Easypay, The Payroll Management System

Problem Statement:

The primary objective of this project is to implement a robust and user-friendly Payroll Management System for XYZ Company, which will address the current challenges and improve the overall efficiency of payroll processes. The application will facilitate Improved Accuracy, Enhanced Compliance, Scalability, Time & Cost Savings and Employee Satisfaction.

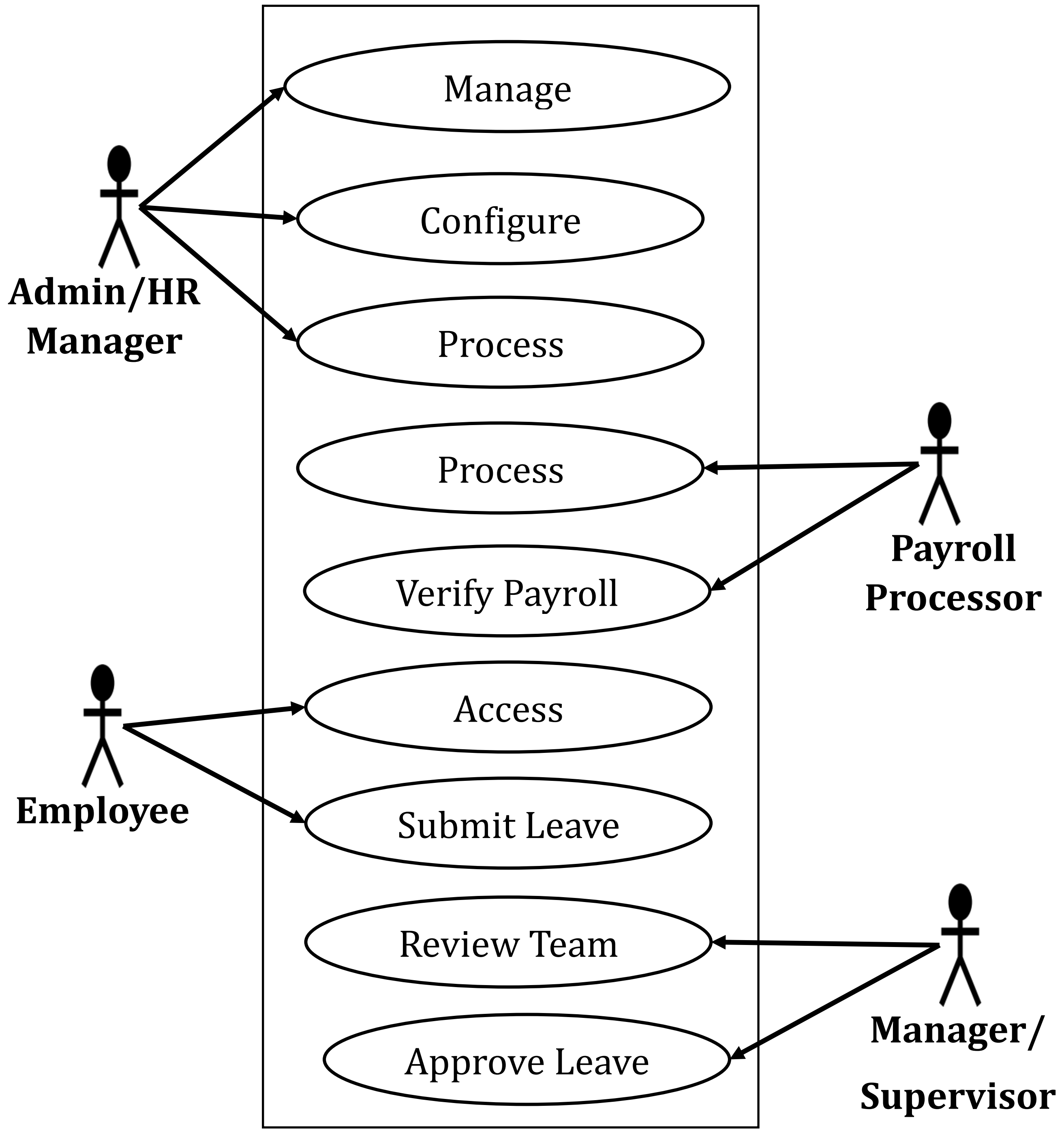
Scope:

- **Automate Payroll Calculations**: Implement an automated system for accurate calculation of employee compensation and benefits to eliminate errors.
- **Streamline Data Entry**: Develop an efficient and error-resistant data entry process for employee information and payroll-related data.
- **Ensure Compliance**: Integrate features to ensure compliance with tax regulations, labor laws, and other statutory requirements.
- **Employee Self-Service Portal**: Create a secure and user-friendly self-service portal for employees to access their payroll information and relevant documents.
- **Timely Salary Disbursement**: Establish a system that guarantees timely and accurate salary disbursement to employees.

Technologies:

- **Frontend**: React.js / Angular.
- **Backend**: Java, Spring Boot/C#, .Net / Python Django for API development.
- **Database**: MySQL / SQL Server.
- **Authentication**: JSON Web Tokens (JWT) for secure user authentication.

Use Case Diagram:



Use Cases:

Actor: Admin/HR Manager

- Use Case: Manage Employee Information
- Use Case: User Management
- Use Case: Define Payroll Policies
- Use Case: Generate Payroll
- Use Case: Compliance Reporting

Payroll Processor:

- Use Case: Calculate Payroll
- Use Case: Verify Payroll Data
- Use Case: Manage Benefits
- Use Case: Update Benefits Information
- Use Case: Process Payments

Employee:

- Use Case: View Pay Stubs
- Use Case: Update Personal Information
- Use Case: Submit Time Sheets
- Use Case: Request Leave

Manager/Supervisor:

- Use Case: Review Team Payrolls
- Use Case: Approve Leave Requests

Development Process:

1. **Admin/HR Manager:**
   - Dashboard: A summary view of payroll-related data and key metrics.
   - User Management: A page to add, edit, and manage user accounts and their roles.
   - Payroll Configuration: Allows the HR manager to define and configure payroll policies.
   - Employee Management: A page for managing employee information, including onboarding and offboarding.
   - Compliance Reporting: Access to generate and view compliance reports.
   - Notification Center: A central hub for system notifications and alerts.
   - Audit Trail: Access to view and search for system activity logs.
2. **Payroll Processor:**
   - Dashboard: A customized dashboard displaying payroll processing status and tasks.
   - Payroll Processing: A page to initiate and manage the payroll calculation process.
   - Payroll Verification: A page to review and validate payroll calculations before finalizing.
   - Audit Trail: Access to view and search for system activity logs.
   - Notifications: Receive alerts related to payroll processing and exceptions.
3. **Employee:**
   - Self-Service Portal: Access to personal payroll information, including pay stubs, tax forms, and leave balances.
   - Personal Information: A page to update personal details, tax withholding preferences, and contact information.
   - Leave Request: Submit and manage leave requests.
   - Notifications: Receive notifications about paydays, leave approvals, and other important events.
   - Time Sheets (if applicable): For hourly or part-time employees to enter work hours.
4. **Manager/Supervisor:**
   - Dashboard: Display key metrics for their teams and payroll processing status.
   - Team Payroll: Access to payroll information for direct reports.
   - Leave Approval: Review and approve leave requests submitted by team members.
   - Notifications: Receive alerts related to their team's payroll and leave requests.

5. **Security and Compliance:**

- User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

- User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
- Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
- Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
  1. Extract the token from the request header.
  2. Verify the token's signature using the server's secret key.
- Payload Verification:
  1. Decode the token and extract user information.
  2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning • Implement security to allow/disallow CRUD operations. • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable. • Implement Unit Test Project for testing the API. • Implement JWT for Security • Implement Logging • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected

3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, Doctor Consultation Page, Patient Appointment Booking Page, Add New Doctor Admin update details page**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, data binding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features.

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application.
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.