International Institute of Information Technology
Bangalore

Software Production Engineering

CS 816

Major Project Report

December 14, 2023

Contents

Employee Management System

1 Team Members

Name	Roll number	Email
Karanjit Saha	IMT2020003	Karanjit.Saha@iiitb.ac.in
Bhavjyot Singh Chadha	IMT2020116	Bhavjyot.Singh@iiitb.ac.in

2 Abstract

The presented project report encapsulates the evolution of the Employee Management application, a dynamic platform crafted through the integration of Java (SpringBoot), React, DevOps practices, Docker, and MySQL. The primary objective of this initiative is to conceive an application catering to the needs of employees, enabling them to log in and effortlessly put in their information for a company like their name, email, etc., and then modify their information in the database as well. The application also provides the feature of reading an employee's information details and also deleting an employee from the database.

The report commences with a comprehensive overview of the diverse technologies employed in crafting this application. Java takes the forefront on the server side, facilitating robust and efficient backend operations. Meanwhile, React is the chosen technology for the client side, ensuring a responsive and engaging user interface. The integration of DevOps tools is explored, emphasizing continuous integration and deployment to enhance the overall development process.

A pivotal aspect discussed in the report is the strategic utilization of Docker containers. These containers play a crucial role in simplifying deployment procedures and enhancing the scalability of the application. As the report unfolds, it delves into the intricacies of each technological facet, providing insights into the rationale behind their selection and their specific contributions to the development and functionality of the Employee Management application.

Continuing the narrative, the report intricately examines the implementation phase of the Employee Management application. This section sheds light on the utilization of diverse SpringBoot modules and React components, elucidating their synergistic role in sculpting the user interface. In particular, the report details the modular architecture of the SpringBoot backend, showcasing how different modules contribute to the overall functionality and performance of the application. Concurrently, the React components are expounded upon, emphasizing their design principles and their pivotal role in creating an intuitive and seamless user experience.

Furthermore, the report delves into the deployment process orchestrated through the use of DevOps tools. It outlines the intricacies of continuous integration and deployment, demonstrating how tools like Jenkins and Github synergize to automate and streamline the release pipeline. Docker containers are examined in this context, elucidating their role in creating a consistent and reproducible deployment environment.

As the report reaches its conclusion, attention is directed towards the challenges encountered throughout the development journey. These challenges are dissected, offering insights into the lessons learned and strategies employed to overcome obstacles. Additionally, the report contemplates potential future improvements aimed at enhancing the application's functionality and usability. This discussion serves as a launchpad for envisioning the application's evolution, presenting a roadmap for refining its features, and ensuring its continued relevance in meeting user needs.

3 System Requirements

3.1 System Configuration

- **Operating System:** Ubuntu 22.04
- **CPU and RAM:** Quad-core processor and 16 GB RAM

3.2 Application Stack

- **Backend:** Java (SpringBoot)
- **Database:** MySQL
- **Frontend:** React JS

3.3 Development Tools

- **Building Tools:** npm, Maven

3.4 DevOps Tools

1. **Version Control System:** Github
2. **CI/CD Pipeline:** Jenkins
3. **Testing Frameworks:**
   - JUnit for Java (SuperTest)
   - Jest for React Testing
4. **Containerization:** Docker
5. **Configuration Management and Infrastructure as Code:** Ansible
6. **Continuous Monitoring:** ELK Stack (Elasticsearch, Logstash, Kibana)

4 Introduction

4.1 What is DevOps

1. **Collaborative Approach:** DevOps is a collaborative practice that involves engineers from both operations and development teams at every stage of the service lifecycle, from design to development and production support.
2. **Integration of Development and Operations:** It is a set of practices that integrates software development (Dev) and information technology operations (Ops). The primary objective is to shorten the system development life cycle while delivering features, fixes, and updates frequently, aligning closely with business objectives.
3. **Automated and Streamlined Processes:** DevOps emphasizes collaboration between development and operations teams to establish an automated and streamlined software delivery process. The overarching goal is to create a continuous delivery pipeline that ensures the rapid, efficient, and error-free delivery of high-quality software.

4. **Enhanced Speed and Efficiency:** The core goal of DevOps is to enhance the speed and efficiency of software delivery. This is achieved by reducing errors and improving communication and collaboration among development, operations, and other stakeholders.

5. **Continuous Delivery Pipeline:** DevOps achieves its objectives through the implementation of tools, automation, continuous integration, continuous delivery, and monitoring. This holistic approach ensures that software is delivered quickly, reliably, and securely.

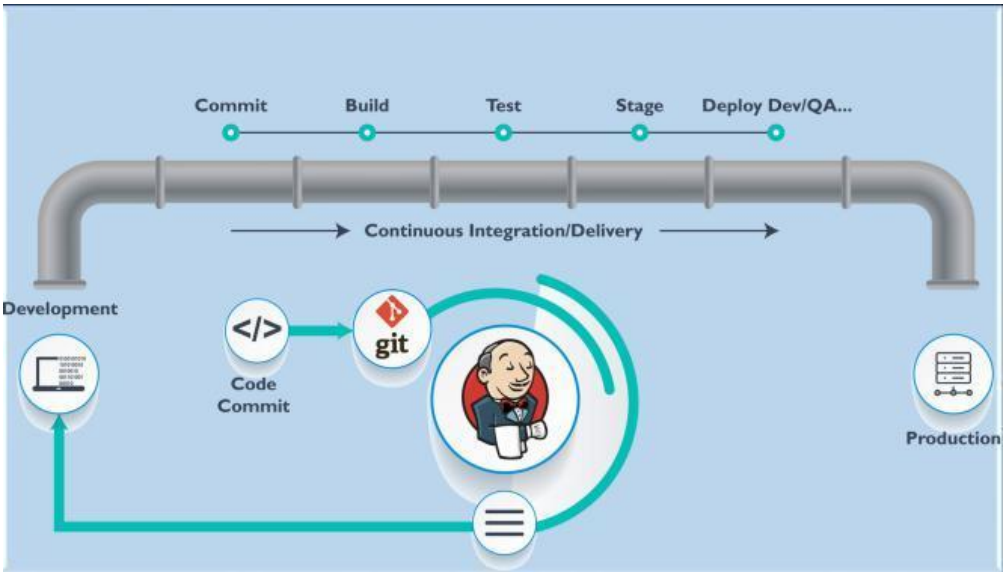


Figure 1: DevOps Workflow Diagram

4.2 DevOps Features

1. **Automation:** DevOps emphasizes the use of automation to streamline the software delivery process and reduce manual errors. This includes automating tasks such as testing, deployment, and monitoring.

2. **Continuous Integration and Continuous Delivery:** DevOps promotes a continuous delivery pipeline, where software changes are integrated and tested continuously, and delivered to production in a timely and efficient manner.

3. **Infrastructure as Code (IaC):** DevOps utilizes infrastructure as code (IaC) to manage and provision infrastructure resources, enabling rapid and consistent deployment and scaling.

4. **Monitoring and Logging:** DevOps emphasizes the importance of monitoring and logging to detect and resolve issues quickly, and to ensure the reliability and performance of software systems.

4.3 Project Description

The Employee Management System is a comprehensive application designed to facilitate the efficient management of employee information with essential CRUD (Create, Read, Update, Delete) operations. This robust system enables employees to input personal information such as name, email, and other relevant details, which are seamlessly integrated into a MySQL database.

4.4 Features

1. **User Authentication:** The system incorporates a secure login mechanism requiring a username and password. This ensures that only authorized individuals can access and manipulate employee data.

2. **Create (C):** Employees can add their information to the system, including but not limited to their full name, email address, and other relevant details. Upon submission, this data is stored in the MySQL database.

3. **Read (R):** Users can view their information as well as the details of other employees. The system provides an intuitive and user-friendly interface for easy navigation and information retrieval.

4. **Update (U):** Employees can modify their information as needed. This feature ensures that the data in the system remains accurate and up-to-date. The changes made are reflected in real-time in the MySQL database.

5. **Delete (D):** The application allows users to remove their profiles or the profiles of other employees from the system. This functionality is essential for maintaining an accurate and relevant employee database.

5 Implementation

5.1 Source Code Management

SCM (Source Code Management) is used to manage the source code of a software project, providing version control for the changes made to the code over time. SCM systems store each version of the code in a repository, along with information such as the date and time of the change, the author who made the change, and a brief description of the change. SCM helps in maintaining the integrity of the codebase, making it easier to manage changes and track issues. It also allows multiple developers to work on the same codebase without interfering with each other's work. Different versions of the code can be compared, merged, and rolled back if needed. This helps in improving collaboration, reducing errors, and increasing productivity.

1. **Version Control:** SCM systems track changes to the source code over time, creating a version history. Each version is associated with information such as the date, time, and author of the change.

2. **Codebase Integrity:** SCM helps maintain the integrity of the codebase by preventing conflicts and ensuring a systematic approach to code changes. Developers can work on their features or bug fixes without affecting the stability of the main codebase.

3. **Change Tracking:** Detailed information about each change, such as commit messages, makes it easier to understand the purpose of a modification. This aids in debugging, auditing, and understanding the evolution of the codebase.

4. **Collaboration:** Enables multiple developers to work on the same codebase simultaneously without conflicts. Changes made by different developers can be merged together to create a cohesive and functional codebase.

5. **Conflict Resolution:** In case of conflicting changes, SCM systems provide tools to merge or resolve conflicts, ensuring a smooth collaboration process.

6. **Comparisons and Differences:** Developers can compare different versions of the code, identifying changes made between versions. This feature helps in understanding the evolution of the code and identifying issues.

7. **Rollback:** SCM allows for rolling back to previous versions of the code if needed, providing a safety net in case of errors or unforeseen issues.

8. **Branching:** Developers can create branches to work on features or experiments independently. This allows for parallel development and the isolation of changes before they are merged into the main codebase.

9. **Traceability:** Traceability of changes is crucial for compliance, auditing, and understanding the context behind specific modifications.

10. **Increased Productivity:** By providing a structured and controlled environment for code changes, SCM systems contribute to increased productivity and a more efficient development process.

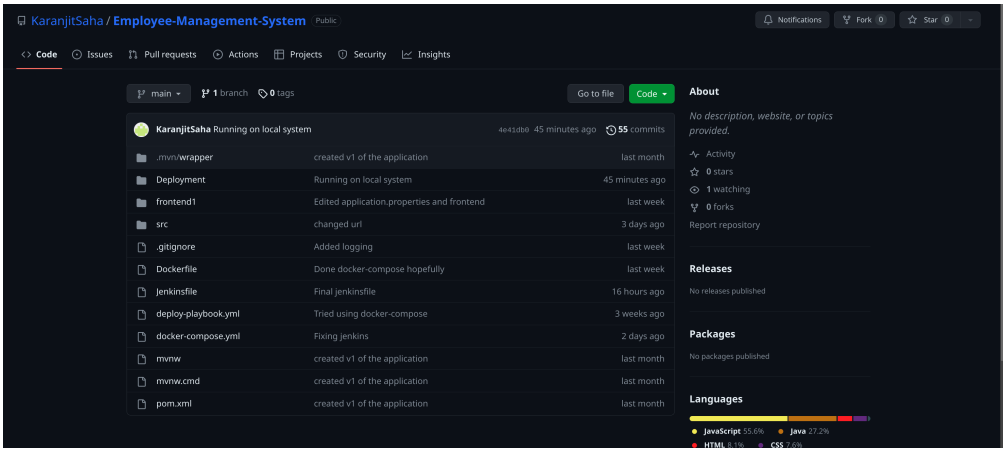


Figure 2: GitHub Repository

5.2 Backend

This project is developed using Spring Boot, Java, and MySQL. Specify the entry point, author, and other details. The folder structure of the backend looks like

1. **application.properties:** Configuration files for setting up the application, including environment variables, database connections, and server settings.

2. **Controllers:** Controllers themselves should only be responsible for work delegation. They should not contain any logic or direct data manipulation. The main purpose of a controller is to get the request, then pass it to the service that will process it, and then return the response that is given by the service. The main aim of the controller is to help simplify the most common tasks that you need to do when setting up routes and functions/classes to

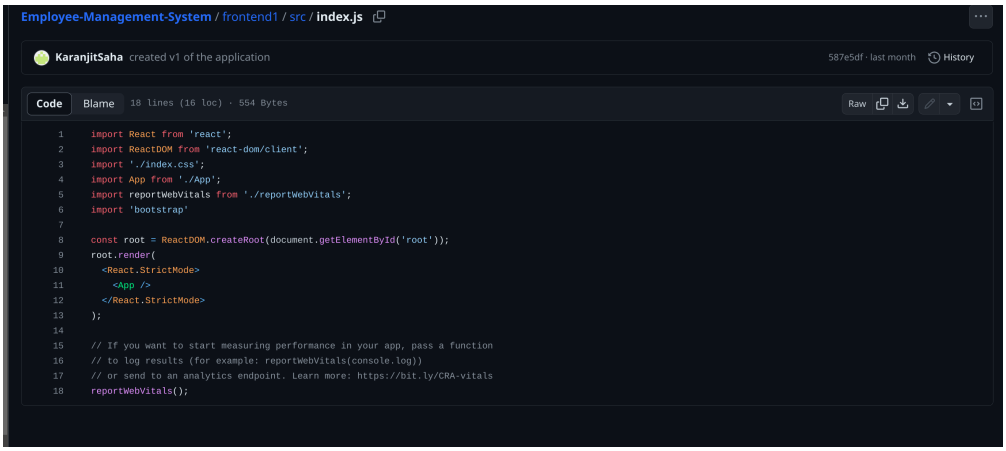


Figure 3: index.js

handle them. Controllers are the thing that directly responds to each HTTP Request that comes into your application, as such each web request will result in (if routed) a new instance of a Controller (Class). Code that handles the business logic of the application, interacting with the models and responding to user requests.

3. **Services:** Generally services contain information that is related to their domain. For example, if we have "User Service" we expect that adding/deleting users happens there like in real life. The same is fair for the codebase. Services (and their methods) handle the business logic which means that they are responsible for transforming data, and performing additional actions (like asking the repository for additional data or another service for processing some logic for it). So if we need to add a user, the service is responsible for getting the data as parameters, formatting them into the user itself, and sending it through an external service or saving it through the repository if it's an internal functionality. Code that runs between the request and response cycle, is often used for tasks such as authentication, error handling, and request validation. Backend developers ensure the website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers. They use code that helps browsers communicate with databases, and store, understand, and delete data.

4. **Repositories:** If there is code that somehow works with the database here is the place for this code to live. No matter if we're fetching the data from the DB or saving/modifying objects, it's the place where SQL queries or ORM operations should be placed. Code that defines the structure of the application's data and handles interactions with the database.

5. **Models:** This can refer to the data layer of the application, including the database and any caching mechanisms used.

Here is the Dockerfile for the backend of the application:

```
FROM openjdk:17
COPY ./target/Employee_Management_System-0.0.1-SNAPSHOT.jar ./
WORKDIR ./
EXPOSE 8082
CMD ["java", "-jar", "Employee_Management_System-0.0.1-SNAPSHOT.jar"]
```

- **Base Image:**
  - `FROM openjdk:17`: Specifies the base image for the Docker image. In this case, it uses the official OpenJDK image with version 17.
- **Copy Application JAR File:**
  - `COPY ./target/Employee_Management_System-0.0.1-SNAPSHOT.jar ./`: Copies the compiled JAR file of the Java application from the local `./target/` directory to the root directory (`./`) in the Docker image.
- **Set Working Directory:**
  - `WORKDIR ./`: Sets the working directory within the container to the root directory (`./`). Subsequent instructions will be executed in this directory.
- **Expose Port:**
  - `EXPOSE 8082`: Informs Docker that the application inside the container will use port 8082. This does not publish the port to the host machine but serves as documentation for developers.
- **Command to Start the Application:**
  - `CMD ["java", "-jar", "Employee_Management_System-0.0.1-SNAPSHOT.jar"]`: Specifies the default command to run when the container starts. In this case, it runs the Java application using the `java -jar` command, pointing to the JAR file copied earlier.

5.3 Frontend

For the frontend, we are using ReactJS. To create a react project run the command **npx create-react-app <package-name>**. Specify dependencies in the **package.json** file. The folder structure of frontend looks like this:

1. **components:** This directory typically contains reusable React components that can be used throughout the application. In our case, it contains admin and student-related pages and components.
2. **public:** The **public** directory typically contains static assets and the HTML file used as the entry point for the application. The contents in the Public directory index.html, Favicon, static Assets.
3. **src:** The **src** directory is where you will find the majority of the source code including the testing of the application, the models required, and the error handling.
4. **service:** All the Axios and fetch calls to the backend API are written in the service and well separated into various JS files depending on URL and functionality.
5. **app.js:** This is the main entry point for the React application, where all the components are composed and rendered to the DOM.

Here is the Dockerfile for the frontend of the application:

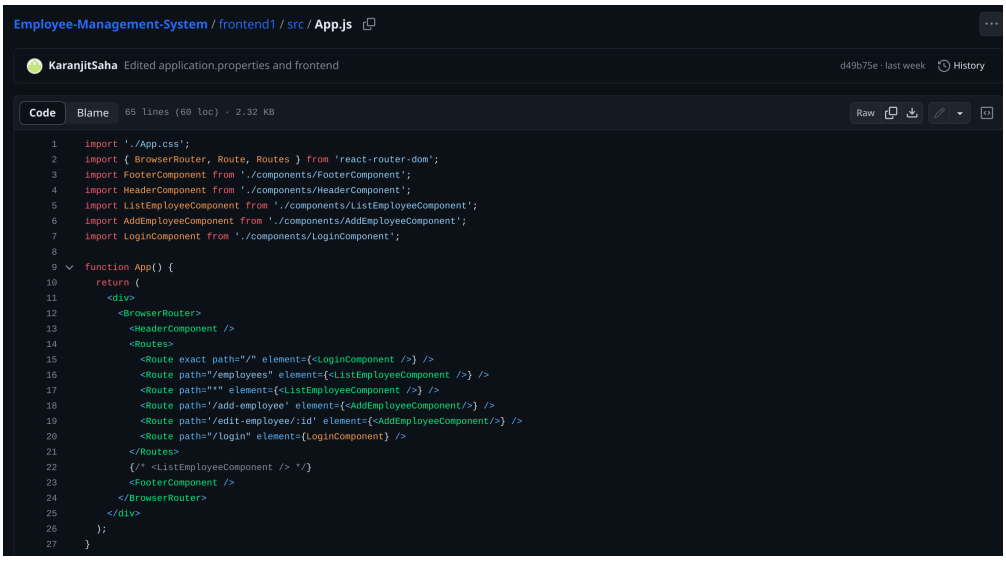


Figure 4: Frontend: App.js

```
FROM node:17-alpine3.12
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json .
COPY package-lock.json .
RUN npm install # -g npm@8.9.0
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **Base Image:**
  - `FROM node:17-alpine3.12`: Specifies the base image for the Docker image. In this case, it uses the official Node.js image with version 17 based on Alpine Linux 3.12.
- **Working Directory:**
  - `WORKDIR /app`: Sets the working directory within the container to `/app`. Subsequent instructions will be executed in this directory.
- **Environment Variable (PATH):**
  - `ENV PATH /app/node_modules/.bin:\$PATH`: Adds the local `node_modules/.bin` directory to the `PATH` environment variable. This is useful for running locally installed npm binaries without specifying the full path.
- **Copy Package Files:**
  - `COPY package.json .`: Copies the `package.json` file from the local directory to the `/app` directory in the container.
  - `COPY package-lock.json .`: Copies the `package-lock.json` file from the local directory to the `/app` directory in the container.

- **Install Dependencies:**
  - `RUN npm install # -g npm@8.9.0`: Runs the `npm install` command to install the Node.js application dependencies specified in the `package.json` file. The `-g npm@8.9.0` part is commented out but suggests installing a specific version of npm globally.
- **Copy Application Code:**
  - `COPY . .`: Copies the entire local directory (application code) to the `/app` directory in the container.
- **Expose Port:**
  - `EXPOSE 3000`: Informs Docker that the application inside the container will use port 3000. This does not publish the port to the host machine but serves as documentation for developers.
- **Command to Start the Application:**
  - `CMD ["npm", "start"]`: Specifies the default command to run when the container starts. In this case, it runs the `npm start` command, which is assumed to start the Node.js application.

5.4 Database

MySQL is an open-source relational database management system that is widely used for storing and managing data in web applications. MySQL is a popular database option for projects due to its scalability, performance, compatibility, security, and community support. It is widely used by small to large businesses, web applications, and software developers around the world. That's why we are using MySQL. To create a database, run the command **create database EMS;**.

5.5 API Documentation

The `EmployeeController` class is a fundamental component of the Employee Management System, designed to handle RESTful API requests related to employee management. This Java class leverages the Spring Framework to create a robust and scalable API, incorporating features such as Cross-Origin Resource Sharing (CORS), logging with Log4j, and Spring Data JPA for seamless interaction with an underlying database. The following points provide a comprehensive breakdown of the key components and functionalities implemented within the `EmployeeController` class.

- **Package and Imports:**
- The class is part of the package `com.KJ.Employee`$_{M}anagement_{S}ystem.controller.Itimportsseveralclasses, i$
- **Class Annotation:**
  - `@RestController`: Indicates that this class is a controller for handling REST requests.
- **CORS Configuration:**
  - `@CrossOrigin("*")`: Allows requests from any origin for Cross-Origin Resource Sharing (CORS).

```
karanjit@pop-os:~$ docker exec -it mysqldb mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| EMS                |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Figure 5: MySQL Databse

- **Request Mapping:**
  - `@RequestMapping("/api/v1/employees")`: Specifies the base URL for all REST API requests.
- **Logger Initialization:**
  - `private static final Logger logger = LogManager.getLogger(EmployeeController.class);`
- **Autowired EmployeeRepository:**
  - `@Autowired private EmployeeRepository employeeRepository;`
- **RESTful Endpoints:**
  - `@GetMapping`: Handles HTTP GET requests to retrieve a list of all employees.
  - `@PostMapping`: Handles HTTP POST requests to create a new employee.
  - `@GetMapping("id")`: Handles HTTP GET requests to retrieve an employee by their ID.
  - `@PutMapping("id")`: Handles HTTP PUT requests to update an existing employee by their ID.
  - `@DeleteMapping("id")`: Handles HTTP DELETE requests to delete an employee by their ID.
- **Exception Handling:**
  - Throws a `ResourceNotFoundException` when attempting to access an employee by ID if not found.
- **Logging:**

```java
package com.KJ.Employee_Management_System.controller;

import com.KJ.Employee_Management_System.EmployeeManagementSystemApplication;
import com.KJ.Employee_Management_System.exception.ResourceNotFoundException;
import com.KJ.Employee_Management_System.model.Employee;
import com.KJ.Employee_Management_System.repository.EmployeeRepository;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

```

Figure 6: EmployeeController.java

  - The logger is used to log messages at different levels (`info`, `debug`) throughout the class.

- **ResponseEntity:**
  - Uses `ResponseEntity` to wrap responses, providing control over the HTTP response.

5.6 Building

Building an application involves compiling and bundling the source code into a deployable package. This package typically includes all the necessary dependencies, configuration files, and other assets required to run the application. Building an application often involves running a build tool or using a CI/CD pipeline to automate the process. Packaging an application involves creating a distribution-ready package of the built application. This package can be in the form of a container image, a ZIP archive, or other formats. Packaging an application often involves running a packaging tool or using a CI/CD pipeline to automate the process. By building and packaging the application, developers can ensure that the application is optimized for production use, secure, and easy to deploy across different environments.

To build and package a Java Spring application using Maven, follow these steps:

1. **Ensure Maven is installed:** Make sure you have Maven installed on your system. You can check if Maven is installed by running the command **mvn -v** in your terminal or command prompt. If Maven is not installed, you can download it from the Apache Maven website and follow the installation instructions.

```java
public class EmployeeController {

    private static final Logger logger =
            LogManager.getLogger(EmployeeController.class);

    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping
    public List<Employee> getAllEmployees(){
        logger.info("Fetching the list of employees from database");
        return employeeRepository.findAll();
    }

    //BUILD CREATE EMPLOYEE REST API
    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee){
        logger.debug("Employee added");
        return employeeRepository.save(employee);
    }

    //BUILD GET EMPLOYEE BY ID REST API
    @GetMapping("{id}")
    public ResponseEntity<Employee> getEmployeeId(@PathVariable long id){
        Employee employee=employeeRepository.findById(id).orElseThrow(()-> new ResourceNotFoundException("Employee not exist with id "+id)) ;
        logger.debug("Employee found");
        return ResponseEntity.ok(employee);
    }
```

Figure 7: EmployeeController.java

```java
 //BUILD UPDATE EMPLOYEE REST API
 @PutMapping("{id}")
 //post mapping vs put mapping. post used to create a resource and put used to update a resource
 public ResponseEntity<Employee> updateEmployee(@PathVariable long id,@RequestBody Employee employeeDetails){
     Employee updateEmployee = employeeRepository.findById(id).orElseThrow(()-> new ResourceNotFoundException("Employee does not exist with id "
     updateEmployee.setFirstName(employeeDetails.getFirstName());
     updateEmployee.setLastName(employeeDetails.getLastName());
     updateEmployee.setEmailId(employeeDetails.getEmailId());
     employeeRepository.save(updateEmployee);
     logger.debug("Successfully updated the employee");
     return ResponseEntity.ok(updateEmployee);
 }

 //BUILD DELETE EMPLOYEE REST API
 @DeleteMapping("{id}")
 public ResponseEntity<HttpStatus> deleteEmployee(@PathVariable long id){
     Employee employee = employeeRepository.findById(id).orElseThrow(()-> new ResourceNotFoundException("Employee not exists "+id));

     employeeRepository.delete(employee);
     logger.debug("Delete employee");
     return new ResponseEntity<>(HttpStatus.NO_CONTENT);

 }
```

Figure 8: EmployeeController.java

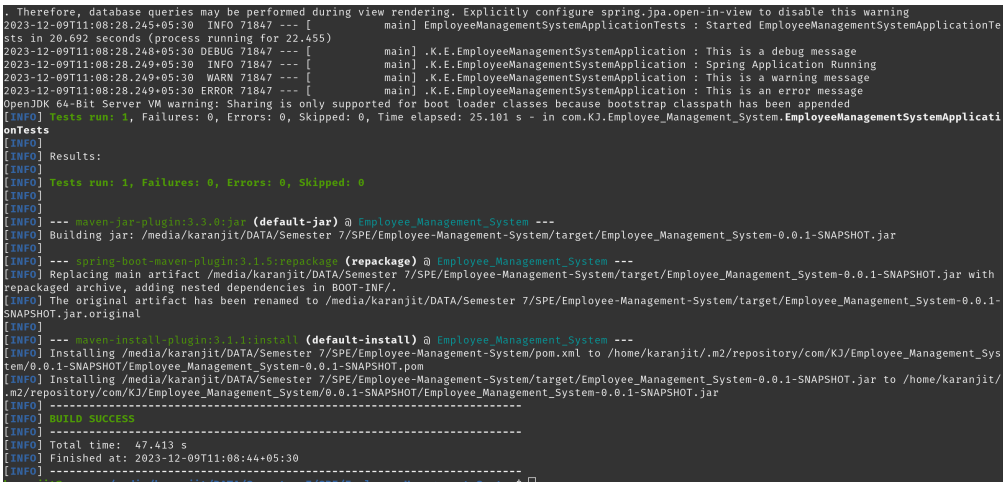


Figure 9: mvn build

2. **Set up the project structure:** Make sure your Java Spring project follows the standard Maven project structure. By convention, your project should have an "src" directory with sub-directories for "main" and "test". The "main" directory contains your application's source code, while the "test" directory contains test code.

3. **Configure the pom.xml file:** In the root directory of your project, you will find the pom.xml file. This is the configuration file for Maven. Open it in a text editor and configure it according to your project's needs. The pom.xml file contains project metadata, dependencies, build plugins, and other configuration settings.

4. **Build the project:** To build your Java Spring application, open a terminal or command prompt, navigate to the project's root directory (where the pom.xml file is located),

   See the following commands :

```
$ mvn clean install
```

   This command cleans the project, compiles the source code, runs tests, and packages the application into a JAR or WAR file. The resulting artifact will be placed in the **target** directory within your project.

5. **Verify the build:** After the build process completes successfully, check the "target" directory for the generated JAR or WAR file. The file will typically have a name based on your project and version.

To build and package a React application using **npm**, you can follow these steps:

1. **Set up your React project:** Ensure you have a working React project with the necessary source code, dependencies, and folder structure. You can create a new React project using tools like Create React App, or set up a custom React project manually.

2. **Install project dependencies:** Open a terminal or command prompt and navigate to the root directory of your React project. Run the following command to install the project dependencies specified in your package.json file:

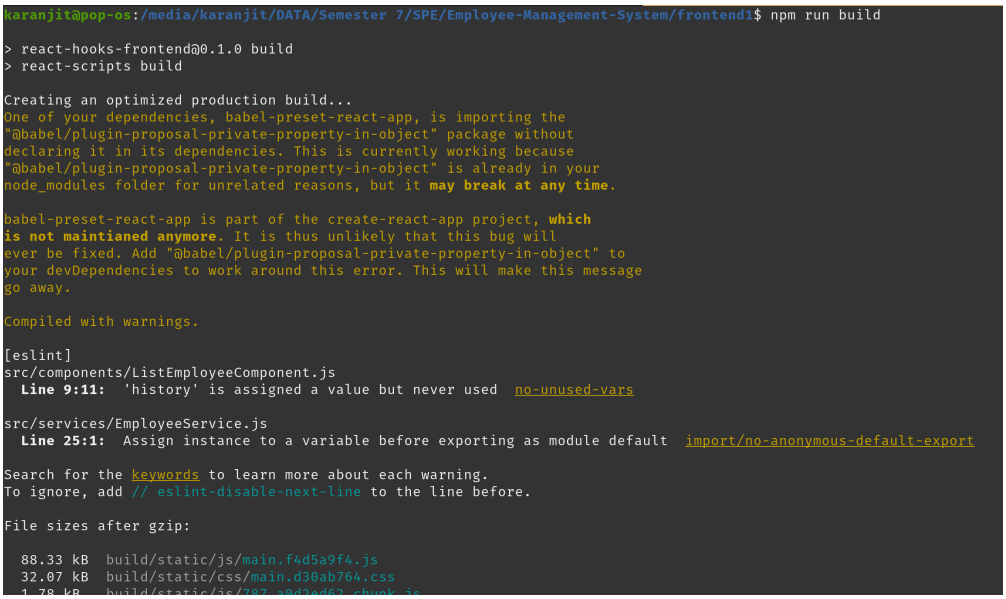


Figure 10: npm run build

See the following commands :

```
$ npm i
```

This will install all the necessary packages required for your React application, including React itself.

3. **Build the application:** Once the dependencies are installed, you can build your React application by running the following command:

   See the following commands :

```
$ npm run build
$ npm start
```

   This command will execute the build script specified in the **scripts** section of your package.json file. The build script typically invokes the appropriate build tools (e.g., webpack, Babel) to bundle and optimize your React code for production.

4. **Packaging the application:** After the build process completes successfully, the compiled and optimized files for your React application will be generated in a build directory (often named “build” or “dist”) within your project. These files can be directly deployed to a web server or used for distribution.

   The build directory will contain an index.html file and other static assets (e.g., JavaScript, CSS files) necessary for running the React application.

```
FROM openjdk:17
COPY ./target/Employee_Management_System-0.0.
1-SNAPSHOT.jar ./
WORKDIR ./
EXPOSE 8082
CMD ["java","-jar","Employee_Management_System-0.0.
1-SNAPSHOT.jar"]
```

Figure 11: Backend Dockerfile

5.7 Docker

Docker is mainly used by programmers to create, distribute, and execute applications easily. It uses OS-level virtualization. Docker platform as a service offers to distribute software in packages known as containers. It will provide software with the environment and all dependencies as a package known as a docker image. We just have to pull this docker image and the application will be ready to execute.

In this project, we are first building a docker image using

```
$ docker build karanjit708/backend-image
```

where **karanjit708** is the docker hub username, **backend-image** is image name. It builds docker images from a Dockerfile. The image created contains the latest version of the source code which can be built successfully to see all the features that have been implemented so far Dockerfile is a document containing a set of commands that are used for building images. We will create two docker files and docker-compose files one for the front end and one for the back end. The Dockerfile for the backend looks like this:

1. `FROM openjdk:17`: This line specifies the base image for the Docker image. In this case, it uses the official OpenJDK 17 image as the base image.

2. `COPY ./target/Employee_Management _System-0.0.1-SNAPSHOT.jar ./`: This line copies the file "./target/Employee_Management_System-0.0.1-SNAPSHOT.jar" from the local file system to the root directory in the Docker image.

3. `WORKDIR ./`: This command sets the working directory inside the Docker image to the current directory (denoted by "./").

4. `EXPOSE 8082`: This line exposes port 8082, indicating that the application inside the Docker container will be accessible on that port.

5. `CMD ["java", "-jar", "Employee_Management_System-0.0.1-SNAPSHOT.jar"]`: This command specifies the default command to be executed when the Docker container starts. It runs the Java application by executing the "java -jar Employee_Management_System-0.0.1-SNAPSHOT.jar" command.

5.8 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to define an entire application stack, including services, networks, and volumes, in a single 'docker-compose.yml' file. This file is then used to configure and start all the services as a single application.

```
version: '3'
services:
  mysqldb:
    image: "mysql"
    container_name: mysqldb
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: Kota@2020
      MYSQL_DATABASE: EMS
    ports:
      - "3306:3306"
    networks:
      - my-network

  frontend:
    image: karanjit708/frontend-image:latest
    container_name: frontend
    ports:
      - "3000:3000"
    networks:
      - my-network

  spring-app:
    image: "karanjit708/backend-image:latest"
    ports:
      - "8082:8082"
    depends_on:
      - mysqldb
    container_name: backend
    environment:
      SPRING_DATASOURCE_URL: ...
     jdbc:mysql://mysqldb:3306/EMS?createDatabaseIfNotExist=true&useSSL=false&allowPu
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: Kota@2020
    volumes:
      - /logs:/logs
    networks:
      - my-network

volumes:
  mysql-data:
    driver: local

networks:
  my-network:
```

```
46      driver: bridge
```

The above shown Docker Compose configuration defines a multi-container application with three services: 'mysqldb', 'frontend', and 'spring-app'. Below is a detailed explanation of each section:

- **Version and Services:**
  - The 'version: '3" at the beginning indicates the use of Docker Compose file version 3 syntax.
  - The 'services' section defines individual containers for each component of the application.
- **MySQL Database Service ('mysqldb'):**
  - Uses the official MySQL image from Docker Hub.
  - Sets the container name to 'mysqldb'.
  - Defines a volume 'mysql-data' to persist MySQL data.
  - Sets environment variables for MySQL root password and database name.
  - Maps port '3306' on the host to '3306' in the container.
  - Connects to the custom network named 'my-network'.
- **Frontend Service ('frontend'):**
  - Uses a custom image 'karanjit708/frontend-image:latest'.
  - Sets the container name to 'frontend'.
  - Maps port '3000' on the host to '3000' in the container.
  - Connects to the custom network named 'my-network'.
- **Spring Boot Application Service ('spring-app'):**
  - Uses a custom image 'karanjit708/backend-image:latest'.
  - Maps port '8082' on the host to '8082' in the container.
  - Depends on the 'mysqldb' service, ensuring it starts after 'mysqldb'.
  - Sets environment variables for the Spring Boot application to connect to the MySQL database.
  - Defines a volume to mount '/logs' in the container.
  - Connects to the custom network named 'my-network'.
- **Volumes and Networks:**
  - The 'volumes' section defines a named volume 'mysql-data' with the 'local' driver for persisting MySQL data.
  - The 'networks' section defines a custom bridge network named 'my-network'.

This Docker Compose configuration creates an environment where a MySQL database, a frontend application, and a Spring Boot application can communicate over the 'my-network' network. Volumes and ports are configured to ensure data persistence and accessibility. The services are orchestrated to start and stop in the specified order with dependencies considered.

5.9 Jenkins

Jenkins is an open-source automation. Continuous integration and continuous delivery (CI/CD) is made possible by automating the software development processes of developing, testing, and deploying. It is a server-based program that runs on Apache Tomcat which is a servlet container. We are using Jenkins for the CI/CD pipeline i.e. from build to deploy. After installing Jenkins, start the Jenkins service and it will run on "http://localhost:8080/". Open it in the browser and log in using admin credentials.

For creating a new pipeline project select on new item and pipeline project. Select necessary options, and provide pipeline script as direct script or file inside the GitHub repository. Click on build now and the pipeline will start building. For our project, we have to install the necessary plugins for Maven, Ansible, Git, and Docker. Go to manage Jenkins, Manage plugins, and search and install plugins. For the docker push command to execute inside the Jenkins pipeline we need to add credentials for the docker hub account. Go to manage Jenkins, manage credentials, add new credentials, and fill in details. Select username with password, and enter docker hub account username, password, and ID, which you will use as credentialID in the pipeline script and description.

```
pipeline {
    environment {
        backend = 'backend-image' // Specify your backend Docker image name/...
    tag
        frontend = 'frontend-image' // Specify your frontend Docker image ...
    name/tag
        mysqlImage = 'mysql:latest' // Specify the MySQL Docker image
        mysqlContainerName = 'mysql-container' // Specify the name for your ...
    MySQL container
        MYSQL_ROOT_PASSWORD = 'Kota@2020'
        MYSQL_PORT = '3306'
        docker_image = ''
        NETWORK = 'deployment_my-network'
    }

    agent any

    stages {
        stage('Stage 0: Pull MySQL Docker Image') {
            steps {
                echo 'Pulling MySQL Docker image from DockerHub'
                script {
                    docker.withRegistry('', 'DockerCred') {
                        docker.image("${mysqlImage}").pull()
                    }
                }
            }
        }

        stage('Stage 0.1: Run MySQL Container') {
            steps {
                script {
                    sh  'docker container stop mysqldb'
                    sh  'docker container rm mysqldb'
                    sh  'docker run --name mysqldb -p 3306:3306 -e ...
    MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD} -d -v "/var/lib/mysql" --...
```

```
        network=${NETWORK} mysql:latest'
33                  }
34              }
35          }
36
37          stage('Stage 1: Git Clone') {
38              steps {
39                  echo 'Cloning the Git repository'
40                  git branch: 'main', url: 'https://github.com/KaranjitSaha/...
        Employee-Management-System.git'
41              }
42          }
43
44          stage('Stage 2: Build Spring Boot backend') {
45              steps {
46                  echo 'Building Spring Boot backend'
47                  sh 'mvn clean install'
48              }
49          }
50
51          stage('Stage 3: Build backend Docker Image') {
52              steps {
53                  echo 'Building backend Docker image'
54                  sh "docker build -t karanjit708/${backend} ."
55              }
56          }
57
58          stage('Stage 4: Build frontend Docker image') {
59              steps {
60                  echo 'Building frontend Docker image'
61                  dir('frontend1') {
62                      echo 'Changing to frontend1 directory'
63                      sh "docker build -t karanjit708/${frontend} ."
64                  }
65              }
66          }
67
68          stage('Stage 5: Push backend Docker image to DockerHub') {
69              steps {
70                  echo 'Pushing backend Docker image to DockerHub'
71                  script {
72                      docker.withRegistry('', 'DockerCred') {
73                          sh 'docker push karanjit708/${backend}'
74                      }
75                  }
76              }
77          }
78
79          stage('Stage 6: Push frontend Docker image to DockerHub') {
80              steps {
81                  echo 'Pushing frontend Docker image to DockerHub'
82                  script {
83                      docker.withRegistry('', 'DockerCred') {
84                          sh 'docker push karanjit708/${frontend}'
85                      }
86                  }
87              }
88          }
89
```

```
90          stage('Stage 7: Clean docker images') {
91              steps {
92                  script {
93                      sh 'docker container prune -f'
94                      sh 'docker image prune -f'
95                  }
96              }
97          }
98
99          stage('Stage 8: Ansible Deployment') {
100             steps {
101                 dir('Deployment'){
102                     sh 'ansible-playbook -i inventory deploy.yml'
103                 }
104             }
105         }
106     }
107 }
```

Listing 1: Jenkinsfile

Pipeline Setup

- `pipeline`: Defines a Jenkins pipeline.
- `environment`: Specifies environment variables for configuration throughout the pipeline.
- `agent any`: Indicates that the pipeline can run on any available agent.

Stages

Stage 0: Pull MySQL Docker Image

- Pulls the MySQL Docker image from DockerHub.
- Uses the specified registry credentials for authentication.

Stage 0.1: Run MySQL Container

- Stops and removes any existing MySQL container.
- Runs a new MySQL container with specified parameters, including the root password and network settings.

Stage 1: Git Clone

- Clones the Git repository from the specified URL.
- Branch 'main' is selected for the clone operation.

Stage 2: Build Spring Boot backend

- Builds the Spring Boot backend using Maven.

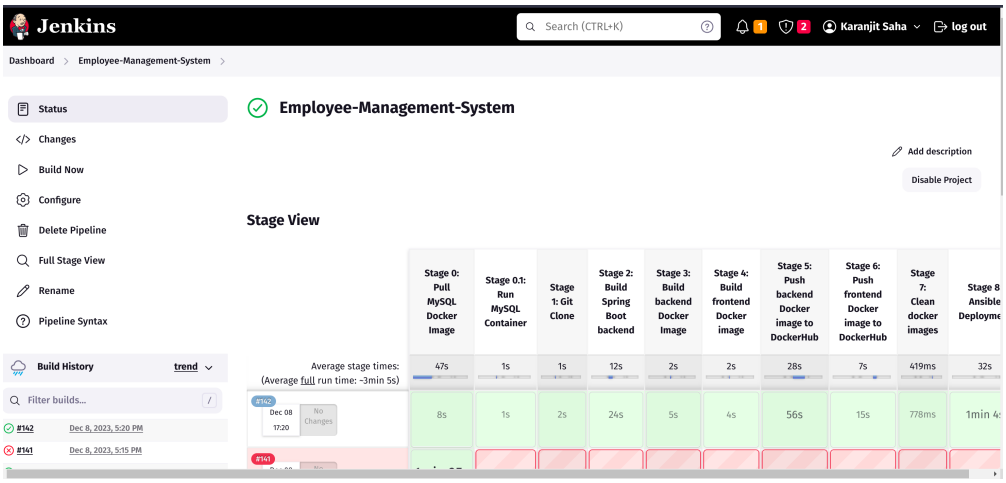


Figure 12: Jenkins Pipeline Build

Stage 3: Build backend Docker Image

- Builds the Docker image for the backend.
- Uses the specified Dockerfile.

Stage 4: Build frontend Docker image

- Builds the Docker image for the frontend.
- Changes the working directory to 'frontend1' before building.

Stage 5 and 6: Push Docker Images to DockerHub

- Pushes the backend and frontend Docker images to DockerHub.
- Uses the specified registry credentials for authentication.

Stage 7: Clean Docker Images

- Prunes (removes) unused Docker containers and images.

Stage 8: Ansible Deployment

- Changes the working directory to 'Deployment'.
- Runs an Ansible playbook for deployment using the specified inventory file.

5.10 Ansible

Ansible is a popular open-source automation tool that simplifies the automation of complex IT tasks, including configuration management, application deployment, and continuous delivery. Ansible is widely used in DevOps and IT operations for its simplicity, ease of use, and flexibility. The key features and uses of Ansible include:

1. **Configuration management:** Ansible can be used to configure servers, network devices, and other infrastructure components, ensuring that they are set up correctly and consistently.
2. **Application deployment:** Ansible can automate the deployment of applications and services to servers and other infrastructure components, ensuring that the process is repeatable, reliable, and consistent.
3. **Continuous delivery:** Ansible can automate the entire continuous delivery pipeline, from code checkout and build to testing, deployment, and monitoring.
4. **Orchestration:** Ansible can orchestrate complex workflows involving multiple tasks and components, ensuring that they are executed in the correct order and with the correct parameters.
5. **Cloud automation:** Ansible can automate the provisioning and management of cloud infrastructure, including virtual machines, containers, and other cloud services.
6. **Security and compliance:** Ansible can be used to enforce security policies and compliance requirements across the infrastructure, ensuring that systems are secure and compliant with relevant regulations and standards.

It is straightforward to deploy since it doesn't require agents or any other specialised security infrastructure, and most significantly because it employs a very basic language (YAML, in the form of Ansible Playbooks), which enables you to express your automation activities in a manner that closely resembles plain English.

This is an Ansible playbook that deploys a Docker image to a container. The playbook is executed on all hosts defined in the inventory file. The playbook uses the Ansible Docker module to manage the Docker environment and the Docker Compose tool to orchestrate the containers.

Playbook Structure

– `---`: Marks the beginning of a YAML document.

– `- name: Deploy docker images`: Defines the name of the Ansible playbook.

– `hosts: Paji`: Specifies the target hosts for the playbook. In this case, it's set to the host named 'Paji'. You can modify this to match your deployment environment.

– Commented out `# hosts: localhost`: Alternative host specification, commented out in this case. If you want to run tasks on the localhost, you can uncomment this line.

Tasks

Task 1: Copy Docker Compose file

– `Name: Copy Docker Compose file from host machine to remote host`: Descriptive name for the task.

– `Module: copy`: Ansible module used for copying files.

– `Source: ../docker-compose.yml`: Source path of the Docker Compose file on the local machine (relative path).

– `Destination: ./`: Destination path on the remote host (current directory).

Deployment > ! deploy.yml

```yaml
---
- name: Deploy docker images
  hosts: Paji
  # hosts: localhost
  # hosts: all

  tasks:
    - name: Copy Docker Compose file from host machine to remote host
      copy:
        src: ../docker-compose.yml
        dest: ./

    - name: Pull the Docker images specifed in docker-compose
      shell:
        cmd: docker-compose pull
        chdir: ./

    - name: Run the pulled Docker images in detached mode
      command: docker-compose up -d --build
```

Figure 13: deploy.yml

```
+ ansible-playbook -i inventory deploy.yml -vvv
ansible-playbook [core 2.15.6]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/var/lib/jenkins/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /var/lib/jenkins/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible-playbook
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
Using /etc/ansible/ansible.cfg as config file
host_list declined parsing /var/lib/jenkins/workspace/Employee-Management-System/Deployment/inventory as it did not pass its verify_file() method
script declined parsing /var/lib/jenkins/workspace/Employee-Management-System/Deployment/inventory as it did not pass its verify_file() method
auto declined parsing /var/lib/jenkins/workspace/Employee-Management-System/Deployment/inventory as it did not pass its verify_file() method
Parsed /var/lib/jenkins/workspace/Employee-Management-System/Deployment/inventory inventory source with ini plugin
Skipping callback 'default', as we already have a stdout callback.
Skipping callback 'minimal', as we already have a stdout callback.
Skipping callback 'oneline', as we already have a stdout callback.

PLAYBOOK: deploy.yml ***********************************************************
1 plays in deploy.yml

PLAY [Deploy docker images] ****************************************************

TASK [Gathering Facts] *********************************************************
task path: /var/lib/jenkins/workspace/Employee-Management-System/Deployment/deploy.yml:2
<192.168.86.70> ESTABLISH SSH CONNECTION FOR USER: bhavjyot
<192.168.86.70> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o KbdInteractiveAuthentication=no -o PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -o
PasswordAuthentication=no -o 'User="bhavjyot"' -o ConnectTimeout=10 -o 'ControlPath="/var/lib/jenkins/.ansible/cp/a1ef2a2258"' 192.168.86.70 '/bin/sh -c '"'"'echo ~bhavjyot && sleep 0'"'"''
<192.168.86.70> (255, b'', b'bhavjyot@192.168.86.70: Permission denied (publickey,password).\r\n')
fatal: [bhavjyot]: UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: bhavjyot@192.168.86.70: Permission denied (publickey,password).",
    "unreachable": true
}

PLAY RECAP *********************************************************************
bhavjyot                   : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 14: SSH Error

Task 2: Pull Docker Images

– `Name: Pull the Docker images specified in docker-compose`: Descriptive name for the task.

– `Module: shell`: Ansible module for running shell commands.

– `Command: docker-compose pull`: Shell command to pull Docker images defined in the Docker Compose file.

– `Working Directory: ./`: Specifies the working directory for the shell command.

Task 3: Run Docker Images

– `Name: Run the pulled Docker images in detached mode`: Descriptive name for the task.

– `Module: command`: Ansible module for running command-line commands.

– `Command: docker-compose up -d --build`: Shell command to run Docker images in detached mode. The `--build` flag ensures that images are built before starting the services.

Task 4: Prune Unwanted Images

– `Name: Prune unwanted images`: Descriptive name for the task.

– `Module: command`: Ansible module for running command-line commands.

– `Command: docker image prune --force`: Shell command to prune (remove) unused Docker images with the `--force` flag to avoid confirmation prompts.

5.11 ELK

Continuous monitoring refers to the ongoing process of collecting and analyzing data from systems and applications to identify and address issues in real time. This practice is critical for maintaining the health and security of IT environments, as it allows teams to quickly detect and respond to potential problems before they become critical. Continuous monitoring involves using tools and techniques such as log analysis, network scanning, vulnerability assessments, and performance monitoring to monitor the health and security of systems and applications.

Some of the key benefits of continuous monitoring include improved system availability and uptime, increased security and compliance, and faster incident response times. After the deployment part is done it is necessary to continuously monitor the application to check whether it is running correctly, giving correct output, and checking performance. We can do this continuous monitoring using the ELK stack. We can analyze the logs using the ELK stack.

Three open-source initiatives are referred to together as ELK: Elasticsearch, Logstash, and Kibana. A search and analytics engine is Elasticsearch. Elasticsearch-style stashes are where data is stored after it has been transformed by a server-side data processing pipeline called Logstash. Users of Elasticsearch may view data using charts and graphs using Kibana.

- **Elasticsearch:** It is a distributed, real-time search and analytics engine. It is used to index and store large amounts of data and provides fast, real-time search and analysis capabilities.
- **Logstash:** It is a data processing pipeline that ingests, filters, and transforms data from various sources and sends it to Elasticsearch or other destinations.

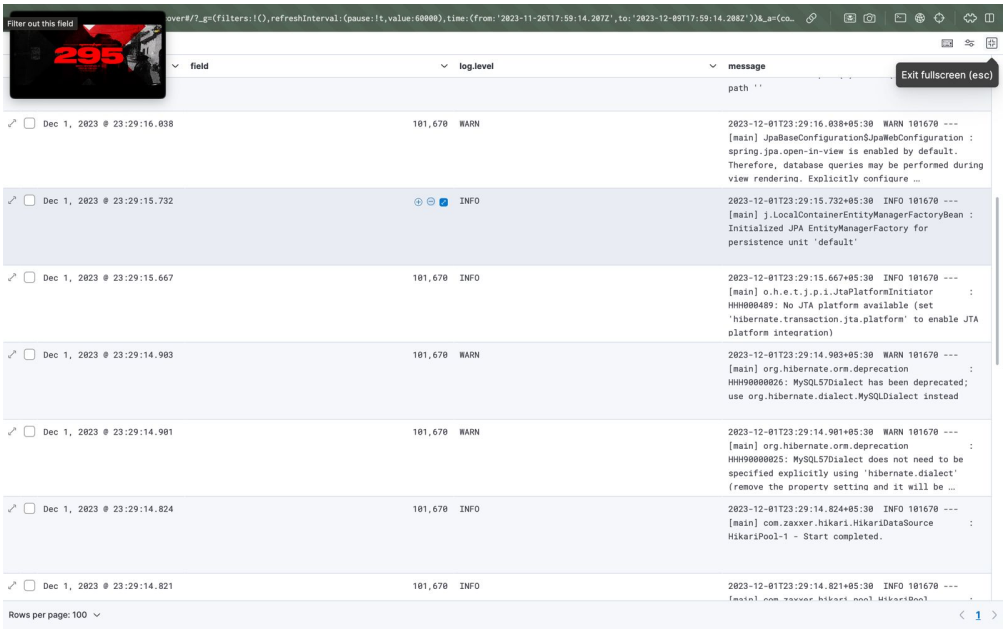
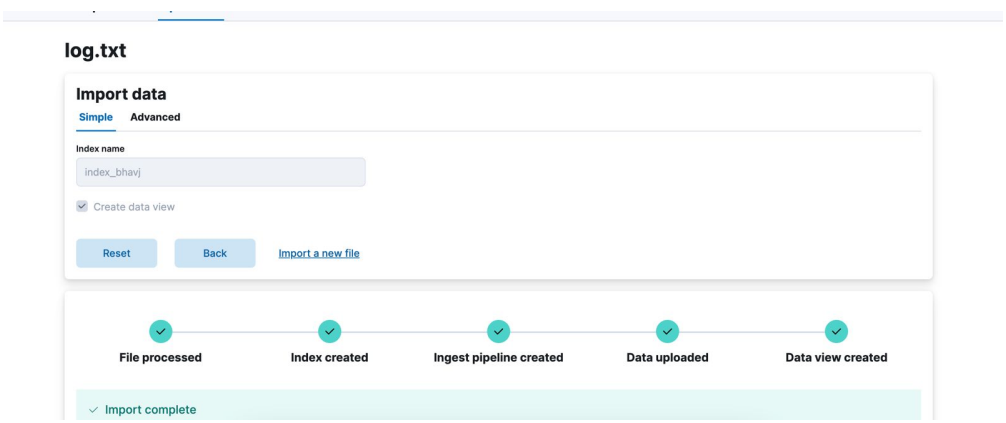


Figure 15: Log file generation



- **Kibana:** Kibana is a web-based user interface that provides visualization and analysis of data stored in Elasticsearch. It allows users to interactively explore and analyze their data through visualizations such as charts, graphs, and tables.

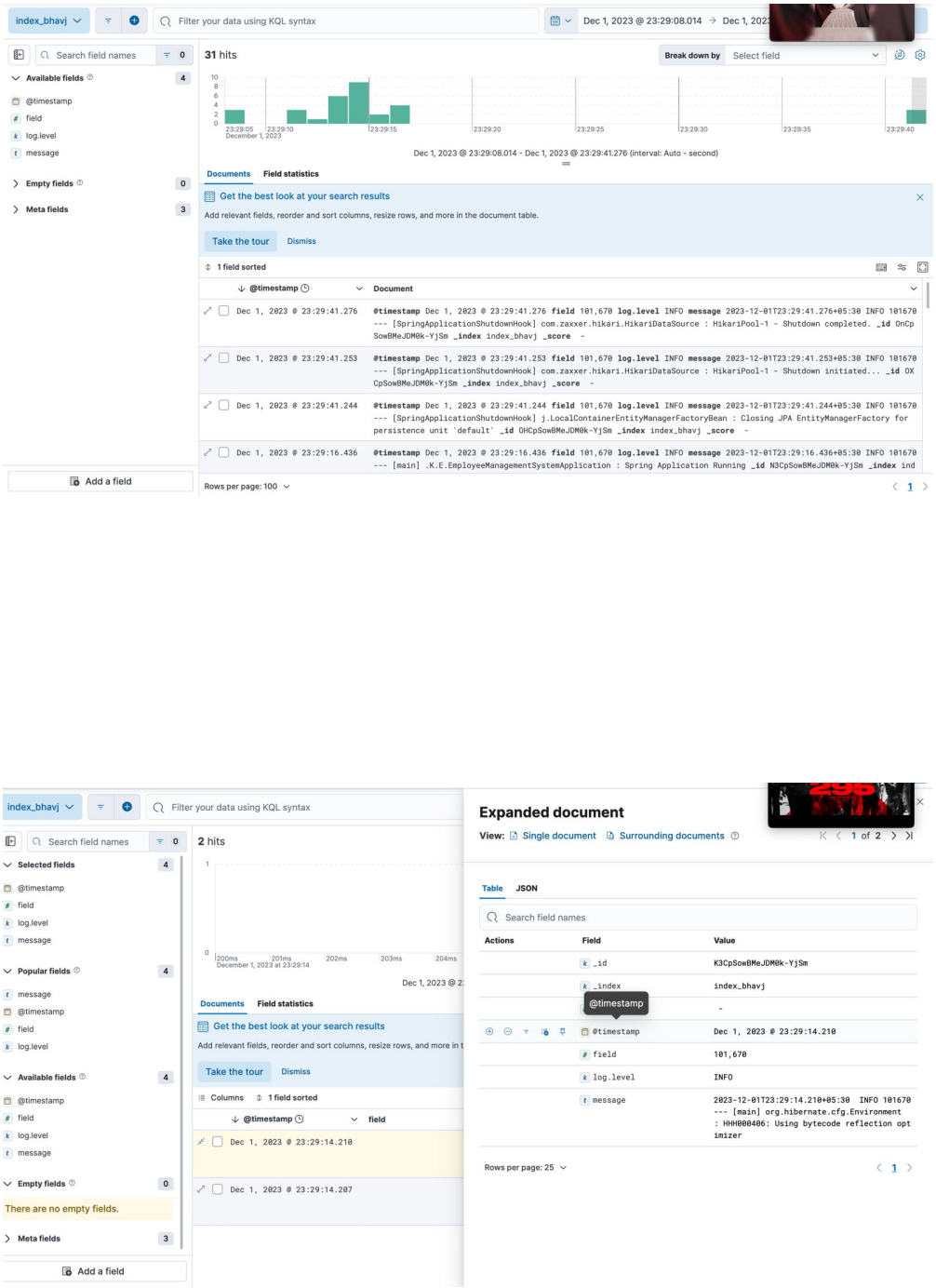


Figure 16: Data pipeline generated from log.txt

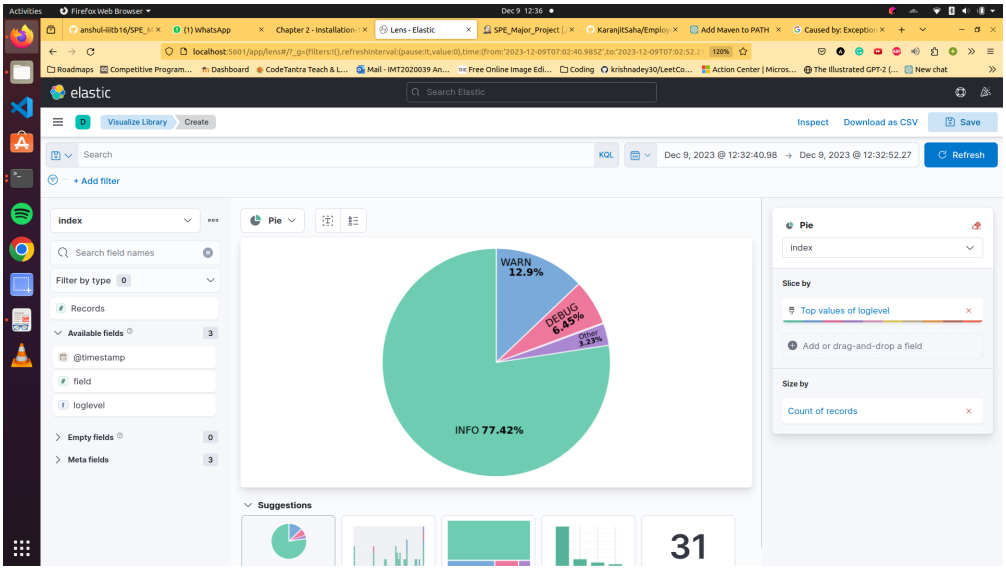


Figure 17: Visualizations from Kibana

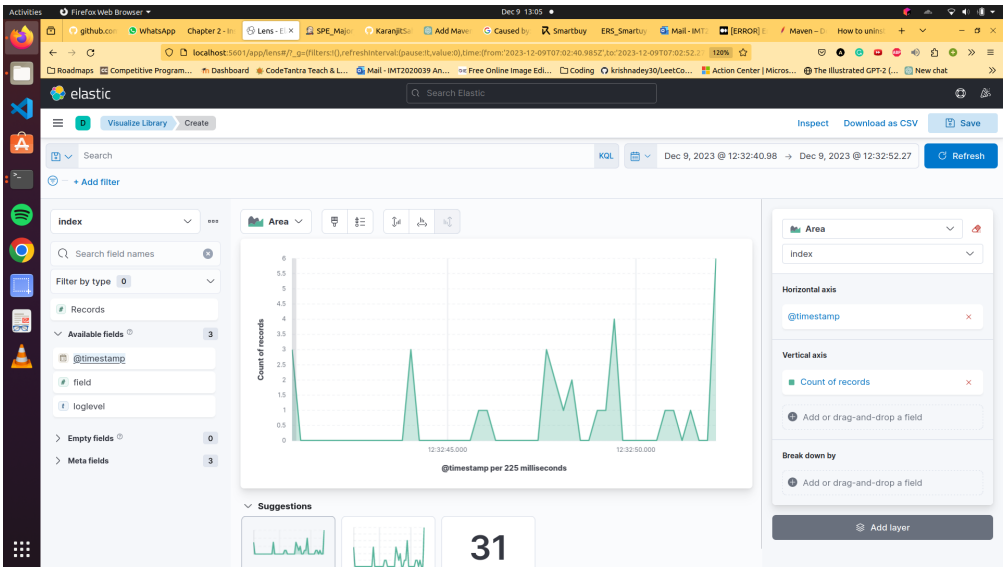


Figure 18: Visualizations from Kibana

6 Testing

- **Package and Imports:**
  - The code is part of the `com.KJ` package and includes various imports for testing, Spring Boot, Mockito, and other related libraries.
- **Annotations:**
  - `@SpringBootTest(classes = EmployeeManagementSystemApplication.class)`: Indicates that the test should bootstrap the entire Spring context and loads the specified class (`EmployeeManagementSystemApplication` in this case).
  - `@AutoConfigureMockMvc`: Configures the test to auto-configure the `MockMvc` instance.
- **Mocking Dependencies:**
  - `@Mock private EmployeeRepository employeeRepository;`: Creates a mock instance of the `EmployeeRepository` to simulate database interactions.
  - `@InjectMocks private EmployeeController employeeController;`: Injects the mock dependencies into the `EmployeeController`.
- **Initialization:**
  - `public void EmployeeControllerTest() { MockitoAnnotations.openMocks(this); }`: Initializes the mocks using `MockitoAnnotations`.
- **Test Method:**
  - `void testGetAllEmployees()`: Tests the `getAllEmployees` method of the `EmployeeController`.
- **Mock Data Setup:**
  - Creates two `Employee` instances (`employee1` and `employee2`) with different details.

```java
@Test
void testGetAllEmployees() {
    // Mock data
    Employee employee1 = new Employee();
    employee1.setId(1L);
    employee1.setFirstName("John");
    employee1.setLastName("Doe");
    employee1.setEmailId("john.doe@example.com");

    Employee employee2 = new Employee();
    employee2.setId(2L);
    employee2.setFirstName("Jane");
    employee2.setLastName("Doe");
    employee2.setEmailId("jane.doe@example.com");

    List<Employee> mockEmployeeList = Arrays.asList(employee1, employee2);

    // Mock the behavior of the employeeRepository
    when(employeeRepository.findAll()).thenReturn(mockEmployeeList);

    // Call the controller method
    List<Employee> result = employeeController.getAllEmployees();

    // Verify the result
    assertEquals(mockEmployeeList.size(), result.size());
    assertEquals(mockEmployeeList.get(0).getFirstName(), result.get(0).getFirstName());
    assertEquals(mockEmployeeList.get(1).getEmailId(), result.get(1).getEmailId());
    // Add more assertions as needed

    }
```

Figure 19: Testing of the code

  - Forms a list of mock employees (`mockEmployeeList`) containing the created `Employee` instances.

- **Mock Behavior:**
  - `when(employeeRepository.findAll()).thenReturn(mockEmployeeList);`: Specifies the expected behavior when the `findAll` method of the mocked `employeeRepository` is called. It will return the list of mock employees.
- **Controller Invocation:**
  - `List<Employee> result = employeeController.getAllEmployees();`: Invokes the `getAllEmployees` method of the `EmployeeController`.
- **Assertions:**
  - `assertEquals(mockEmployeeList.size(), result.size());`: Verifies that the size of the returned list matches the size of the mock employee list.
  - `assertEquals(mockEmployeeList.get(0).getFirstName(), result.get(0).getFirstName());`: Verifies that the first element's first name in the returned list matches the first element's first name in the mock list.
  - Additional assertions can be added based on the specific requirements of the test.

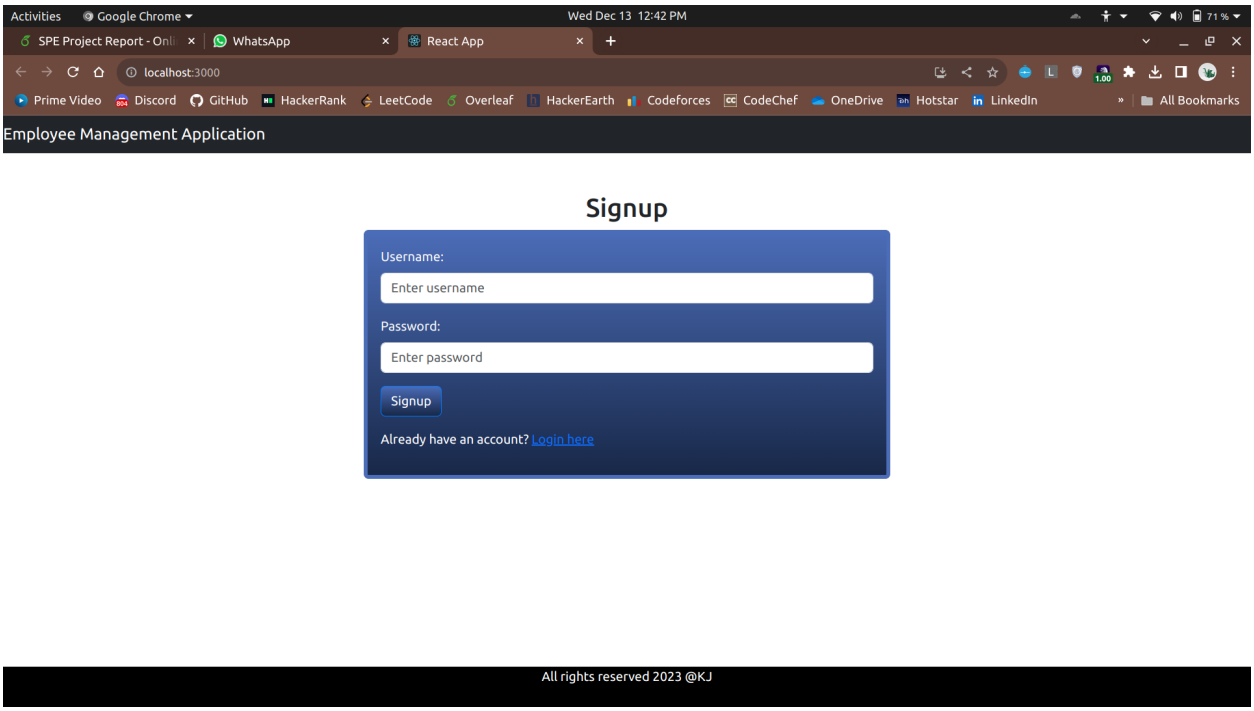


Figure 20: Signup Page

7 Results

Below are the screenshots of the final deployed website. The website initially has a Sign-Up page for new users and also has a Login link for previously existing users. The users can log in to the system and can add their information to the system. The users can now view the entire database and can perform the update and the delete operations. The update option will help the users to update the information that they previously entered and the delete operation helps them to delete an employee from the database.

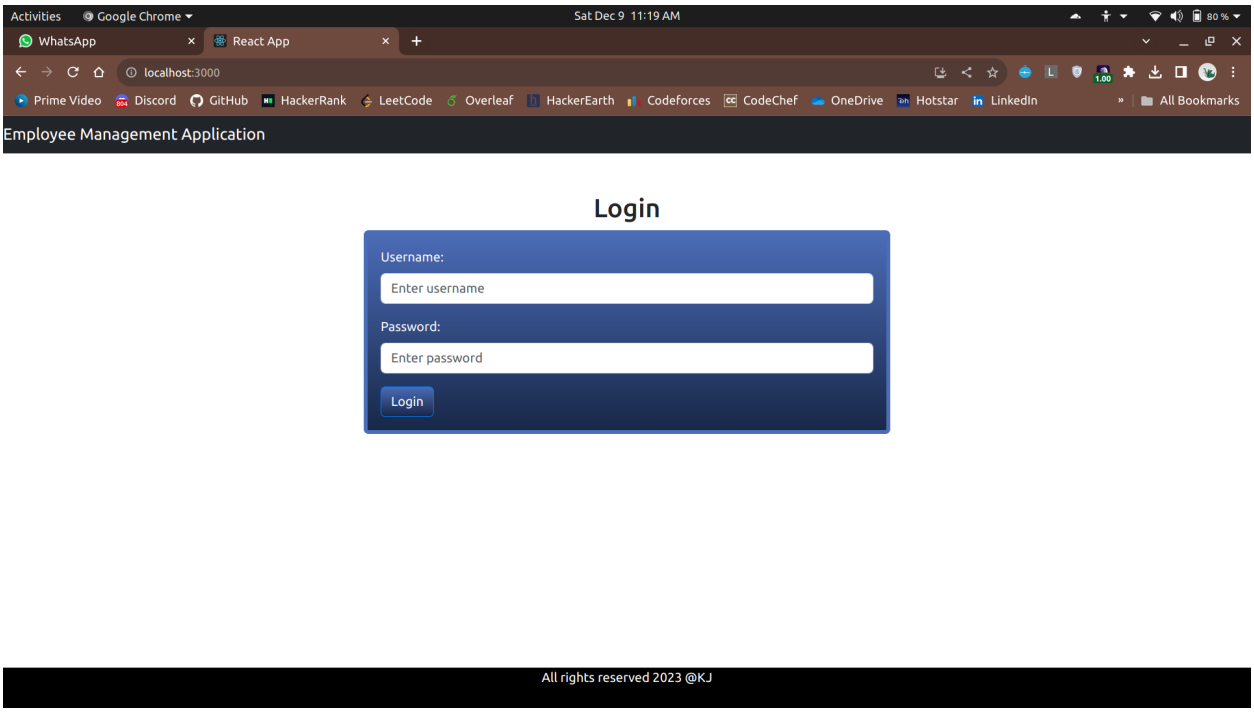


Figure 21: Login Page

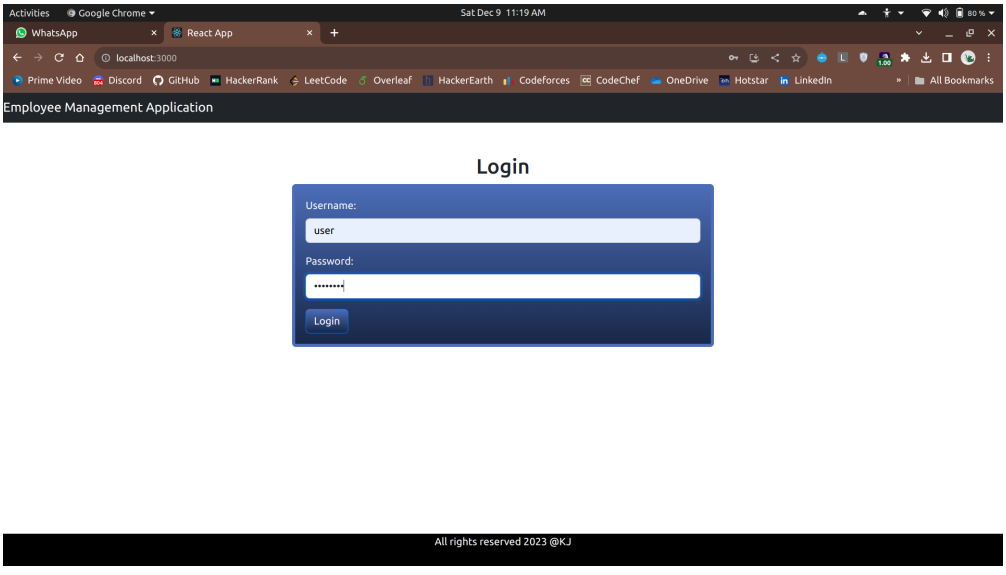


Figure 22: Enter credentials in the Login page

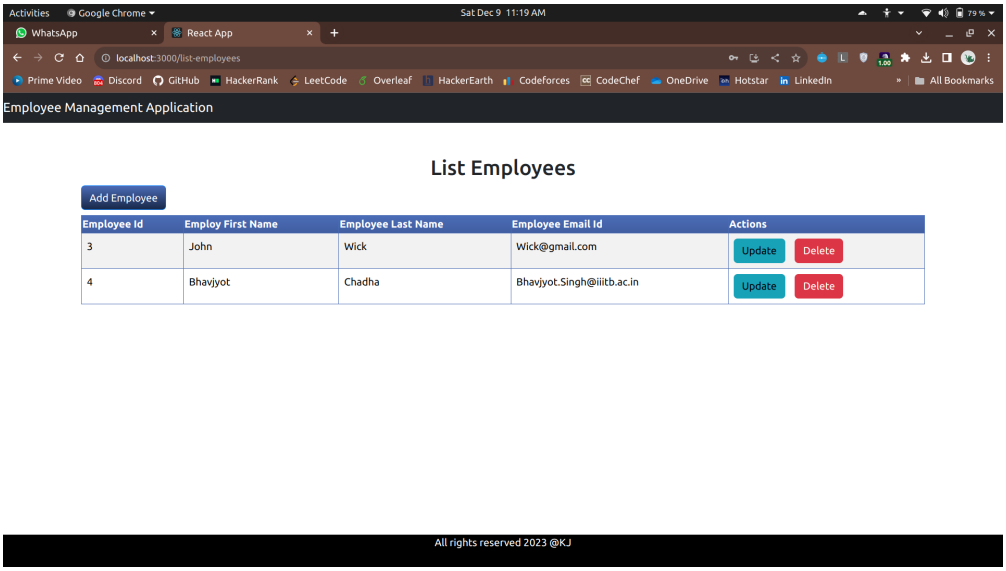


Figure 23: List of Employees

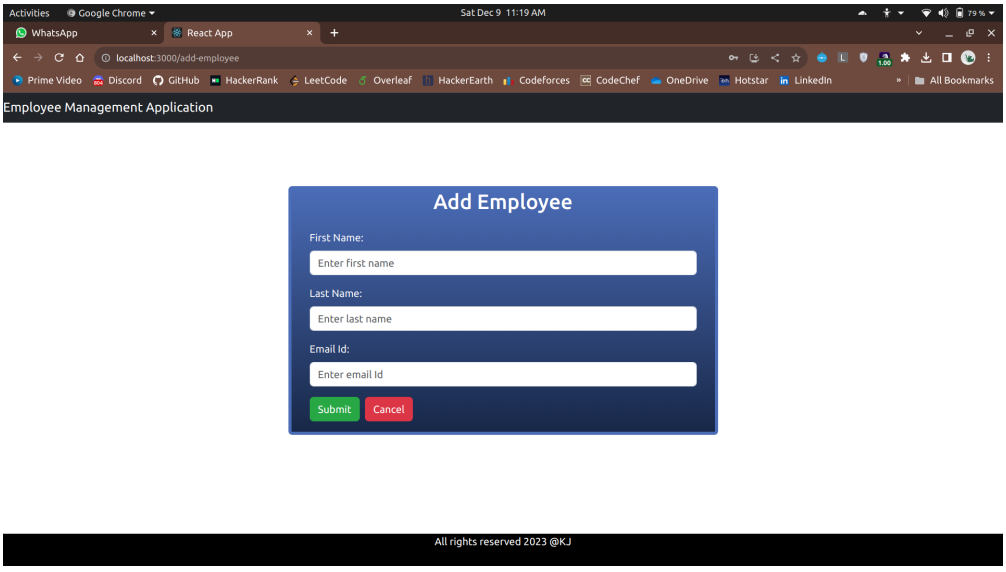


Figure 24: Add an employee to the database

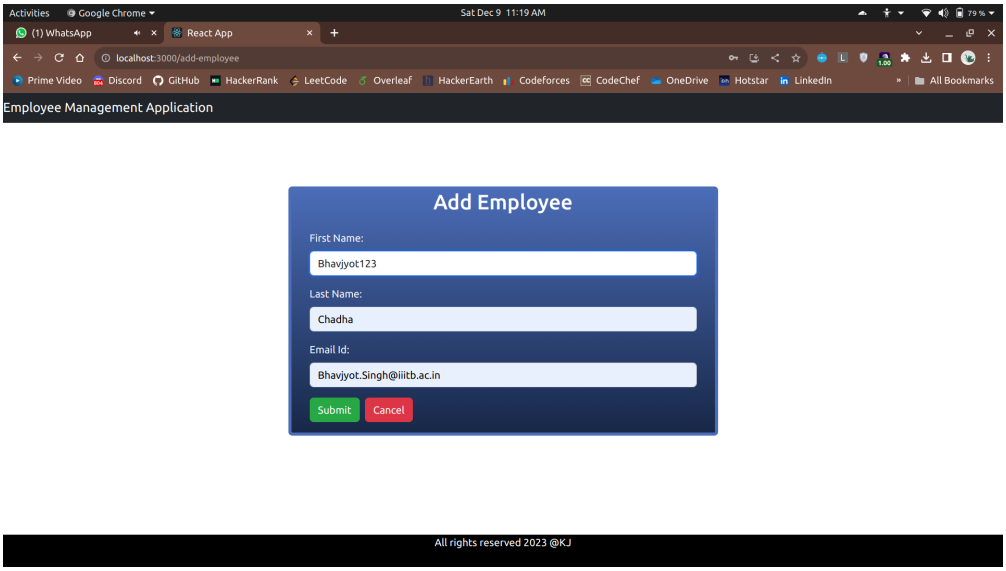


Figure 25: Adding credentials of the new employee

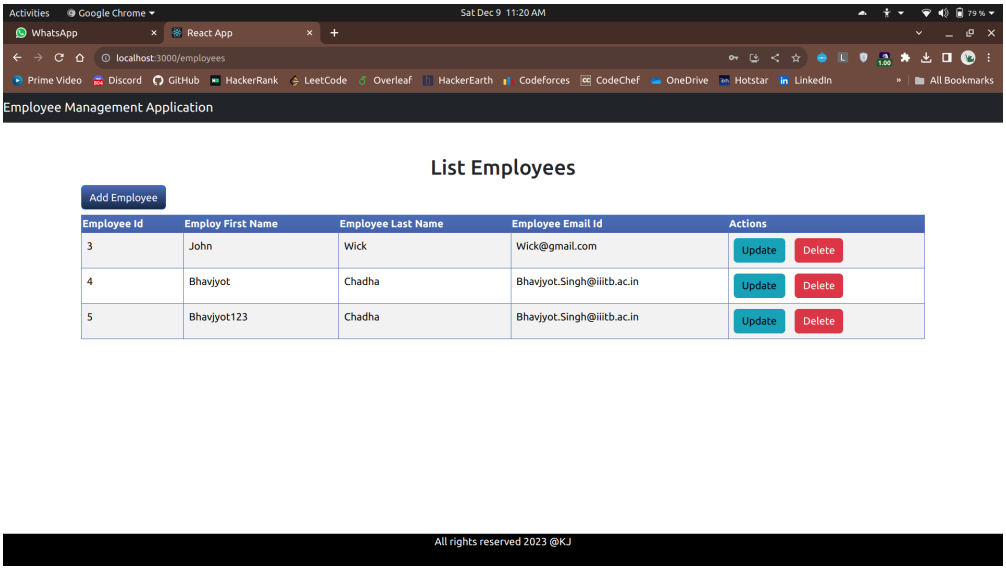


Figure 26: Employee added to the database

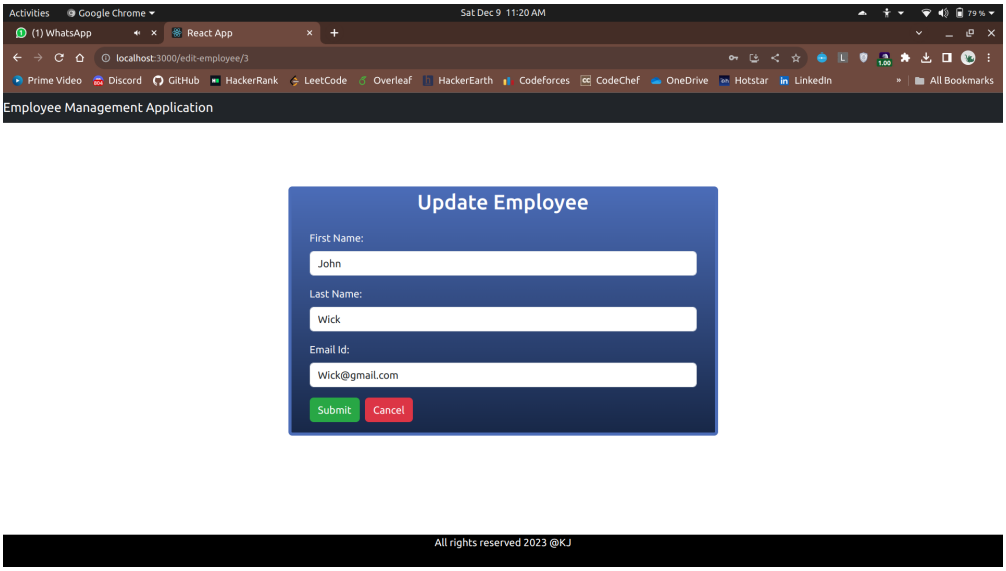


Figure 27: Update information of an existing employee

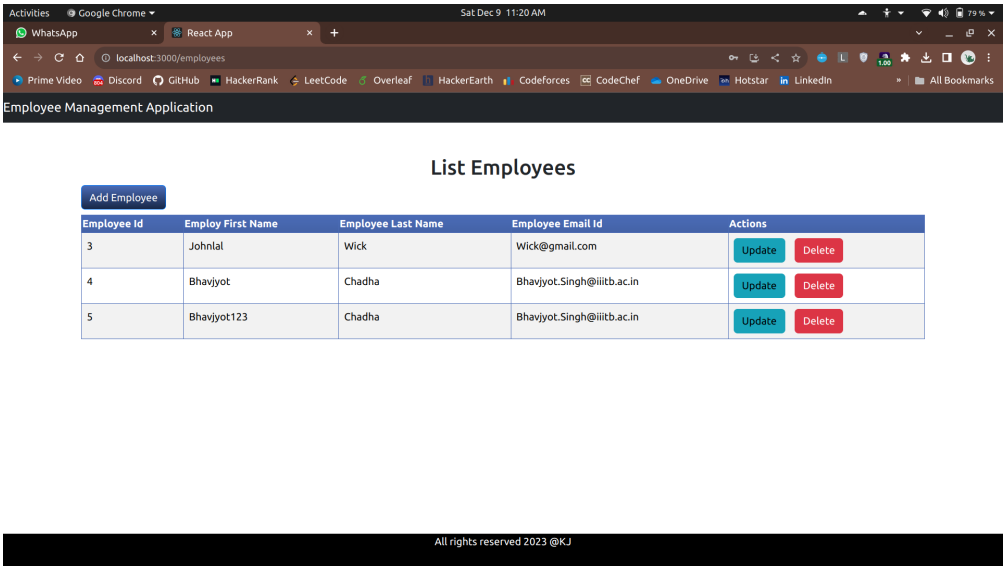


Figure 28: Updated information of the employee

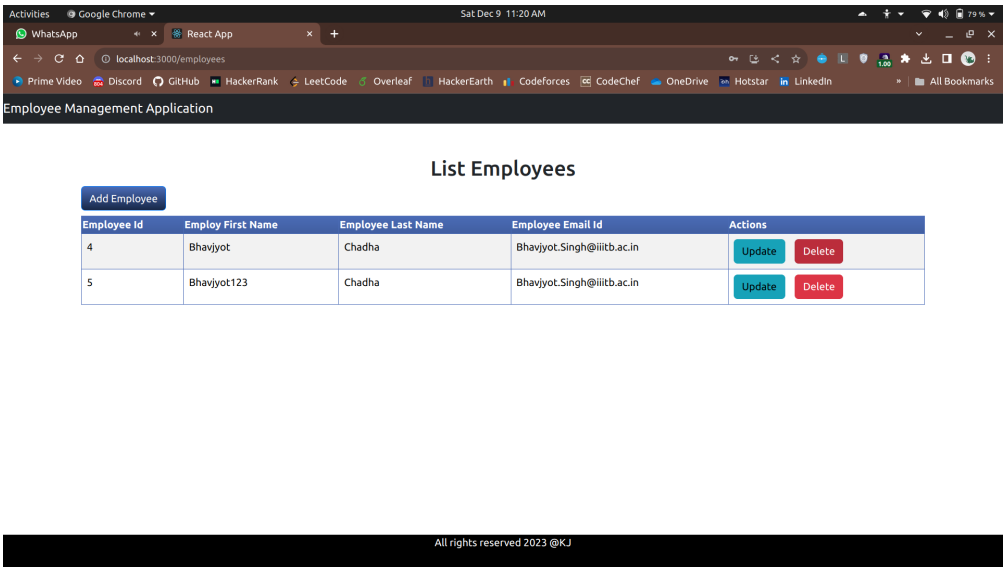


Figure 29: Delete employee from the database

8 Challenges

1. One of the biggest challenges was to establish real-time communication between the server and the clients.
2. The task of setting up an SSH connection between two systems was also a difficult task at first.
3. The user interface must be intuitive, easy to use, and visually appealing.
4. App required a robust and scalable data storage system to store user information. This includes using the database MySQL. Designing database models was a difficult task.
5. Configuring the various components of the ELK stack, namely Elasticsearch, Logstash, and Kibana was a complex process.
6. Integrating the ELK stack with the application being monitored was challenging.

9 Future Scope

The Employee Management System (EMS) project holds significant potential for future enhancements and expansions, catering to the evolving needs of workforce management. The following areas represent the future scope of the project:

1. **Advanced Security Features:** Strengthening security through the integration of advanced authentication mechanisms such as multi-factor authentication (MFA) or biometric authentication to safeguard sensitive employee data.
2. **Integration with HR Systems:** Exploring integration with broader Human Resources (HR) systems, connecting the EMS with payroll, performance management, and other HR-related applications for a more unified ecosystem.

3. **Mobile Application Development:** Developing a dedicated mobile application to extend accessibility, enabling employees and administrators to manage information on the go and incorporating features like push notifications.

4. **Data Analytics and Reporting:** Implementing data analytics tools to provide valuable insights into employee performance, attendance patterns, and other metrics. Incorporating reporting functionalities to support data-driven decision-making.

5. **Employee Self-Service Portal:** Expanding the system to include a self-service portal, empowering employees to update personal information, access performance records, and submit HR-related requests independently.

6. **Attendance and Leave Management:** Integrating an attendance and leave management system to automate tracking, approval, and reporting of employee attendance and leave requests.

7. **Performance Evaluation Module:** Adding a performance evaluation module for managers to conduct and track performance reviews, set goals, and provide feedback, contributing to a more comprehensive workforce management solution.

8. **Workflow Automation:** Exploring workflow automation for routine HR processes, such as onboarding and offboarding, to improve efficiency and reduce manual intervention.

9. **Machine Learning for Predictive Analysis:** Leveraging machine learning algorithms for predictive analysis to forecast employee trends, identify potential issues, and make proactive decisions.

10. **Scalability and Cloud Integration:** Designing the system with scalability in mind and exploring cloud integration to ensure efficient handling of a growing number of users and data volumes.

11. **Regulatory Compliance:** Regularly updating the system to comply with evolving labor laws and regulations, ensuring the organization remains compliant with legal requirements.

12. **Employee Well-being and Engagement:** Introducing features related to employee well-being and engagement, such as surveys and feedback mechanisms, to contribute to a positive organizational culture.

10 Links

1. Github Repo - GitHub Link
2. Frontend Image - Frontend Image
3. Backend Image - Backend Image

11 References

1. **Spring Boot -** Official Documentation: `https://spring.io/projects/spring-boot`
2. **Jenkins -** Official Website: `https://www.jenkins.io/`

3. **Docker -** Official Documentation: https://docs.docker.com/
4. **Ansible -** Official Documentation: https://docs.ansible.com/
5. **ELK Stack -** Official Documentation: https://www.elastic.co/
6. **ChatGPT**