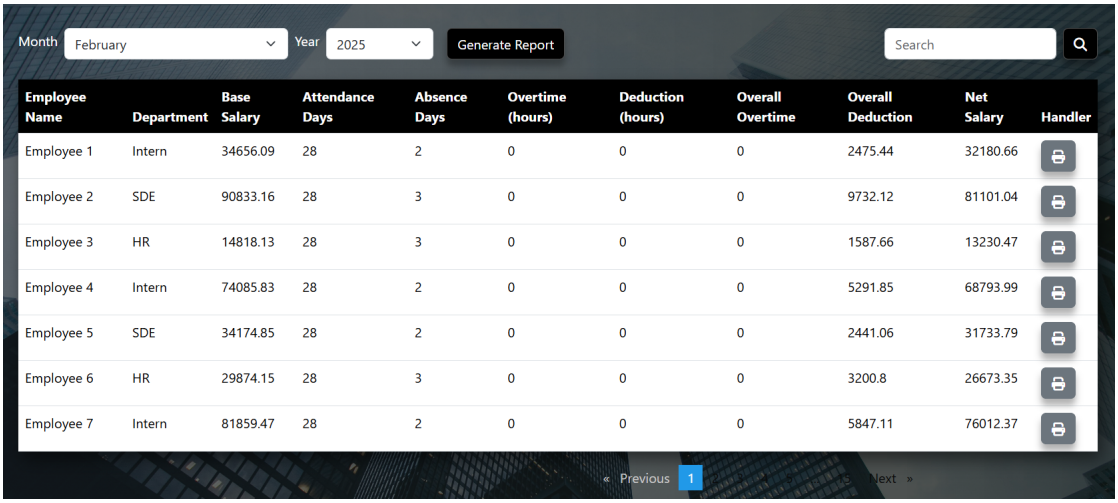


Figure 4.10: salary report

```
SELECT
    EmpId,
    FullName,
    DepartmentName,
    BaseSalary,
    AttendanceDays,
    AbsentDays,
    TotalOvertimeHours,
    TotalLateHours,
    ShiftDuration,

    -- Pre-calculate the hourly rate and reuse it
    ROUND(BaseSalary / (AttendanceDays * ShiftDuration), 2) AS HourlyRate,

    -- Salary Calculation
    ROUND(BaseSalary+((BaseSalary / (AttendanceDays * ShiftDuration)) * TotalOvertimeHours)-((
            (BaseSalary / (AttendanceDays * ShiftDuration)) * TotalLateHours
        )
        + (
            (BaseSalary / AttendanceDays) * AbsentDays
        )),2)
    AS CalculatedSalary,

    -- Additional Pay (Overtime)
    ROUND(
        (BaseSalary / (AttendanceDays * ShiftDuration)) * TotalOvertimeHours,
        2
    ) AS AdditionalPay,

    -- Deduction Pay (Lateness & Absence)
    ROUND(
        (
            (BaseSalary / (AttendanceDays * ShiftDuration)) * TotalLateHours
        )
        + (
            (BaseSalary / AttendanceDays) * AbsentDays
        ),
        2
    ) AS DeductionPay

FROM
    AttendanceData
    order by EmpId;
```

Figure 4.11: stored procedure code

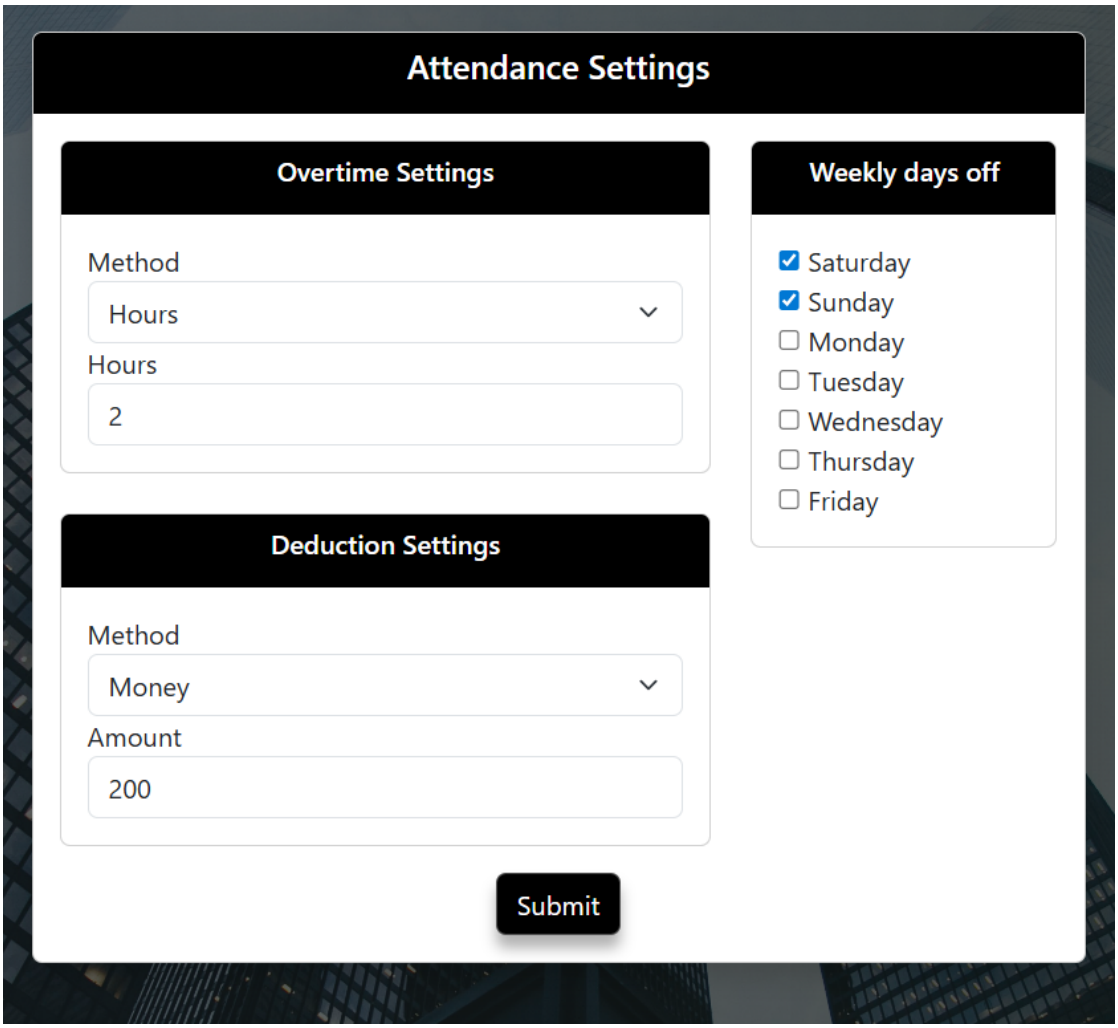


Figure 4.12: settings

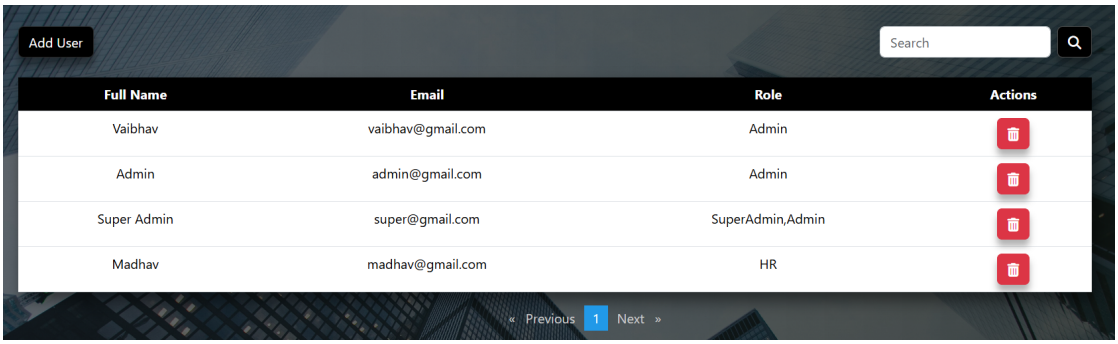


Figure 4.13: users

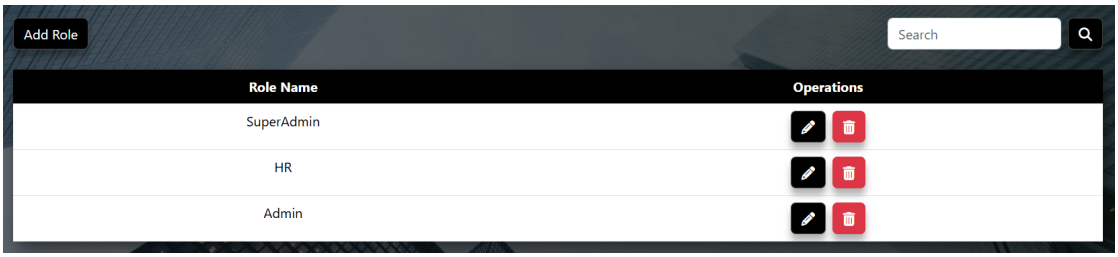


Figure 4.14: roles

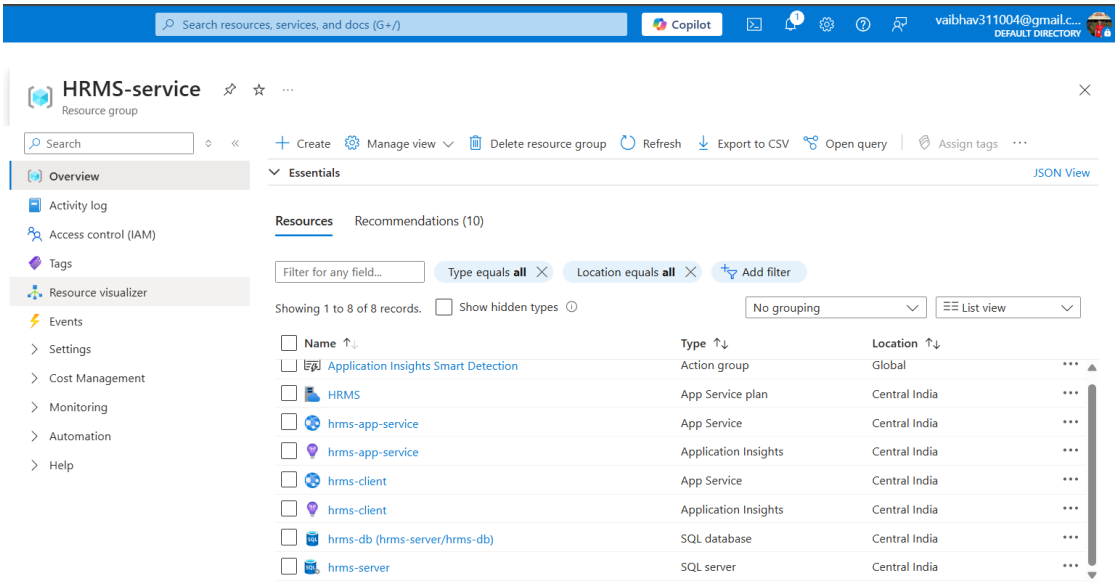


Figure 4.15: azure online hosting

5 Testing

5.1 Overview

The testing phase encompassed unit and stress testing, focusing on key metrics such as database query performance and maintainability scores from Visual Studio 2022. Integration testing was performed manually, ensuring seamless functionality by verifying the application after each module integration.

5.2 API Testing

The system's APIs were thoroughly tested using **Swagger**, an API documentation and testing tool. Swagger provided an interactive interface to validate API endpoints, ensuring that each endpoint responded correctly to different request types, including GET, POST, PUT, and DELETE. It also facilitated the verification of request payloads, response codes, and data consistency.

5.3 Frontend Testing

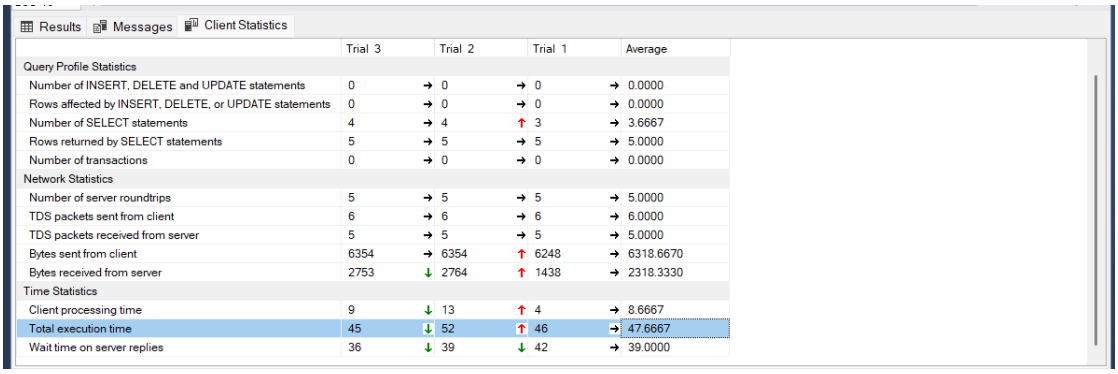
Once the APIs were successfully tested, the backend was integrated with the **Angular** frontend. The frontend underwent rigorous manual testing to assess the accuracy of data representation, navigation flows, and overall user experience. **Lighthouse** was used to generate performance and seo score for the inital page.

5.4 Database Testing

To validate database performance and scalability, stress testing was performed using synthetic data. ChatGPT was utilized to generate large datasets resembling real-world scenarios. Queries were executed to measure database response times and identify potential bottlenecks.

A significant observation was made during the payroll calculation testing. Initially, payroll was processed using a backend loop-based approach, which took approximately three minutes to handle 10,000 employee records. After optimization using SQL stored procedures, the same payroll calculations were completed in less than 10 seconds. This remarkable performance improvement was a key success indicator of the testing phase.

Figure 5.1 displays the execution of a stored procedure for a test case.



	Trial 3	Trial 2	Trial 1	Average
Query Profile Statistics				
Number of INSERT, DELETE and UPDATE statements	0	0	0	0.0000
Rows affected by INSERT, DELETE, or UPDATE statements	0	0	0	0.0000
Number of SELECT statements	4	4	3	3.6667
Rows returned by SELECT statements	5	5	5	5.0000
Number of transactions	0	0	0	0.0000
Network Statistics				
Number of server roundtrips	5	5	5	5.0000
TDS packets sent from client	6	6	6	6.0000
TDS packets received from server	5	5	5	5.0000
Bytes sent from client	6354	6354	6248	6318.6670
Bytes received from server	2753	2764	1438	2318.3330
Time Statistics				
Client processing time	9	13	4	8.6667
Total execution time	45	52	46	47.6667
Wait time on server replies	36	39	42	39.0000

Figure 5.1: database payroll client analysis using ssms query

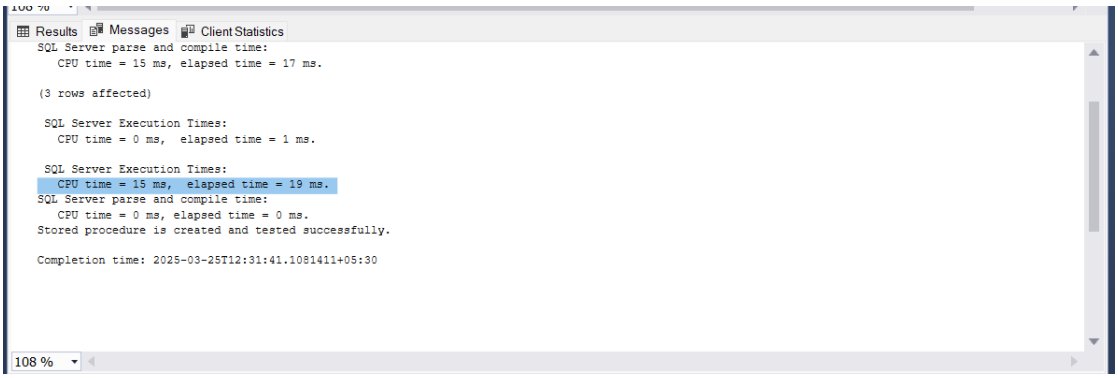


Figure 5.2: database payroll execution time - 19ms

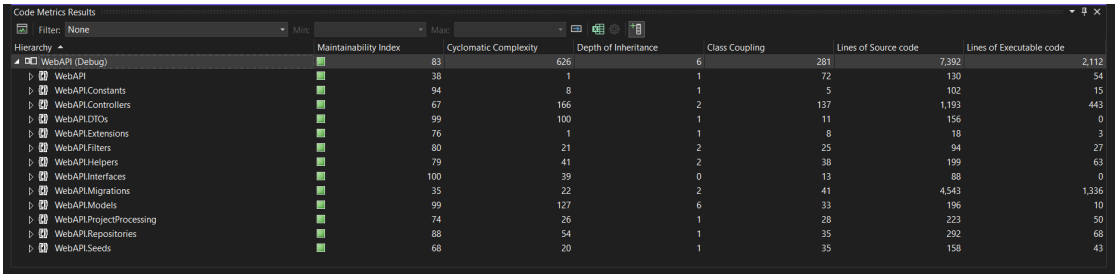


Figure 5.3: Backend Code Maintainability Score

- **Maintainability Index**: Higher is better (ranges from 0 to 100). - 83
- **Depth of Inheritance**: Measures class hierarchy depth. - 6

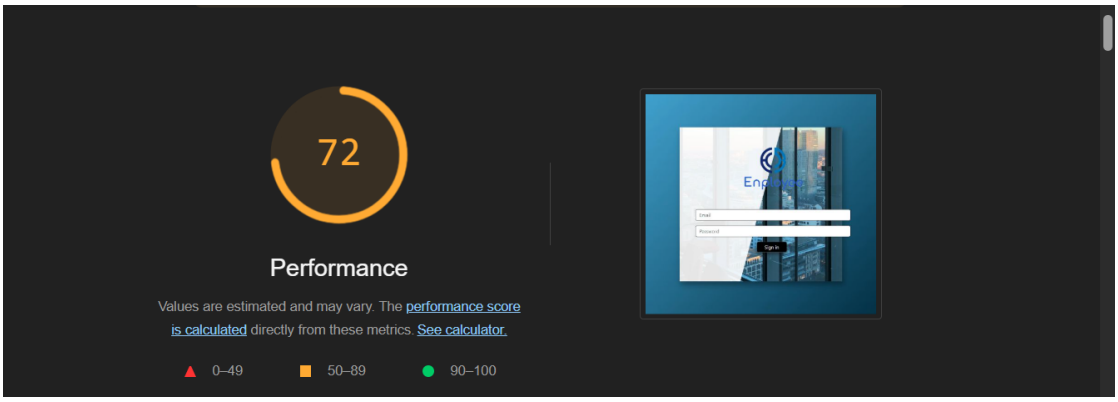


Figure 5.4: Frontend Lighthouse Performance score for inital page

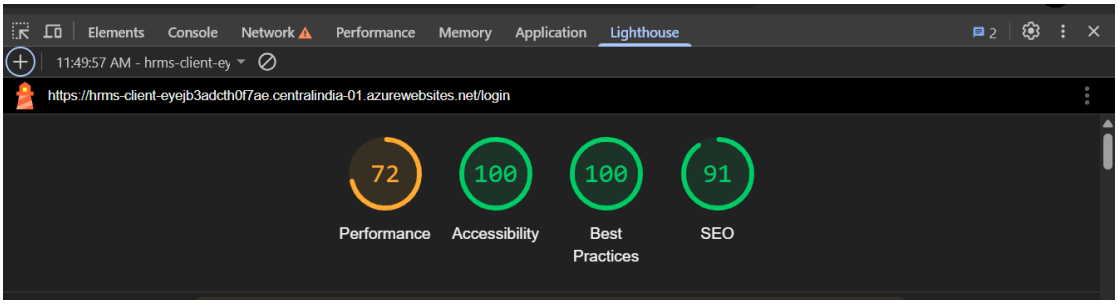


Figure 5.5: Frontend Lighthouse overall score for inital page

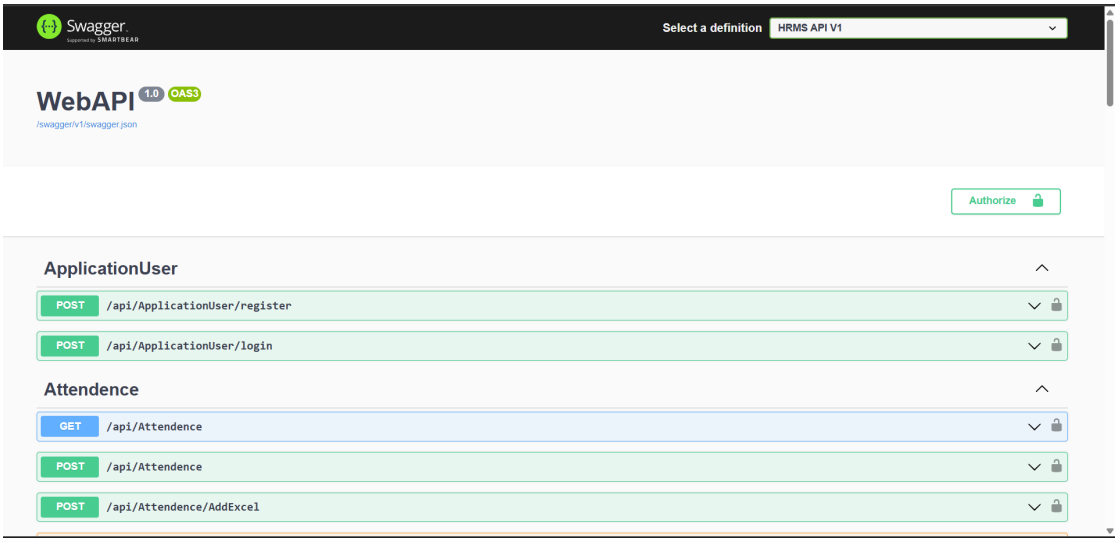


Figure 5.6: swagger-1

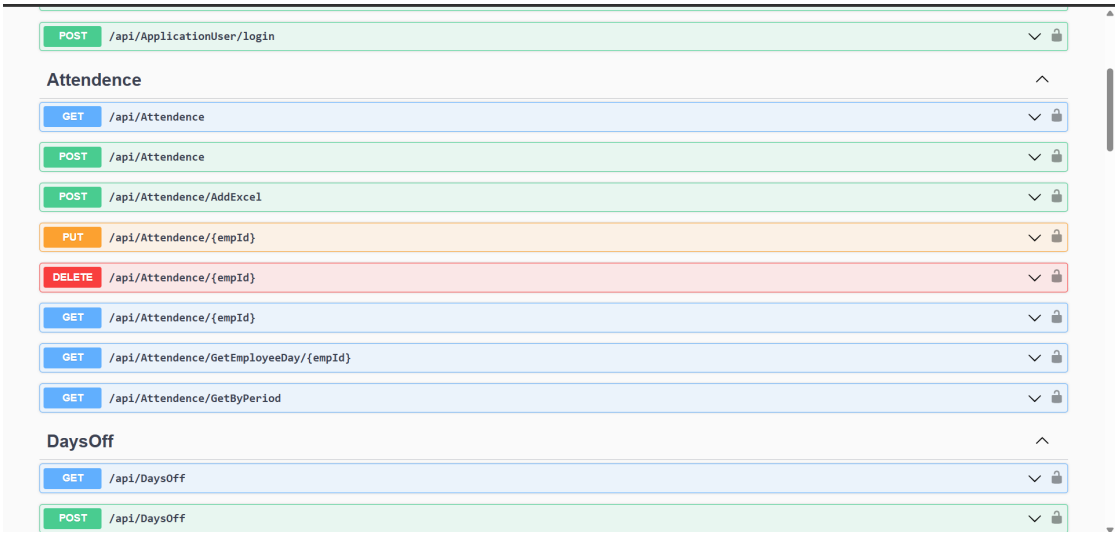


Figure 5.7: Swagger-2

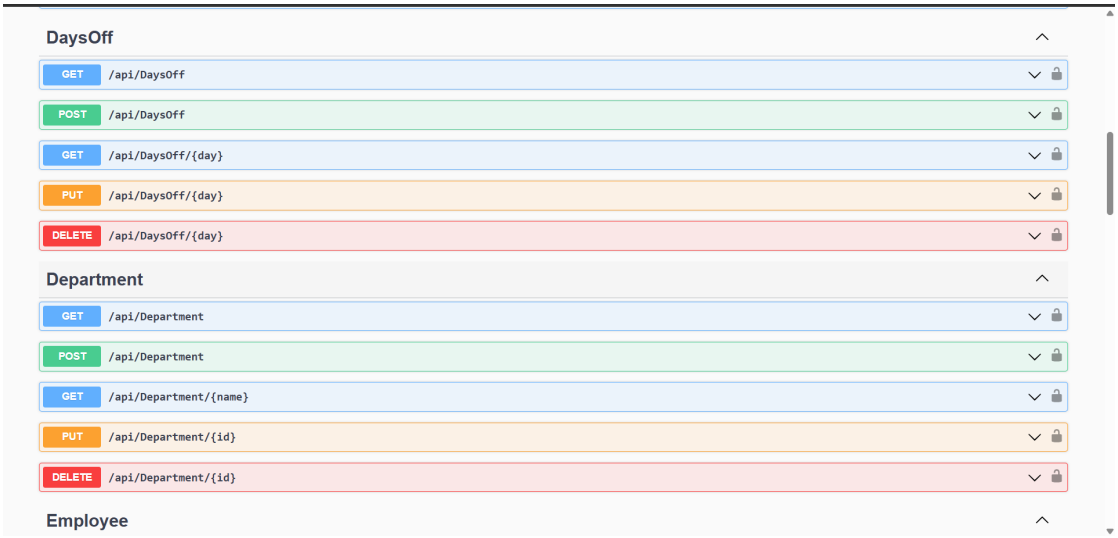


Figure 5.8: Swagger-3

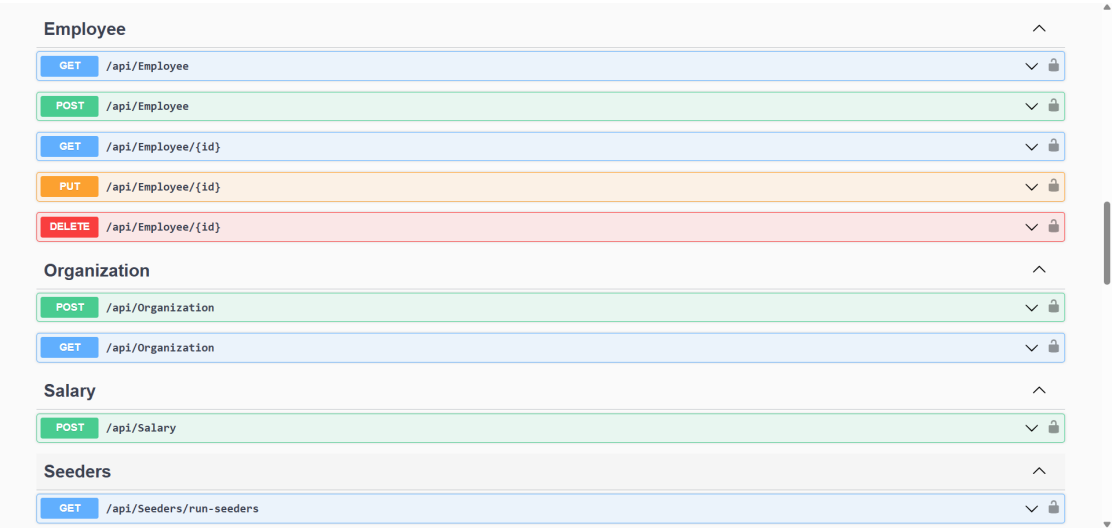


Figure 5.9: Swagger-4

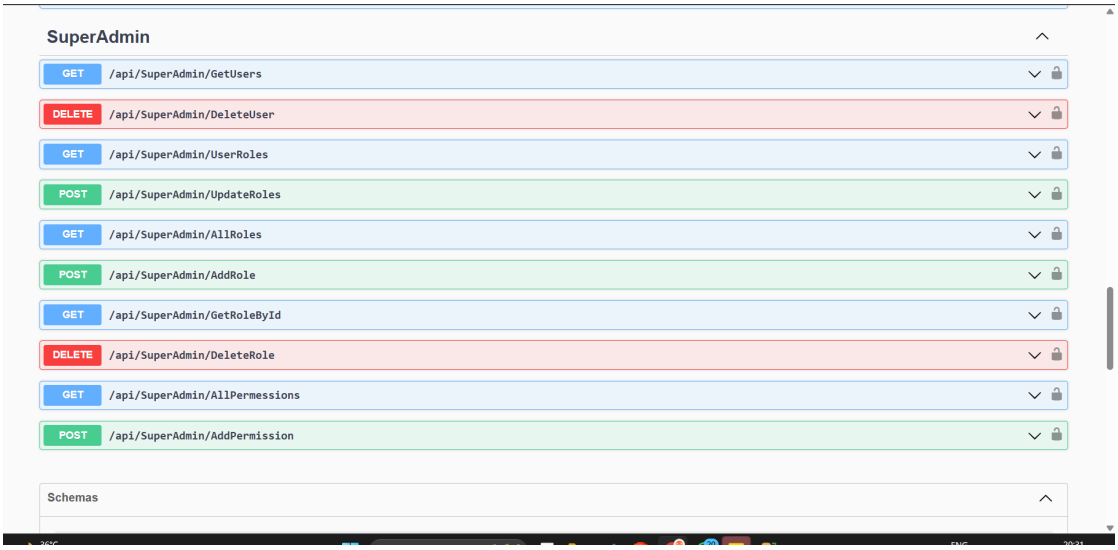


Figure 5.10: Swagger-5

Figure 5.11: Swagger-6

6 Conclusion and Future Extensions

6.1 Conclusion

The HR Management System has successfully achieved its objectives of automating and streamlining key HR processes. From attendance tracking using Excel sheet uploads to optimized payroll processing through SQL stored procedures, the system has demonstrated significant efficiency improvements. Role-based access management ensures secure and restricted access to sensitive information, while the intuitive user interface simplifies HR operations.

Furthermore, the reduction in payroll processing time from three minutes to under ten seconds for large datasets highlights the effectiveness of the system's design and implementation. The successful integration of .NET Core, Angular, and SQL Server has provided a scalable and robust solution suitable for organizations of all sizes.

6.2 Future Extensions

6.2.1 AI Chatbot for Employee Report Analysis

An AI chatbot can serve as a virtual HR assistant, capable of answering employee-related queries by analyzing reports and contract data. The chatbot would be particularly beneficial for HR managers, as it could provide instant insights without the need to manually search through records.

Implementation Approach

Gemini API with Vector Database. Convert your document to vector embeddings and provide them as input to the Gemini API. Then you can generate accurate answers to you questions, with proper reference to the original phrase in the employee policy.

6.2.2 Further Optimize the payroll calculations to include concurrency

To further optimize payroll processing, concurrency can be introduced to handle large datasets efficiently. By dividing the payroll calculation task into smaller, independent operations and executing them in parallel, significant time reductions can be achieved. Implementing multi-threading or parallel processing within SQL Server or using application-level concurrency mechanisms will ensure faster and more efficient payroll generation, especially for organizations with thousands of employees.