Low-Level Design (LLD) for Employee Management System

1 Controllers

Controllers handle incoming HTTP requests, process them (often involving database operations), and return appropriate responses (usually views or JSON data).

1.1 EmployeeController

Handles operations related to employees.

- **ListEmployees**: Lists all employees with sorting and pagination.
- **EditEmployee**: Edits an employee's details.
- **AddEmployee**: Creates a new employee.
- **DeleteEmployee**: Deletes an employee.

1.2 SalaryController

Handles operations related to salaries.

- **ListSalary**: Lists all salaries with sorting and pagination.
- **EditSalary**: Edits a salary's details.
- **EditSalary1**:It adds new Transaction payable record in Salaries Table.

1.3 DepartmentController

Handles operations related to departments.

- **ListDepartments**: Lists all departments with sorting and pagination.
- **EditDepartment**: Edits a department's details.
- **AddDepartment**: Creates a new department.
- **DeleteDepartment**: Deletes a department.

1.4 AuthController

Handles operations related to user authorization and authentication.

- **Login**: Authenticates a user if they are a current employee and validates their username and password.
- **Register**: Registers a user if they are a current employee but do not have credentials.
- **Logout**: Logs out the user and deletes the associated cookies.

2 Models

Models represent the data structure of the application. They are typically mapped to database tables.

2.1 Employee

Properties:

- **Emp_Id**: int
- **Name**: string
- **DepartmentId**: int
- **Position**: string

2.2 Salary

Properties:

- **Sal_Id**: int
- **Emp_Id**: int
- **Amount**: decimal
- **Timestamp**: string

2.3 Department

Properties:

- **Dept_Id**: int
- **Name**: string

2.4 Auth

Properties:

- **Emp_Id**: int
- **Name**: string

3 Views

Views are responsible for rendering the UI. They use Razor syntax to embed C# code within HTML.

3.1 Employee Views

- **ListEmployees.cshtml**: Displays a list of employees.
- **EditEmployee.cshtml**: Form for editing an employee's details.
- **AddEmployee.cshtml**: Form for creating a new employee.

3.2 Salary Views

- **ListSalary.cshtml**: Displays a list of salaries.
- **EditSalary.cshtml**: Form for editing a salary's details.

3.3 Department Views

- **ListDepartments.cshtml**: Displays a list of departments.
- **EditDepartment.cshtml**: Form for editing a department's details.
- **AddDepartment.cshtml**: Form for creating a new department.

3.4 Auth Views

- **Login.cshtml**: Displays the login form for users to enter their credentials.
- **Register.cshtml**: Displays the registration form for new users to create an account.

4 Database Context

The database context is used to interact with the database using Entity Framework Core.

4.1 ApplicationDbContext

- **DbSets**:
  - **DbSet¡Employee¿.Employees**
  - **DbSet¡Salary¿.Salaries**
  - **DbSet¡Department¿.Departments**
  - **DbSet¡Auth¿.Auths**

5 Routing

Routing defines how HTTP requests are mapped to controller actions.

5.1 Routes

5.1.1 Employee Routes

- **/Employee/ListEmployee**: Maps to EmployeeController.ListEmployees
- **/Employee/EditEmployee/{id}**: Maps to EmployeeController.EditEmployee
- **/Employee/AddEmployee**: Maps to EmployeeController.AddEmployee
- **/Employee/DeleteEmployee/{id}**: Maps to EmployeeController.DeleteEmployee

5.1.2 Salary Routes

- **/Salary/ListSalary**: Maps to SalaryController.ListSalary
- **/Salary/EditSalary/{id}**: Maps to SalaryController.EditSalary
- **/Salary/EditSalary1/{id}**:Maps to SalaryController.EditSalary1

5.1.3 Department Routes

- **/Dept/ListDept**: Maps to DepartmentController.ListDepartments
- **/Dept/EditDept/{id}**: Maps to DepartmentController.EditDepartment
- **/Dept/AddDept**: Maps to DepartmentController.AddDepartment
- **/Dept/DeleteDept/{id}**: Maps to DepartmentController.DeleteDepartment

5.1.4 Authentication Routes

- **/auth/login**: Maps to AuthController.Login
- **/auth/logout**: Maps to AuthController.Logout
- **/auth/register**: Maps to AuthController.Register

6 Interaction Flow

6.1 List Employees

1. User navigates to **/Employee/ListEmployee**.
2. **EmployeeController.ListEmployees** is invoked.
3. The method fetches employee data from **ApplicationDbContext**.
4. The data is passed to **ListEmployees.cshtml** for rendering.

6.2 Edit Employee

1. User navigates to **/Employee/EditEmployee/{id}**.
2. **EmployeeController.EditEmployee** is invoked with the employee ID.
3. The method fetches the employee data from **ApplicationDbContext**.
4. The data is passed to **EditEmployee.cshtml** for rendering the form.
5. User submits the form, and the data is updated in the database.

6.3 Create Employee

1. User navigates to **/Employee/AddEmployee**.
2. **EmployeeController.AddEmployee** is invoked.
3. The method displays the form for creating a new employee.
4. User submits the form, and the data is saved in the database.

6.4 Delete Employee

1. User navigates to **/Employee/DeleteEmployee/{id}**.
2. **EmployeeController.DeleteEmployee** is invoked with the employee ID.
3. The method displays a confirmation page.
4. User confirms, and the employee is deleted from the database.

6.5 List Salaries

1. User navigates to **/Salary/ListSalary**.
2. **SalaryController.ListSalary** is invoked.
3. The method fetches salary data from **ApplicationDbContext**.
4. The data is passed to **ListSalary.cshtml** for rendering.

6.6 Edit Salary

1. User navigates to **/Salary/EditSalary/{id}**.
2. **SalaryController.EditSalary** is invoked with the salary ID.
3. The method fetches the salary data from **ApplicationDbContext**.
4. The data is passed to **EditSalary.cshtml** for rendering the form.
5. User submits the form, and the data is updated in the database.

6.7 List Departments

1. User navigates to **/Dept/ListDept**.
2. **DepartmentController.ListDepartments** is invoked.
3. The method fetches department data from **ApplicationDbContext**.
4. The data is passed to **ListDepartments.cshtml** for rendering.

6.8 Edit Department

1. User navigates to **/Dept/EditDept/{id}**.
2. **DepartmentController.EditDepartment** is invoked with the department ID.
3. The method fetches the department data from **ApplicationDbContext**.
4. The data is passed to **EditDepartment.cshtml** for rendering the form.
5. User submits the form, and the data is updated in the database.

6.9 Create Department

1. User navigates to **/Dept/AddDept**.
2. **DepartmentController.AddDepartment** is invoked.
3. The method displays the form for creating a new department.
4. User submits the form, and the data is saved in the database.

6.10 Delete Department

1. User navigates to **/Dept/DeleteDept/{id}**.
2. **DepartmentController.DeleteDepartment** is invoked with the department ID.
3. The method displays a confirmation page.
4. User confirms, and the department is deleted from the database.

6.11 Login

1. User navigates to **/auth/login**.
2. **AuthController.Login** is invoked.
3. The method displays the login form.
4. User submits the form, and the method authenticates the user.
5. If successful, the user is redirected to the home page.

6.12 Logout

1. User navigates to **/auth/logout**.
2. **AuthController.Logout** is invoked.
3. The method ends the user's session and redirects to the login page.

6.13 Register

1. User navigates to **/auth/register**.
2. **AuthController.Register** is invoked.
3. The method displays the registration form.
4. User submits the form, and the method registers the new user.
5. If successful, the user is redirected to the login page.

7 Paging and Sorting

7.1 Pager Component

The Pager component is responsible for handling pagination in the application. It helps in dividing large sets of data into smaller, manageable pages.

7.1.1 Pager Class

The Pager class encapsulates the logic for pagination, including calculating the total number of pages, the current page, and the range of items to display.

Properties

- **TotalItems**: The total number of items to paginate.
- **CurrentPage**: The current page number.
- **PageSize**: The number of items per page.
- **TotalPages**: The total number of pages.
- **StartPage**: The starting page number in the pagination control.
- **EndPage**: The ending page number in the pagination control.

Methods

- **Pager(int totalItems, int? page, int pageSize = 10)**: Constructor to initialize the Pager with total items, current page, and page size.

7.2 Sorting

Sorting is used to arrange data in a specific order, either ascending or descending, based on a chosen criterion.

7.2.1 Sorting Implementation

Sorting functionality allows users to view data in a desired order, improving data navigation and usability.

Properties

- **SortBy**: The field by which the data should be sorted.

7.3 Usage in Controllers

7.3.1 ListEmployees

- **Paging**: The ListEmployees action method uses the Pager component to handle pagination of employee data.
- **Sorting**: The ListEmployees action method applies sorting to the employee data based on user-selected criteria.

7.3.2 ListSalary

- **Paging**: The ListSalary action method uses the Pager component to handle pagination of salary data.
- **Sorting**: The ListSalary action method applies sorting to the salary data based on user-selected criteria.

7.3.3 ListDepartments

- **Paging**: The ListDepartments action method uses the Pager component to handle pagination of department data.
- **Sorting**: The ListDepartments action method applies sorting to the department data based on user-selected criteria.