Employee Management System

GRAMA MALINA BIANCA

HORVAT DIANA ADRIANA

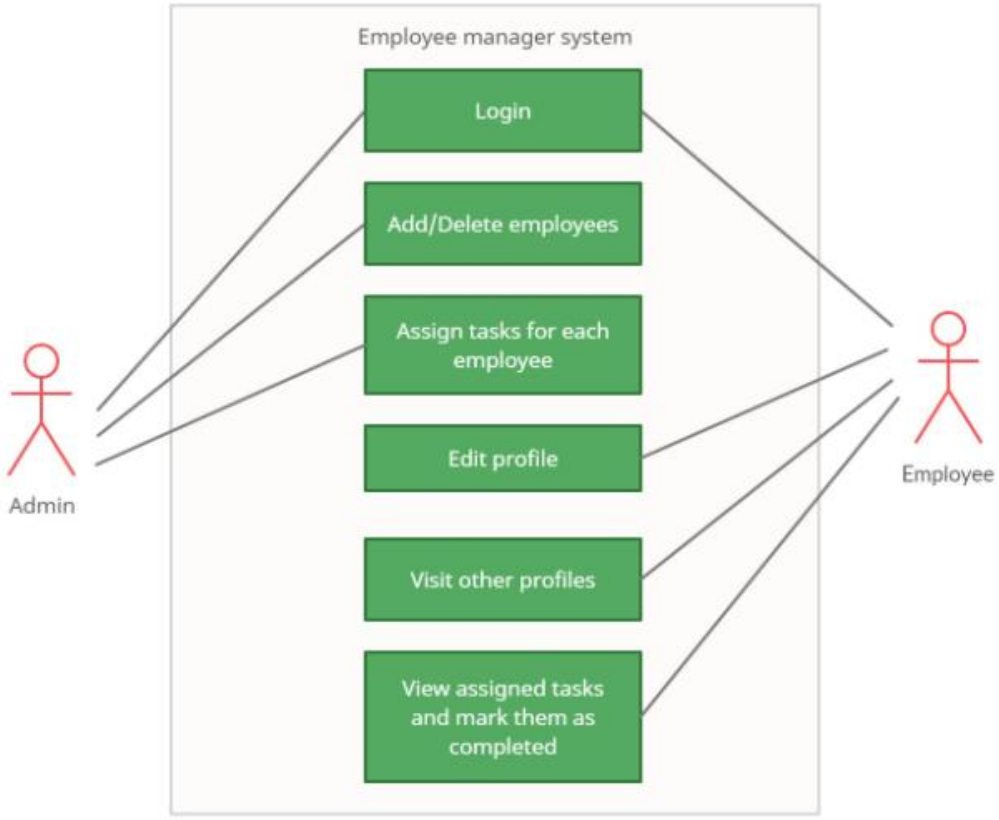
Technical University of Cluj-Napoca

1. INTRODUCTION

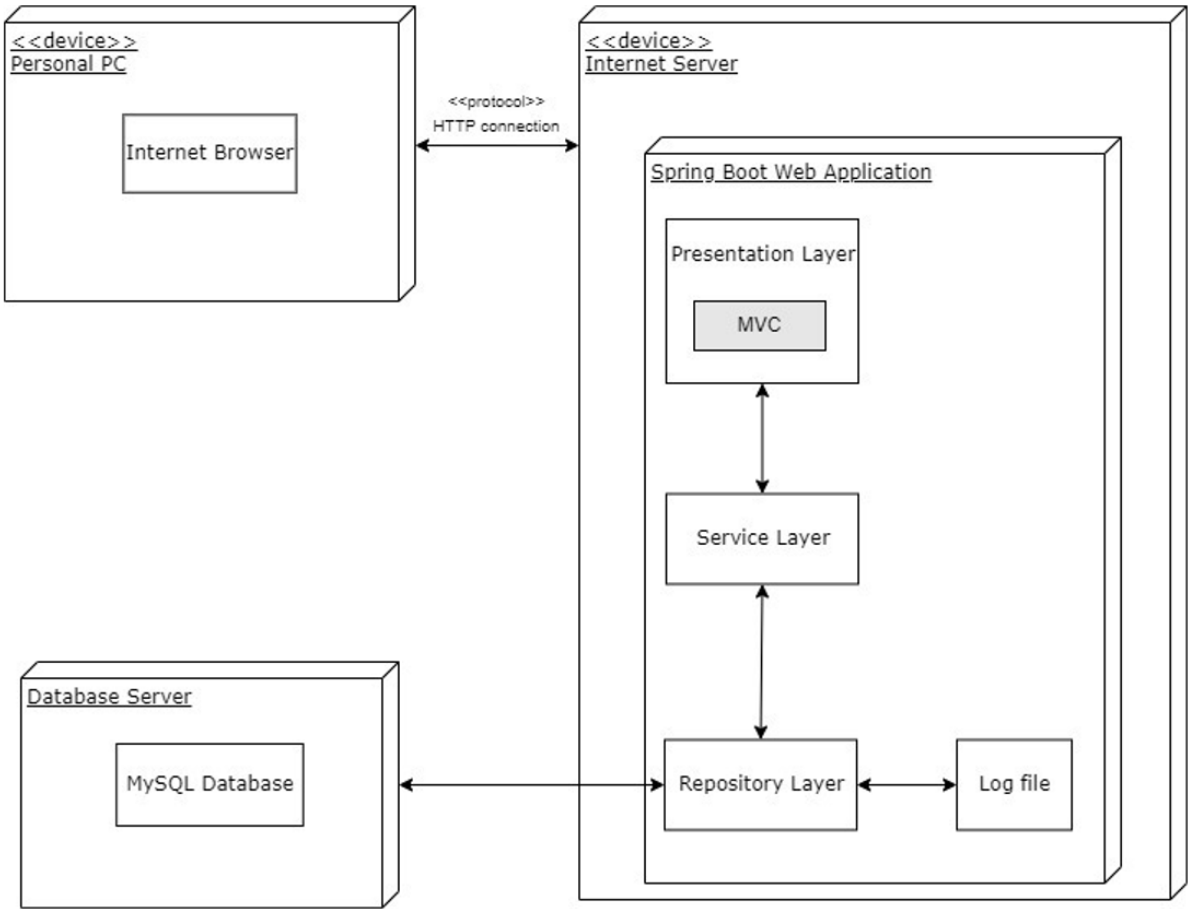
As the name suggests, we developed an application for managing employees and their tasks. The admin can add employees, delete and edit their profiles and assign tasks to each one of them, including itself. On the other side, the employees can see the profiles of their colleagues, edit their profile, see the list of the assigned tasks and mark them as completed.

2. DIAGRAMS

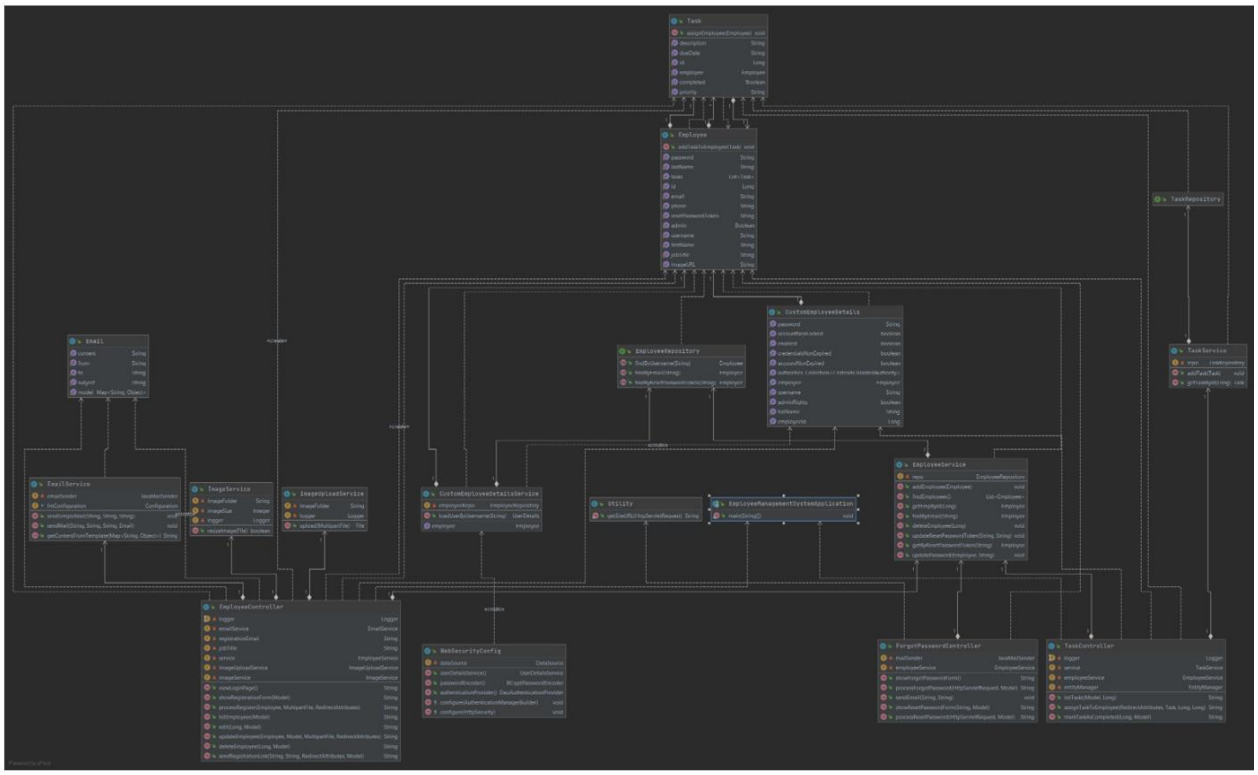
2.1. USE CASE DIAGRAM



2.2. DEPLOYMENT DIAGRAM



2.3. UML DIAGRAM



3. GENERAL APP DESCRIPTION

The app has two static pages, home and login, which can be accessed by any user. The registration page is restricted, meaning that other users cannot register just via a registration link sent by email by the admin of the app. The same system is applied when an employee forgot their password and a link is sent on their email to reset it.

Beside these features, as we mentioned before the admin can delete an employee, edit the job title of them and assign tasks with a deadline and a high or low priority. Employees can just edit their profile, see the assigned tasks and mark them as completed.

4. IMPLEMENTATION

The application is developed on three layers:

- Presentation Layer – MVC architectural pattern
- Service Layer
- Repository Layer

4.1. PRESENTATION LAYER

4.1.1 CONTROLLER PACKAGE

4.1.1.1 EMPLOYEECONTROLLER CLASS

```
@Transactional
@Controller
public class EmployeeController {
    private static final Logger logger =
LoggerFactory.getLogger(EmployeeManagementSystemApplication.class);

    @Autowired
    private EmailService emailService;

    private String registrationEmail;
    private String jobTitle;

    @Autowired
    private EmployeeService service;

    @Autowired
    private ImageUploadService imageUploadService;

    @Autowired
    private ImageService imageService;

@GetMapping("/login")
public String viewLoginPage() {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if(authentication == null || authentication instanceof
AnonymousAuthenticationToken) {
        logger.info("User enter on the login page");
        return "login";
    }
    logger.warn("User was already logged in so it was redirected to the home
page");
    return "redirect:/";
}
```

```
@GetMapping("/employees")
public String listEmployees(Model model) {
    List<Employee> listEmployees = service.findEmployees();
    model.addAttribute("listEmployees", listEmployees);

    Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
    CustomEmployeeDetails currentEmployee = (CustomEmployeeDetails)
auth.getPrincipal();
    Boolean isAdmin = currentEmployee.getAdminRights();
    Long emplId = currentEmployee.getEmployeeId();
    Employee emp = service.getEmpById(emplId);
    Boolean noTasks = emp.getTasks().isEmpty();

    Employee currentLoggedEmployee = currentEmployee.getEmployee();

    model.addAttribute("noTasks", noTasks);
    model.addAttribute("isAdmin", isAdmin);
    model.addAttribute("currentEmployee", currentLoggedEmployee);

    Task task = new Task();
    model.addAttribute("task", task);

    logger.info("User accessed the employees list");

    return "employees";
}

@GetMapping("/edit/{id}")
public String edit(@PathVariable Long id, Model model){
    Employee emp = service.getEmpById(id);

    model.addAttribute("emp", emp);

    logger.info("User wants to edit his profile");

    return "edit";
}

@PostMapping("/update")
public String updateEmployee(@ModelAttribute Employee emp, Model model,
@RequestParam("image")MultipartFile imageFile, RedirectAttributes
redirectAttributes) {
    emp.setId(emp.getId());

    Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
    CustomEmployeeDetails currentEmployee = (CustomEmployeeDetails)
auth.getPrincipal();
    Boolean isAdmin = currentEmployee.getAdminRights();

    if(imageFile.isEmpty()) {
        redirectAttributes.addFlashAttribute("noImageUploadedErrorMessage",
"Please choose file to upload.");
        logger.error("There was an error while uploading the user image");
        return "redirect:/edit/{id}";
    }
```

```
    File file = imageUploadService.upload(imageFile);
    if(file == null) {
        redirectAttributes.addFlashAttribute("uploadFailErrorMessage", "Upload
failed.");
        logger.error("There was an error while uploading the user image");

        return "redirect:/edit/{id}";
    }

    boolean resizeResult =  imageService.resizeImage(file);
    if(!resizeResult) {
        redirectAttributes.addFlashAttribute("resizeFailed", "Resize
failed.");
        logger.error("There was an error while resizing the user image");

        return "redirect:/edit/{id}";
    }
    String[] splitted = file.toString().split("static");

    emp.setImageURL(splitted[1]);

    model.addAttribute("isAdmin", isAdmin);
    service.addEmployee(emp);

    logger.info("User edited his profile");

    return "redirect:/employees";
}

@GetMapping("/delete/{id}")
public String deleteEmployee(@PathVariable Long id, Model model){
    service.deleteEmployee(id);

    Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
    CustomEmployeeDetails currentEmployee = (CustomEmployeeDetails)
auth.getPrincipal();
    Boolean isAdmin = currentEmployee.getAdminRights();

    model.addAttribute("isAdmin", isAdmin);

    logger.info("Admin deleted the employee with id = " + id);

    return "redirect:/employees";
}

@PostMapping("/sendRegistrationLink")
public String sendRegistrationLink(@RequestParam(value = "newEmployeeEmail")
String newEmployeeEmail,
                                   @RequestParam(value =
"newEmployeeJobTitle") String newEmployeeJobTitle,
                                   RedirectAttributes redirectAttributes,
Model model){
    if(!newEmployeeEmail.isEmpty() && !newEmployeeJobTitle.isEmpty()){
        registrationEmail = newEmployeeEmail;
        jobTitle = newEmployeeJobTitle;
```

```
        Email email = new Email();
        emailService.sendMail(newEmployeeEmail, "Registration link", "Hello!
", email);

        redirectAttributes.addFlashAttribute("success", "Email sent
successfully");
    }else{
        logger.error("Email address is not valid");
        redirectAttributes.addFlashAttribute("error", "The email field is
required");
    }

    Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
    CustomEmployeeDetails currentEmployee = (CustomEmployeeDetails)
auth.getPrincipal();
    Boolean isAdmin = currentEmployee.getAdminRights();

    model.addAttribute("isAdmin", isAdmin);
    logger.info("Admin send a registration link successfully");

    return "redirect:/employees";
}
}
```

4.1.1.2 ForgotPasswordController Class

```
@Controller
public class ForgotPasswordController {
    @Autowired
    private JavaMailSender mailSender;

    @Autowired
    private EmployeeService employeeService;

    @GetMapping("/forgot_password")
    public String showForgotPasswordForm() {
        return "forgot_password_form";
    }

    @PostMapping("/forgot_password")
    public String processForgotPassword(HttpServletRequest request, Model
model) {
        String email = request.getParameter("email");
        String token = RandomString.make(30);

        try {
            employeeService.updateResetPasswordToken(token, email);
            String resetPasswordLink = Utility.getSiteURL(request) +
"/reset_password?token=" + token;
            sendEmail(email, resetPasswordLink);
            model.addAttribute("message", "We have sent a reset password link
to your email.");
        } catch (UsernameNotFoundException ex) {
            model.addAttribute("error", ex.getMessage());
```

```
        } catch (MessagingException e) {
            model.addAttribute("error", "Error while sending email");
        }

        return "forgot_password_form";

    }

    public void sendEmail(String email, String resetPasswordLink) throws
MessagingException {
        MimeMessage message = mailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message);
        Employee emp = employeeService.findByEmail(email);
        String username = emp.getUsername();

        helper.setFrom("gramabmalina@gmail.com");
        helper.setTo(email);

        String subject = "Reset Password";

        String content = "<h1>Hello!</h1>" +
                         "<p style=\"font-size: medium\">" +
                         " You are receiving this email because you requested
your password to be changed. <br>" +
                         "Your username is: " + username +". <br>" +
                         "To change your password, please click on the
following link." +
                         "</p>" +
                         "<div class=\"container text-center\">" +
                             "<a href=\"" + resetPasswordLink + "\"
style=\"font-size: medium\">" +
                                 "Change Password" +
                             "</a>" +
                         "</div>" +
                         "<p style=\"font-size: medium\">" +
                            "If you have not made this request, please ignore
this e-mail." +
                         "</p>";
        helper.setSubject(subject);
        helper.setText(content, true);
        mailSender.send(message);
    }


    @GetMapping("/reset_password")
    public String showResetPasswordForm(@Param(value = "token") String token,
Model model) {
        Employee employee = employeeService.getByResetPasswordToken(token);
        model.addAttribute("token", token);

        if (employee == null) {
            model.addAttribute("message", "Invalid Token");
            return "message";
        }

        return "reset_password_form";
    }
```

```
    @PostMapping("/reset_password")
    public String processResetPassword(HttpServletRequest request, Model
model) {
        String token = request.getParameter("token");
        String password = request.getParameter("password");

        Employee employee = employeeService.getByResetPasswordToken(token);
        model.addAttribute("title", "Reset your password");

        if (employee == null) {
            model.addAttribute("message", "Invalid Token");
            return "message";
        } else {
            employeeService.updatePassword(employee, password);

            model.addAttribute("message", "You have successfully changed your
password.");
        }

        return "message";
    }
}
```

4.1.1.3 Utility Class

```
public class Utility {
    public static String getSiteURL(HttpServletRequest request) {
        String siteURL = request.getRequestURL().toString();
        return siteURL.replace(request.getServletPath(), "");
    }
}
```

4.1.1.4 WebSecurityConfig Class

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{
    @Autowired
    private DataSource dataSource;

    @Bean
    public UserDetailsService userDetailsService() {
        return new CustomEmployeeDetailsService();
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
```

```
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.authenticationProvider(authenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/employees").authenticated()
                .anyRequest().permitAll()
                .and()
                .formLogin()
                .loginPage("/login")
                .usernameParameter("username")
                .permitAll()
                .and()
                .logout().logoutRequestMatcher(new
AntPathRequestMatcher("/logout")).permitAll();
    }
}
```

4.1.2 Model Package

4.1.2.1 Employee Class

```
@Entity
@Table(name = "employees")
public class Employee implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 50)
    private String lastName;

    @Column(nullable = false, length = 50)
    private String firstName;

    @Column(nullable = false, length = 50)
    private String username;

    @Column(nullable = false, length = 1)
    private Boolean admin;
```

```
    public Boolean getAdmin() {
        return admin;
    }

    public void setAdmin(Boolean admin) {
        this.admin = admin;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Column(nullable = false, length = 500)
    private String password;

    @Column(nullable = false, length = 50)
    private String email;

    @Column(nullable = false, length = 50)
    private String jobTitle;

    @Column(nullable = false, length = 12)
    private String phone;

    @Column(nullable = false, length = 1000)
    private String imageURL;

    @Column(name = "reset_password_token", length = 30)
    private String resetPasswordToken;

    @JsonIgnore
    @OneToMany(mappedBy = "employee")
    private List<Task> tasks = new ArrayList<Task>();;

    public List<Task> getTasks() {
        return tasks;
    }

    public void setTasks(List<Task> tasks) {
        this.tasks = tasks;
    }

    public void addTaskToEmployee(Task task){
        tasks.add(task);
        task.setEmployee(this);
    }

    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 getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getJobTitle() {
        return jobTitle;
    }

    public void setJobTitle(String jobTitle) {
        this.jobTitle = jobTitle;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getImageURL() {
        return imageURL;
    }

    public void setImageURL(String imageURL) {
        this.imageURL = imageURL;
```

```
    }

    public String getResetPasswordToken() {
        return resetPasswordToken;
    }

    public void setResetPasswordToken(String resetPasswordToken) {
        this.resetPasswordToken = resetPasswordToken;
    }
}
```

4.1.3 View Package

The view package contains multiple html (with css and javascript) files that implement the different web pages of the application. For example, here is a listing of the index.html file:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="ISO-8859-1">
    <title>Employee Management System</title>
    <link rel="stylesheet" type="text/css"
href="/webjars/bootstrap/css/bootstrap.min.css"/>
    <script type="text/javascript"
src="/webjars/jquery/jquery.min.js"></script>
    <script type="text/javascript"
src="/webjars/bootstrap/js/bootstrap.min.js"></script>
</head>
<body>

<!--Navbar-->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Employee Management</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul  class="nav navbar-nav navbar-right" style="padding-left:
68%">
                <li class="nav-item">
                    <a class="nav-link" readonly aria-current="page"
th:href="@{/login}">List Of Tasks</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" aria-current="page"
th:href="@{/employees}">List Of Employees</a>
                </li>
                <li class="nav-item" sec:authorize="!isAuthenticated()">
                    <a class="nav-link" th:href="@{/login}">Login</a>
```

```
                </li>
                <li class="nav-item" sec:authorize="isAuthenticated()">
                    <a class="nav-link" th:href="@{/logout}">Logout</a>
                </li>
<!--                <div sec:authorize="isAuthenticated()">-->
<!--                    Text visible only to authenticated users.-->
<!--                </div>-->
            </ul>
        </div>
    </div>
</nav>
<!--Navbar-->


</body>
</html>
```

4.2 Service Layer

4.2.1 CustomEmployeeDetailsService Class

```
public class CustomEmployeeDetailsService implements UserDetailsService{
    @Autowired
    private EmployeeRepository employeeRepo;

    private Employee employee;


    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        employee = employeeRepo.findByUsername(username);
        if (employee == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return new CustomEmployeeDetails(employee);
    }

    public Employee getEmployee() {
        return employee;
    }
}
```

4.2.2 EmployeeService Class

```
@Service
@Transactional
public class EmployeeService {
    @Autowired
    private EmployeeRepository repo;

    public void addEmployee(Employee e) {
```

```
        repo.save(e);
    }

    public List<Employee> findEmployees(){
        return repo.findAll();
    }

    public Employee getEmpById(Long id){
        Optional<Employee> e = repo.findById(id);
        return e.orElse(null);
    }

    public Employee findByEmail(String email){
        return repo.findByEmail(email);
    }

    public void deleteEmployee(Long id){
        repo.deleteById(id);
    }

    public void updateResetPasswordToken(String token, String email) throws
UsernameNotFoundException {
        Employee employee = repo.findByEmail(email);
        if (employee != null) {
            employee.setResetPasswordToken(token);
            repo.save(employee);
        } else {
            throw new UsernameNotFoundException("Could not find any employee
with the email " + email);
        }
    }

    public Employee getByResetPasswordToken(String token) {
        return repo.findByResetPasswordToken(token);
    }

    public void updatePassword(Employee employee, String newPassword) {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String encodedPassword = passwordEncoder.encode(newPassword);
        employee.setPassword(encodedPassword);

        employee.setResetPasswordToken(null);
        repo.save(employee);
    }

}
```

BIBLIOGRAPHY

1. https://www.baeldung.com/spring-security-thymeleaf
2. https://www.baeldung.com/spring-security-logout
3. https://www.codejava.net/frameworks/spring-boot/spring-security-custom-login-page
4. https://www.codejava.net/frameworks/spring-boot/spring-security-forgot-password-tutorial
5. https://www.codejava.net/frameworks/spring-boot/spring-boot-security-role-based-authorization-tutorial
6. https://www.codejava.net/frameworks/spring-boot/spring-security-add-roles-to-user-examples