Project Report

Dharmsinh Desai University, Nadiad

Faculty of Technology

Department of Computer Engineering

Btech. CE Semester – VI

Subject: System Design Practices

Project Title: HR Management System

By

Thakkar Anish , CE088 , ID No. 22CEUON124

Vaibhav Makwana , CE024 , ID No. 22CEUBG138

Guided by: Prof. Jigar M. Pandya, Associate Professor

Dharmsinh Desai University

Faculty of Technology, College Road, Nadiad – 387001, Gujarat

CERTIFICATE

This is to certify that the term work carried out in the subject of **System Design Practices** and submitted is the bonafide work of **Mr. Anish Thakkar,** Roll No.: **CE082,** and **Mr. Vaibhav Makwana,** Roll No.: **CE024** Identity No.: **22CEUON124, 22CEUBG138** of B.Tech. **Semester VI** in the branch of **Computer Engineering** during the academic year 2024-2025.

Prof. Jigar M. Pandya	Dr. C. K. Bhensdadia
Associate Professor	Head, CE Department

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

- FR-EM1: Complete CRUD operations for employee data
- FR-EM2: Role-based access controls

2.1.2 Payroll Management

- FR-PM1: Bulk salary updates for efficiency
- FR-PM2: Automated payroll calculations considering weekdays, overtime, and days off
- FR-PM3: Integration with attendance data for salary processing

2.1.3 Attendance Management

- FR-AM1: Excel-based attendance data upload mechanism
- FR-AM2: Real-time integration with payroll systems

2.1.4 Role and Authorization Management

- FR-RM1: Role assignment by Superadmin
- FR-RM2: Default system roles (Superadmin and Admin)
- FR-RM3: Fine-grained control through claims
- FR-RM4: Role-based dashboard views

2.1.5 Department Management

- FR-DM1: CRUD operations for Departments

2.2 Non-Functional Requirements

2.2.1 Usage

NFR-USE1: Continuous system availability

2.2.2 Performance

NFR-PERF1: Automated attendance management in seconds
NFR-PERF2: Payroll processing under 10s for 10k records

2.2.3 Scalability

NFR-SCL1: Scalable architecture for future enhancements

3 Design

3.1 ER Diagram

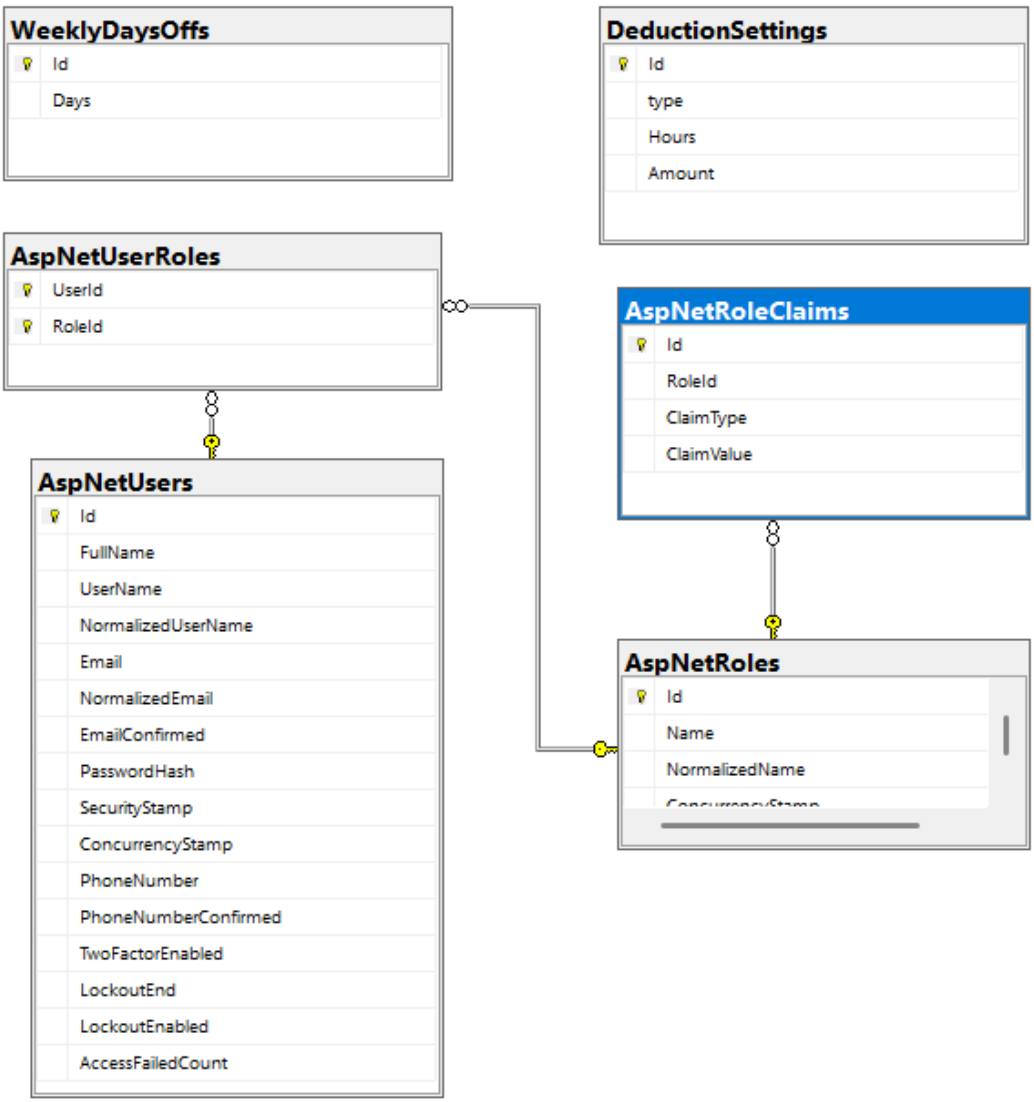


Figure 3.1: Entity Relationship Diagram - 1

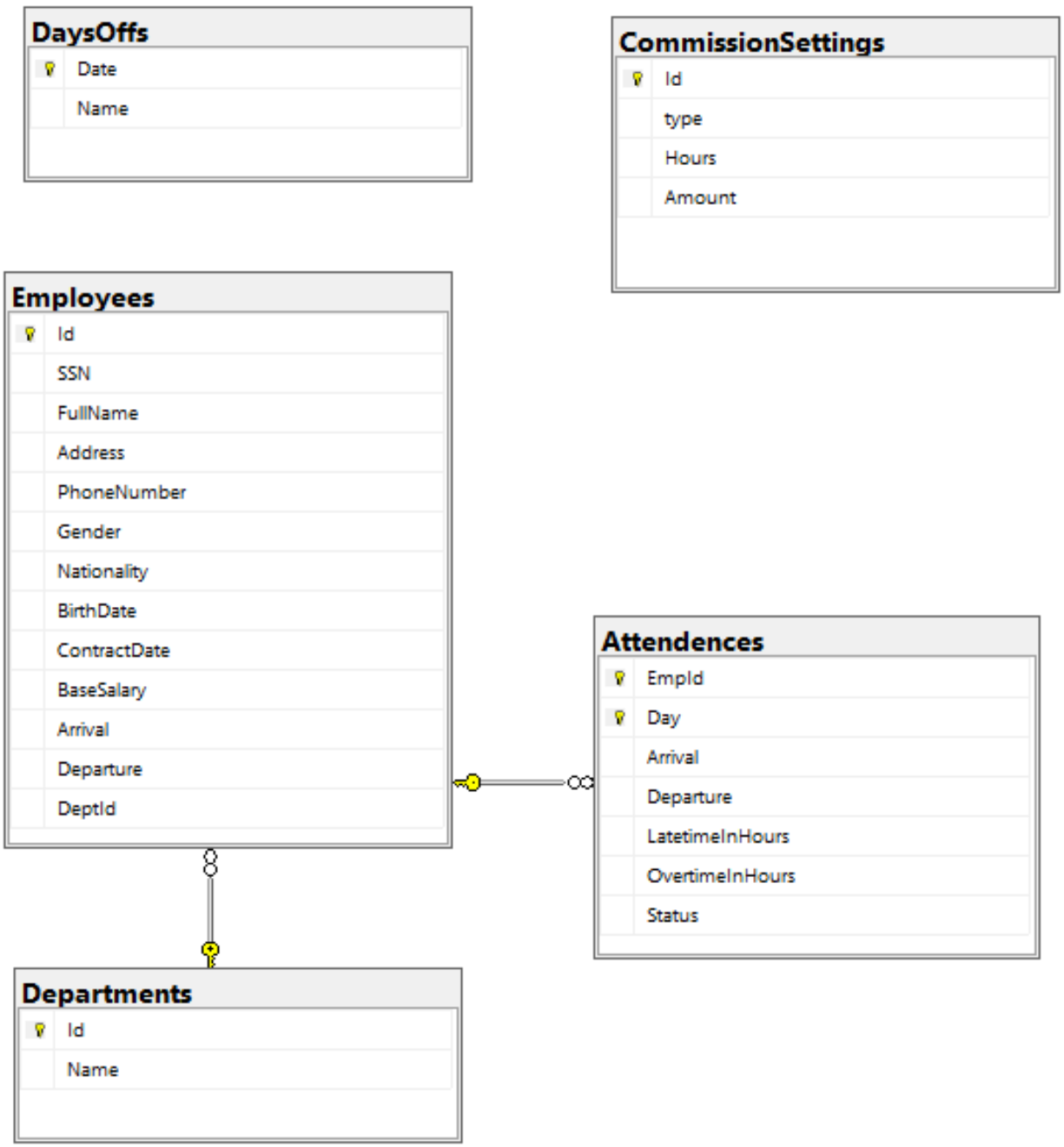


Figure 3.2: Entity Relationship Diagram - 2

3.2 Class Diagram

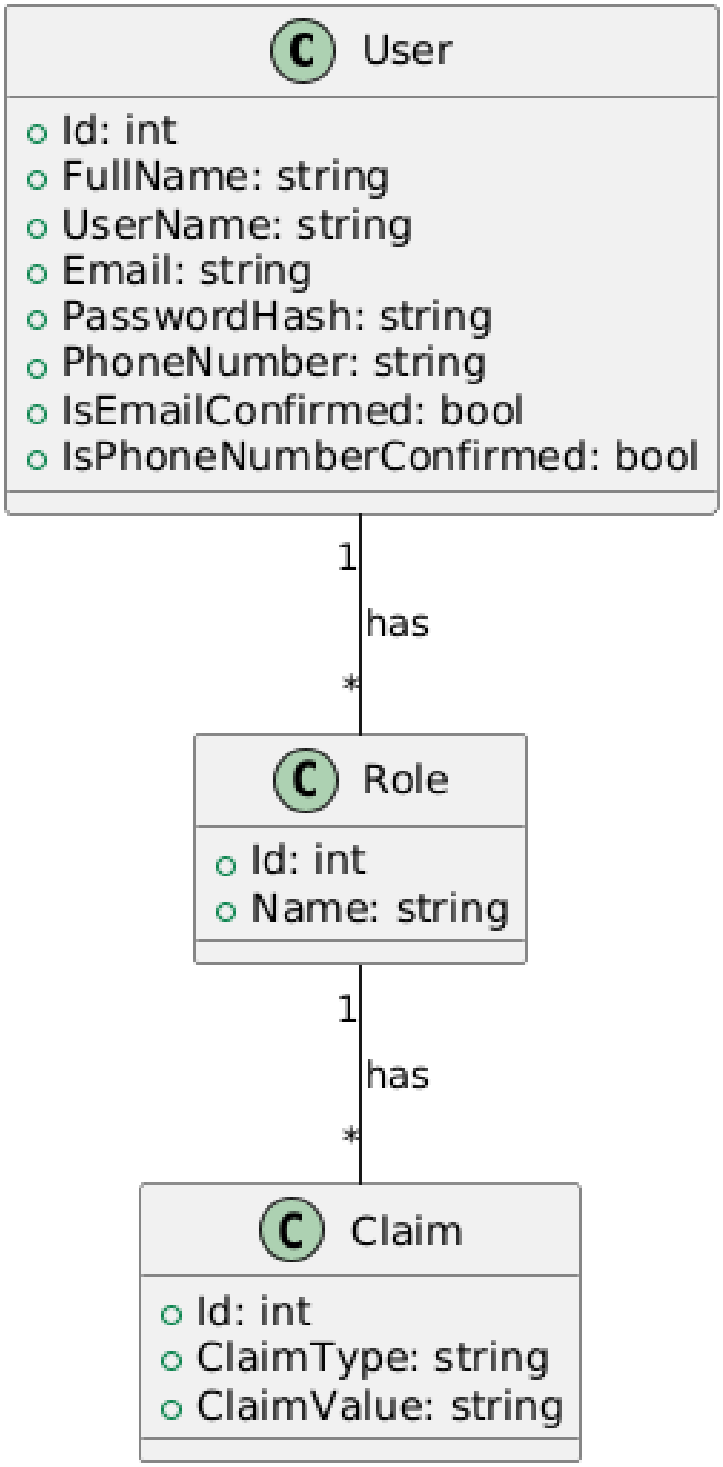


Figure 3.3: Class Diagram - 1

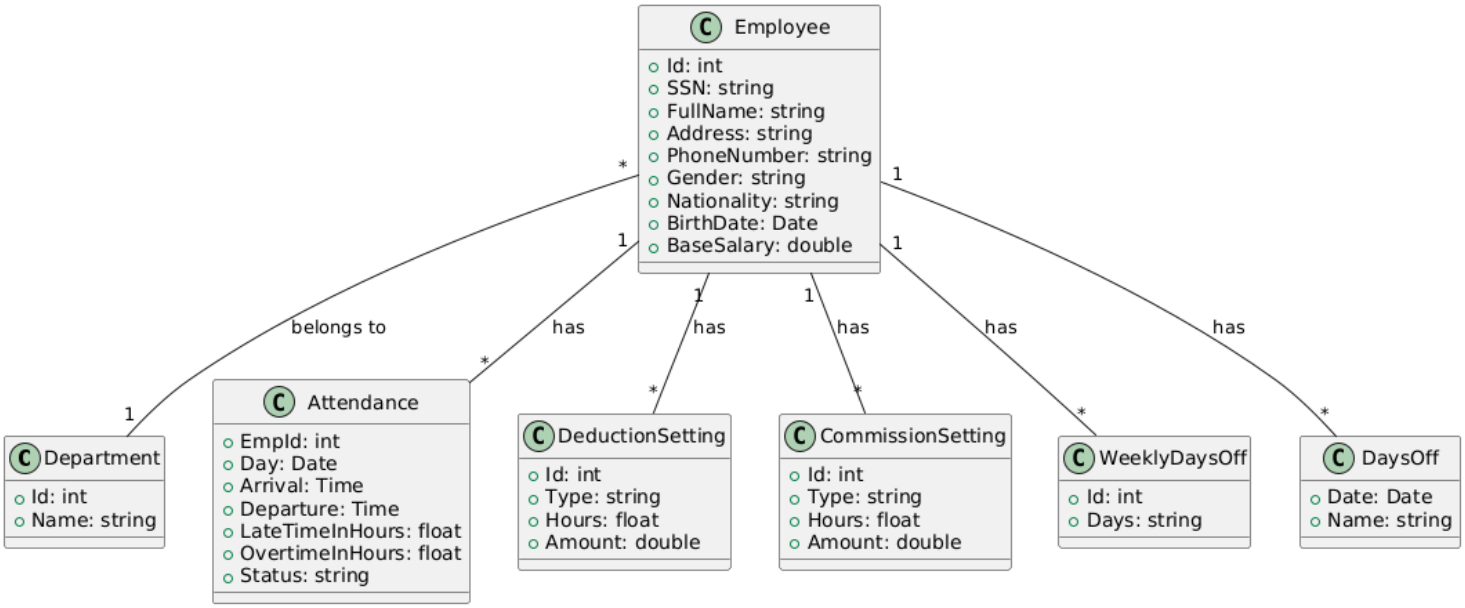


Figure 3.4: Class Diagram - 2

3.2.1 Attendance

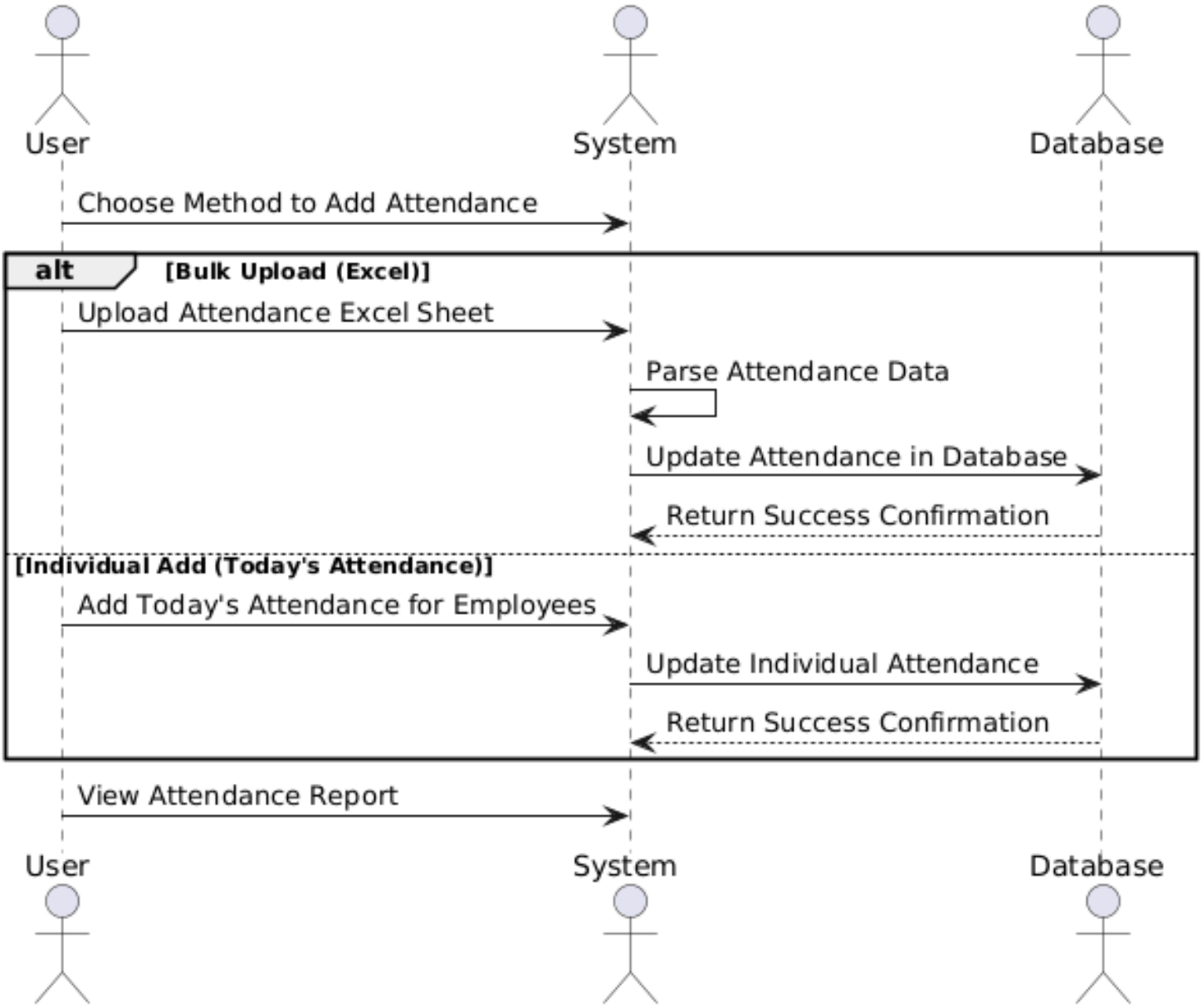


Figure 3.5: Sequence Diagram - Attendance

3.2.2 Payroll

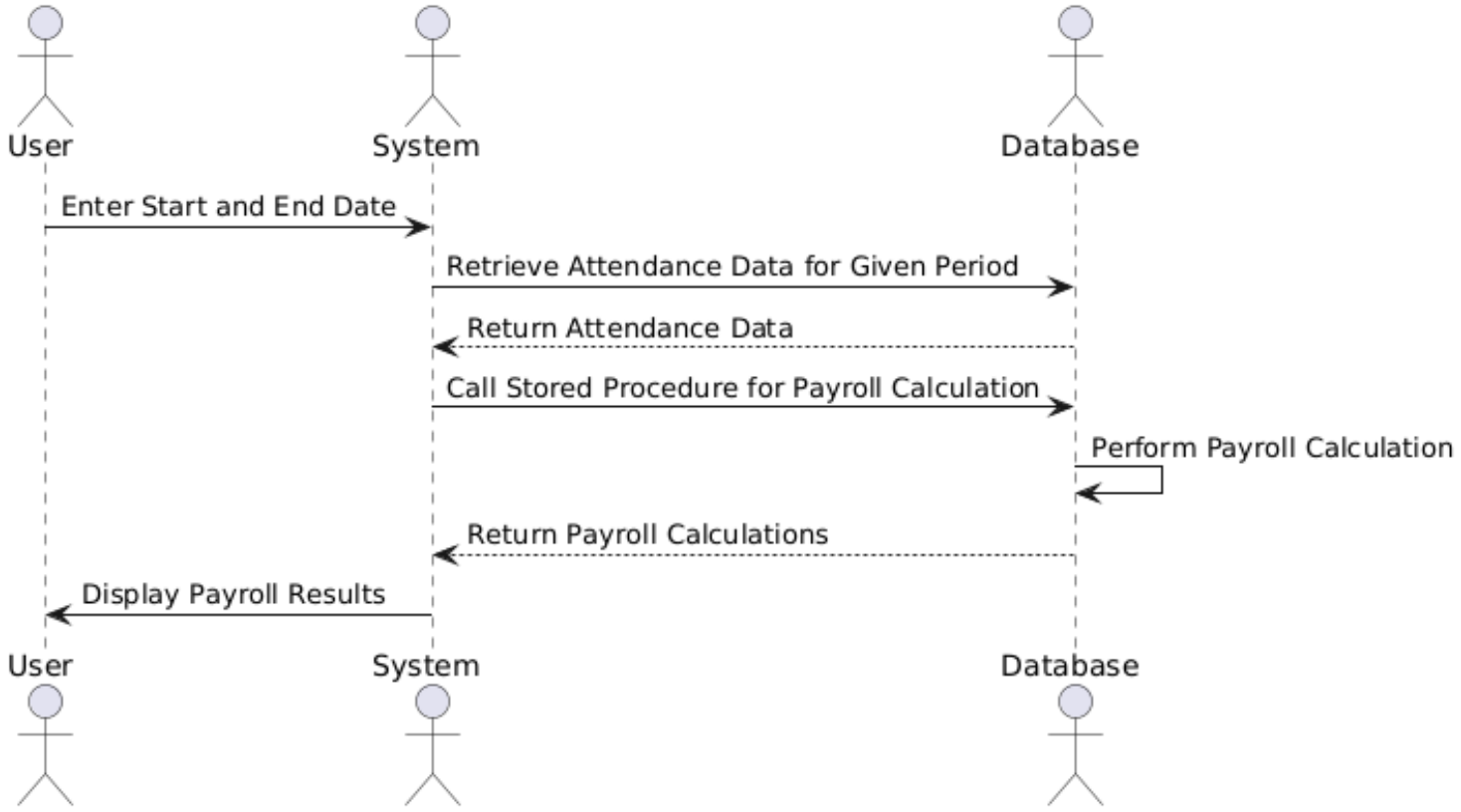


Figure 3.6: Sequence Diagram - Payroll

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

Software Version Description Document

- Deployment Architecture: Azure App Service + SQL Server
- Installation Steps: Docker image hrms:1.0
- Configuration: appsettings.Production.json
- Dependencies: .NET 8 Runtime, Node 18.x

4.2 Screenshots

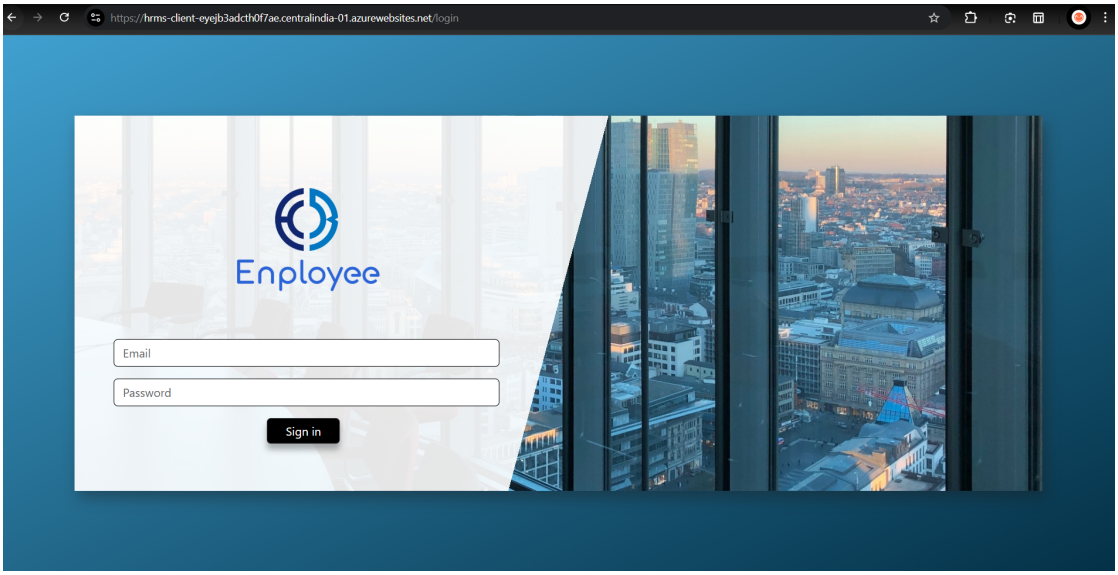


Figure 4.1: Login page implementing Role-based access controls (Req. 2.1.4)

Simple login page made using Identity Nuget package and Angular UI.

```
SecurityKey securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:Secret"]));
SigningCredentials signingCredentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

// Create token
JwtSecurityToken token = new JwtSecurityToken(
    issuer: configuration["JWT:ValidIssuer"], // URL of the Web API
    audience: configuration["JWT:ValidAudiance"], // URL of the consumer (Angular)
    claims: claims,
    expires: DateTime.Now.AddDays(1),
    signingCredentials: signingCredentials
);

return Ok(new
{
    token = new JwtSecurityTokenHandler().WriteToken(token),
    expiration = token.ValidTo,
    fullName = user.FullName,
    role = roles[0] // return the first role of the list
});
```

Figure 4.2: JWT Token stored for 1 day for storing logged in sessions(Req. 2.1.4)

A JWT security token is generated using a secret key called "JWT:Secret", which is stored in the configuration settings. The token, along with the logged-in user's ID, is stored in the database and is valid for one day.

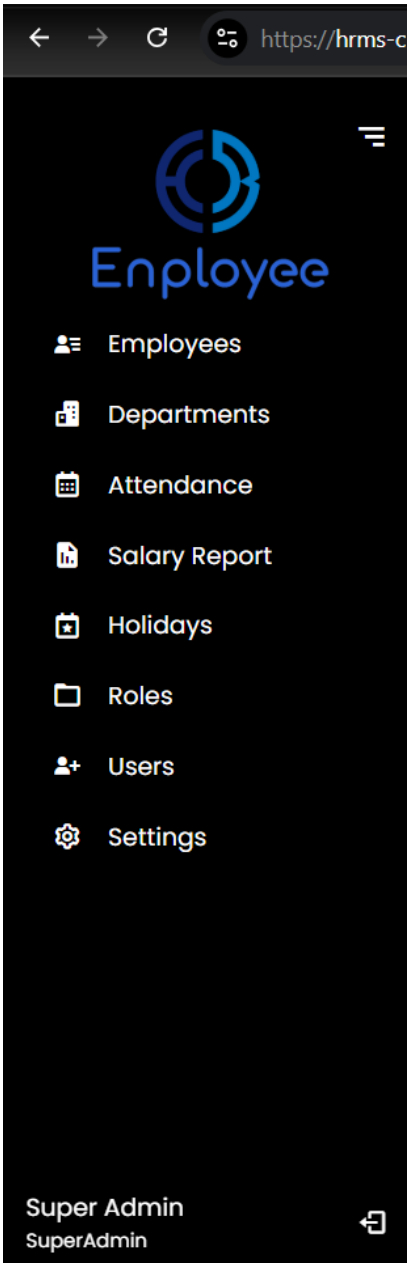


Figure 4.3: Navigation bar with Role-based views (Req. **??**)

A hamburger menu is to navigate to each of the provided features. Currently features are - Employees' CRUD, Departments' CRUD, Attendance management, Salary Report generation; Holidays, Roles and Users Curd, and Settings.

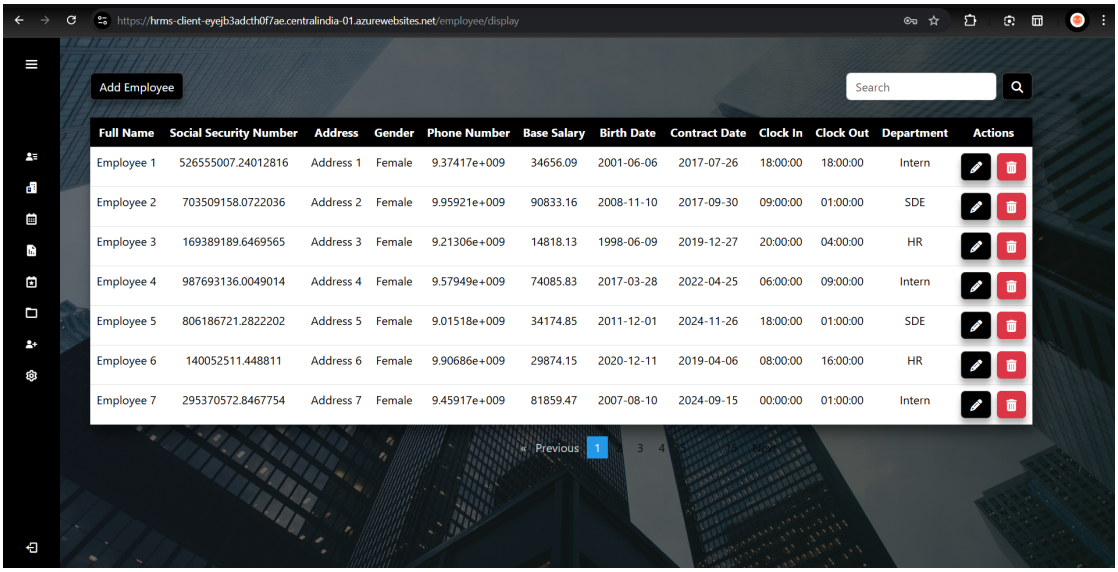


Figure 4.4: Employee list supporting CRUD operations (Req. 2.1.1)

Employess can be added, removed or its information can be edited. By Default both roles - super admin and admin can do this task.

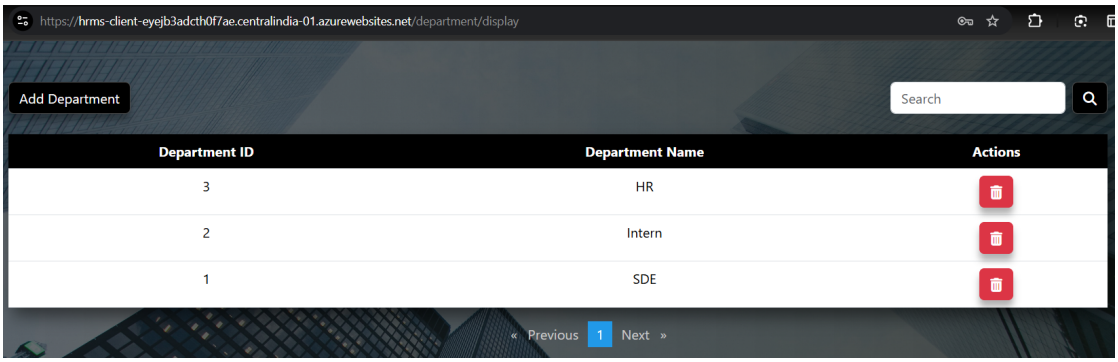


Figure 4.5: Department management interface (Req. 2.1.5)

Basic Department addition and deletion can be done.

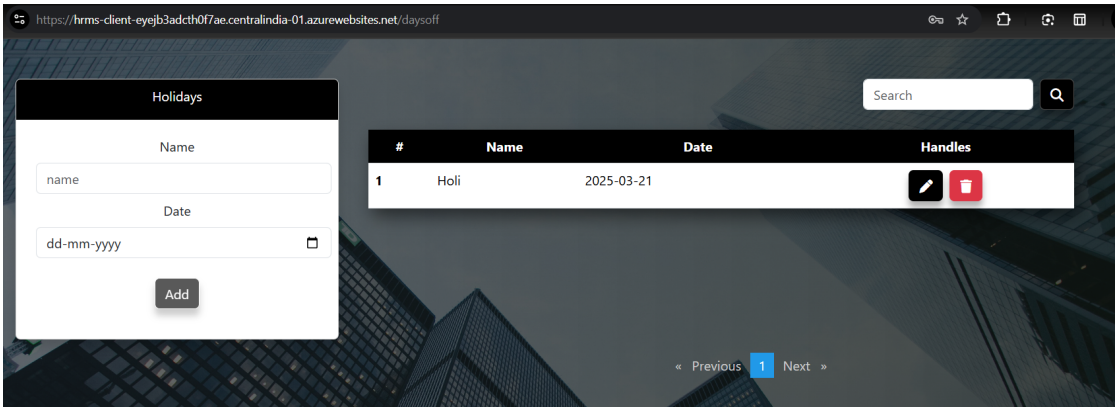


Figure 4.6: Holiday management affecting payroll calculations (Req. 2.1.2)

Basic holidays crud. These holidays are useful in attendance management, if a person is not present on these holidays, he will not be marked absent.

	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 4.7: Excel template for attendance upload (Req. 2.1.3)

Format of Excel sheet used for bulk addition of attendance. Generally this is auto-generated by a biometric attendance machine.

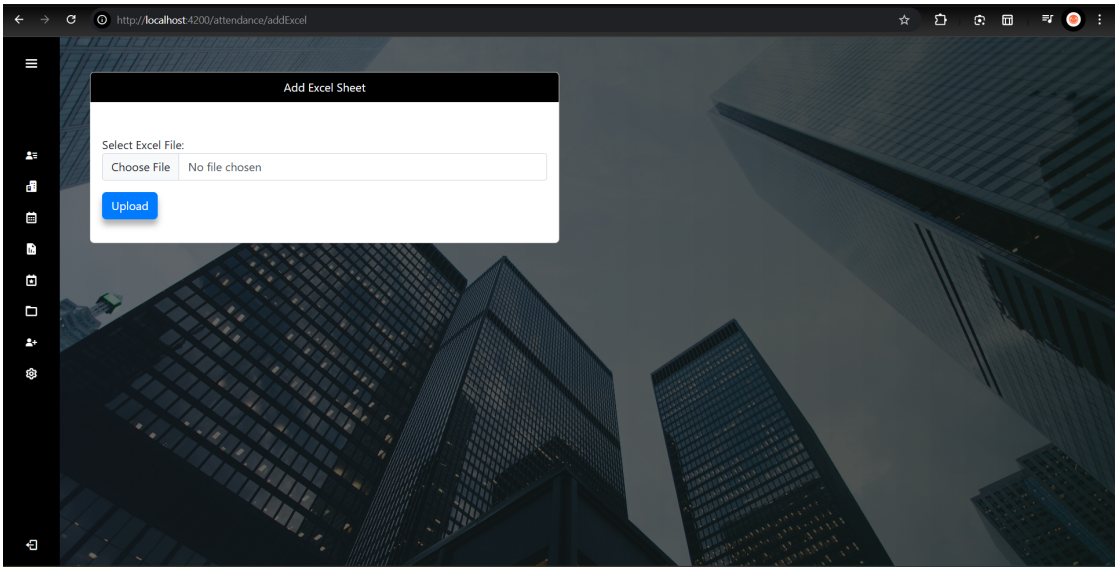


Figure 4.8: Bulk attendance upload form (Req. 2.1.3, 2.2.2)

Simple HTML form with submit having header of form-data. Excel sheet is submitted in form data format and is thus sent piece by piece.

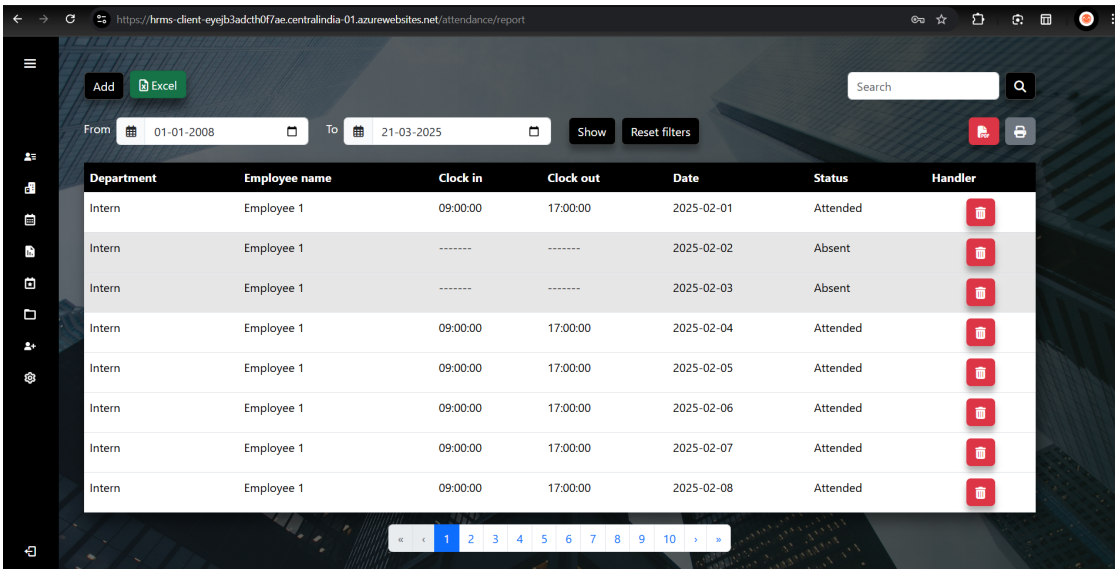


Figure 4.9: Attendance records integrated with payroll (Req. 2.1.3, 2.1.2)

Attendance record viewing list. This list is updated based on the available data in database. If you wish wish to update any single attendance, you can update that in excel sheet re-upload that in our system. For adding single attendance record - current day attendance can be added using the button on top left corner.

```
using (var stream = file.OpenReadStream())
{
    var excelData = new List<List<object>>();
    using (var reader = ExcelReaderFactory.CreateReader(stream))
    {
        var resultDataset = reader.AsDataSet(new ExcelDataSetConfiguration
        {
            FilterSheet = (tableReader, sheetIndex) => true,
            UseColumnDataType = true,

            ConfigureDataTable = _ => new ExcelDataTableConfiguration
            {
                UseHeaderRow = true,
                //filter columns
                FilterColumn = (rowReader, columnIndex) =>
                {
                    return columnIndex == 0 || columnIndex == 1 || columnIndex == 2 || columnIndex == 3 || columnIndex == 4;
                }
            }
        });

        dataTable = resultDataset.Tables[0];
    }
```

Figure 4.10: Excel processing code (Req. 2.1.3, 2.2.2)

The `AddExcel` method processes an uploaded Excel file to extract and store employee attendance records. The key steps are:

1. **File Validation:** The method first checks if the uploaded file is valid.
2. **Reading Excel Data:** The file is read using `ExcelReaderFactory`, extracting only necessary columns.
3. **Fetching Database Records:** Existing employee records, holidays, and weekly off days are retrieved.
4. **Data Conversion:** Each row is converted into an `Attendence` object, handling null values and parsing dates.
5. **Holiday Check:** Attendance on holidays or off days is skipped.
6. **Overtime and Late Time Calculation:**
   - Overtime is computed based on the departure time.
   - Late time is determined using the arrival time.
7. **Updating or Adding Attendance:** If an attendance record already exists, it is updated; otherwise, a new record is added.
8. **Saving Changes:** The attendance data is saved to the database, and the method returns an HTTP 200 response.

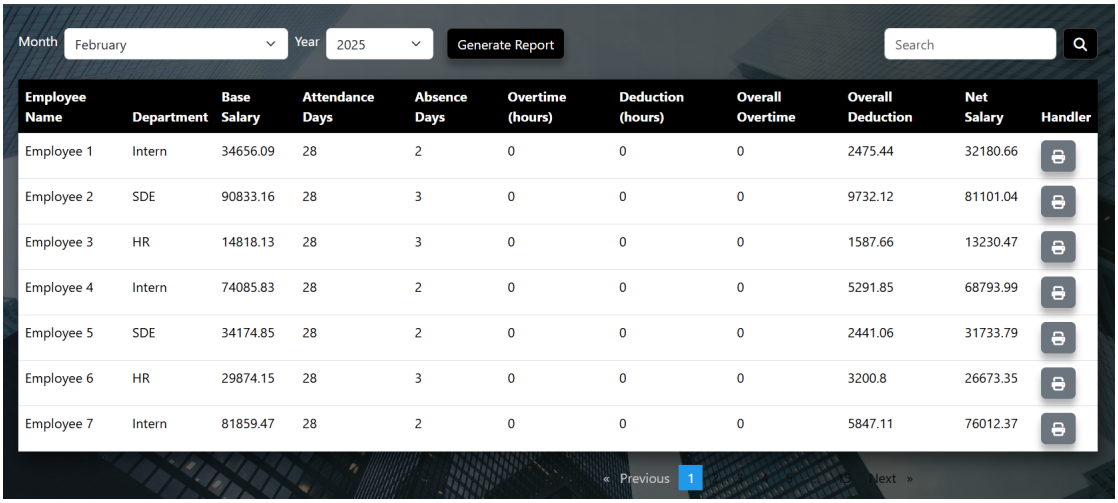


Figure 4.11: Automated payroll report (Req. 2.1.2, 2.2.2)

Generate payroll from submitted first date and last date. An stored procedure is used, so this generation takes just a few seconds.

```
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 4.12: Payroll calculation stored procedure (Req. 2.1.2, 2.2.2)

The stored procedure `GetEmployeePayslip` calculates employees' payroll details for a given date range. It follows these steps:

1. **Attendance Data Extraction:** Uses a Common Table Expression (CTE) to retrieve employee attendance details such as total attendance days, absent days, overtime, and late hours.
2. **Shift Duration Calculation:** Computes the work shift duration using the `DATEDIFF` function.
3. **Payroll Calculation:**
   - **Hourly Rate:** Computed as $\frac{\text{BaseSalary}}{\text{AttendanceDays} \times \text{ShiftDuration}}$.
   - **Additional Pay:** Based on overtime hours.
   - **Deduction Pay:** Includes penalties for lateness and absence.

- **Final Salary:** Calculated as:

$$\text{BaseSalary} + \text{AdditionalPay} - \text{DeductionPay}$$

4. **Final Output:** Returns a detailed payslip report ordered by Employee ID.

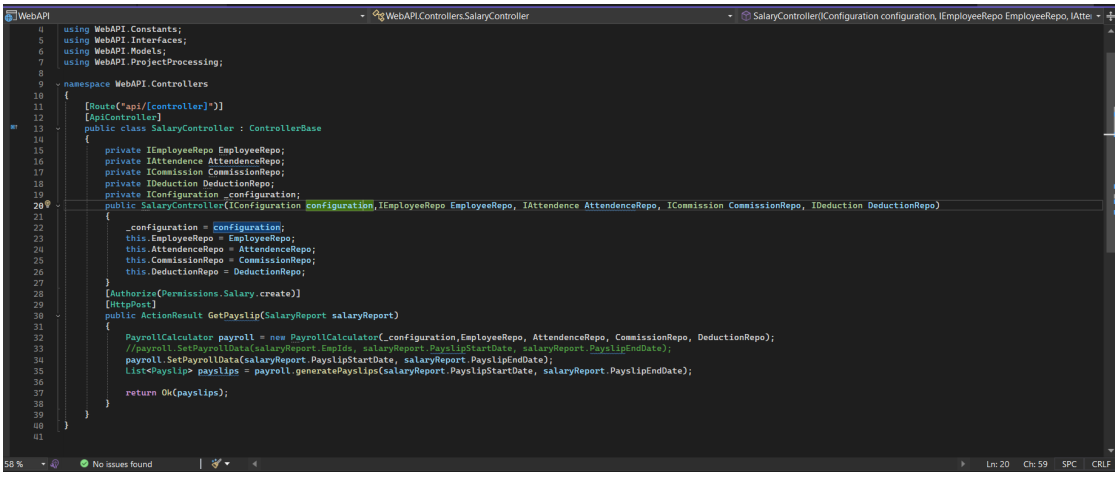


Figure 4.13: Payroll calculation controller (Req. 2.1.2, 2.2.2)

The `SalaryController` in ASP.NET Core handles payslip generation with the following functionality:

1. **Dependency Injection:** Injects repositories for employee data, attendance, commissions, and deductions.
2. **Authorization:** The `GetPayslip` method requires permission `Permissions.Salary.create`.
3. **Payroll Calculation:**
   - Initializes `PayrollCalculator` with dependencies.
   - Calls `SetPayrollData` to set the payslip period.
   - Generates payslips using `generatePayslips`.
4. **Response:** Returns a JSON list of payslips.

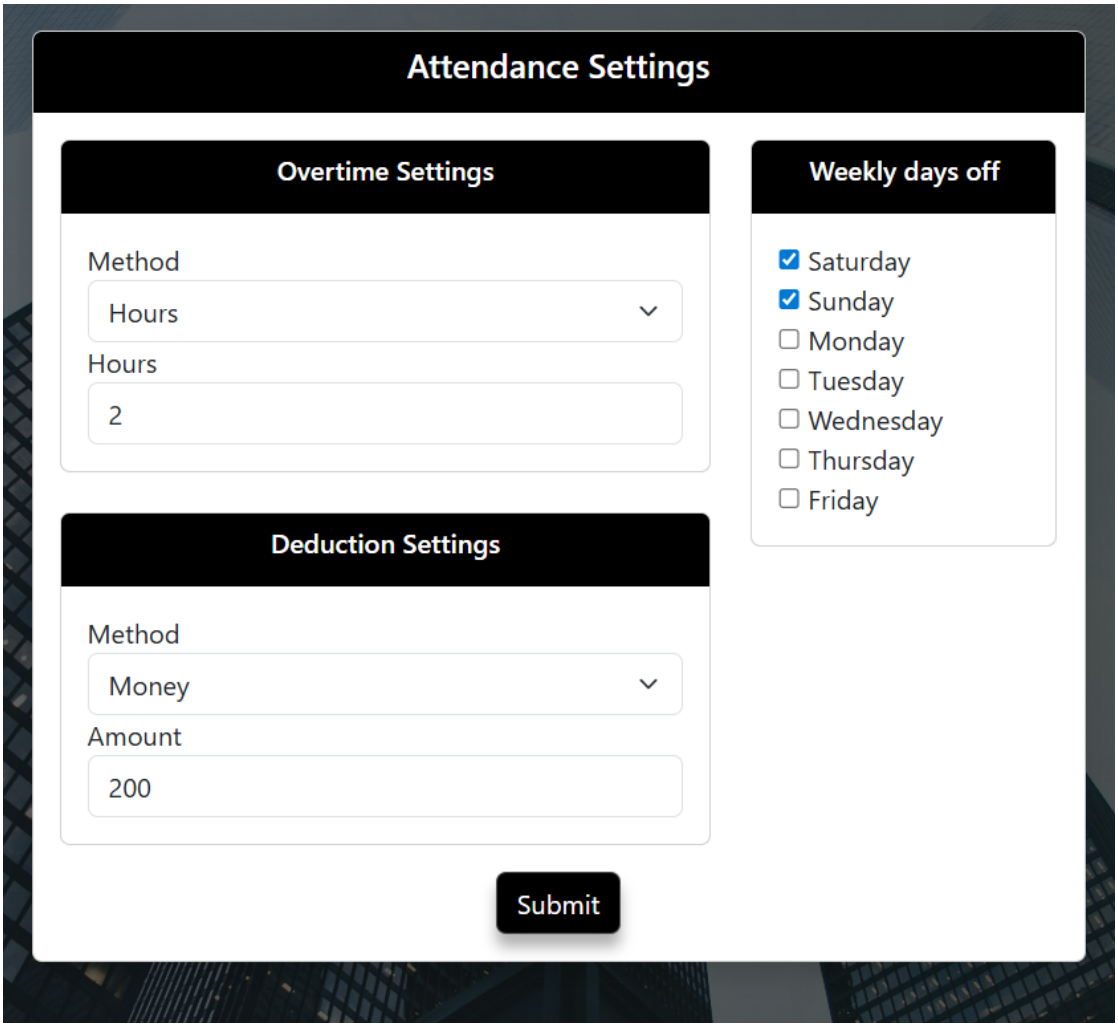


Figure 4.14: System settings with access controls (Req. 2.1.4)

These settings allow you to control the overtime reward and undertime deduction for salary payroll calculations. You can also mention the weekdays-off for you office.

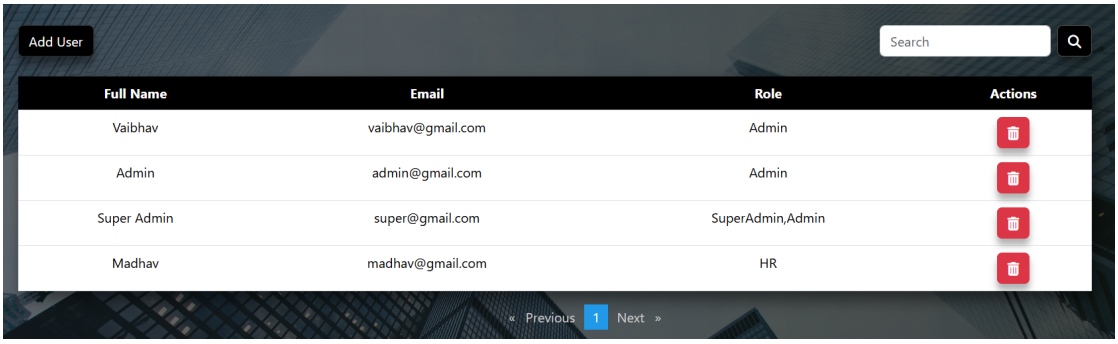


Figure 4.15: User role management interface (Req. 2.1.4)

You can assign which user gets what role.

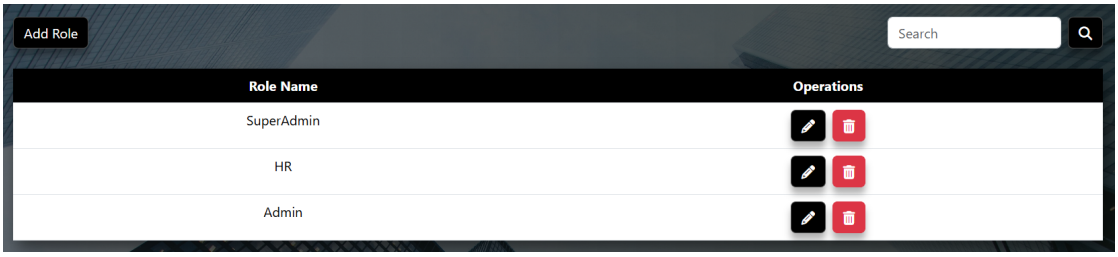
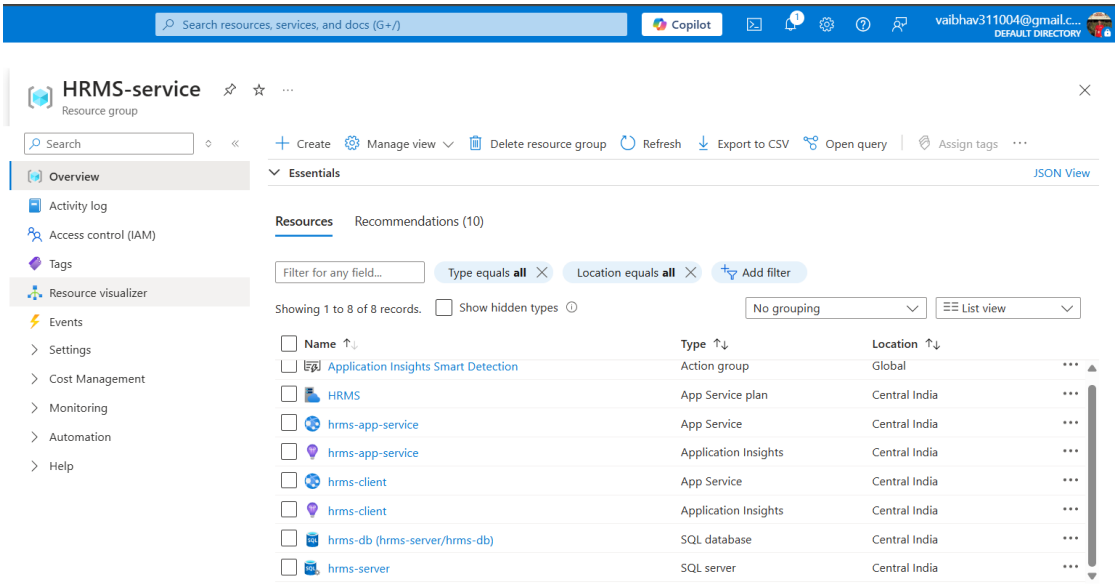


Figure 4.16: Predefined system roles (Req. 2.1.4)

You can also assign what are the features each role can access.



This is a screenshot of Azure site, showing our deployment. 1.5

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.

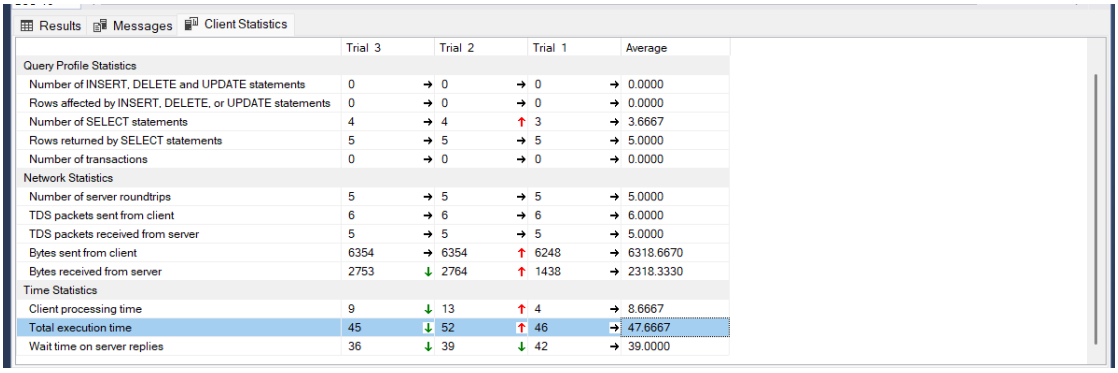
A significant observation was made during the payroll calculation testing. Initially, payroll was processed using a backend loop-based approach, which took approximately three minutes to handle 10,000 employee records. After optimization using SQL stored procedures, the same payroll calculations were 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.

5.5 Test Cases

Table 5.1: Important Test Cases with Requirement Mapping

Test ID	**Steps**	**Expected Result**	**Actual Result**	**Req. ID**
TC-001	Login with valid credentials	Redirect to dashboard	Passed	FR-1.1
TC-002	Upload Excel attendance	Attendance added in ¡10s	Passed	FR-3.1
TC-003	Bulk salary update (10k records)	Execution time ¡15s	Passed	NFR-2

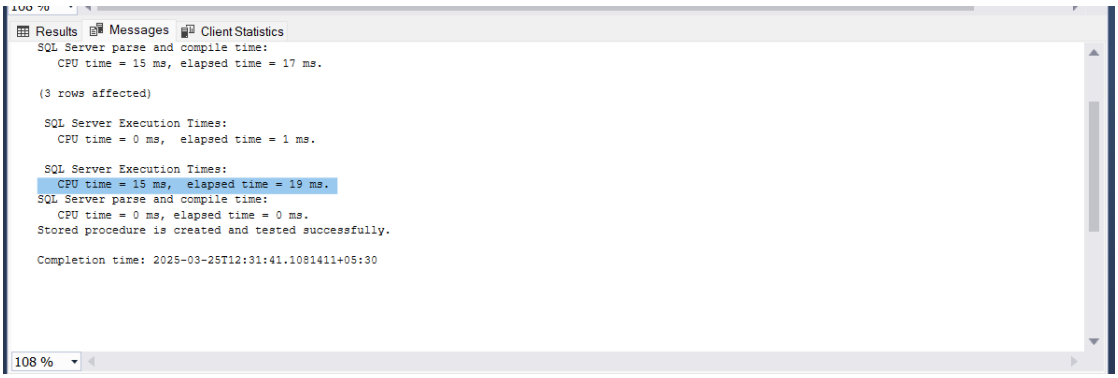


Results | Messages | Client Statistics

	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

Ran client analysis on SQL Server Management Studio



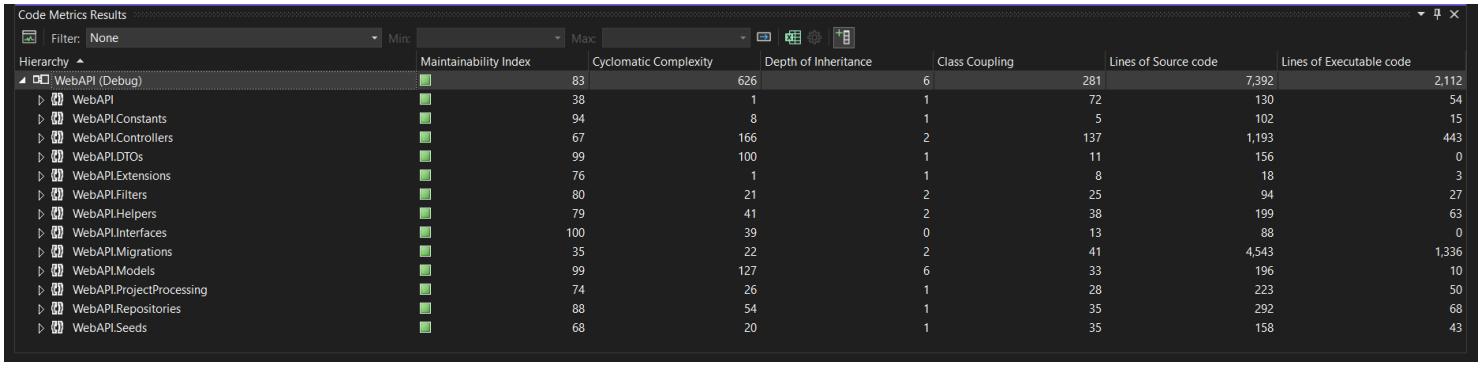
Results | Messages | Client Statistics

```
SQL Server parse and compile time:
   CPU time = 15 ms, elapsed time = 17 ms.

(3 rows affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 19 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Stored procedure is created and tested successfully.

Completion time: 2025-03-25T12:31:41.1081411+05:30
```

108 %

Figure 5.2: database payroll execution time - 19ms

Stored Procedure taking 19ms in total.



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
WebAPI (Debug)	83	626	6	281	7,392	2,112
WebAPI	38	1	1	72	130	54
WebAPI.Constants	94	8	1	5	102	15
WebAPI.Controllers	67	166	2	137	1,193	443
WebAPI.DTOs	99	100	1	11	156	0
WebAPI.Extensions	76	1	1	8	18	3
WebAPI.Filters	80	21	2	25	94	27
WebAPI.Helpers	79	41	2	38	199	63
WebAPI.Interfaces	100	39	0	13	88	0
WebAPI.Migrations	35	22	2	41	4,543	1,336
WebAPI.Models	99	127	6	33	196	10
WebAPI.ProjectProcessing	74	26	1	28	223	50
WebAPI.Repositories	88	54	1	35	292	68
WebAPI.Seeds	68	20	1	35	158	43

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

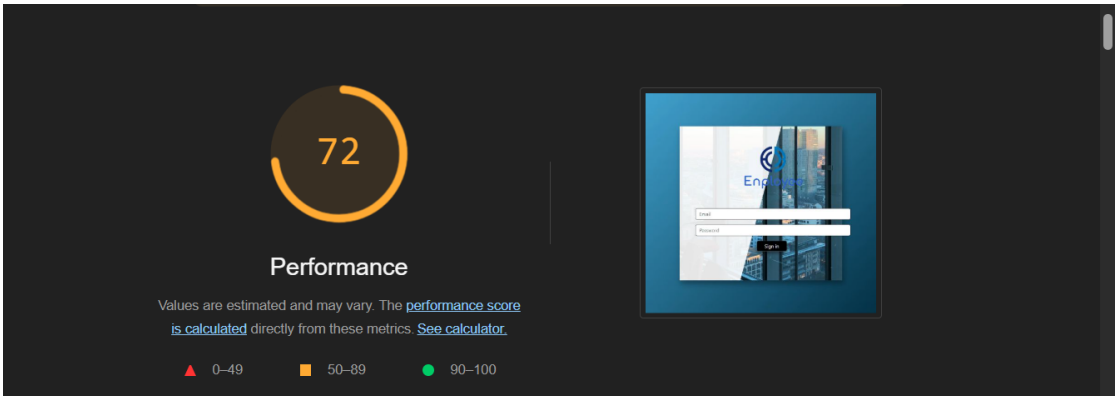


Figure 5.4: Frontend Lighthouse Performance score for inital page

Performace Score on Lighthouse, by Google.
For first page i.e., login page of our site

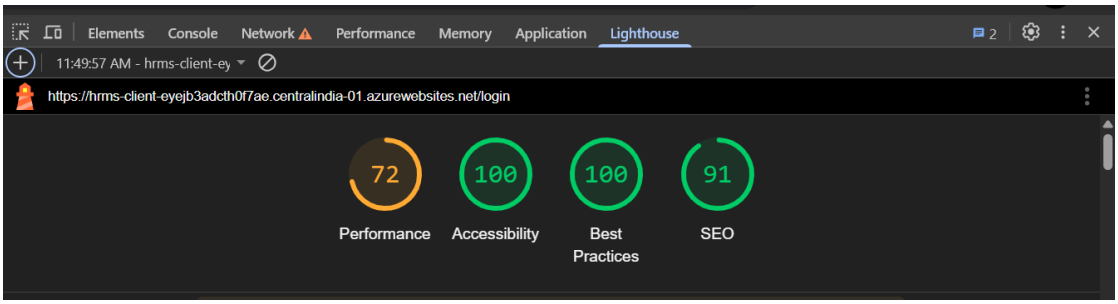


Figure 5.5: Frontend Lighthouse overall score for inital page

Other scores on Ligthhouse,
Accessibility: 100%
Best Practices: 100%
SEO: 91%

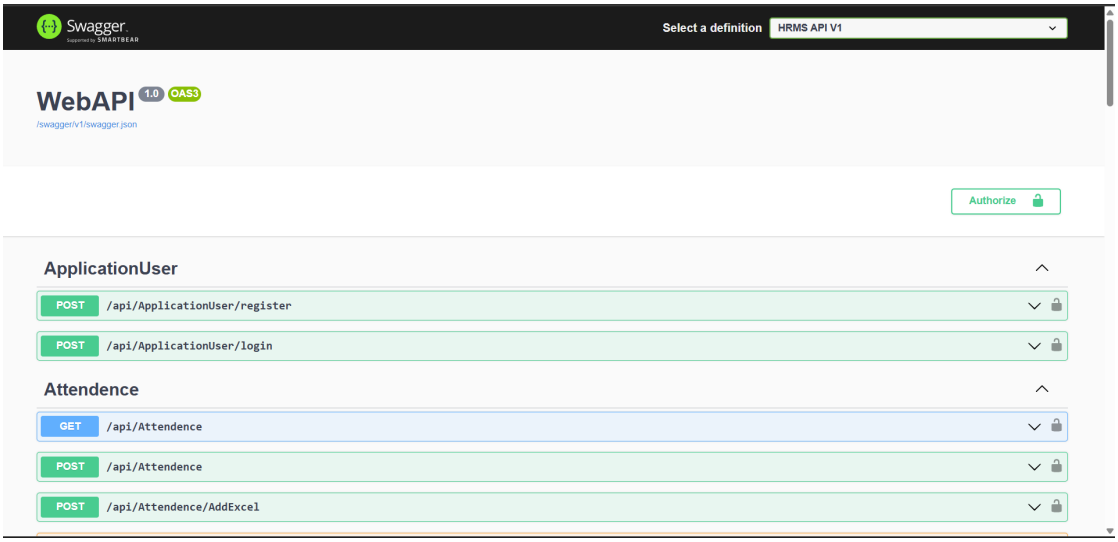


Figure 5.6: swagger-1

Application User controller

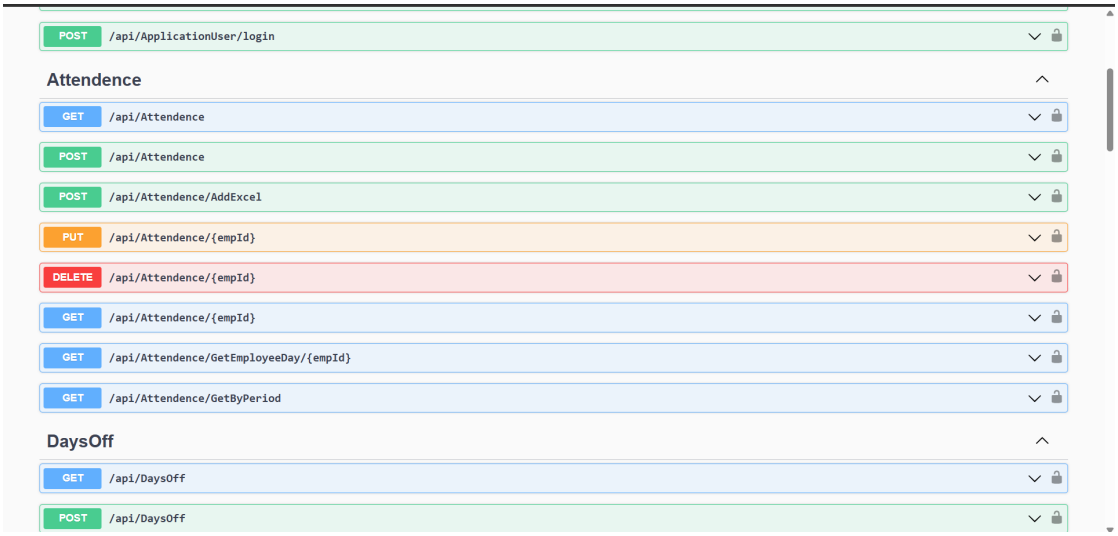


Figure 5.7: Swagger-2

Attendance Controller

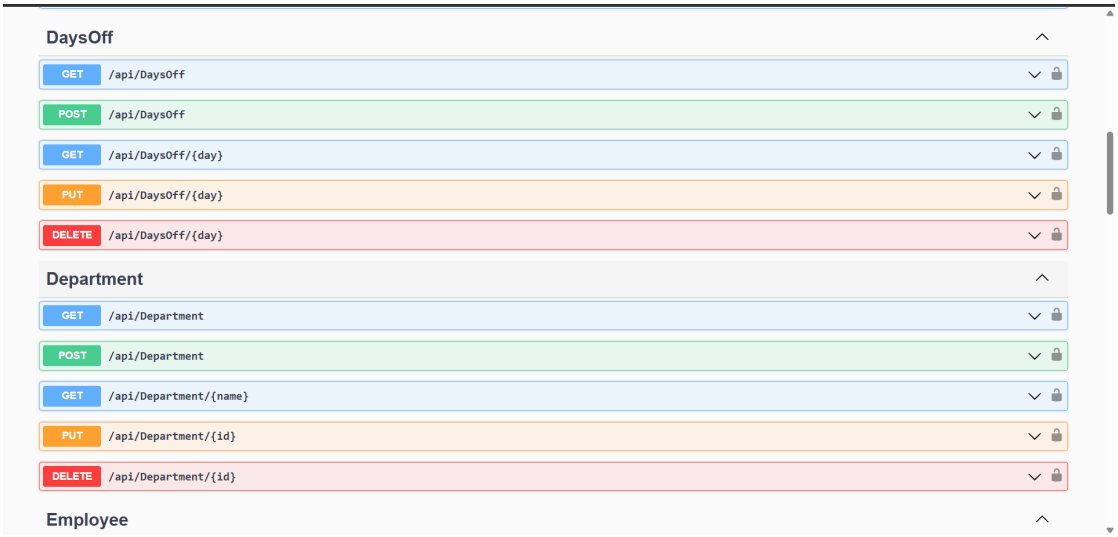


Figure 5.8: Swagger-3

Days off and Department Controller

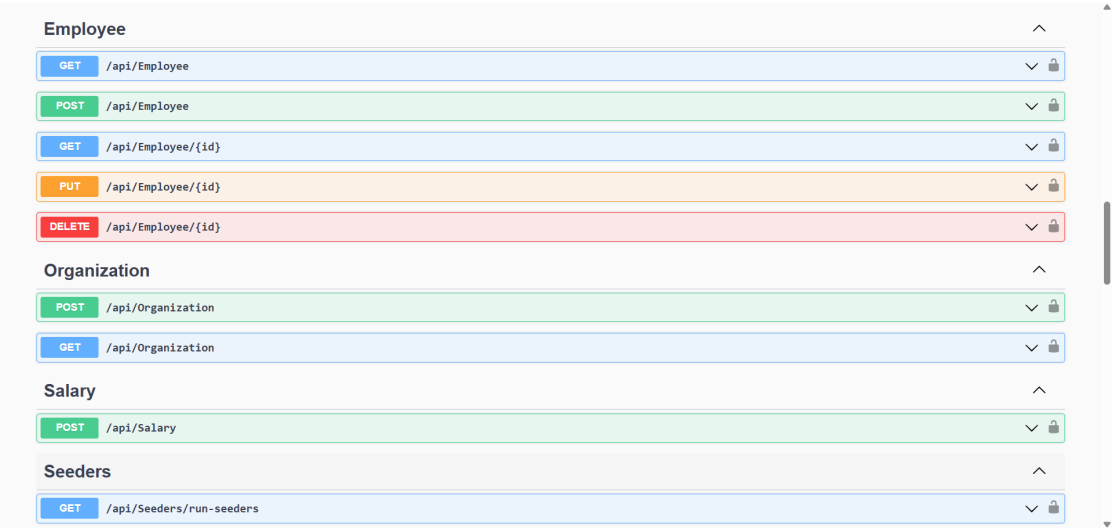


Figure 5.9: Swagger-4

Employee, Organization and Salary(Payroll) controller

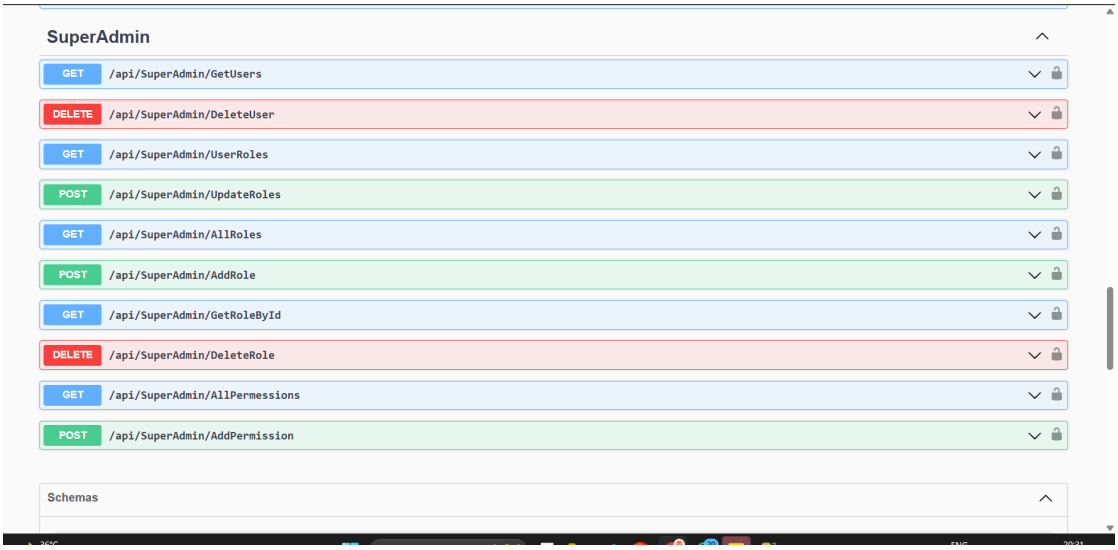


Figure 5.10: Swagger-5

Super Admin Controller, used for User and Roles management.

Figure 5.11: Swagger-6

Some DTOs (Data Transfer Objects) used.

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 multithreading 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.

Bibliography

[1] Angular Team. *Angular Documentation*. https://angular.io/docs, 2023.

[2] Microsoft. *.NET Core Guide*. https://learn.microsoft.com/en-us/dotnet/core/, 2024.

[3] Microsoft. *SQL Server Stored Procedures*. https://learn.microsoft.com/en-us/sql/, 2023.