

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
Associate Professor

Dr. C. K. Bhensdadia
Head, CE Department

Contents

1	Introduction	1
1.1	Overview	1
1.2	Project Objectives	1
1.3	System Overview	1
1.4	Definitions, Acronyms, and Abbreviations	2
1.5	Link	2
2	Requirement Specification and Analysis	3
2.1	Functional Requirements	3
2.1.1	Employee Management	3
2.1.2	Payroll Management	3
2.1.3	Attendance Management	3
2.1.4	Role and Authorization Management	3
2.1.5	Department Management	3
2.2	Non-Functional Requirements	4
2.2.1	Usage	4
2.2.2	Performance	4
2.2.3	Scalability	4
3	Design	5
3.1	ER Diagram	5
3.2	Class Diagram	7
3.2.1	Attendance	9
3.2.2	Payroll	10
4	Implementation details	11
4.1	Technologies Used	11
4.1.1	Frontend Technologies	11
4.1.2	Backend Technologies	11
4.1.3	Database	11
4.1.4	Development Environment	11
4.2	Screenshots	12
5	Testing	26
5.1	Overview	26
5.2	API Testing	26
5.3	Frontend Testing	26

5.4	Database Testing	26
5.5	Test Cases	27
6	Conclusion and Future Extensions	34
6.1	Conclusion	34
6.2	Future Extensions	34
6.2.1	AI Chatbot for Employee Report Analysis	34
6.2.2	Further Optimize the payroll calculations to include concurrency .	34

List of Figures

Figure 3.1 Entity Relationship Diagram - 1	5
Figure 3.2 Entity Relationship Diagram - 2	6
Figure 3.3 Class Diagram - 1	7
Figure 3.4 Class Diagram - 2	8
Figure 3.5 Sequence Diagram - Attendance	9
Figure 3.6 Sequence Diagram - Payroll	10
Figure 4.1 Login page implementing Role-based access controls (Req. 2.1.4)	12
Figure 4.2 JWT Token stored for 1 day for storing logged in sessions(Req. 2.1.4)	12
Figure 4.3 Navigation bar with Role-based views (Req. ??)	14
Figure 4.4 Employee list supporting CRUD operations (Req. 2.1.1)	15
Figure 4.5 Department management interface (Req. 2.1.5)	15
Figure 4.6 Holiday management affecting payroll calculations (Req. 2.1.2)	16
Figure 4.7 Excel template for attendance upload (Req. 2.1.3)	17
Figure 4.8 Bulk attendance upload form (Req. 2.1.3, 2.2.2)	18
Figure 4.9 Attendance records integrated with payroll (Req. 2.1.3, 2.1.2)	18
Figure 4.10 Excel processing code (Req. 2.1.3, 2.2.2)	19
Figure 4.11 Automated payroll report (Req. 2.1.2, 2.2.2)	20
Figure 4.12 Payroll calculation stored procedure (Req. 2.1.2, 2.2.2)	21
Figure 4.13 Payroll calculation controller (Req. 2.1.2, 2.2.2)	22
Figure 4.14 System settings with access controls (Req. 2.1.4)	23
Figure 4.15 User role management interface (Req. 2.1.4)	24
Figure 4.16 Predefined system roles (Req. 2.1.4)	24
Figure 5.1 database payroll client analysis using ssms query	28
Figure 5.2 database payroll execution time - 19ms	28
Figure 5.3 Backend Code Maintainability Score	29
Figure 5.4 Frontend Lighthouse Performance score for initial page	30
Figure 5.5 Frontend Lighthouse overall score for initial page	30
Figure 5.6 swagger-1	31
Figure 5.7 Swagger-2	31
Figure 5.8 Swagger-3	32
Figure 5.9 Swagger-4	32
Figure 5.10 Swagger-5	33
Figure 5.11 Swagger-6	33

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, frontend and database deployed on azure for enhanced accessibility and reliability.

1.4 Definitions, Acronyms, and Abbreviations

- **HR:** Human Resources
- **employee:** 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

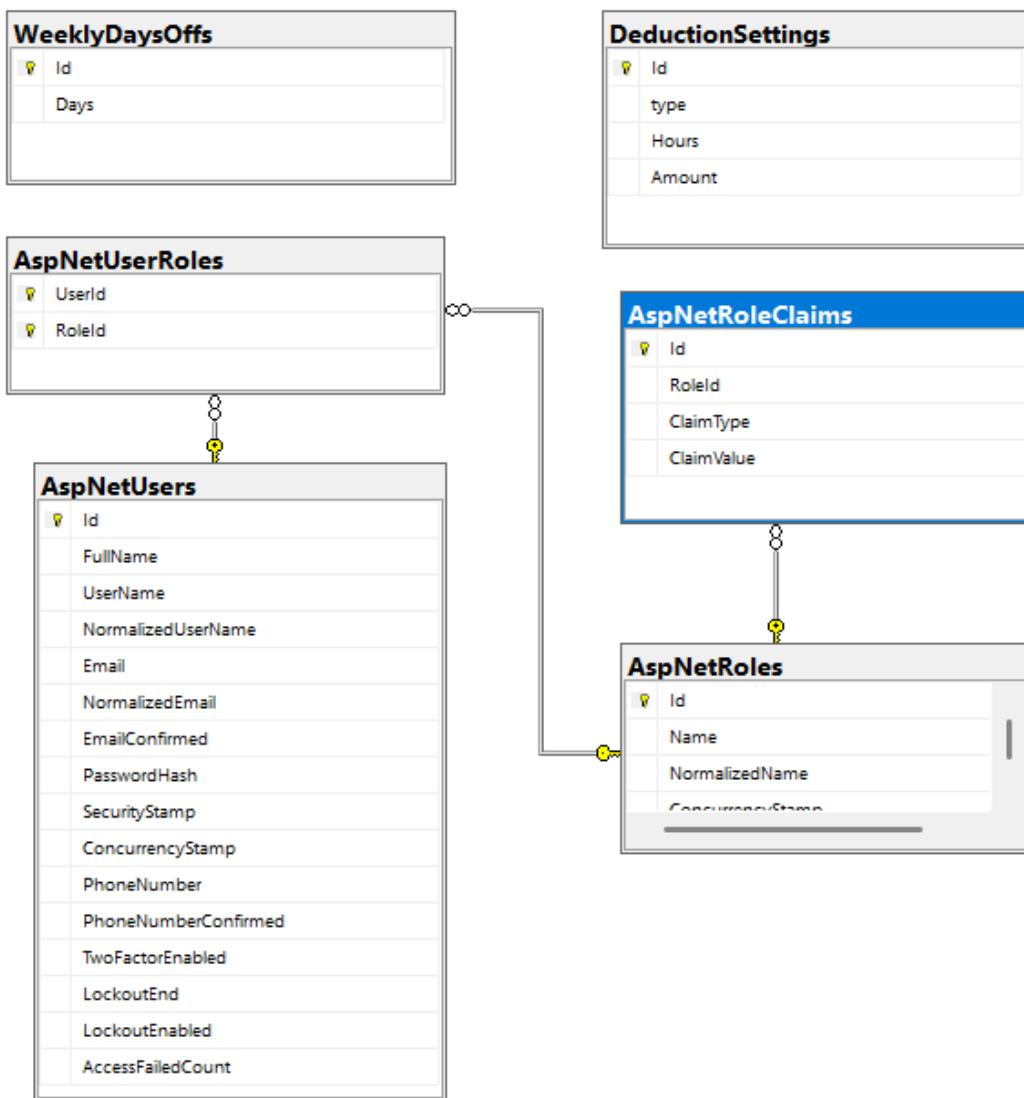


Figure 3.1: Entity Relationship Diagram - 1

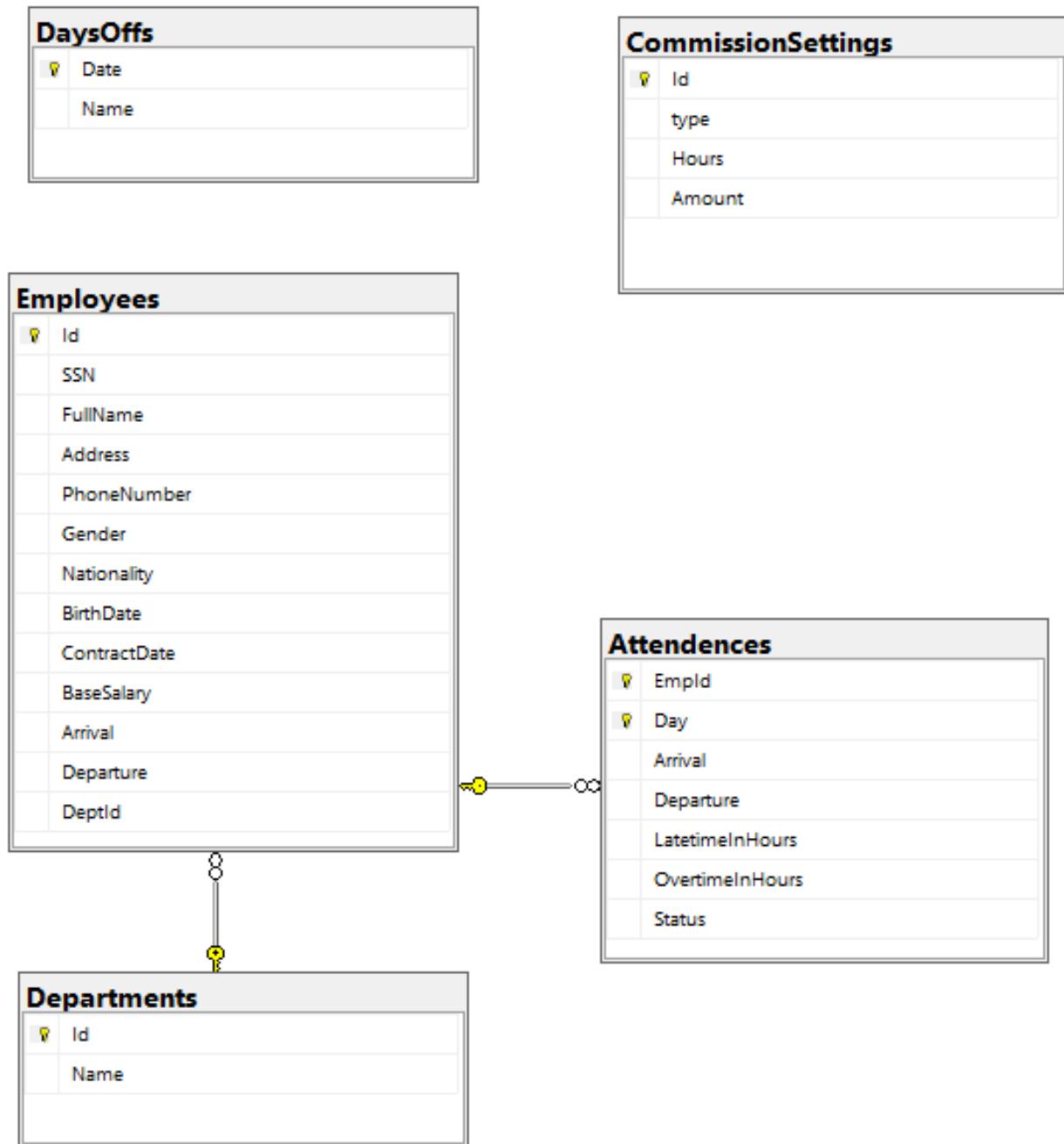


Figure 3.2: Entity Relationship Diagram - 2

3.2 Class Diagram

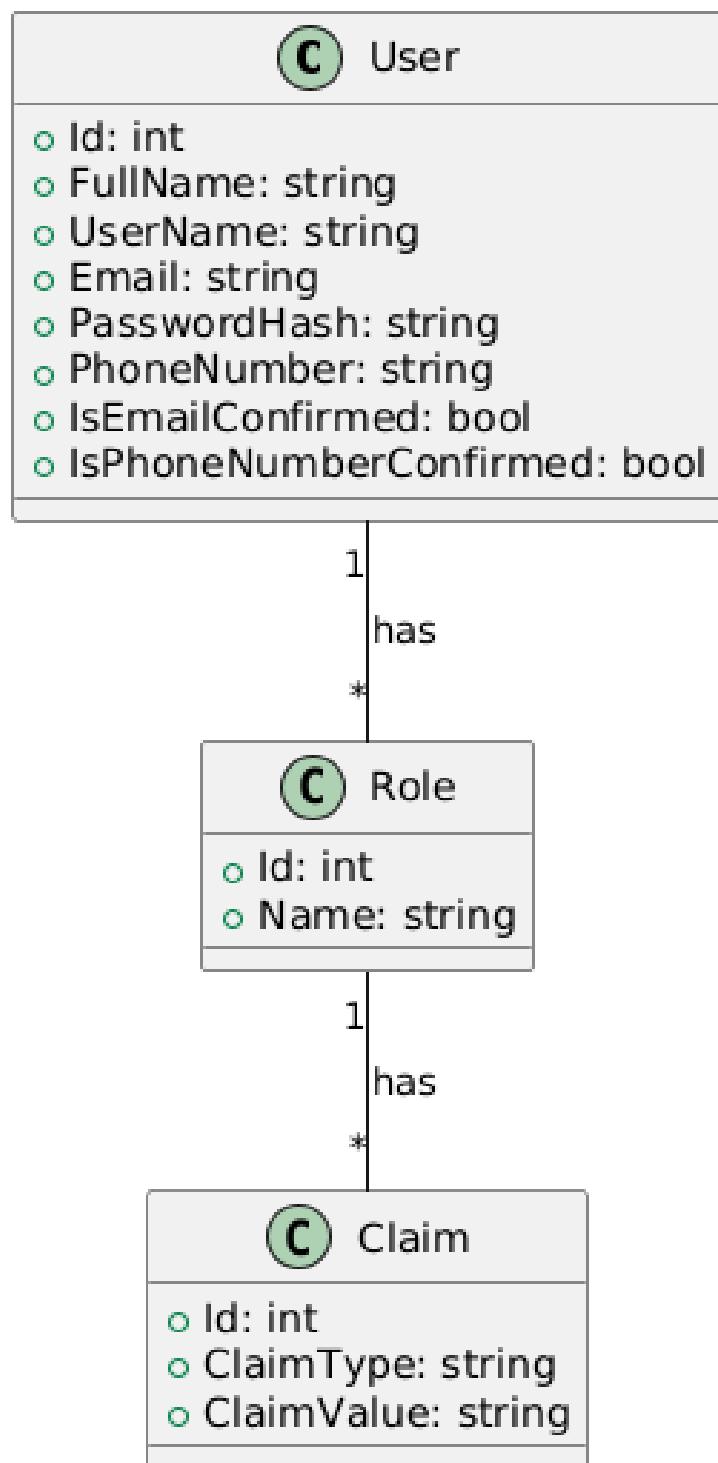


Figure 3.3: Class Diagram - 1

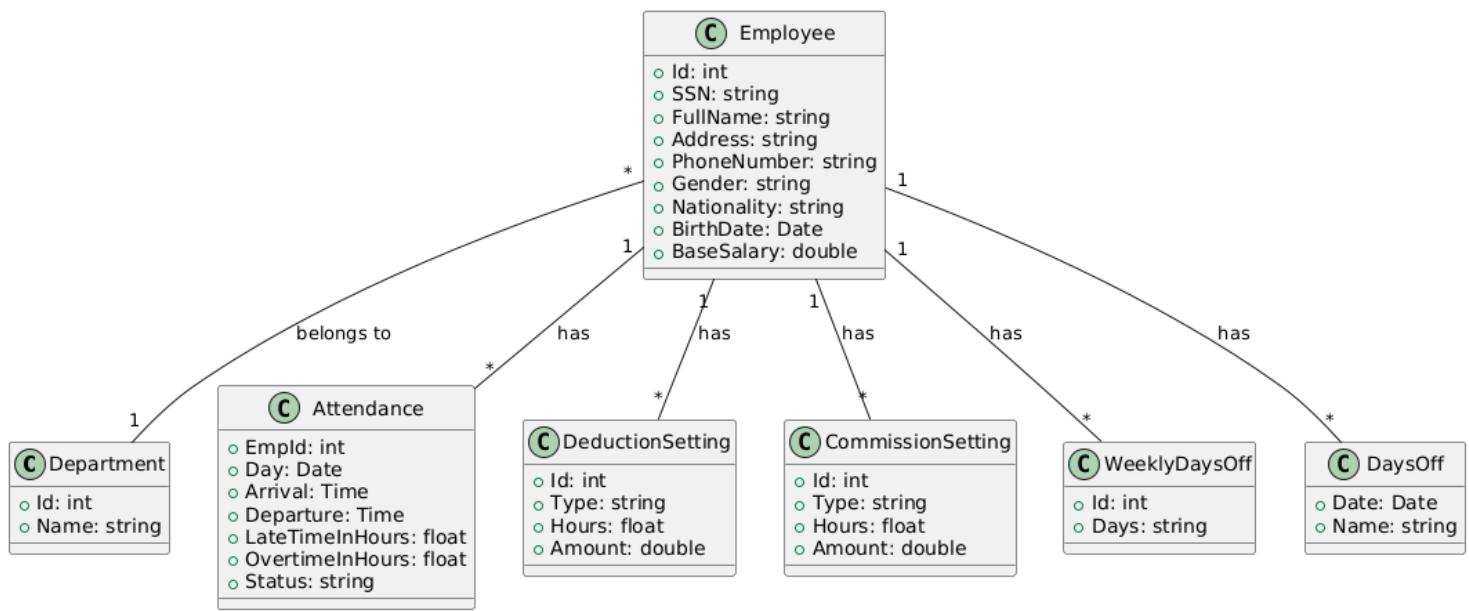


Figure 3.4: Class Diagram - 2

3.2.1 Attendance

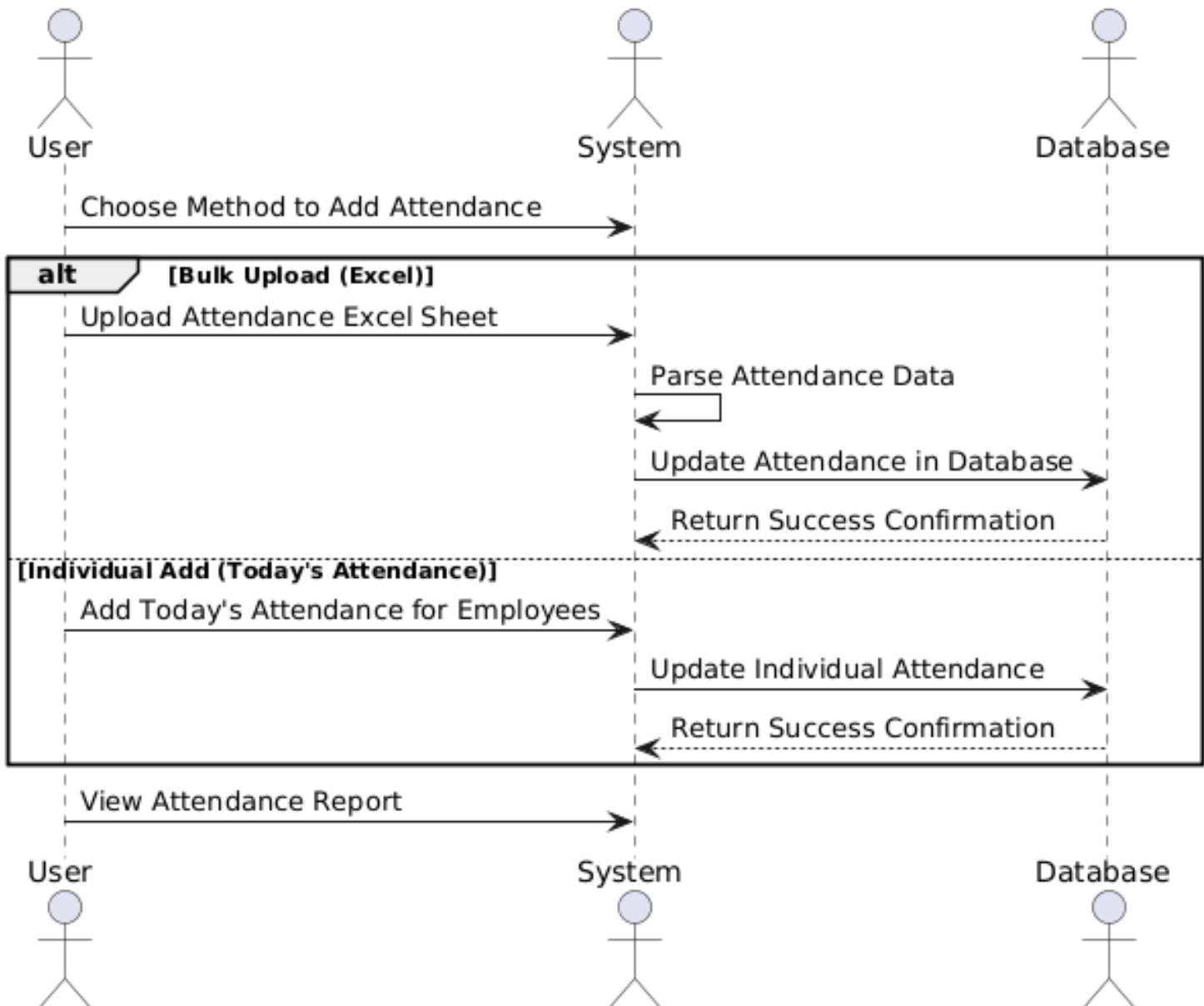


Figure 3.5: Sequence Diagram - Attendance

3.2.2 Payroll

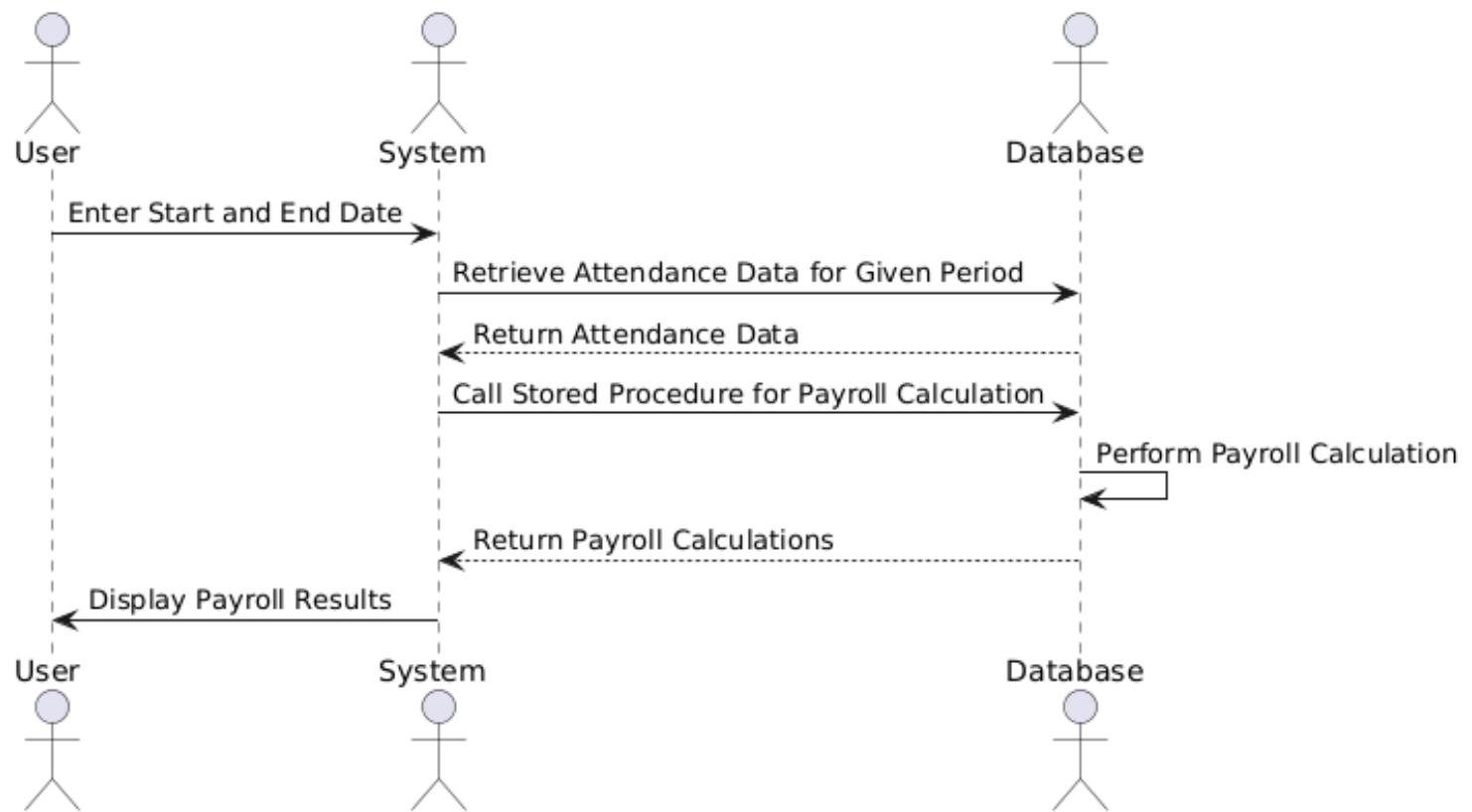


Figure 3.6: Sequence Diagram - Payroll

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

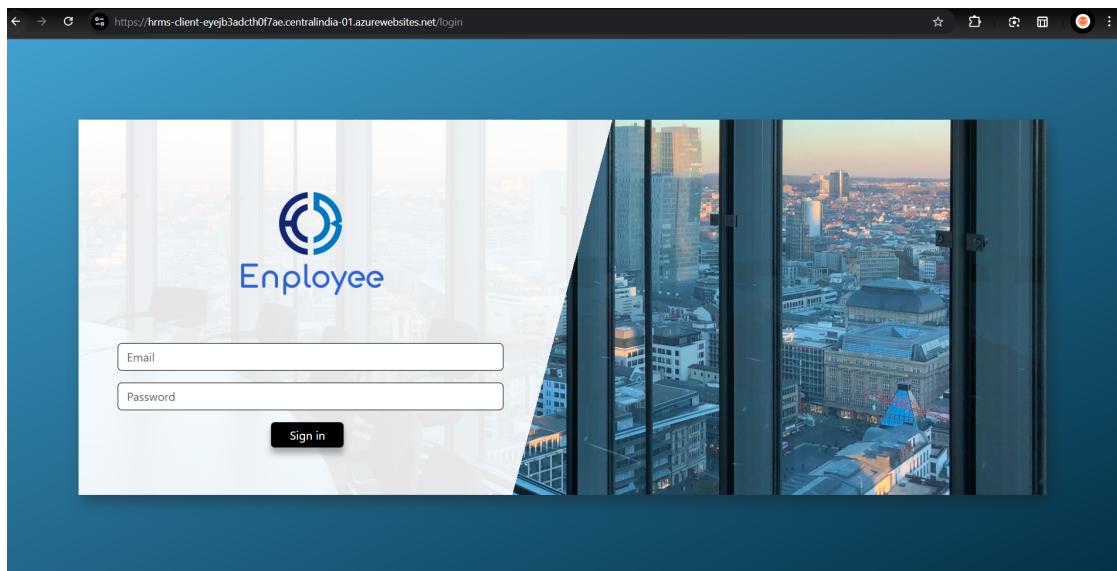


Figure 4.1: Login page implementing Role-based access controls (Req. 2.1.4)

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

A screenshot of a code editor showing a C# method for generating a JWT token. The code uses the `SymmetricSecurityKey` and `SigningCredentials` classes from the `System.IdentityModel.Tokens.Jwt` namespace. It creates a token with the issuer set to the configuration key "JWT:ValidIssuer", the audience set to "JWT:ValidAudience", and a claim for the user's full name. The token is signed using the `HmacSha256` algorithm. The token is then written to a file using a `JwtSecurityTokenHandler` and its expiration is set to one day. Finally, the first role from the user's list is returned.

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.

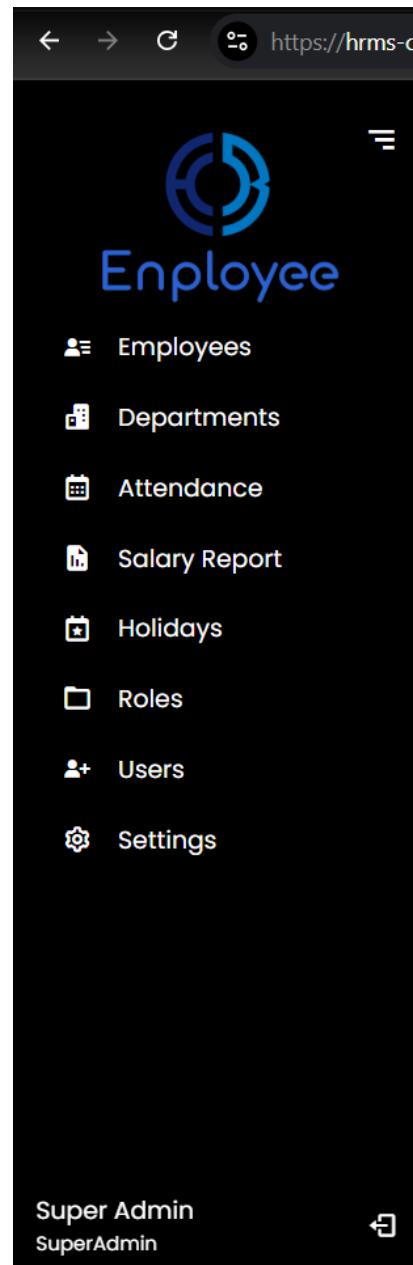


Figure 4.3: Navigation bar with Role-based views (Req. ??)

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.

The screenshot shows a web-based employee management system. At the top left is a sidebar with various icons. In the center is a table titled "Employee list". The table has columns for Full Name, Social Security Number, Address, Gender, Phone Number, Base Salary, Birth Date, Contract Date, Clock In, Clock Out, Department, and Actions. The "Actions" column contains icons for edit and delete. Below the table is a navigation bar with "Previous", page numbers (1, 2, 3, 4, 5, Next), and a search bar.

Full Name	Social Security Number	Address	Gender	Phone Number	Base Salary	Birth Date	Contract Date	Clock In	Clock Out	Department	Actions
Employee 1	52655007.24012816	Address 1	Female	9.37417e+009	34656.09	2001-06-06	2017-07-26	18:00:00	18:00:00	Intern	
Employee 2	703509158.0722036	Address 2	Female	9.95921e+009	90833.16	2008-11-10	2017-09-30	09:00:00	01:00:00	SDE	
Employee 3	169389189.6469565	Address 3	Female	9.21306e+009	14818.13	1998-06-09	2019-12-27	20:00:00	04:00:00	HR	
Employee 4	987693136.0049014	Address 4	Female	9.57949e+009	74085.83	2017-03-28	2022-04-25	06:00:00	09:00:00	Intern	
Employee 5	806186721.2822202	Address 5	Female	9.01518e+009	34174.85	2011-12-01	2024-11-26	18:00:00	01:00:00	SDE	
Employee 6	140052511.448811	Address 6	Female	9.90686e+009	29874.15	2020-12-11	2019-04-06	08:00:00	16:00:00	HR	
Employee 7	295370572.8467754	Address 7	Female	9.45917e+009	81859.47	2007-08-10	2024-09-15	00:00:00	01:00:00	Intern	

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

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

The screenshot shows a web-based department management system. At the top left is a sidebar with various icons. In the center is a table titled "Department list". The table has columns for Department ID and Department Name. The "Actions" column contains icons for delete. Below the table is a navigation bar with "Previous", page numbers (1, 2, 3, 4, 5, Next), and a search bar.

Department ID	Department Name	Actions
3	HR	
2	Intern	
1	SDE	

Figure 4.5: Department management interface (Req. 2.1.5)

Basic Department addition and deletion can be done.

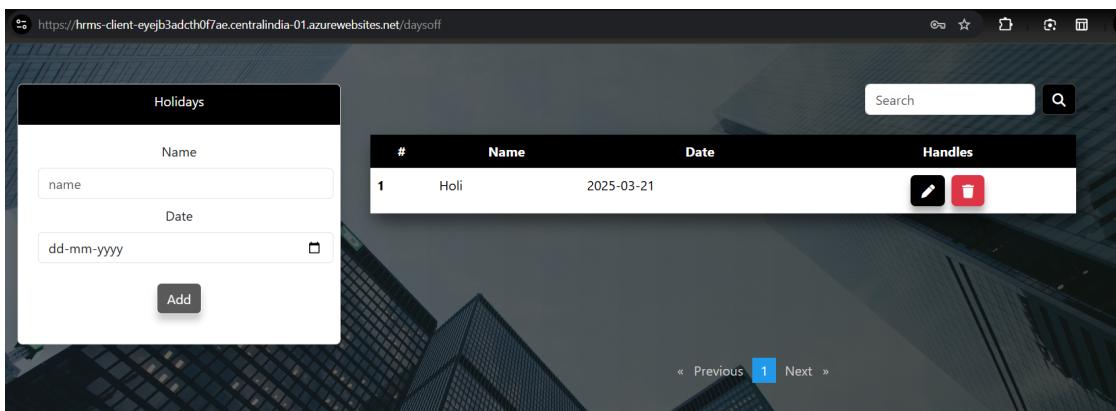


Figure 4.6: Holiday management affecting payroll calculations (Req. 2.1.2)

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	Empld	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.

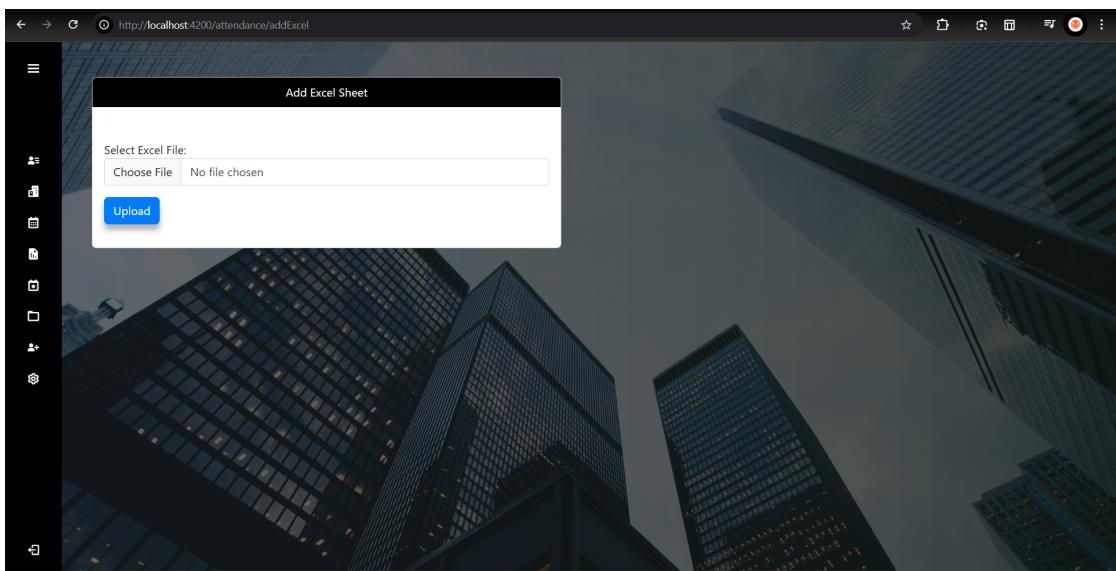


Figure 4.8: Bulk attendance upload form (Req. 2.1.3, 2.2.2)

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.

Department	Employee name	Clock in	Clock out	Date	Status	Handler
Intern	Employee 1	09:00:00	17:00:00	2025-02-01	Attended	
Intern	Employee 1	-----	-----	2025-02-02	Absent	
Intern	Employee 1	-----	-----	2025-02-03	Absent	
Intern	Employee 1	09:00:00	17:00:00	2025-02-04	Attended	
Intern	Employee 1	09:00:00	17:00:00	2025-02-05	Attended	
Intern	Employee 1	09:00:00	17:00:00	2025-02-06	Attended	
Intern	Employee 1	09:00:00	17:00:00	2025-02-07	Attended	
Intern	Employee 1	09:00:00	17:00:00	2025-02-08	Attended	

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 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 `Attendance` 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.

The screenshot shows a web-based payroll reporting application. At the top, there are dropdown menus for 'Month' (set to 'February') and 'Year' (set to '2025'), a 'Generate Report' button, and a search bar with a magnifying glass icon. Below the header is a table with the following data:

Employee Name	Department	Base Salary	Attendance Days	Absence Days	Overtime (hours)	Deduction (hours)	Overall Overtime	Overall Deduction	Net Salary	Handler
Employee 1	Intern	34656.09	28	2	0	0	0	2475.44	32180.66	
Employee 2	SDE	90833.16	28	3	0	0	0	9732.12	81101.04	
Employee 3	HR	14818.13	28	3	0	0	0	1587.66	13230.47	
Employee 4	Intern	74085.83	28	2	0	0	0	5291.85	68793.99	
Employee 5	SDE	34174.85	28	2	0	0	0	2441.06	31733.79	
Employee 6	HR	29874.15	28	3	0	0	0	3200.8	26673.35	
Employee 7	Intern	81859.47	28	2	0	0	0	5847.11	76012.37	

At the bottom, there are navigation links: '< Previous', a page number '1', and 'Next >'.

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)

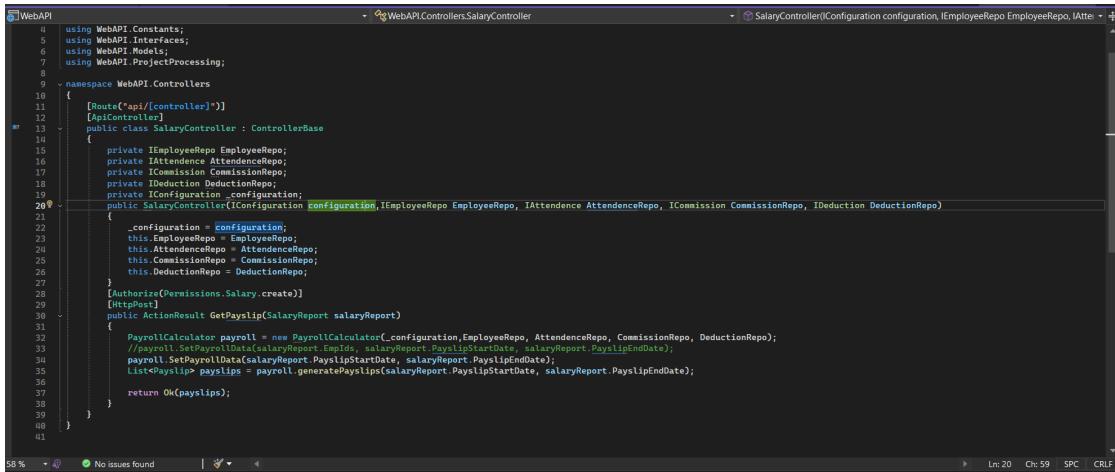
The stored procedure `GetEmployeePayslip` calculates employees' payroll details for a given date range. It follows these steps:

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.



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6  using Microsoft.Extensions.Logging;
7  using WebAPI.Constants;
8  using WebAPI.Interfaces;
9  using WebAPI.Models;
10 using WebAPI.ProjectProcessing;
11
12 [Route("api/[controller]")]
13 [ApiController]
14 public class SalaryController : ControllerBase
15 {
16     private IEmployeeRepo EmployeeRepo;
17     private IAttendance AttendanceRepo;
18     private ICCommission CommissionRepo;
19     private IDeduction DeductionRepo;
20     private IConfiguration _configuration;
21
22     public SalaryController(IConfiguration configuration, IEmployeeRepo EmployeeRepo, IAttendance AttendanceRepo, ICCommission CommissionRepo, IDeduction DeductionRepo)
23     {
24         _configuration = configuration;
25         this.EmployeeRepo = EmployeeRepo;
26         this.AttendanceRepo = AttendanceRepo;
27         this.CommissionRepo = CommissionRepo;
28         this.DeductionRepo = DeductionRepo;
29     }
30
31     [Authorize(Permissions.Salary.create)]
32     [HttpPost]
33     public ActionResult<List<Payroll>> GetPayslip(SalaryReport salaryReport)
34     {
35         PayrollCalculator payroll = new PayrollCalculator(_configuration, EmployeeRepo, AttendanceRepo, CommissionRepo, DeductionRepo);
36         payroll.SetPayrollData(salaryReport.PayslipStartDate, salaryReport.PayslipEndDate);
37         List<Payroll> payslips = payroll.generatePayslips(salaryReport.PayslipStartDate, salaryReport.PayslipEndDate);
38
39         return Ok(payslips);
40     }
41 }

```

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.

Attendance Settings

Overtime Settings

Method

Hours

Hours

2

Weekly days off

- Saturday
- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday

Deduction Settings

Method

Money

Amount

200

Submit

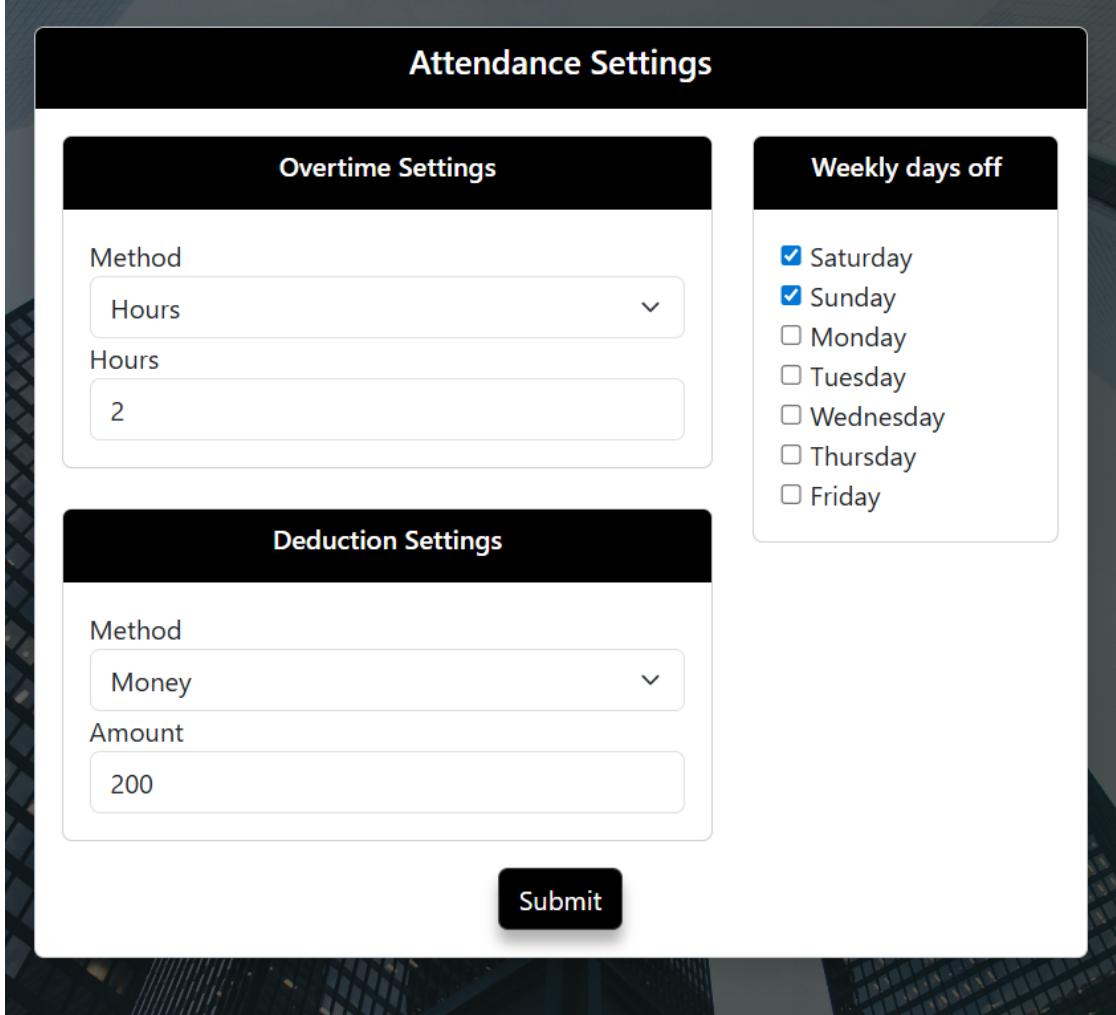


Figure 4.14: System settings with access controls (Req. 2.1.4)

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

The screenshot shows a user management interface with a dark header bar containing 'Add User' and a search bar. Below is a table with columns: Full Name, Email, Role, and Actions. The table lists four users:

Full Name	Email	Role	Actions
Vaibhav	vaibhav@gmail.com	Admin	
Admin	admin@gmail.com	Admin	
Super Admin	super@gmail.com	SuperAdmin,Admin	
Madhav	madhav@gmail.com	HR	

At the bottom, there are navigation links: « Previous, 1, Next ».

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

You can assign which user gets what role.

The screenshot shows a role management interface with a dark header bar containing 'Add Role' and a search bar. Below is a table with columns: Role Name and Operations. The table lists three predefined roles:

Role Name	Operations
SuperAdmin	
HR	
Admin	

Figure 4.16: Predefined system roles (Req. 2.1.4)

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

The screenshot shows the Azure portal interface for a resource group named 'HRMS-service'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, and Resource visualizer. The main content area displays a table of resources with columns for Name, Type, and Location. The table includes the following data:

Name	Type	Location
Application Insights Smart Detection	Action group	Global
HRMS	App Service plan	Central India
hrms-app-service	App Service	Central India
hrms-app-service	Application Insights	Central India
hrms-client	App Service	Central India
hrms-client	Application Insights	Central India
hrms-db (hrms-server/hrms-db)	SQL database	Central India
hrms-server	SQL server	Central India

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 initial 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 ≤ 10 s	Passed	FR-3.1
TC-003	Bulk salary update (10k records)	Execution time ≤ 15 s	Passed	NFR-2

	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

```

100 %
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

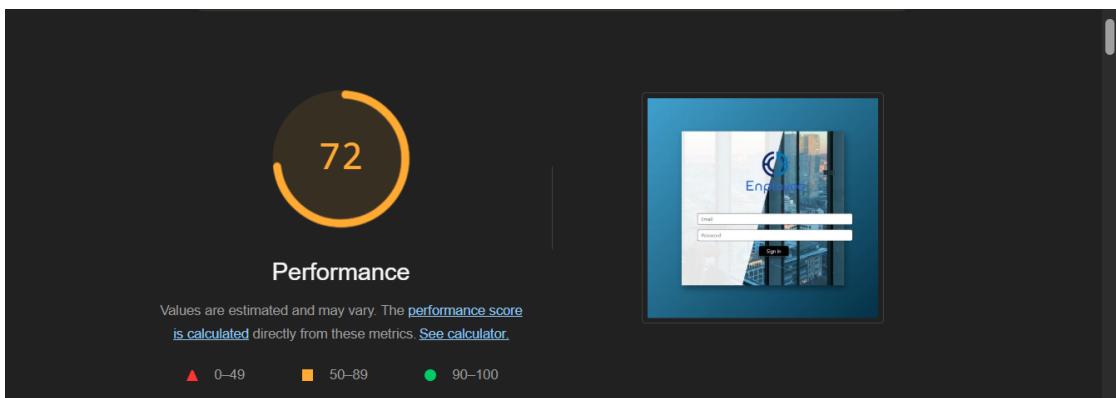


Figure 5.4: Frontend Lighthouse Performance score for initial page

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

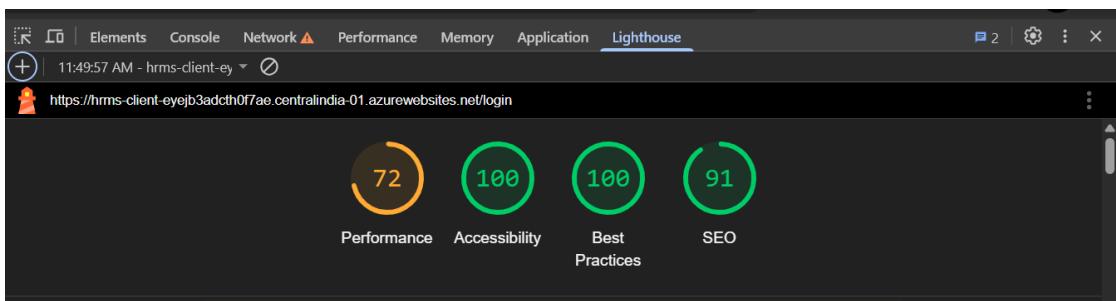


Figure 5.5: Frontend Lighthouse overall score for initial page

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

Swagger
Supported by SMARTBEAR

Select a definition HRMS API V1

WebAPI 1.0 OAS

/swagger/v1/swagger.json

Authorize

ApplicationUser

POST /api/ApplicationUser/register

POST /api/ApplicationUser/login

Attendance

GET /api/Attendance

POST /api/Attendance

POST /api/Attendance/AddExcel

Figure 5.6: swagger-1

Application User controller

POST /api/ApplicationUser/login

Attendance

GET /api/Attendance

POST /api/Attendance

POST /api/Attendance/AddExcel

PUT /api/Attendance/{empId}

DELETE /api/Attendance/{empId}

GET /api/Attendance/{empId}

GET /api/Attendance/GetEmployeeDay/{empId}

GET /api/Attendance/GetByPeriod

DaysOff

GET /api/DaysOff

Figure 5.7: Swagger-2

Attendance Controller

The screenshot shows the Swagger UI interface for a RESTful API. It is organized into three main sections: DaysOff, Department, and Employee.

- DaysOff**:
 - GET /api/DaysOff
 - POST /api/DaysOff
 - GET /api/DaysOff/{day}
 - PUT /api/DaysOff/{day}**
 - DELETE /api/DaysOff/{day}**
- Department**:
 - GET /api/Department
 - POST /api/Department
 - GET /api/Department/{name}
 - PUT /api/Department/{id}**
 - DELETE /api/Department/{id}**
- Employee**:
 - GET /api/Employee
 - POST /api/Employee**
 - GET /api/Employee/{id}
 - PUT /api/Employee/{id}**
 - DELETE /api/Employee/{id}**

Figure 5.8: Swagger-3

Days off and Department Controller

The screenshot shows the Swagger UI interface for a RESTful API. It is organized into four main sections: Employee, Organization, Salary, and Seeders.

- Employee**:
 - GET /api/Employee
 - POST /api/Employee**
 - GET /api/Employee/{id}
 - PUT /api/Employee/{id}**
 - DELETE /api/Employee/{id}**
- Organization**:
 - POST /api/Organization**
 - GET /api/Organization
- Salary**:
 - POST /api/Salary**
- Seeders**:
 - GET /api/Seeders/run-seeders

Figure 5.9: Swagger-4

Employee, Organization and Salary(Payroll) controller

The screenshot shows the Swagger UI interface for a SuperAdmin API. At the top, there is a header with the title "SuperAdmin". Below the header, a list of API endpoints is displayed in a table format. The table has two columns: the first column shows the HTTP method and endpoint path, and the second column contains a dropdown arrow and a lock icon. The endpoints listed are:

Method	Endpoint Path
GET	/api/SuperAdmin/GetUsers
DELETE	/api/SuperAdmin/DeleteUser
GET	/api/SuperAdmin/UserRoles
POST	/api/SuperAdmin/UpdateRoles
GET	/api/SuperAdmin/AllRoles
POST	/api/SuperAdmin/AddRole
GET	/api/SuperAdmin/GetRoleId
DELETE	/api/SuperAdmin/DeleteRole
GET	/api/SuperAdmin/AllPermissions
POST	/api/SuperAdmin/AddPermission

Below the table, there is a section titled "Schemas" which lists various Data Transfer Objects (DTOs) used in the API.

Figure 5.10: Swagger-5

Super Admin Controller, used for User and Roles management.

The screenshot shows the Swagger UI interface focusing on the "Schemas" section. This section lists several Data Transfer Objects (DTOs) used throughout the application. Each item in the list is a link to its detailed schema definition, indicated by a right-pointing arrow.

- AttendanceDTO >
- CheckBoxDTO >
- CommissionDTO >
- DaysOffDTO >
- DeductionDTO >
- DepartmentDTO >
- EmployeeDTO >
- LoginDTO >
- OrganizationSettings >

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 your 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.