**A**

**Training Report**

**On**

EMPLOYEE MANAGEMENT SYSTEM

Submitted in partial fulfilment of the requirement for the award of degree  
 Of B.tech

**Bachelor of Technology**

**Computer Science and Engineering**

**LOVELY PROFESSIONAL UNIVERSITY**

**PHAGWARA, PUNJAB**

****

**Duration:-10-Jan-2023 to 28-May-2023**

**SUBMITTED BY**

Name of the student: Rishikesh Singh

Registration Number: 11902664

Signature of the student: Rishikesh

**Student Declaration**

**To whom so ever it may concern**

**I**, Rishikesh singh, 11902664, hereby declare that the work done by me on "Employee management system" from 15 Feb-2023 to present, under the supervision of Komal Arora Lovely Professional University Phagwara, Punjab, is a record of original work for the partial fulfilment of the requirements for the award of the degree Computer Science and Engineering.

Name of the Student (Registration Number)

Rishikesh singh (11902664)

Signature of the student Dated: 11/05/2023

**Preface**

I have made this report file on the topic; **EMPLOYEE MANAGEMENT SYSTEM;**I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and every one has ended on a successful note. I express my sincere gratitude to **Komal arora** who assisting me throughout the preparation of this topic. I thank her for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***ABSTRACT***

**Employee Management System** is a distributed application, developed to maintain the details of employees working in any organization. It maintains the information about the personal details of their employees, also the details about the payroll system which enable to generate the payslip. The application is actually a suite of applications developed using SpringBoot.

It is simple to understand and can be used by anyone who is not even familiar with simple  
employees system. It is user friendly and just asks the user to follow step by step operations by giving him few options. It is fast and can perform many operations of a company.

This software package has been developed using the powerful coding tools of Springboot at Front End and Microsoft Sql Server at Back End. The software is very user friendly. The package contains different modules like Employee details. This version of the software has multi-user approach. For further enhancement or development of the package, user’s feedback will be considered.

**Employee management system**

**Chapter 1: 6-8**

* Introduction, Background, Statement of Problem, Purpose, General Objective, Specific Objective Description of the Existing System, Major Function of the Current System, Users of the Current System, Drawbacks of Current System, Acronym

**Chapter 2: 9-22**

* Proposed System Overview, Functional Requirements, Non-Functional Requirements, Performance Requirements, System Model (Scenarios, Use Case Model, Object Model, Data Dictionary, Class Diagram, Dynamic Model, Sequence Diagram, Activity Diagram, State Chart Diagram, User Interface)

**Chapter 3: 23-25**

* System Design Overview, Purpose of the System, Design Goal, Proposed System Architecture (System Process, Subsystem Decomposition, Hardware Software Mapping, Persistent Data Management, Component Diagram, Deployment Diagram, Boundary Condition, Database Design, Access Control)

**Chapter 4: 26-40**

* Implementation Overview, Tool and Technology Utilized During System Development, Implementation Detail

**Chapter 5: 41**

* Conclusion

**..-\_-..**

**Employee management system**

# Chapter -1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Introduction

Every firm maintains a record of its employees. Employee records are essential for managing employees. These records are necessary for any firm in order to manage the workforce, compute pay, and assess employee performance. The HR team must manage all of these records, which is a difficult task and time-consuming procedure that can be sped up by adopting the EMS, or employee management system. Any organization's success depends on its human resources, which are an essential component. Organisations make significant investments in staff management. A human resource information system, or HRIS, controls accounting and inventory. EMS is an informatics tool that helps owners, HRs, and managers save time, energy, and money. All firms, including public and private , require employee management systems. However, they have been keeping records using the antiquated, traditional method of pen and paper for years. Recently, however, there has been a significant increase in the use of automatic systems that can handle salary calculation on their own.

Because these systems are expensive and need on-going maintenance, they are challenging to implement in many places. I therefore have developed EMS, which is built on a mobile application, to address this issue. Each employee's wage and daily attendance will be computed. This technology was created so that smaller organisations may also use it. It helps small-scale industries manage their staff and is less expensive than other systems. This solution attempts to address this problem so that the HR staff can concentrate on other processes rather than resolving disputes because there are many disagreements in small industries owing to erroneous calculation of pay and overtime. An organisation benefits from streamlining the record-keeping procedure. Organisations must apply this personnel management approach to increase the effectiveness of their job. This essay explains the system's development process, issues encountered, and advantages of using it.

## Background

There is a system that has been used in the past that addressed the employee management system to the necessary employee by writing on a piece of paper for permission, registration, attendance, and shortly. The world is growing more technological in modern times, and information technology is one of those technologies that is driving the way. Nowadays, a computer is available everywhere. We have now put forth a system that is entirely computer-controlled and enables us to submit employee data. The planned system will offer beneficial chances for low-cost, quick-duration, and highly visible commutes.

## Purpose

The main goals of this application are registering new employees, and ensuring that workers complete their tasks on time. The administrator of this application establishes branches and designates a Branch Manager for each branch. The branch manager will add staff to his branch and give them duties to do. The primary user of this online application is the administrator, and the branch manager will maintain employee records.

**1. The project's goal**

**1.1. Overarching goal**

* To switch from a manual to a computerised approach for managing employees.
* Using a computerised management system for simple and efficient employee management.

**1.2.** **Clearly defined goal**

* To handle the tasks of the employee in an effective manner
* To easily register new employees
* To conveniently retrieve or search the employee information.
* To attend to the employee whether or not he or she is absent.
* To erase the information of the employee who leaves the job.
* Eliminating the personnel management system's tedious and worn-out working methodology. Preventing various faults when using a manual approach.
* Compared to the previous system, the new one makes managing human resources easier.
* The new system is secure and only authorised individuals are able to access it.

### Technical Feasibility

I acquired knowledge of programming languages like Springboot, CSS, HTML, MySQL, and thymeleaf that were necessary for the project's effective completion.

## Scope and limitation

The authorised user can access the web-based computerised staff management system from any location within an operating system.

The system, which is intended to run on the organisation, would be centrally governed and regulated.

## Operating environment

* Hardware and software tools
* Software tools
* Client side tools

Table1.1. Client side tools

|  |  |  |
| --- | --- | --- |
| Tool | Version and abbreviation | Used for |
| Hypertext markup language | Html4/html5 | For configuration to develop front end |
| Cascading style sheet | Css/css3 | For layout design, content decoration in user interface design |
| Spring | Spring Boot 2.0 | Java-based frameworks to create micro services and web apps. |

Table1.2. Server side tools

|  |  |  |
| --- | --- | --- |
| Tool | Version and abbreviation | Used for |
| Hypertext pre-processor | Html, CSS | Front end And programming language, |
| MySQL  server | MySQL 8.1 | Backend or project database |

# **Chapter 2: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PROPOSED SYSTEM**

The suggested system is made to take the position of the manual employee management system with a computerised system. It is also made to save employee information in a database to lessen the issues that the manual system has.

The programme creates several forms to input employee data into the database and to extract necessary employee data from the database.

**Functional necessity**

Functions or activities that the system must provide are referred to as functional requirements.

The following functional criteria are part of EMS.

* Registration of new employees.
* Check out the personnel details.
* Refresh the employee details.
* Sign in and out.

## 

## Non-functional requirement

The non-functional requirement lists restrictions on how the project can be implemented. Among these are the requirements that the central server be placed in a safe location, the system be extensible and maintained, the network architecture be a private network, and the installation of client devices at each user's end. Each user should have access to technical support personnel. Additionally, the user should be familiar with the fundamentals of computers, and employees should receive training on the management application's demo version. The system meets the following non-functional criteria.

 **Reliable**

The system must be capable of handling mistakes when they do arise.

** Usability**

The system's menu and available options should be clear to the users. The system must have an intuitive user interface so that users do not strain when interacting with it.

** Performance**

The system need to react quickly. Mostly depends on the user's device's processor and connection speed.

 **Accessibility & Availability**

The system must be operational at all times.

### Use case model

* **Actor identification**
* Manager/administrator
* branch Manager
* Employee
* **Use case identification**
* Registration
* Delete record
* Update record
* View information
* View employee detail
* View policy
* View their detail
* **Use case description**
* **Registration**

|  |  |
| --- | --- |
| U ID: | 01 |
| Use case Name: | Registration |
| Actors: | Employee, branch manager, manager |
| Pre-condition | The employee must be well informed on who they are. entire information on him/her, etc. |
| Post condition | The database must have a record of the registered employee. and sends a message that was successfully registered. |
| Main flow | 1. A worker or management accesses the system  2. The sign-up page that was shown on the system  3. After the user selects something from the menu, the registration form is shown.  4. Completely fill out the employee's or his/her details.   1. Click register button |

* **Login**

|  |  |
| --- | --- |
| U ID: | 02 |
| Use case Name: | Login |
| Actors: | Employee, manager, branch manager |
| Pre-condition | The user must be registered in the university first. |
| Post condition | The user logged successfully in to their respective page. |
| Main flow | 1. A worker or management accesses the system  2. The index page features a login form.  3. The proper login and password are entered by the employee and management.  4. Press the login button.  5. The message "Logged in successfully" is shown |
|  |  |
| Exceptional condition | * If the user does not provide the right username and password, the system displays a message asking them to do so. * If the employee or management registers first, the system displays a message asking them to do so. * When there is no connectivity, the system is inoperative. |

* **Update record**

|  |  |
| --- | --- |
| U ID: | 03 |
| Use case Name: | Update employee record |
| Actors: | Branch manager |
| Post condition | The revised record is successfully recorded in the database by the management. |
| Main flow | 1. The manager opened the system and logged in.  2. The system's update menu was visible.  3. After the user clicks the menu, the form is shown.  4. The manager completes the update form by entering the data.  5. Press the update button.  6. The message "Detail updated successfully" is shown. |
| Exceptional flow | 1. Error message will be displayed if the manager fills out the form incorrectly.  2. If the manager does not complete the form, the system shows a notification asking them to do so.  3. When there is no connectivity, the system is inoperable. |

* **Delete record**

|  |  |
| --- | --- |
| U ID: | 04 |
| Use case Name: | Delete employee record. |
| Actors: | Branch Manager |
| Pre-condition | A prior registration is required for the information that will be removed. |
| Post condition | The manager deletes the record successfully. |
| Main flow | 1. The manager opened the system and logged in.  2. The management page's delete menu was visible.  3. After the user clicks the menu, the form is shown.  4. Type in the worker's number.  5. Press the erase button.  6. A successful record deletion notification appears. |
| Exceptional flow | * The system shows a notification saying that the employee is not registered if they do not enter their own employee number. * When there is no connectivity, the system is inoperable. |

### Data Dictionary

User is the central object in the EMS module. Users have the following information

* Identification information includes Employee name, Mobile number, address, date of birth and Joining date.
* Contact information includes e-mail address, cellular phone.
* User authentication information includes user ID, username, and password.

**Employee**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field name | Data type | Field size | Constraint | Description |
| EmployeeName | Char | 50 | Not null | Full name of the employee |
| Employee No | Int | 50 | Not null | The identification name of employee |
| Username | Varchar | 50 | Primary key | Username the employee |
| Password | Varchar | 50 | Not null | Password of the employee |
| Phone Number | Int | 10 | Not null | Mobile or home phone number of the employee |
| Address | Char | 30 | Not null | Address of the address of the employee |
| Date Of Birth | Date |  | Not null | Birth date of the employee |
| Sex | Varchar | 10 | Not null | Gender of employee |
| Age | Int |  | Not null | Age of employee |
| Date Of join | Date |  | Not null | The employee join date |
| Department | Varchar | 50 | Not null | The department of the employee h  e/she works |

**Table data dictionary for signup/registration/create account**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field name | Data type | Field size | Constraint | Description |
| Email | Varchar | 50 | Primary key | Email of Manager |
| Password | Varchar | 50 | Not null | Password of Manager |

### Class Diagram

The primary level of object-oriented analysis and design is the class diagram. The class model depicts the link between the class's operation, characteristics, aggregation and association, and inheritance. It demonstrates how relationships and static structure interact. Class, object, operation, and dependencies are the primary operations that are displayed on a class diagram.



## Dynamic model

Sequence diagrams, state chart diagrams, and activity diagrams are used to depict dynamic models; thus, in the prior part, we briefly discussed these diagrams for a few use cases.

#### Fig 01 .Sequence diagram for registration



#### Fig 02: Sequence diagram for login



*Fig 03: Sequence diagram for update record*



#### Fig 05. Sequence diagram for delete record



***Fig: activity diagram for registration*** *Fig: Activity diagram for login*



*Fig: State chart diagram for registration Fig: State chart record*

**Login Page**

**About page**

**Web Search**

**Home Page**



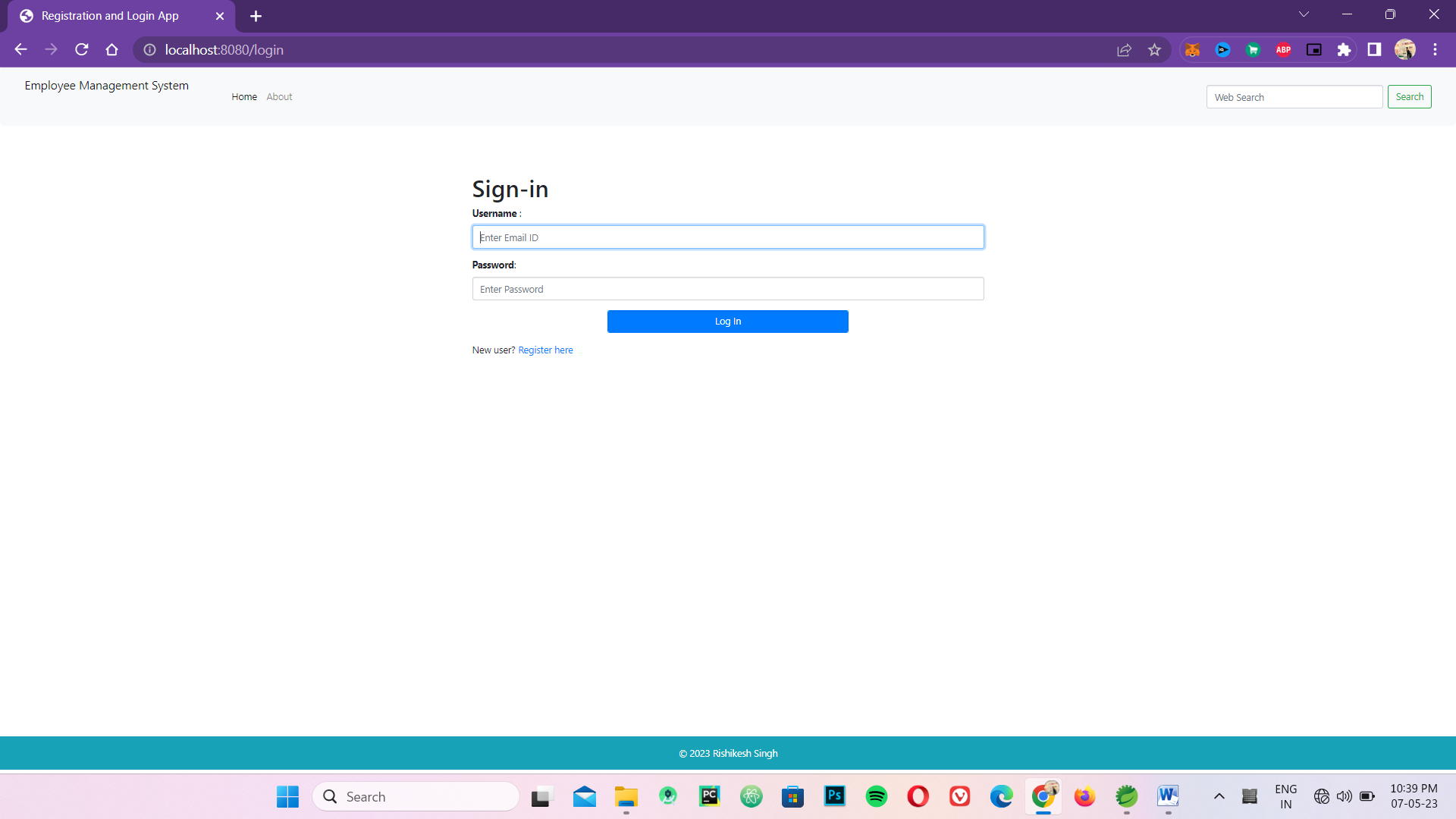
**Login Button**

**login**

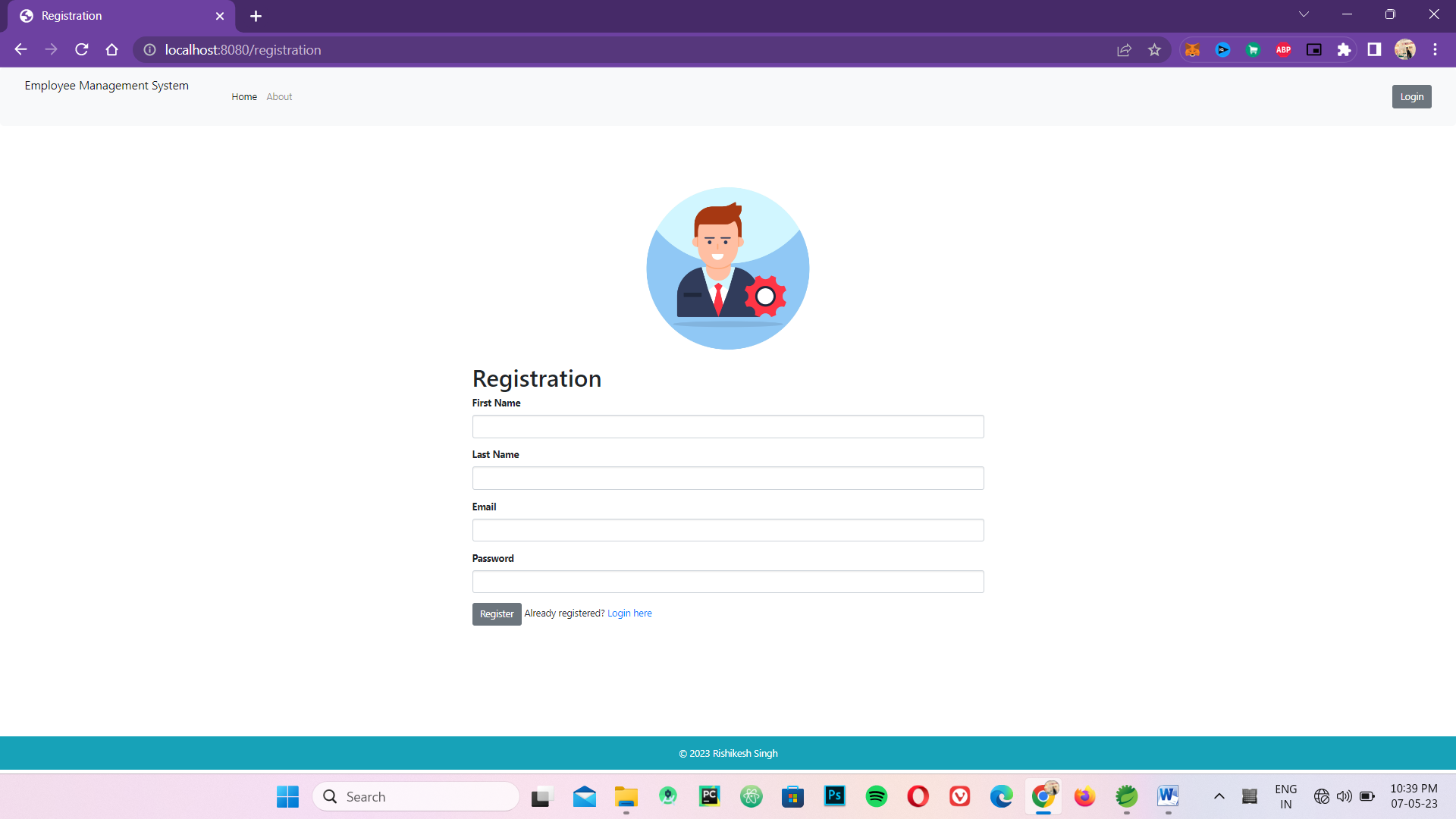
**Register**

**Password**

**Username**



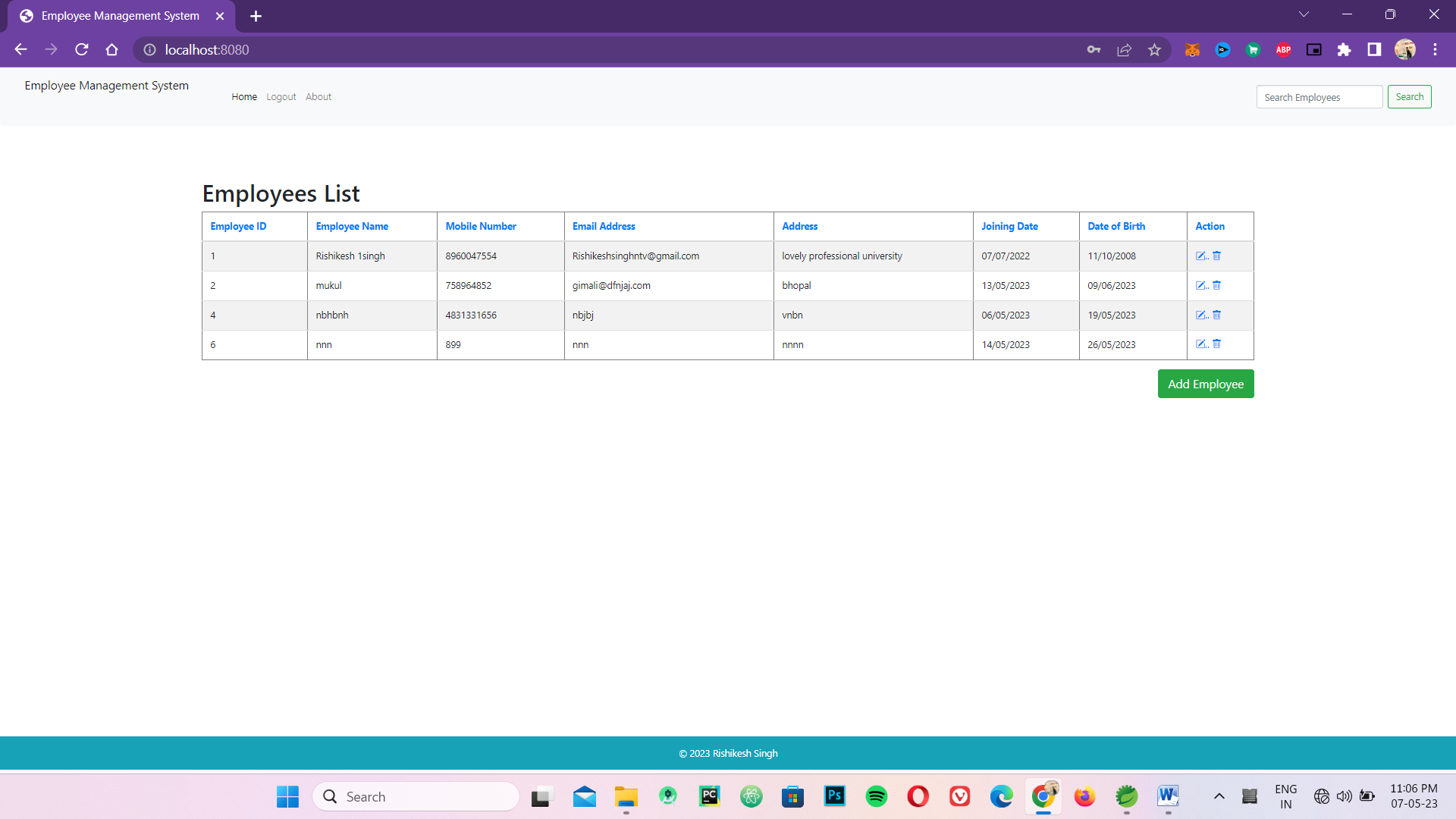
**Login**



**Registration Page**



**About Page**



# Chapter - 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 

# System design

The process of developing software to address one or more sets of issues is known as software design. The software requirements analysis (SRA) is one of the crucial components of software design. Lists of specifications that are used in software engineering are an element of the software development process. This is the EMS system design. The suggested system design and object design are included in the module. In this chapter, we'll try to take into account things like establishing the design aim, subsystem decomposition and identification, persistent data storage, hardware/software mapping, access control, and boundary conditions in EMS.

## Purpose of the system

The design focuses on the capabilities, and depending on the context in which the solution will be hosted, there may be several solutions for the same problem. The architectural system of the proposed systems is the primary focus of the software system design.It also outlines the fundamental choices taken while designing the architectural system, ensuring that users and system designers can comprehend the goal using a technique known as visual representation or abstraction. The system's ease of implementation is also advantageous.

### Design Goal

The goal of software design is dealing with the logical organization of the software which in turn affects the non-functional requirements. These requirements include:

 Performance

 Dependability

 Maintainability

 End user

#### Performance

The programme completes its tasks at a time that the user finds acceptable. The programme doesn't use a lot of memory.

* Response time: Depending on the strength of the network that is available, the should reply quickly.
* Storage space: It requires a CPU with at least 2GB of RAM and at least 20MB of HD storage to perform tasks effectively.

#### Dependability

The following capabilities are necessary for EMS to perform properly:

* **Robustness**: The software must be able to function under pressure or accept inconsistent or invalid input. It may, for instance, be built to be resilient to low memory situations.
* **Availability**: The system is available around-the-clock if a connection is accessible and the power is on since it is scheduled with time and throughout the period that it is scheduled.
* **Security**: The programme can withstand malicious influences and actions. Without entering their own username and password, the user is unable to do any actions, and one employee cannot view another employee's details.

#### Maintainability

A measure of how straightforward it is to make functional or bug changes.

* **Extensibility**: New features may be added to the software without significantly altering the architecture below.
* **Modifiability**: The system allows for the addition or modification of new features or functionalities.
* **Reusability**: The programme may be modified or added to to offer new features with little to no change.
* **Portability**: The system is designed to work with any web browser, independent of the platform or version that it is running on.
* **Readability**: The system code may be read by selecting "view the source code" from the menu that appears when you click on the current web page.

### System process



**At http://localhost:8080/**

## Integration testing

If they all function properly separately, they ought to function properly when together. Obviously, combining them is difficult. "This can be done in two ways:

**Top down integration:** Modules are included into the structure in one of two ways: depth first or breadth first. This is done by working downward down the control hierarchy, starting with the primary control module.

**Bottom-up integration**: starts with building and testing atomic modules, or modules at the base of the program's hierarchy. Modules are integrated from the bottom up, hence there is no need for stubs because processing needed for modules subordinate to a particular level is always accessible.

* **Testing includes**
* **Verification** and **Validation**
* **Verification: -**is a process of confirming that software meets its specification.
* **Validation:** - is the process of confirming that software meets the customer’s requirements.

# Chapter- 4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Implementation**

## Overview

Since creating a functional software product is the primary goal of software development, implementation is the last and most crucial phase of the Software Development Life Cycle (SDLC). Since the other phases are meant to aid the implementation phase, the entire effort will be ineffective without it. Implementing interfaces and, most significantly, code are included.

## Tool and technology utilized during System development

For the creation of the online ordering system, we make use of a variety of tools and technology. These tools are mostly used for creating websites. Although standalone application development tools exist, they are not widely used. To improve technical capabilities, we employ the most recent version of these development tools. The system will be created using the following important software development tools: - WAMP server, which includes MYSQL, HTML, CSS, and Spring Boot, is the primary development tool for our system because it will be a web-based system.

## . Implementation detail

**This code is for a Spring Boot application**

Package:-  ***net.javaguides.springboot***

**Module 1:-class : ( EmployeeManagementWebApplication.jave)**

* This code initializes a Spring Boot application and starts the Spring application context, which allows you to build and deploy a web application using Spring Boot.

**package** net.javaguides.springboot;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** EmployeeManagementWebappApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EmployeeManagementWebappApplication.**class**, args);

}

}

Package:- **net.javaguides.springboot.config**

**Module 2:-Class ; (SecurityConfiguration.java)**

* This code is for a class named “**Security Configuration”** that is used to configure Spring Security in a web application.

The @Autowired annotation is used for automatic dependency injection of the UserService bean, which provides user-related services such as authentication and authorization.

@Configuration

EnableWebSecurity

//marker annotation

**public** **class** SecurityConfiguration {

@Autowired // enables automatic dependency injection

// can autowire relationships between collaborating beans

**private** UserService userService;

@Bean

**public** SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

http.csrf().disable().((authorize) -> authorize.requestMatchers("/registration\*\*").permitAll()

.requestMatchers("/js/\*\*").permitAll().requestMatchers("/css/\*\*").permitAll().requestMatchers("/img/\*\*")

.permitAll().requestMatchers("/").permitAll().requestMatchers("/showNewEmployeeForm").permitAll()

.requestMatchers("/saveEmployee").permitAll().requestMatchers("/deleteEmployee/{id}").permitAll()

.requestMatchers("/showFormForUpdate/{id}").permitAll().requestMatchers("/page/{pageNo}").permitAll()

.requestMatchers("/About").permitAll().requestMatchers("/Home").permitAll()

).formLogin(form -> form.loginPage("/login").permitAll())

.logout(logout -> logout.logoutRequestMatcher(**new** AntPathRequestMatcher("/logout")).permitAll());

**return** http.build();

}

@Autowired

**public** **void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(userService).passwordEncoder(*passwordEncoder*());

}

}

Package:-  ***net.javaguides.springboot.controller***

**Module 3:- Class :- (EmployeeController.java)**

* This code is for a Spring MVC controller named ***EmployeeController***. It contains several methods with different mappings to handle HTTP requests.

@Controller // which marks it as a Spring MVC controller.

**public** **class** EmployeeController {

@Autowired // which indicates that Spring should inject an instance of the

// "EmployeeService" interface into the controller.

**private** EmployeeService employeeService;

// display list of employees

@GetMapping("/")

**public** String viewHomePage(Model model) {

model.addAttribute("listEmployees", employeeService.getAllEmployees());

**return** "index";

}

**public** **static** String *uploadDirectory* = System.*getProperty*("user.dir") + "/src/main/templates/imagedata";

@GetMapping("/showNewEmployeeForm")

**public** String showNewEmployeeForm(Model model) {

// create model attribute to bind form data

Employee employee = **new** Employee();

model.addAttribute("employee", employee);

**return** "new\_employee";

}

@PostMapping("/saveEmployee")

**public** String saveEmployee(@ModelAttribute("employee") Employee employee) {

// save employee to database

employeeService.saveEmployee(employee);

**return** "redirect:/";

}

@GetMapping("/showFormForUpdate/{id}")

**public** String showFormForUpdate(@PathVariable **long** id, Model model) {

// get employee from the service

Employee employee = employeeService.getEmployeeById(id);

// set employee as a model attribute to pre-populate the form

model.addAttribute("employee", employee);

**return** "update\_employee";

}

@GetMapping("/deleteEmployee/{id}")

**public** String deleteEmployee(@PathVariable(value = "id") **long** id) {

// call delete employee method

**this**.employeeService.deleteEmployeeById(id);

**return** "redirect:/";

}

@GetMapping("/page/{pageNo}")

**public** String findPaginated(@PathVariable(value = "pageNo") **int** pageNo, @RequestParam("sortField") String sortField,

@RequestParam("sortDir") String sortDir, Model model) {

**int** pageSize = 5;

Page<Employee> page = employeeService.findPaginated(pageNo, pageSize, sortField, sortDir);

List<Employee> listEmployees = page.getContent();

List<Employee> employees = employeeService.search(keyword);

model.addAttribute("employees", employees);

**return** "redirect:/";

}

}

model.addAttribute("currentPage", pageNo);

model.addAttribute("totalPages", page.getTotalPages());

model.addAttribute("totalItems", page.getTotalElements());

model.addAttribute("sortField", sortField);

model.addAttribute("sortDir", sortDir);

model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" : "asc");

model.addAttribute("listEmployees", listEmployees);

**return** "index";

}

// Search option

@GetMapping("/search")

**public** String search(Model model, @RequestParam("keyword") String keyword) {

// Search option

@GetMapping("/search")

**public** String search(Model model, @RequestParam("keyword") String keyword) {

List<Employee> employees = employeeService.search(keyword);

model.addAttribute("employees", employees);

**return** "redirect:/";

}

}

The controller is using Spring's @Autowired annotation to inject an instance of the EmployeeService interface into the controller. This interface provides methods for performing CRUD (Create, Read, Update, Delete) operations on the employee database.

**Module 4:- Class :- (*MainController.java)***

* This code defines a Spring MVC controller named ***MainController***, which has three handler methods**.**

The handler method is mapped to the HTTP GET request for the path "/login",” /aboutpage”, “/homepage”.

@Controller // which marks it as a Spring MVC controller.

**public** **class** MainController {

@GetMapping("/login") //for mapping HTTP GET requests onto specific handler method.

**public** String login() {

**return** "login";

}

@Controller

**public** **class** AboutController {

@GetMapping("/About")

**public** String aboutPage() {

**return** "About";

}

}

@Controller

**public** **class** HomeController {

@GetMapping("/Home")

**public** String aboutPage() {

**return** "Home";

}

}

}

**Module 5:- Class:- (UserRegistrationController.java)**

@Controller

@RequestMapping("/registration")

**public** **class** UserRegistrationController {

**private** UserService userService;

**public** UserRegistrationController(UserService userService) {

**super**();

**this**.userService = userService;

}

@ModelAttribute("user")

**public** UserRegistrationDto userRegistrationDto() {

**return** **new** UserRegistrationDto();

}

@GetMapping

**public** String showRegistrationForm() {

**return** "registration";

}

@PostMapping

**public** String registerUserAccount(@ModelAttribute("user") UserRegistrationDto registrationDto) {

userService.save(registrationDto);

**return** "redirect:/registration?success";

}

}

This is a Spring MVC controller for handling user registration. The **@Controller** annotation marks the class as a Spring MVC controller, and the **@RequestMapping** annotation specifies the base URL mapping for the controller. In this case, the base URL mapping is "/registration".

Package:-  ***net.javaguides.springboot.Dto***

**Module 6:-Class :- (*UserRegistrationDto.java)***

This code defines a Java class called “***UserRegistrationDto”*** that represents a Data Transfer Object (DTO) for user registration information. The class contains private fields for the user's first name, last name, email, password, address, date of birth, and date of joining. The class also contains public getters and setters for each of these fields, as well as two constructors: a default constructor and a parameterized constructor that takes in values for all of the fields. The class is located in the package “***net.javaguides.springboot.dto”.***

**public** **class** UserRegistrationDto {

**private** Long id;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

**private** String address;

**private** LocalDate dateOfBirth;

**private** LocalDate joiningDate;

**public** UserRegistrationDto() {

}

**public** UserRegistrationDto(Long id, String firstName, String lastName, String email, String password, String address, LocalDate dateOfBirth, LocalDate joiningDate) {

**this**.id=id;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

**this**.password = password;

**this**.address = address;

**this**.dateOfBirth = dateOfBirth;

**this**.joiningDate = joiningDate;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;}

**public** String getFirstName() {

**return** firstName;}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;}

**public** String getLastName() {

**return** lastName;}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** LocalDate getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(LocalDate dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

**public** **void** setDateOfJoining(LocalDate joiningDate) {

**this**.joiningDate = joiningDate;

}

}

}

**public** LocalDate getDateOfJoining() {

**return** joiningDate;

}

Package:-  ***net.javaguides.springboot.model***

**Module 7:- Class- (Employee.java)**

This code defines a Java class named Employee which represents an employee entity in a Spring Boot application. It uses the JPA (Java Persistence API) annotations to map the class fields to corresponding columns in a relational database table.

@Entity

@Table(name = "employees")

**public** **class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **long** id;

@Column(name = "first\_name")

**private** String firstName;

@Column(name = "last\_name")

**private** String lastName;

@Column(name = "email")

**private** String email;

@Column(name = "address")

**private** String address;

@Column(name = "joining\_date")

**private** LocalDate joiningDate;

@Column(name = "date\_of\_birth")

**private** LocalDate dateOfBirth;

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** LocalDate getJoiningDate() {

**return** joiningDate;

}

**public** **void** setJoiningDate(LocalDate joiningDate) {

**this**.joiningDate = joiningDate;

}

**public** LocalDate getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(LocalDate dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", email=" + email

+ ", address=" + address + ", joiningDate=" + joiningDate + ", dateOfBirth=" + dateOfBirth + "]";

}

}

The class has the following fields:

**id**: a long integer that represents the unique identifier of the employee.

**firstName**: a string that represents the first name of the employee.

**lastName**: a string that represents the last name of the employee.

**email**: a string that represents the email address of the employee.

**address**: a string that represents the address of the employee.

**joiningDate**: a LocalDate object that represents the date on which the employee joined the organization.

**dateOfBirth**: a LocalDate object that represents the date of birth of the employee.

**Module 8:-Class :- (Role.java)**

This code defines a JPA entity class Role. It is mapped to a database table named role. The ***@Entity*** annotation specifies that this is a JPA entity, and the **@Table** annotation specifies the name of the database table.

The **@GeneratedValue** annotation with **GenerationType.IDENTITY** strategy is used to indicate that the ID field will be automatically generated by the database.

@Entity

@Table(name = "role")

**public** **class** Role {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

**public** Role() {

}

**public** Role(String name) {

**super**();

**this**.name = name;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**Module 9:- Class:- (User.java)**

This is a Java class called User in the package **net.javaguides.springboot.model**. The class is annotated with the **@Entity** annotation which indicates that this class is a JPA entity and it will be managed by the entity manager.

The class has several fields annotated with various annotations such as **@Id**, and **@Column**.

The **@*Id*** annotation is used to specify the primary key of the entity.

The **@*Column*** is used to specify the properties of the column mapped to the field.

@Entity

@Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames = "email"))

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

@Column(name = "first\_name")

**private** String firstName;

@Column(name = "last\_name")

**private** String lastName;

**private** String email;

**private** String password;

**private** LocalDate joining;

**private** String address;

**private** LocalDate dateOfBirth;

@ManyToMany(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***)

@JoinTable(

name = "users\_roles",

joinColumns = @JoinColumn(

name = "user\_id", referencedColumnName = "id"),

inverseJoinColumns = @JoinColumn(

name = "role\_id", referencedColumnName = "id"))

**private** Collection<Role> roles;

**public** User() {

}

**public** User(String firstName, String lastName, String email, String password, LocalDate dateOfBirth, String address, Collection<Role> roles) {

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

**this**.password = password;

**this**.dateOfBirth = dateOfBirth;

**this**.address = address;

**this**.roles = roles;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** LocalDate getJoining() {

**return** joining;

}

**public** **void** setJoining(LocalDate joining) {

**this**.joining = joining;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

}

Package:-  ***net.javaguides.springboot.repository***

**Module 10:- Class :-(EmployeeRepository.java)**

This code defines an interface ***EmployeeRepository*** in the ***net.javaguides.springboot.repository*** package that extends the ***JpaRepository*** interface with the **Employee** entity class and the primary key data type ***Long***.

By extending ***JpaRepository***, the **EmployeeRepository** interface inherits several methods for performing basic CRUD (Create, Read, Update, Delete) operations on the Employee entity such as ***save(),*** ***findAll(), findById(), deleteById(),*** etc.

The **EmployeeRepository** interface doesn't have any additional methods defined beyond the inherited ones. However, it's possible to define custom methods that use JPA query methods or custom JPQL queries to perform more complex queries on the database.

**package** net.javaguides.springboot.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** net.javaguides.springboot.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

}

Package:-  ***net.javaguides.springboot.service***

**Module 11:- Class:-(EmployeeService.java)**

**public** **interface** EmployeeService {

List<Employee> getAllEmployees();

**void** saveEmployee(Employee employee);

Employee getEmployeeById(**long** id);

**void** deleteEmployeeById(**long** id);

Page<Employee> findPaginated(**int** pageNo, **int** pageSize, String sortField, String sortDirection);

**public** List<Employee> getAllEmployees(String keyword);

List<Employee> search(String keyword);

}

This code defines the interface for the ***EmployeeService.***

**Module 12:- Class:- (EmployeeServiceImpl.java)**

This code is a service implementation for a Spring Boot application with CRUD functionality for an Employee model. It includes methods for getting all employees, saving an employee, getting an employee by ID, deleting an employee by ID, and finding employees with pagination and search functionality**.**

The code starts with importing necessary dependencies and defining a service class named **EmployeeServiceImpl**. The service class implements the **EmployeeService** interface, which contains method signatures for CRUD functionality.

@Service

**public** **class** EmployeeServiceImpl **implements** EmployeeService {

@Autowired

**private** EmployeeRepository employeeRepository;

@Override

**public** List<Employee> getAllEmployees() {

List<Employee> temp = employeeRepository.findAll();

System.***out***.println(temp);

**return** temp;

}

@Override

**public** **void** saveEmployee(Employee employee) {

**this**.employeeRepository.save(employee);

}

@Override

**public** Employee getEmployeeById(**long** id) {

Optional<Employee> optional = employeeRepository.findById(id);

Employee employee = **null**;

**if** (optional.isPresent()) {

employee = optional.get();

} **else** {

**throw** **new** RuntimeException(" Employee not found for id :: " + id);

}

**return** employee;

}

@Override

**public** **void** deleteEmployeeById(**long** id) {

**this**.employeeRepository.deleteById(id);

}

@Override

**public** Page<Employee> findPaginated(**int** pageNo, **int** pageSize, String sortField, String sortDirection) {

Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.***ASC***.name()) ? Sort.*by*(sortField).ascending() :

Sort.*by*(sortField).descending();

Pageable pageable = PageRequest.*of*(pageNo - 1, pageSize, sort);

**return** **this**.employeeRepository.findAll(pageable);

}

**Module 13:-class:- (UserService.java)**

This code defines an interface for a service class that handles user-related operations. It extends the ***UserDetailsService*** interface provided by Spring Security, and has a method save that takes a ***UserRegistrationDto*** object and returns a ***User*** object.

**package** net.javaguides.springboot.service;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** net.javaguides.springboot.dto.UserRegistrationDto;

**import** net.javaguides.springboot.model.User;

**public** **interface** UserService **extends** UserDetailsService{

User save(UserRegistrationDto registrationDto);

}

**Module 14:- Class:- (UserServiceImpl.java)**

This code is an implementation of the **UserService** interface using the **UserRepository** and the **BCryptPasswordEncoder**. It provides two methods - save and **loadUserByUsername**.

The **save** method takes a **UserRegistrationDto** object, creates a new **User** object with the data from the DTO, encodes the password using the **BCryptPasswordEncoder**, adds a role of **ROLE\_USER**, and saves the user to the database using the **UserRepository**.

@Service

**public** **class** UserServiceImpl **implements** UserService{

**private** UserRepository userRepository;

@Autowired

**private** BCryptPasswordEncoder passwordEncoder;

**public** UserServiceImpl(UserRepository userRepository) {

**super**();

**this**.userRepository = userRepository;

}

@Override

**public** User save(UserRegistrationDto registrationDto) {

User user = **new** User(registrationDto.getFirstName(),

registrationDto.getLastName(), registrationDto.getEmail(),

passwordEncoder.encode(registrationDto.getPassword()), registrationDto.getDateOfBirth(),

registrationDto.getAddress(), Arrays.*asList*(**new** Role("ROLE\_USER")));

**return** userRepository.save(user);

}

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

User user = userRepository.findByEmail(username);

**if**(user == **null**) {

**throw** **new** UsernameNotFoundException("Invalid username or password.");

}

**return** **new** org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), mapRolesToAuthorities(user.getRoles()));

}

**private** Collection<? **extends** GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles){

**return** roles.stream().map(role -> **new** SimpleGrantedAuthority(role.getName())).collect(Collectors.*toList*());

}

}

**Chapter- 5**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conclusion**

An employee management system is a valuable tool for organizations looking to streamline their processes and increase efficiency. By providing a centralized database for employee records, it can help reduce errors, save time, and ultimately save money. Moreover, it can provide transparency to employees about their wages and benefits, increasing their satisfaction and motivation. Additionally, HR departments can use the system to monitor employee performance and identify areas for improvement, enabling them to better support and develop their employees. Overall, implementing an employee management system can be a smart investment for organizations looking to enhance their operations and create a more positive workplace culture.

* An employee management system can help organizations comply with labour laws and regulations, by ensuring that employee data is accurately recorded and reported. This can help avoid costly fines and legal issues.
* By automating administrative tasks such as scheduling, time-tracking, and leave management, an employee management system can reduce the workload on HR staff and improve their productivity.
* An employee management system can help identify trends and patterns in employee data, such as turnover rates and absenteeism, which can inform HR strategies and help, identify areas for improvement.
* By providing a platform for communication and feedback between employees and managers, an employee management system can help foster a culture of openness and accountability within the organization.
* Some employee management systems may include features such as training and development modules, which can help employees, build new skills and improve their performance over time.