WorkWize – Project Management Tool

Major Project

CS 816: Software Production Engineering

Guided By

B. Thangaraju
Professor
International Institute of Information Technology, Bangalore

Under the Assistance of

Samaksh Dhingra
Teaching Assistant

International Institute of Information Technology, Bangalore

Created by:

Vraj Jatin Naik
MT2023050
Vraj.Naik@iiitb.ac.in

Arjun Gangani
MT2023153
Arjun.Gangani@iiitb.ac.in

ABSTRACT

In industrial settings, traditional project management methods often lack efficiency and responsiveness due to manual processes and disjointed workflows. Existing systems may not sufficiently address the specific needs of industrial projects, leading to challenges in coordination, task assignment, and progress tracking. There is a growing demand for platforms that can accommodate real-time user feedback and updates.

The project "WorkWize" aims to address these challenges by developing a comprehensive project management platform tailored for industrial environments. The primary objective is to enhance project workflows and facilitate effective management by all stakeholders. WorkWize incorporates roles for employees, managers, and administrators, each with specific functionalities to streamline project execution and oversight.

Employees can view assigned tasks, update progress, and visualize their task history, while managers can create, update, and delete projects, track progress, and manage time and effort distribution. Administrators oversee user management and system administration, ensuring smooth operation and handling technical issues.

Utilizing a robust tool stack including AngularJS for the front end, Spring Boot for the back end, and MySQL for the database, WorkWize leverages DevOps tools such as GitHub, Jenkins, Docker, Docker Compose, Ansible, and the ELK stack to automate the CI/CD pipeline, ensure efficient application management, and provide comprehensive log analysis. This integration of tools and methodologies results in a streamlined, efficient, and responsive project management solution.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Workflow management is the process of optimizing data paths in a given process. It involves creating and optimizing the sequence of tasks required to complete a particular project. A Workflow Management System (WMS) automates this process by creating a form to hold data and automating a sequential path of tasks for the data to follow until it is fully processed. By automating repetitive processes, a WMS such as WorkWize frees up time and resources for more important tasks. Additionally, it ensures that uncompleted tasks are automatically followed up on, reducing the risk of errors or delays.

WorkWize is an efficient WMS that provides a range of features for workflow management and automation. Its customizable forms allow for the creation of unique workflows that meet the specific needs of any project. The system provides real-time monitoring of progress, allowing for easy tracking of project status and identification of potential bottlenecks. The system also generates detailed performance metrics, providing insights into how to optimize the workflow further.

The scope of WorkWize is broad, as it is suitable for any industry, whether small-scale or large-scale, that requires workflow management and automation. By using WorkWize, organizations can increase productivity, improve efficiency, and reduce errors and delays.

2. APPLICATION FEATURE

2.1 ROLES

I. Employee:

Role Description: Employees execute assigned tasks within projects under the guidance of managers.

II. Manager:

Role Description: Managers oversee projects and task assignments, ensuring smooth workflow and project progress.

III. Admin:

Role Description: Administrators have the highest level of access and are responsible for managing users, including adding and removing employees and managers, as well as overseeing the overall system administration.

2.2 ROLE-WISE FUNCTIONALITIES

I. EMPLOYEE:

- **Tasks Related:**

Description: Employees can view tasks assigned to them.
Actions:

i. View Assigned Tasks:
Description: Employees can see tasks assigned to them under projects.

ii. Update Task Progress:
Description: Employees can update the progress status of tasks assigned to them.

iii. Visualize Personal Progress:
Description: Employees can visualize their progress as per the tasks assigned to them.

- **View Profile:**

  Description: Employees can view their personal details and task history.

  Actions:
  - View personal details
  - View task history

- **Reset Password:**

  Description: Employees can reset their passwords if forgotten.

  Actions:
  - Request password reset
  - Set new password

II. MANAGER:

- **Create, Update, and Delete Projects:**

  Description: Managers can create new projects, update existing projects, and delete projects they have created.

  Actions:
  - Create new projects
  - Edit existing projects
  - Delete projects from the system

- **Manage Employees:**

  Description: Managers can add employees to projects and manage them by assigning tasks related to the project.

  Actions:
  - Add employees to projects
  - Assign tasks to employees

- **Project Progress Monitoring:**

  Description: Managers can track the progress of their projects.

  Actions:
  - View project progress

- **Time and Effort Distribution:**

  Description: Managers can allocate and adjust time distribution and visualize effort distribution.

  Actions:
  - Give/edit distribution of time in the project effort table
  - Visualize effort distribution

- **Reset Password:**

  Description: Managers can reset their passwords if forgotten.

  Actions:
  - Request password reset
  - Set new password

III. ADMIN:

- **Manage Users:**

  Description: Admins can add, view, edit, and remove users (both employees and managers) from the system.

  Actions:
  - Add new users
  - View user profiles
  - Edit user profiles
  - Delete user accounts

- **System Administration:**

  Description: Admins oversee the overall system administration, ensuring its smooth operation and handling any technical issues.

  Actions:
  - Handle system configurations
  - Perform maintenance tasks
  - Address technical issues and escalations

3. ARCHITECTURE

3.1 SYSTEM ARCHITECTURE

- **FRONTEND**
    - **Framework:** Angular
    - **Description:** The frontend is a single-page application (SPA) built using Angular. It handles user interactions, renders dynamic content, and communicates with the backend via RESTful APIs.

- **BACKEND**
    - **Framework:** Spring Boot
    - **Description:** The backend consists of Spring Boot Web REST Controllers. It processes HTTP requests from the frontend, handles business logic, and interacts with the database to retrieve and store data.

- **DATABASE**
    - **Database Management System:** MySQL
    - **Description:** All application data is stored in a MySQL database. This includes user information, application settings, transaction records, and other relevant data.

- **CONTAINERIZATION AND DEPLOYMENT**
    - **Containerization Tool:** Docker
    - **Container Registry:** DockerHub
    - **Deployment Tool:** Ansible
    - **Container Orchestration**: Docker Compose

- **DEPLOYMENT WORKFLOW**
    - **Development:** Developers create and test the Angular frontend and Spring Boot backend locally.
    - **Dockerization:** Separate Docker images are created for the frontend, backend, and database.
    - **Publishing:** Docker images are published to DockerHub.
    - **Deployment:** Using Ansible, the Docker images are deployed from DockerHub to an Ubuntu Server.
    - **Orchestration:** Docker Compose is used on the Ubuntu Server to manage and run the containers, ensuring they work together within the same network.

3.2 SYSTEM CONFIGURTION

3.2.1 PROJECT TECHNOLOGY STACK

- **FRONTEND**

  - **Framework**: Angular
  - **Languages**: HTML, CSS, TypeScript
  - **Build Tool**: npm
  - **Description**: The frontend of Work Wize is a single-page application built using Angular. It manages user interface components, handles user interactions, and communicates with the backend via RESTful APIs.

- **BACKEND**

  - **Framework**: Spring Boot Web Service
  - **Languages**: Java
  - **Build Tool:** Maven
  - **Description:** The backend is implemented using Spring Boot, which provides RESTful web services. It handles business logic, processes HTTP requests from the frontend, and interacts with the MySQL database.

- **DATABASE**

  - **Database Management System:** MySQL
  - **Description:** MySQL is used as the relational database management system for storing and managing the application's data. This includes user data, application settings, and transaction records.

- **TESTING**

  - **Unit Testing:** JUnit (for backend), Karma (for frontend)
  - **Behavioral Testing:** Jasmine (for frontend)
  - **Description:** Testing frameworks are used to ensure code quality and functionality. JUnit is used for backend testing, while Karma and Jasmine are used for frontend testing.

- **LOGGING**

  - **Technology:** Log4j2
  - **Description:** Log4j2 is used for logging application events, which helps in monitoring and debugging the application. It provides a reliable and flexible logging framework.

- **API DOCUMENTATION**

  - **Platform:** Postman API Platform
  - **Description:** Postman is used for documenting and testing APIs. It provides a user-friendly interface to create, share, test, and document APIs, ensuring clear communication and understanding of the API endpoints.

- **INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES)**

  - **IDEs:** IntelliJ, VS Code
  - **Platform:** Postman
  - **Description:** Development is conducted using IntelliJ and VS Code for coding, with Postman utilized for API testing and documentation.

- **PROGRAMMING LANGUAGES AND FRAMEWORKS**

  - **Languages:** Python, JavaScript (React Library - npm runtime environment), Java (Spring Boot framework)
  - **Description:** Various programming languages and frameworks are used in different parts of the application to leverage their respective strengths.

- **BUILDING TOOLS**

  - **Tools:** Maven, npm
  - **Description:** Maven and npm are used for building and managing dependencies in the backend and frontend, respectively.

3.2.2 INFRASTRCTURE DETAILS

- **OPERATING SYSTEM:**

  Ubuntu 20.04 (Both Host Machine and Controller Machine / Server)

  - Description: The application is hosted and managed on Ubuntu 20.04 servers, providing a stable and secure environment for deployment.

- **HARDWARE CONFIGURATION**

  - **CPU:** 8-core Processor
  - **RAM:** 8GB RAM (16GB of RAM is preferable)
  - **Description:** The server hardware specifications ensure adequate performance and scalability for the application.

3.2.3 DEV-OPS TOOLS

- **SOURCE CONTROL MANAGEMENT**:
    - **Tool**: GitHub
    - **Description:** GitHub is used for version control and source code management, facilitating collaboration among developers.

- **CONTINUOUS INTEGRATION:**
    - **Tool:** Jenkins
    - **Description:** Jenkins is used for continuous integration, automating the process of building and testing the application.

- **CONTAINERIZATION:**
    - **Tool: Docker**
    - **Description:** Docker is used to containerize the application, ensuring consistency across different environments.

- **CONTINUOUS DEPLOYMENT:**
    - **Tool:** Ansible
    - **Description:** Ansible automates the deployment process, ensuring reliable and repeatable deployments.

- **MONITORING:**
    - **Tool**: ELK stack (Elasticsearch, Logstash, Kibana)
    - **Description:** The ELK stack is used for monitoring and analyzing logs, providing insights into the application's performance and operational health.

4. DESIGN

4.1 USE CASE DIAGRAM

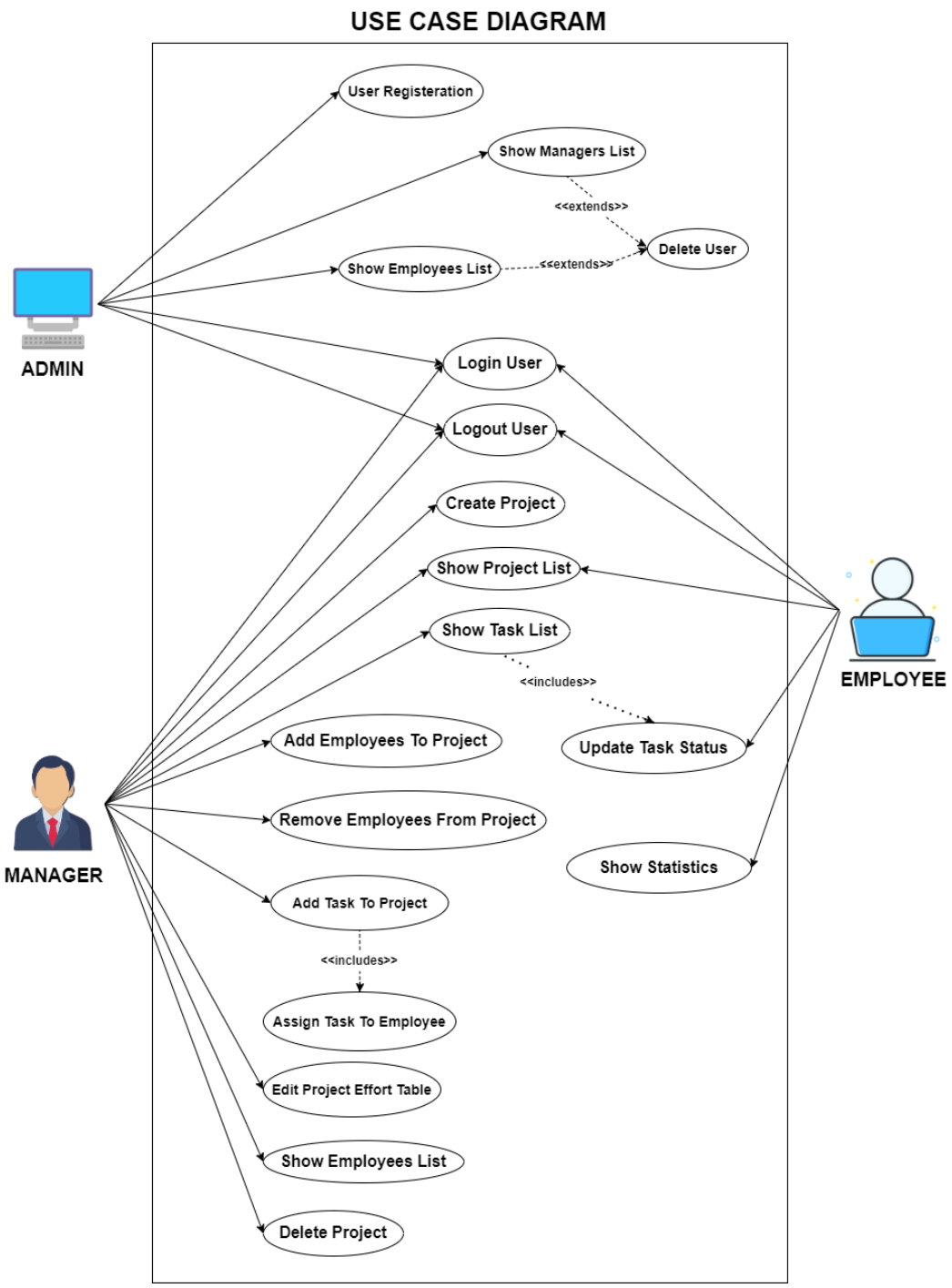

Figure 4-1Use Case Diagram

4.2 CLASS DIAGRAM

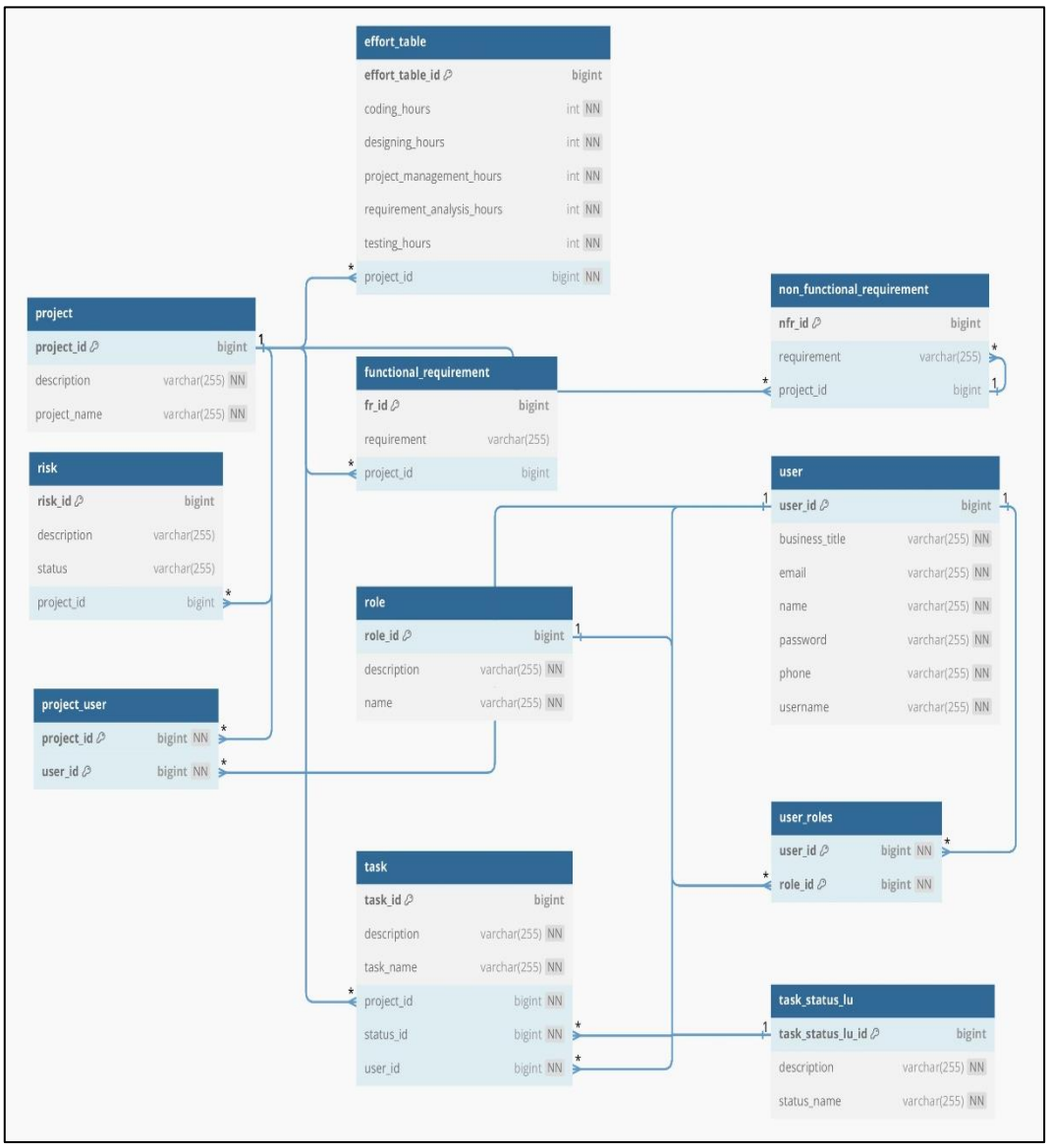

Figure 4-2 Class Diagram

5. SOFTWARE DEVELOPMENT LIFE CYCLE

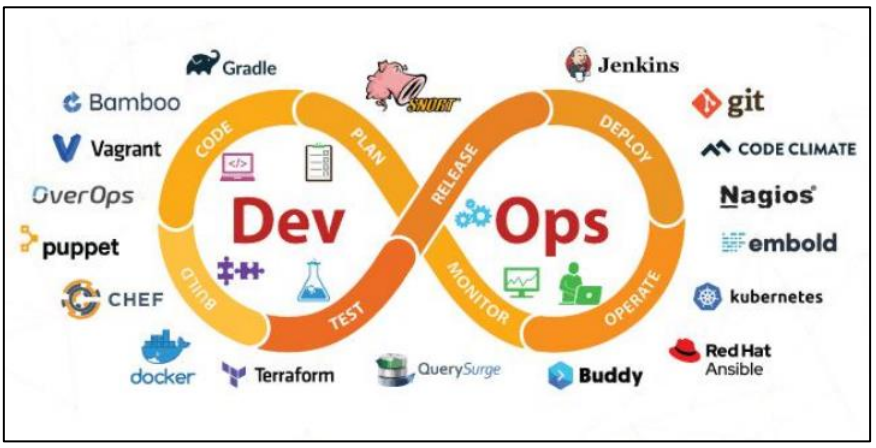

Figure 5-1 DevOps Tool Chain

In the development lifecycle of WorkWize, DevOps practices are seamlessly integrated to automate and enhance efficiency across various stages. WorkWize leverages Git and GitHub for code management, enabling collaborative version control and efficient code management. Maven orchestrates the build process, while JUnit facilitates automated testing, ensuring high code quality and functionality. For delivery, WorkWize utilizes DockerHub as the central repository for storing and managing Docker images, simplifying the delivery process. Ansible automates deployment, ensuring consistency and reliability, with WorkWize deployed within Docker containers on Ubuntu servers. Additionally, the ELK Stack, comprising Elasticsearch, Logstash, and Kibana, is utilized for monitoring and visualizing logs, enabling effective monitoring and troubleshooting. Through this integrated approach, WorkWize benefits from increased efficiency, faster deployment cycles, and improved collaboration among development and operations teams.

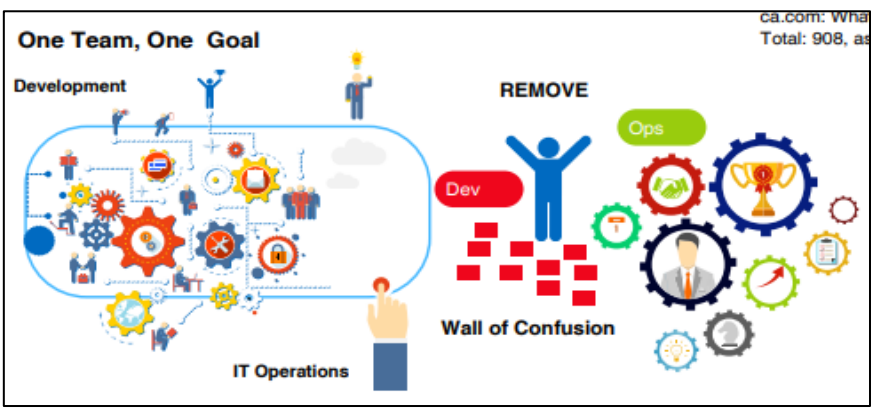

Figure 5-2 DevOps Benefits

5.1 INSTALLATION AND SETUP:

5.1.1 GITHUB

1. **Create a new repository on GitHub**:

- Log in to your GitHub account.
- Click on the "+" icon on the page and select "New repository".
- Fill in the repository name (e.g., "WorkWize"), add an optional description, choose visibility (public or private), and initialize with a README if needed.
- Click on "Create repository".

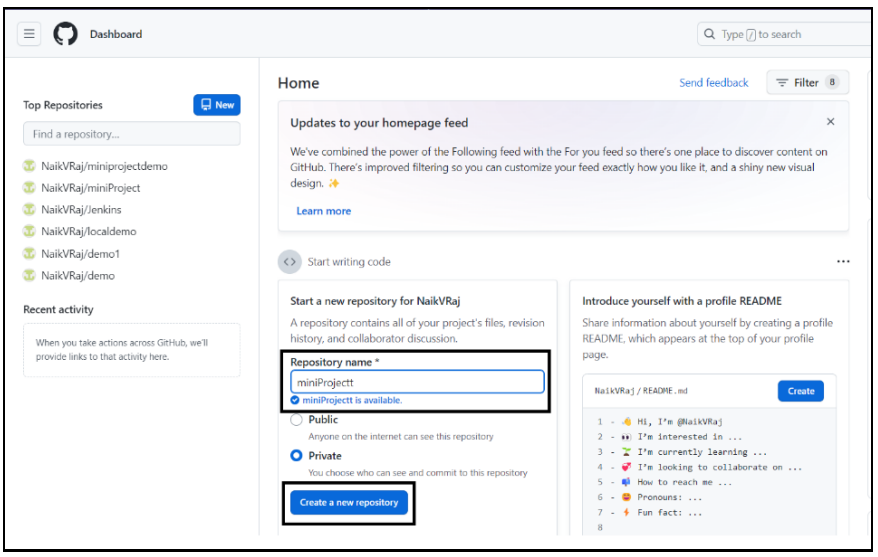

Figure 5-3 GitHub new Repository

2. **Set up local repository and link it to the remote repository:**

- Open your terminal or command prompt.
- Navigate to the directory where you want to create your local repository.
- Execute the following commands:

```
echo "# WorkWize" >> README.md
git init
git add README.md
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/your-username/mini
                -Project.git
git push -u origin main
```

3. **Develop code and manage source code:**

- Write your code according to project requirements within your local repository.
- After making changes, stage the changes for commit.

  git add .

- Commit the changes along with a descriptive message:

  git commit −m "Your descriptive message here"

- Push the changes to the remote repository:

  git push origin main

4. **Viewing commit history:**

- To view commit history, you can use the following command:

  git log

- This will display a list of commits along with their commit IDs, authors, dates, and commit messages.
- You can also use various options with git log to customize the output as needed.

5. **Checking out to a previous commit:**

- To go back to a previous commit, you can use the following command:

  git checkout < commit_id >

- Replace <commit_id> with the commit ID you want to revert to.

- After checking out to a previous commit, you can create a new branch if you want to make changes from that point onwards without affecting the current branch.

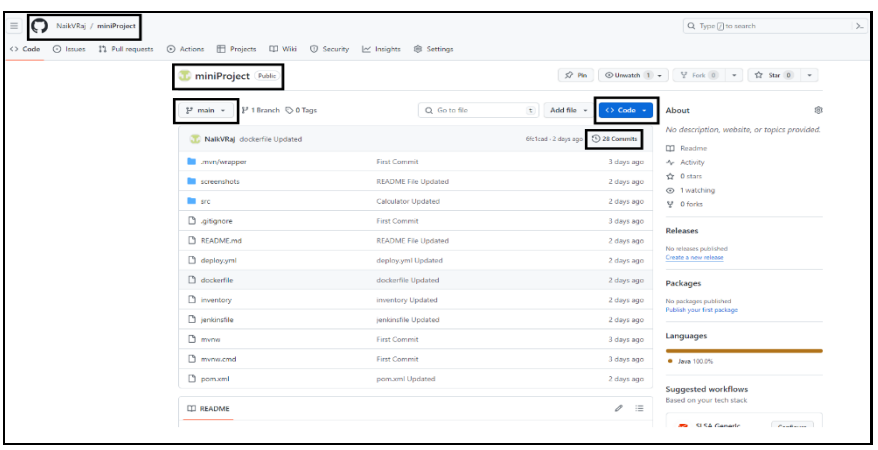

Figure 5-4 Repository Page

Commit History Commit-id Commit by whom? When? Commit message

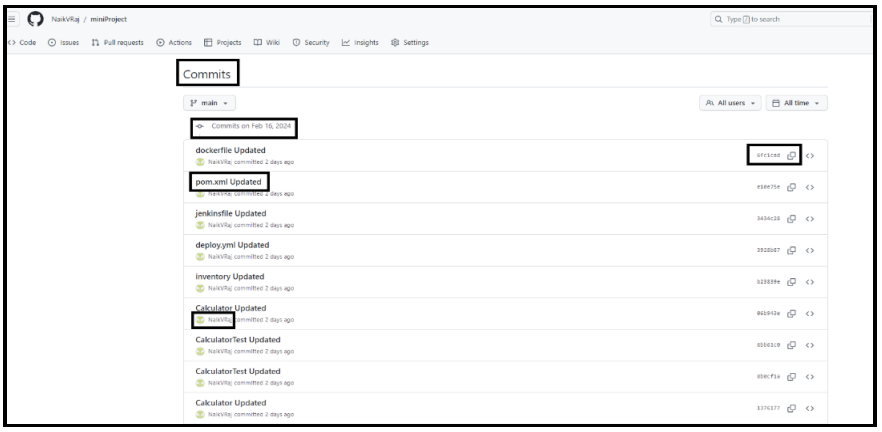

Figure 5-5 Commit History

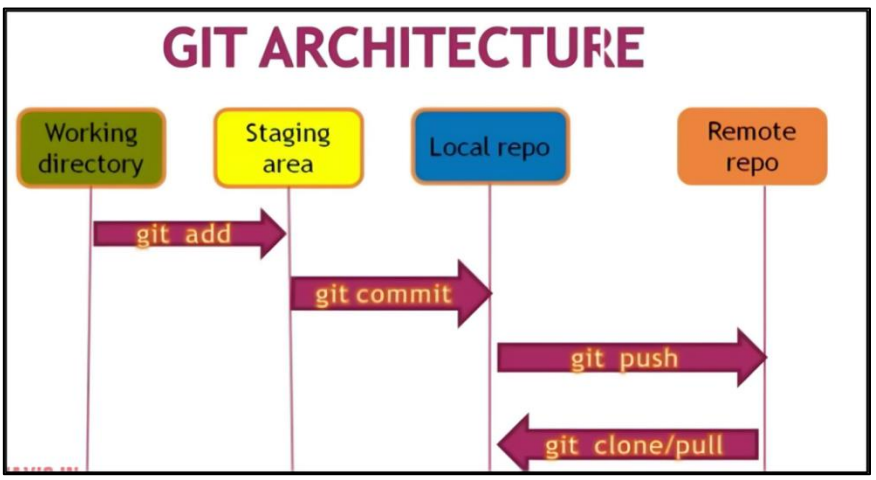

Figure 5-6 Git Architecture

5.1.2 JENKINS

1. Update Package List and Install Java:

- Update Package List: This ensures that your system has the latest information about available packages and their versions. It's crucial before installing any new software.

- Install Java: Jenkins requires Java to run. Installing OpenJDK 17 ensures that the Java runtime environment is available on your system, which Jenkins will utilize.

sudo apt update

sudo apt install openjdk − 17 − jdk

2. Add Jenkins Repository Key and Repository:

- Add Jenkins Repository Key: This step adds the Jenkins repository key to your system, allowing your package manager to verify the integrity of Jenkins packages during installation.

- Add Jenkins Repository: By adding the Jenkins repository to your system's package sources, you enable your package manager to find and install Jenkins packages.

$$wget\ -q\ -O\ -\ https://pkg.jenkins.io/debian\\ -stable/jenkins.io.key\ |\ sudo\ apt-key\ add\ -$$

$$sudo\ sh\ -c\ 'echo\ deb\ https://pkg.jenkins.io/debian\\ -stable\ binary/\\ >\ /etc/apt/sources.list.d/jenkins.list'$$

3. Install Jenkins:

- Update Package List Again: Refreshes the package list to include the newly added Jenkins repository.

- Install Jenkins: This command installs the Jenkins continuous integration server on your system, allowing you to use Jenkins for automation and building projects.

$$sudo\ apt\ update$$
$$sudo\ apt\ install\ jenkins$$

4. Start and Check Jenkins Service:

- Start Jenkins Service: Initiates the Jenkins service, allowing it to run in the background and serve web requests.

- Check Jenkins Service: Verifies whether the Jenkins service has started successfully and is currently running.

sudo service jenkins start

sudo service jenkins status

5. Access and Unlock Jenkins:

- Access Jenkins: Opens Jenkins in your web browser, allowing you to interact with its web-based interface.

- Unlock Jenkins: Jenkins is initially locked to prevent unauthorized access. Finding and entering the initial administrator password unlocks Jenkins, enabling you to configure it.

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

6. Setup Jenkins and Install Essential Tools:

- Follow Setup Wizard: The setup wizard guides you through the initial configuration of Jenkins, such as setting up the admin user and selecting plugins.

- Install Required Plugins: Installing essential plugins like Maven, Git, Ansible, and Docker ensures that Jenkins has the necessary functionality for building and deploying projects.

Figure 5-7 Ansible Installation

Figure 5-8 Maven Installation

Figure 5-9 Docker Installation

7. Install Additional Required Plugins:

- Navigate to Jenkins dashboard and go to "Manage Jenkins" → "Manage Plugins".
- Select the "Available" tab and search for the following plugins: Maven, Git, Ansible, Docker.
- Install these plugins to enable their functionality within Jenkins.

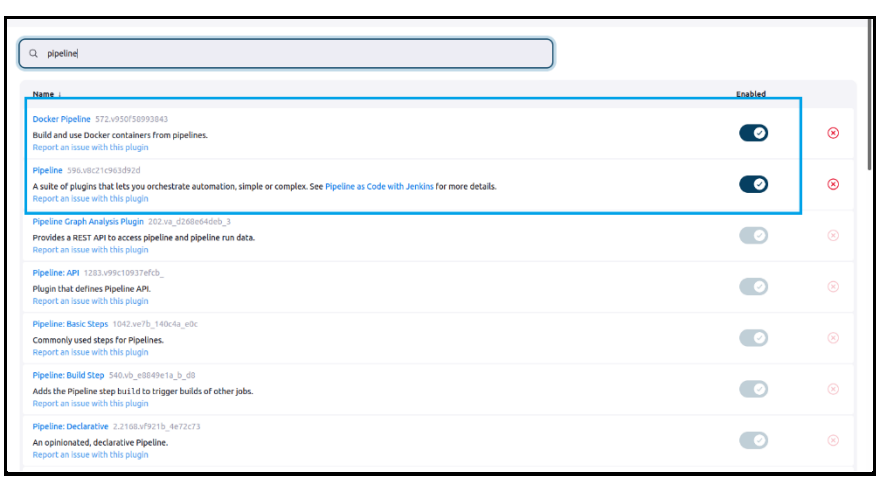

Figure 5-10 Pipeline Plugin Installation

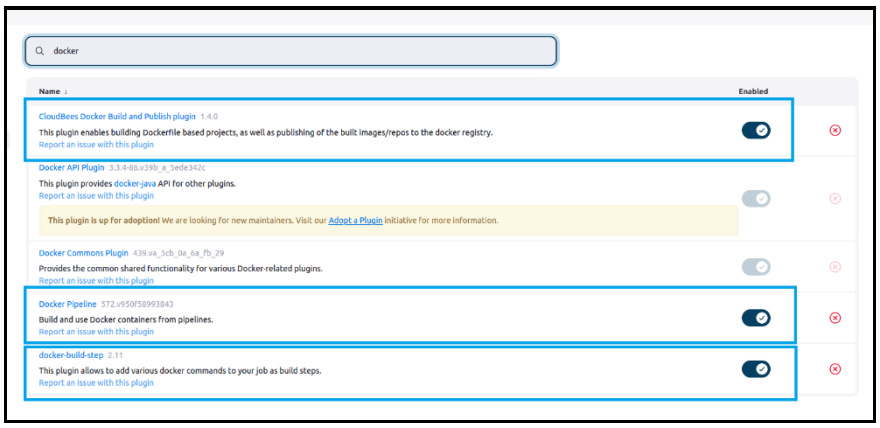

Figure 5-11 Docker Plugin Installation

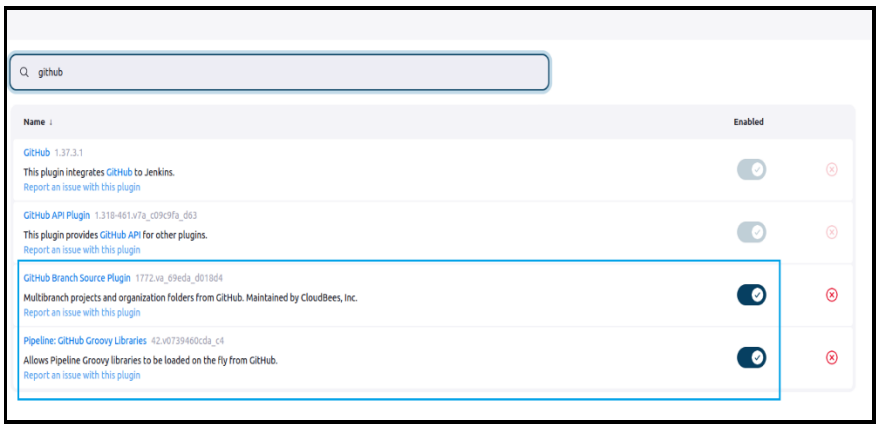

Figure 5-12 Github Plugins Installation

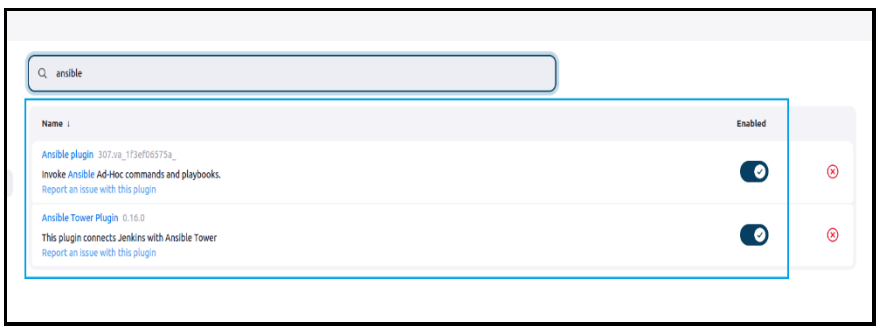

Figure 5-13 Ansible Plugins Installation

8. Add Global Tool Configuration from Manage Jenkins:

- Go to “Manage Jenkins” → “Global Tool Configuration”.
- Configure the tools as follows:

  - For Maven:

    Set the name as “maven”.

    Specify the MAVEN_HOME directory.

- For Git:

  Add the Git executable path.

- For Ansible and Docker:

  If applicable, specify the executable paths for Ansible and Docker.

9. Configure Jenkins location and GitHub server:

- For Jenkins Location:
  - Set the system administrator email address to a Gmail account.
- For GitHub Server:
  - Go to “Manage Jenkins” → “Configure System”.
  - Scroll down to the "GitHub" section.
  - Click on "Add GitHub Server" and provide credentials for accessing GitHub repositories.
  - Configure additional settings as necessary, such as webhook management and advanced connection settings.

Figure 5-14 Adding Admin e-mail Address

Figure 5-15 Adding GitHub Server Credentials

10. Add Credentials:

- Go to “Manage Jenkins” → “Manage Credentials”.
- Under “System”, click on “Global credentials”.
- Add credentials for GitHub and Docker Hub:
- For GitHub:
    - Add a username/password credential or SSH key credential with access to the repository.
- For Docker Hub:
    - Add a username/password credential with access to Docker Hub.

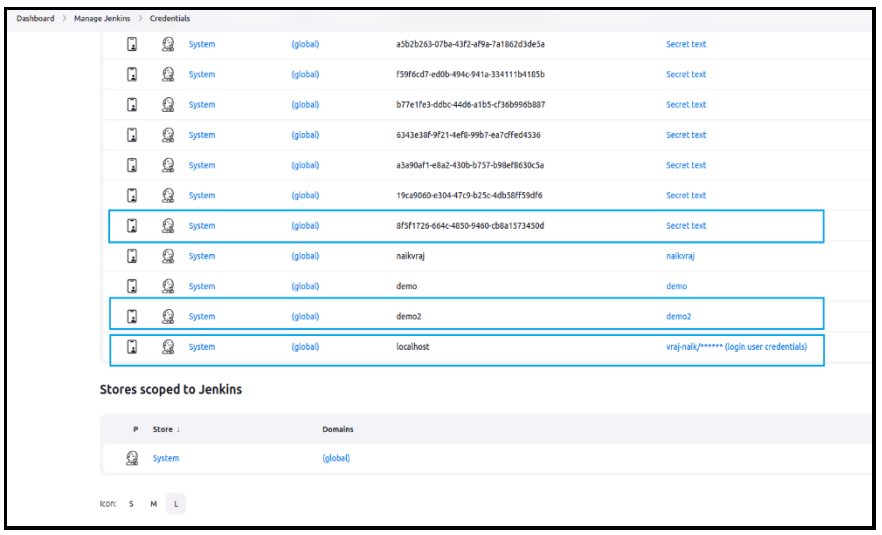

Figure 5-16 Selecting Required Credentials

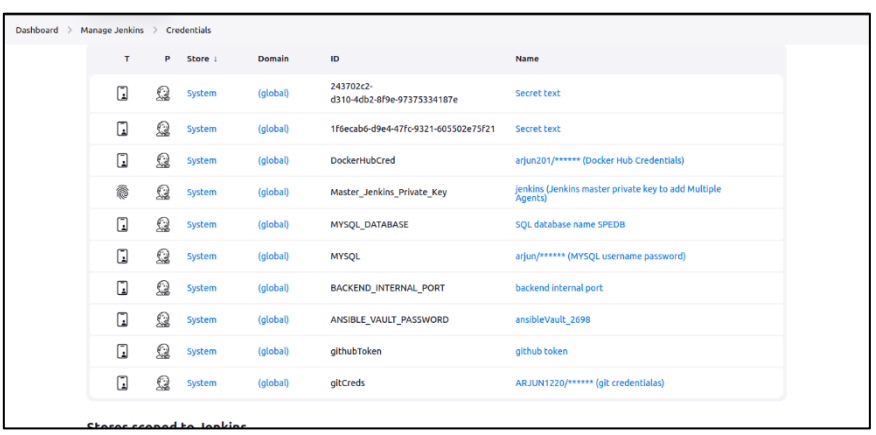

Figure 5-17 Credentials of other Services

5.1.3 DOCKER

Docker is a tool that can package an application and its dependencies in a virtual container that can run on any linux server. This will enable flexibility and portability on where the application can run either on premise, public/private cloud, base metal etc. Docker is a platform which provides OS level virtualization to deliver software as packages. Our goal is to create a docker image of the project which contains all the source code, libraries and dependencies packaged as an immutable file. We then push this latest image onto the docker hub. This pushing procedure can be automated after every build step which automatic call removes previously pushed docker images and replaces it with the latest build one.

In order to install docker on our system we need to follow below commands:

- Install & Start Docker:

  sudo apt install docker. io

  sudo systemctl start docker

  sudo systemctl status docker

- To display all docker images in the system:

  sudo docker images

- Docker Build Image: (Assuming Current Directory has Dockerfile)

  docker build − t imagename .

- We need to make sure that we add Jenkins to the docker group so that jenkins can use docker for build docker image.
- To do this use the below command

  sudo username − aG docker jenkins

Docker file: We use dockerfile to create a docker images. The dockerfile tells the build should be built on what image. In the dockerfile we mention all the commands in succession which needs to be run in order to create an image.

5.1.4 ANSIBLE

- For working with the ansible we have some prerequisites to be followed. We need to have the python and SSH installed on the managed nodes and the ansible engine.
- For installing SSH use the below command:

  sudo apt install openssh − server

- To generate the SSH key use the below command:

  $$ssh - keygen - t\ rsa$$

- To install ansible using the following command:

  $$sudo\ apt\ install\ ansible$$

- Once the ssh is installed and the playbooks are written, we shall try to make the Jenkins automatically login to our managed node and the ansible plugin in the Jenkins can run the playbook. For this, we have to copy the ssh key that was generated on the remote server to our Jenkins server.

- This can be done from the below command

  $$sudo\ su\ jenkins$$

5.1.5 DOCKER COMPOSE

- Make sure you have Docker installed on your system before proceeding with Docker Compose installation.

- On Linux
- Download the Docker Compose Binary:

  $$curl - L\ \text{https://github.com/docker/compose/releases/download/v2.18.1/docker-compose-\$(uname -s)-\$(uname -m)} - o\ /usr/local/bin/docker - compose$$

- Apply executable permissions to the binary:

  $$sudo\ chmod + x\ /usr/local/bin/docker - compose$$

- Verify the installation:

  $$docker - compose - version$$

5.1.6 ANGULAR

- Update the System:

  $$sudo\ apt - get\ update$$
  $$sudo\ apt - get\ upgrade$$

- Prerequisite installations:

  Before starting with Angular, make sure that Node.js is already installed in the system.

$$sudo\ apt - get\ install\ software - properties - common$$
$$curl - sL\ https{:}//\ deb\ .nodesource.com/setup\_12.x\ |\ sudo - E\ bash -$$

- Install Angular CLI

  $$npm\ install - g\ @angular/cli$$

- Verify angular installations

  $$ng\ --version$$

- Running the application on localhost

  $$ng\ serve$$

5.1.7 MAVEN

1. Install Maven:
   - Maven is a build automation tool used primarily for Java projects, including Spring Boot applications.

     $$sudo\ apt - get\ update$$
     $$sudo\ apt - get\ install\ maven$$

   - Verify the installation
     Check the installed version of Maven to ensure its installed correctly.

     $$mvn - v$$

2. Running the Spring Boot Application Locally:

   - Once Maven is installed, you can use it to run your Spring Boot application locally.
     Navigate to your Spring Boot project directory

   - Clean and install the project
     Use Maven to clean the project and install all dependencies.

   - Run the Spring Boot application
     Run your Spring Boot application using Maven.

     $$cd\ /path/to/your/spring - boot - project$$
     $$mvn\ clean\ install$$
     $$mvn\ spring - boot{:}run$$

5.2 FRONT END CODE

Angular is a TypeScript-based free and open-source web application framework, maintained by Google. Its primary purpose is to develop single-page applications (SPAs). As a framework, Angular offers several clear advantages, including robustness, scalability, and maintainability. It provides a standard structure for developers to work with, facilitating efficient development and collaboration. One of Angular's key features is its component-based architecture, which promotes modularity and code reusability. Additionally, Angular comes with a comprehensive ecosystem of tools and libraries, such as Angular Material for UI components and RxJS for reactive programming. These features enable users to create large applications in a maintainable manner, with built-in support for features like routing, forms, HTTP client, and state management. Overall, Angular empowers developers to build modern, feature-rich web applications with ease and efficiency.

- **DIRECTORY STRUCTURE:**
  - All the building components are stored in /components directory
  - All the Services which are responsible for API Calls are stored in /services directory
  - All the Routings are configured in app-routing.module.ts file
  - All the modules are in app.modules.ts
  - Testing is enabled done in *spec.ts files in each component.
  - All the assets are stored in /assets folder - Router Outlet is configured in
  - app.component.html

- **EXPLORING COMPONENTS:**
  - For creating new Component

    $$ng\ g\ c < path > / < filename >$$

  - Following files are created on generating new component:
    a. *.html -> Component HTML File
    b. *.scss -> Component Styling File
    c. *.spec.ts -> Component testing and spec file
    d. *.ts -> Component Typescript file

- **EXPLORING SERVICE FILE:**
  - For creating a service:

    $$ng\ g\ s < path > / < filename >$$

  - Here “g” stands for generate and “s” stands for service
  - Following files are created:
    a. *.service.ts -> Typescript file for writing services
    b. *.service.spec.ts

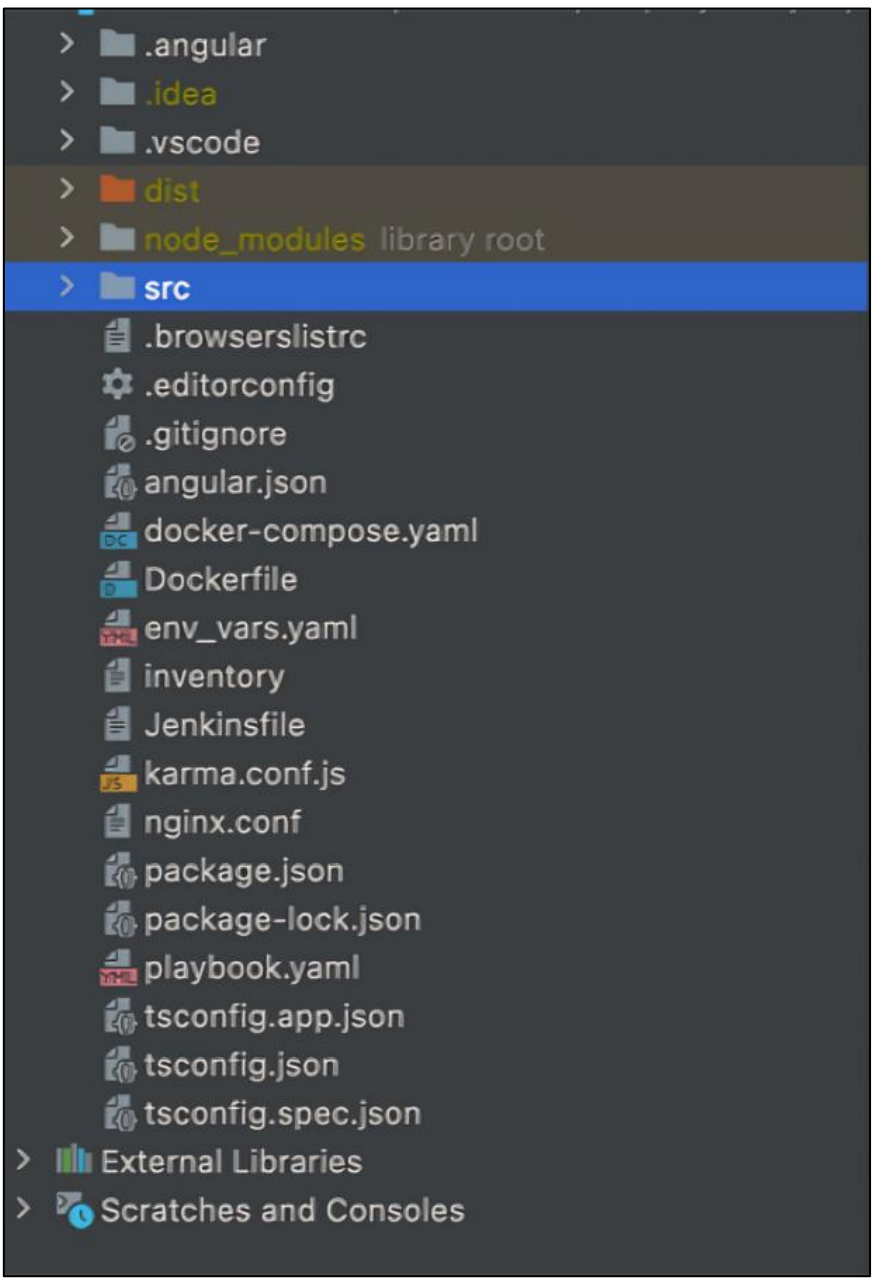

Figure 5-18 Front End Directory Structure

- **API CALLS:**
  - **Making the API Calls to the Backend:**

```
addUser(userData: any) {
    let token = localStorage.getItem("SessionUser");
    let header = new HttpHeaders({
        Authorization: "Bearer " + token
    });

    return this.http.post(`${this.url}/users/register`,
userData, { 'headers': header });
}
```

Figure 5-19 API Call to Back End

  - **Calling the API from service file to Component:**

```
getEmployeeData() {
    this.adminService.getEmployeeData().subscribe(
        (response: any) => {
            console.log(response);
            this.employeeData = [];
            response.forEach((element: any) => {
                let employeeDetails = {
                    id: element.id,
                    name: element.name,
                    email: element.email,
                    businessTitle: element.businessTitle
                };
                this.employeeData.push(employeeDetails);
            });
            // this.displayManager = true;
            this.employeeDataSource.data =
this.employeeData;
            console.log(response);
        },
        (error: any) => {
            this._snackBar.open(error, 'Close', {
                horizontalPosition:
this.horizontalPosition,
                verticalPosition: this.verticalPosition,
                duration: 2 * 1000,
            });
        }
    );
}
```

Figure 5-20 API Call from Service to Component

5.3 BACK-END CODE

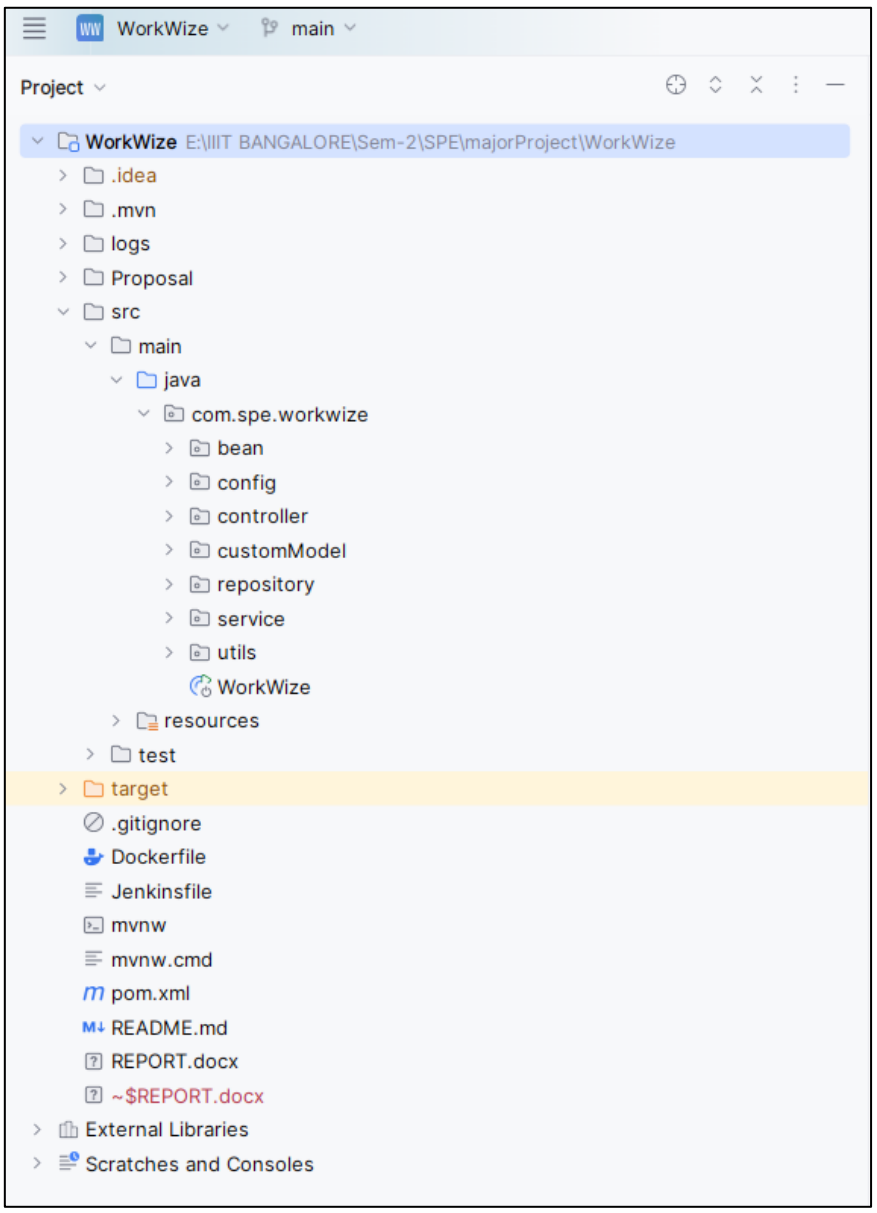

Figure 5-21 Backend Directory Structure

1. **API Controllers *(/controllers)*:**

   - API controllers handle incoming HTTP requests and define the endpoints for your application.
   - Each controller class corresponds to a specific resource or functionality (e.g., user, product, order).
   - Controllers are responsible for validating input, invoking service methods, and returning appropriate responses.
   - Example: If you have an endpoint for creating a new user, you'd define a UserController with methods for handling POST requests related to user creation.

2. **Business Logic *(/service)*:**

    - The service layer contains the core business logic of your application.
    - Services encapsulate complex operations, data processing, and interactions with other components.
    - They often interact with repositories (explained next) to retrieve or update data.
    - Example: A UserService might have methods for user authentication, profile updates, and password resets.

3. **DB Schemas *(/bean)*:**

    - The /bean directory likely contains data models or entity classes.
    - These classes represent the structure of your database tables.
    - Annotations (such as @Entity, @Table, and @Column) define the mapping between Java objects and database tables.
    - Example: If you have a user entity, it would define properties like id, username, and email.

4. **JPA Repositories *(/repository)*:**

    - JPA (Java Persistence API) repositories provide an abstraction layer for database operations.
    - They allow you to perform CRUD (Create, Read, Update, Delete) operations on entities.
    - Repositories extend interfaces like JpaRepository and provide methods for querying data.
    - Example: A UserRepository might have methods like findByUsername or findAllByRole.

5. **JWT Configuration *(/config)*:**

    - The /config directory holds configuration details related to JWT (JSON Web Token) authentication.
    - JWTs are commonly used for securing APIs and managing user sessions.
    - Configuration includes settings like token expiration time, secret keys, and token validation rules.
    - Example: You'd configure how to generate and validate JWTs for user authentication.

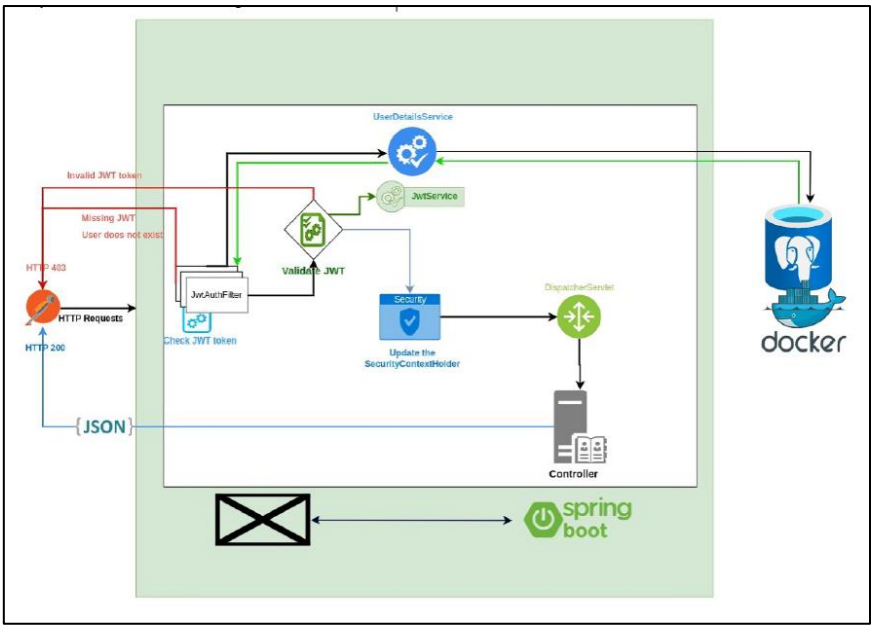

Figure 5-22 JWT Token Verification

i. Generating JWTs (Server-side)

- **User Authentication:**

When a user logs in or provides valid credentials, authenticate them against your user database.

- **Generate a JWT:**

Use a JWT library (e.g., jsonwebtoken in Node.js) to create a JWT.
Include relevant claims (payload) such as user ID, roles, and expiration time.
Sign the token using a secret key known only to your server.

- **Send the JWT to the Client:**

Return the JWT to the client (usually in the response body or a custom header).
The client will store this token for subsequent requests.

ii. Using JWTs in Requests (Client-side)

- **Include the JWT in Requests:**
  For each API request, include the JWT in the Authorization header:
  Authorization: Bearer YOUR_JWT_TOKEN

iii. Validating JWTs (Server-side)

- **Extract the Token:**
  In your server, extract the JWT from the Authorization header. Verify that the token is well formed (contains 3 parts: header, payload signature).

- **Verify the Signature:**
  Use the same secret key used during token generation to verify the token's signature.
  If the signature is valid, proceed; otherwise, reject the request.

- **Check Expiration:**
  Ensure the token hasn't expired (check the exp claim).
  Reject expired tokens.

- **Additional Validation (Optional):**
  Validate other claims (e.g., issuer, audience) if needed.
  Check user roles or permissions stored in the token.

iv. Handling Unauthorized Requests

- **Unauthorized Requests:**
  If the token is missing, invalid, or expired, return an HTTP 401 (Unauthorized) response.
  Prompt the client to re-authenticate or obtain a new token.

v. Refreshing Tokens (Optional)

- **Token Refresh:**
  To avoid frequent logins, issue a short-lived refresh token alongside the access token.
  When the access token expires, use the refresh token to obtain a new access token.

vi. Logging Out (Optional)

- **Logout:**
  To log out a user, invalidate the token on the server side.
  Maintain a blacklist of revoked tokens or use a short token expiration time.

vii. Best Practices

- **Keep Secrets Secure:**
  Safeguard the secret key used for signing tokens. Avoid hardcoding it in your code or exposing it publicly.
- **Short Token Lifetimes:**
  Set a reasonable expiration time (not too long). Regularly rotate keys and invalidate old tokens.
- **Use HTTPS:**
  Always use HTTPS to prevent token interception.

6. **Application Properties *(application.properties)*:**

   - This file contains various configuration properties for your Spring Boot application.
   - You can set database connection details, logging levels, security settings, and more.
   - Example properties: spring.datasource.url, spring.jpa.hibernate.ddl-auto, server. port.

7. **Data Initialization (*data.sql)*:**

   - The data.sql file is used for initializing data in your database during application startup.
   - You can insert initial records, create default users, or set up reference data.
   - Example: If your application requires predefined roles (e.g., admin, user), you'd insert them in this file.

5.4 DEVOPS PIPELINE

DevOps is a set of practices and tools that enhance collaboration between software development (Dev) and IT operations (Ops) teams. Its primary purpose is to automate and streamline the software development lifecycle, resulting in faster and more reliable software releases.
Key components and tools in our DevOps workflow include:

- **Source Code Management:** GitHub - A platform for version control and collaborative development.
- **Continuous Integration (CI):** Jenkins - An automation server for building and testing code continuously.
- **Containerization:** Docker - A platform for packaging applications into containers for consistent deployment.
- **Container Orchestration:** Docker Compose - A tool for defining and running multi-container Docker applications.
- **Continuous Deployment (CD):** Ansible - An automation tool for application deployment, configuration management, and orchestration.
- **Logging and Monitoring:** ELK Stack (Elasticsearch, Logstash, Kibana) - Tools for searching, analyzing, and visualizing log data.
- **Frontend Artifact Management:** npm - A package manager for JavaScript that handles project dependencies.
- **Backend Artifact Management:** Maven - A build automation tool primarily used for Java projects.

These tools and practices help ensure efficient development, deployment, and monitoring of applications, fostering a culture of continuous improvement and collaboration.

5.4.1 SOURCE CODE MANAGEMENT

Source code management is the practice of tracking modifications to source code.Using SCM, we can maintain the current state of the software using which developers can work on newer versions for features. Multiple developers from the same team can work on a single project collaborating among them and then resolving the conflicts in the code when they push them to the repository. For Software configuration management we shall be using Github.

We have created a similar one for the WorkWize Application and can be found at

FRONTEND: https://github.com/ARJUN1220/SPE_frontend
BACKEND: https://github.com/NaikVRaj/majorProject.git
The SCM handles our code and is used to connect as input to Jenkins.

Once the developer creates the project and adds any features these can be pushed to the Github by following the below steps:

git clone https://github.com/NaikVRaj/majorProject

git init

git remote add origin https://github.com/NaikVRaj/majorProject

- For Cloning the Project:

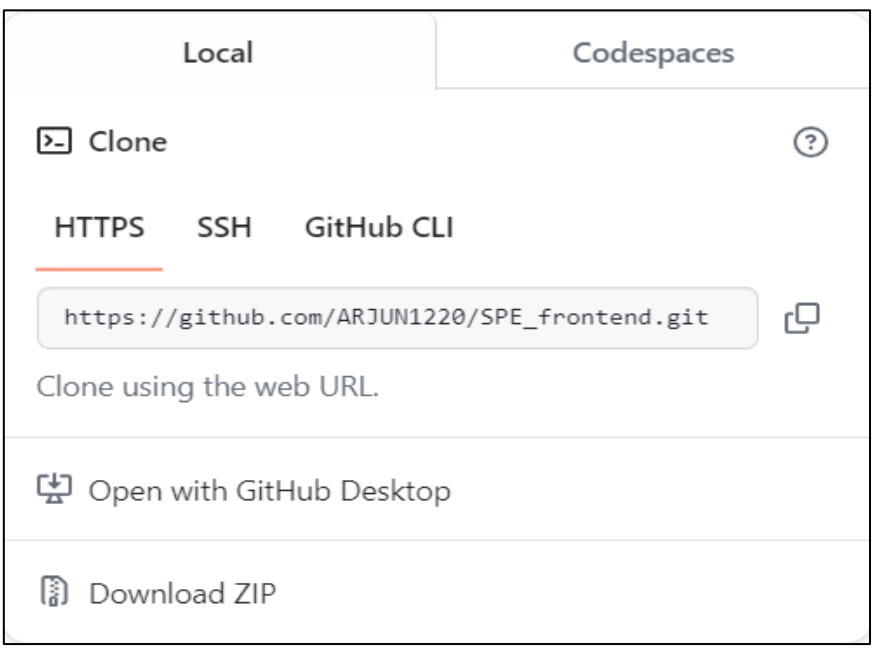

Figure 5-23 Front End Git Clone

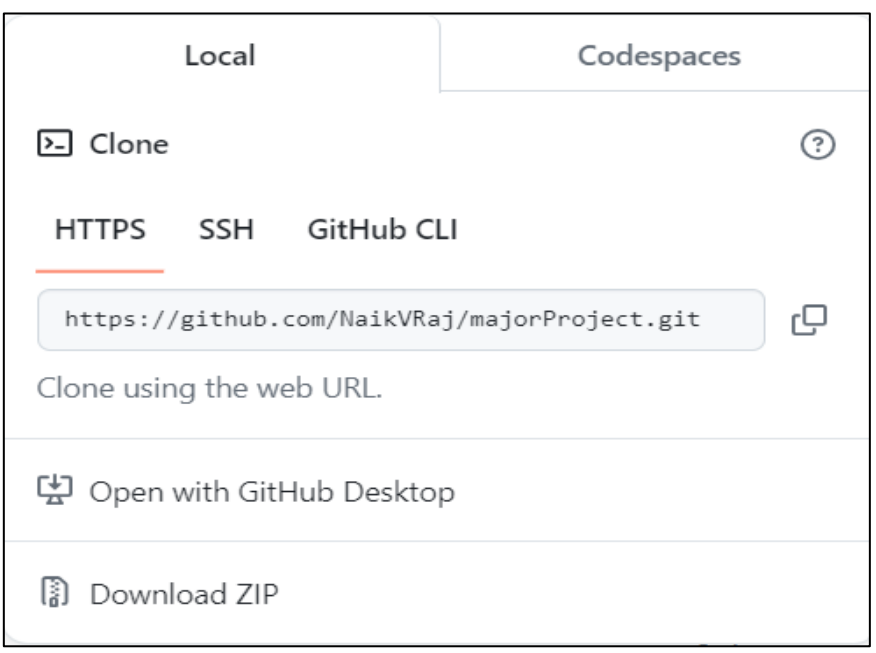

Figure 5-24 Back End Git Clone

5.4.2 CONTINUOUS INTEGRATION

Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied. Few Popular tools for CI are Jenkins, Travis CI, GitLab CI, Github actions, and many more.

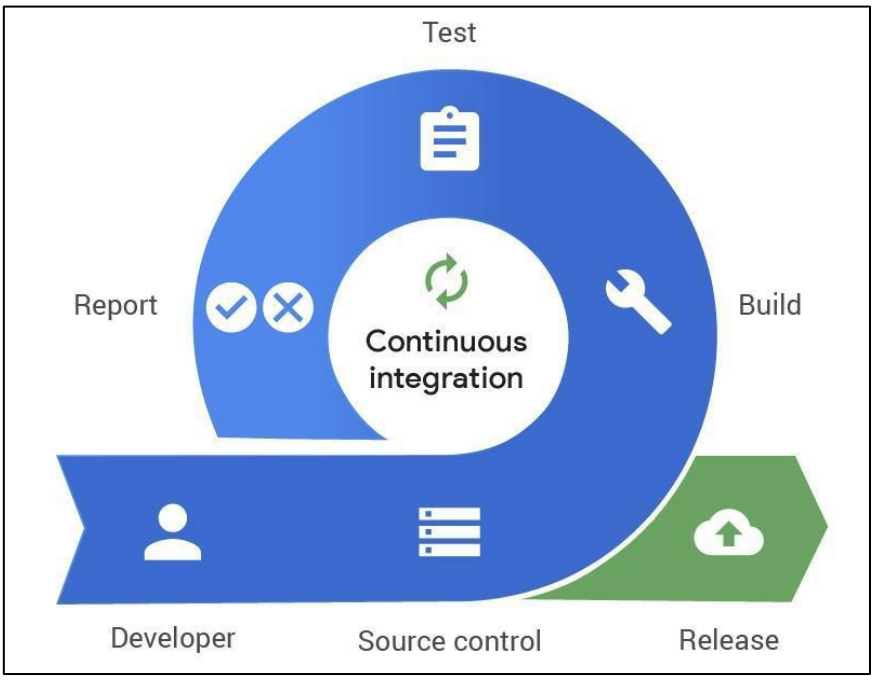

Figure 5-25 Continuous Integration

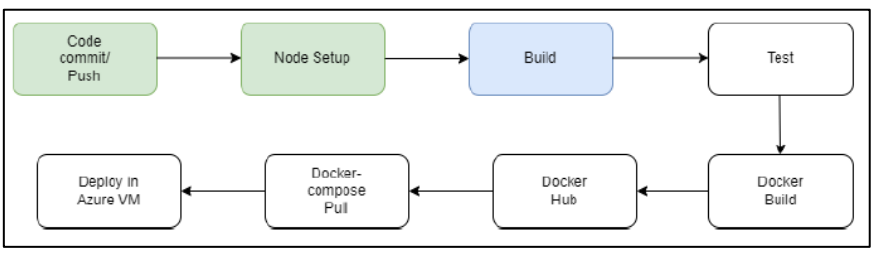

Figure 5-26 Front End DevOps Pipeline

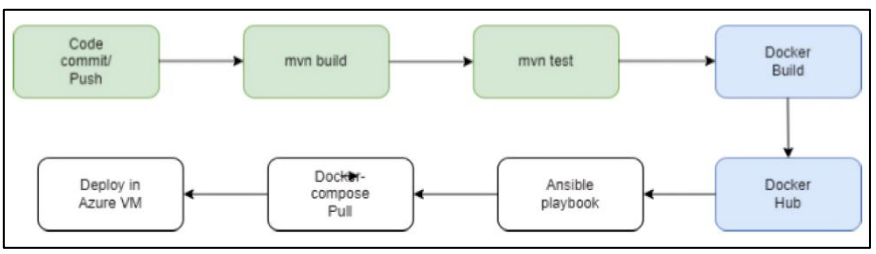

Figure 5-27 Back End DevOps Pipeline

After the developer commits & pushes the code, the jenkins workflow will be triggered and the build will start.

This command builds and serves the application. It rebuilds the application if changes occur. Here project is the name of the application as defined in angular.json.

This command compiles an angular application/library into an output directory named dist at the given path.

Options used & Syntax:

– configuration=production: A named build target, as specified in the "configurations" section of angular.json.
Each named target is accompanied by a configuration of option defaults for that target. Setting this explicitly overrides the "--prod" flag.d

Here, we are using productions we want to get production environment settings while building an angular application/binary.

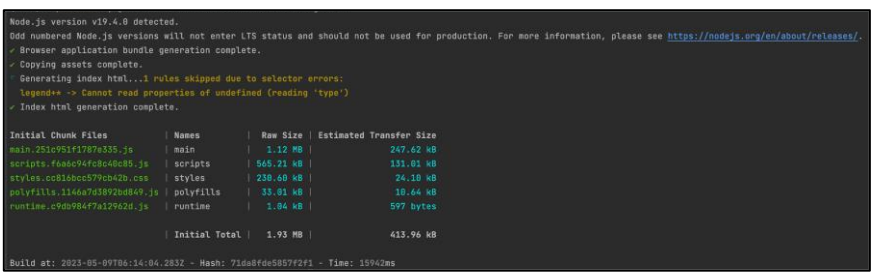

Figure 5-28 Production Environment

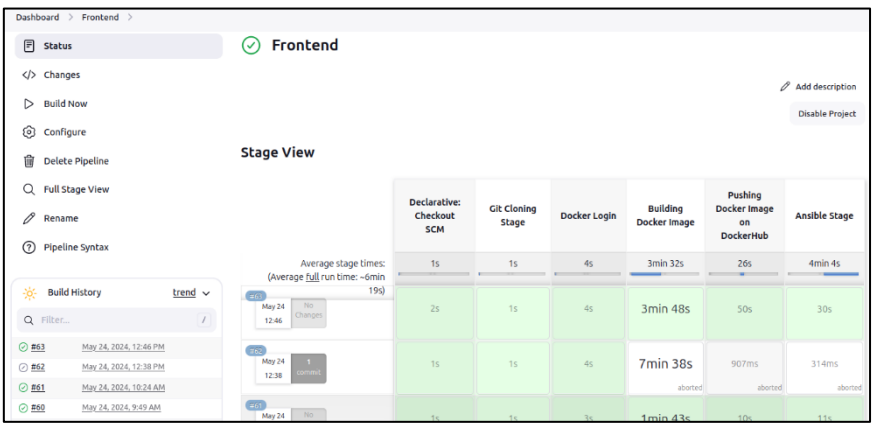

Figure 5-29 Front End Jenkins Pipeline

```
pipeline{
    environment{
            DOCKERHUB = credentials('DockerHubCred')
            ANSIBLE_VAULT_PASSWORD = credentials('ANSIBLE_VAULT_PASSWORD')
    }
    agent any
    stages{
        stage("Git Cloning Stage"){
            steps{
                git 'https://github.com/ARJUN1220/SPE_frontend.git'
            }
        }
        stage("Docker Login"){
            steps{
               sh 'docker login -u $DOCKERHUB_USR -p $DOCKERHUB_PSW'
            }
        }
        stage("Building Docker Image"){
            steps{
                sh 'docker build -t arjun201/aj-frontend:1.0 .'
            }
        }
        stage("Pushing Docker Image on DockerHub"){
            steps{
                sh 'docker push arjun201/aj-frontend:1.0'
            }
        }
        stage("Ansible Stage") {
            steps {
                script{
                    withEnv(["ANSIBLE_HOST_KEY_CHECKING=False"]) {
                        ansiblePlaybook(
                            inventory: 'inventory',
                            playbook: 'playbook.yaml',
                            vaultCredentialsId: 'ANSIBLE_VAULT_PASSWORD'
                        )
                    }
                }
            }
        }

    }
}
```

Figure 5-30 Front End Jenkins File

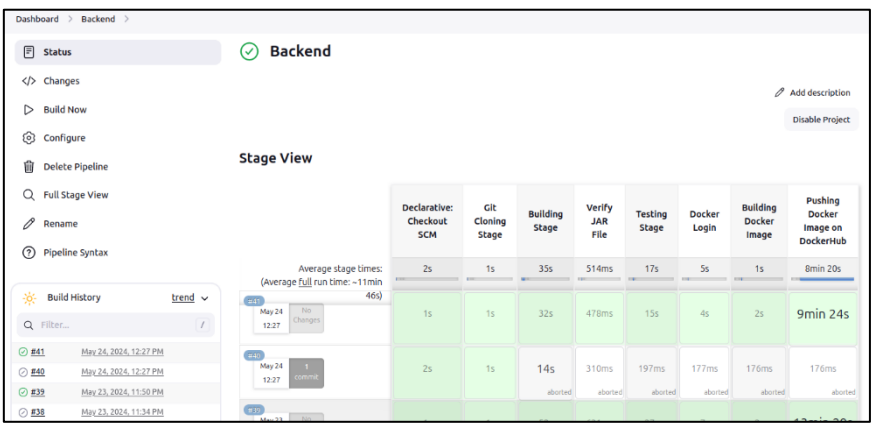

Figure 5-31 Back End Jenkins Pipeline

```
pipeline{
    environment{
            DOCKERHUB = credentials('DockerHubCred')
            MYSQL_DATABASE = credentials('MYSQL_DATABASE')
            MYSQL = credentials('MYSQL')
            BACKEND_INTERNAL_PORT = credentials('BACKEND_INTERNAL_PORT')
    }
    agent any
    stages{
        stage("Git Cloning Stage"){
            steps{
                git 'https://github.com/ARJUN1220/SPE_backend.git'
            }
        }
        stage("Building Stage"){
            steps{
                sh 'mvn clean -DMYSQL_DATABASE=$MYSQL_DATABASE -
DBACKEND_INTERNAL_PORT=$BACKEND_INTERNAL_PORT -DMYSQL_USR=$MYSQL_USR -
DMYSQL_PSW=$MYSQL_PSW'
                sh 'mvn compile -DMYSQL_DATABASE=$MYSQL_DATABASE -
DBACKEND_INTERNAL_PORT=$BACKEND_INTERNAL_PORT -DMYSQL_USR=$MYSQL_USR -
DMYSQL_PSW=$MYSQL_PSW'
                sh 'mvn package -DMYSQL_DATABASE=$MYSQL_DATABASE -
DBACKEND_INTERNAL_PORT=$BACKEND_INTERNAL_PORT -DMYSQL_USR=$MYSQL_USR -
DMYSQL_PSW=$MYSQL_PSW'
            }
        }

        stage("Verify JAR File") {
            steps {
                sh 'ls -lh target/workwise-0.0.1-SNAPSHOT.jar'
            }
        }

        stage("Testing Stage"){
            steps{
                sh 'mvn test -DMYSQL_DATABASE=$MYSQL_DATABASE -
DBACKEND_INTERNAL_PORT=$BACKEND_INTERNAL_PORT -DMYSQL_USR=$MYSQL_USR -
DMYSQL_PSW=$MYSQL_PSW'
            }
        }
        stage("Docker Login"){
            steps{
                sh 'docker login -u $DOCKERHUB_USR -p $DOCKERHUB_PSW'
            }
        }
        stage("Building Docker Image"){
            steps{
                sh 'docker build --no-cache -t arjun201/aj-backend:1.0 .'
            }
        }
        stage("Pushing Docker Image on DockerHub"){
            steps{
                sh 'docker push arjun201/aj-backend:1.0'
            }
        }

    }
}
```

Figure 5-32 Back End Jenkins File

5.4.3 CONTAINERIZATION USING DOCKER

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that include everything the software needs to run: libraries, system tools, code, and runtime. Containers ensure that your application runs consistently in any environment, from development to production. Using Docker, you can streamline your development workflow by automating the deployment and scaling of applications.

Using Docker, you can quickly deploy and scale applications into any environment, ensuring your code will run reliably. This is achieved through the flexibility of building and pushing images to Docker Hub. Docker Hub is a cloud-based repository where you can store and distribute Docker images.

The process of creating Docker images involves writing a Dockerfile. A Dockerfile is a simple text file that consists of a set of instructions to build Docker images. These instructions might include specifying a base image, copying application files, installing dependencies, setting environment variables, and defining the command to run the application.

```
FROM node:latest as build
WORKDIR /usr/local/app
COPY ./ /usr/local/app/
RUN npm config set legacy-peer-deps true
RUN npm install --configuration=production
RUN npm run build --configuration=production
FROM nginx:latest
RUN rm -rf /usr/share/nginx/html/* && rm -rf /etc/nginx/nginx.conf
COPY ./nginx.conf /etc/nginx/nginx.conf
COPY --from=build /usr/local/app/dist/workwise /usr/share/nginx/html
EXPOSE 80
```

Figure 5-33 Docker File Front End

```
FROM openjdk:17
WORKDIR /app
ADD target/workwise-0.0.1-SNAPSHOT.jar /app/workwise-0.0.1-SNAPSHOT.jar
EXPOSE 8086
CMD ["java", "-jar", "/app/workwise-0.0.1-SNAPSHOT.jar", "--spring.profiles.active=docker"]
```

Figure 5-34 Docker File Backend

5.4.4 CONTAINER ORCHESTRATION USING DOCKER COMPOSE

Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down. The big advantage of using Compose is you can define your application stack in a file, keep it at the root of your project repo (it's now version controlled), and easily enable someone else to contribute to your project. Someone would only need to clone your repo and start the compose app. We have created a docker-compose.yml file by which we pull the Docker hub frontend container and connect it with the backend & database.

```yaml
version: '3'
services:
  workwize-db:
    container_name: workwize-db
    image: mysql
    volumes:
      - workwize-db-vol:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}

  workwize-backend:
    container_name: workwize-backend
    image: aj1234/workwize-backend:1.0
    working_dir: /workwize/backend
    volumes:
      - workwize-backend-vol:/workwize/backend
    stdin_open: true
    expose:
      - 8086
    ports:
      - 8086:8086
    depends_on:
      - workwize-db
    restart: unless-stopped
    environment:
      DB_SERVICE_NAME: ${DB_SERVICE_NAME}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      BACKEND_INTERNAL_PORT: ${BACKEND_INTERNAL_PORT}

  workwize-frontend:
    container_name: workwize-frontend
    image: murphy961/workwize-frontend:1.0
    working_dir: /workwize/frontend
    stdin_open: true
    expose:
      - 8087
    ports:
      - 8087:80
    depends_on:
      - workwize-backend
    restart: unless-stopped
    volumes:
      - workwize-db-vol
      - workwize-backend-vol

volumes:
  workwize-db-vol:
  workwize-backend-vol:
```

Figure 5-35 Docker Compose File

1. **Version and Services:**

   The Docker Compose file is set to version 3.
   It defines three services: WorkWize-db, WorkWize-backend, and WorkWize-frontend.

2. **Containers:**

   i. **WorkWize-db:**

   - This container runs the MySQL database.
   - It uses the official MySQL image.
   - Data persistence is ensured by mounting a volume named WorkWize-db-vol to the /var/lib/mysql directory.
   - Environment variables are set for the MySQL root password, user, password, and database name.

   ii. **WorkWize-backend:**

   - This container runs the backend application.
   - It uses a custom image (aj/WorkWize-backend:1.0).
   - The working directory inside the container is set to /WorkWize/backend.
   - A volume named WorkWize-backend-vol is mounted to the /WorkWize/backend directory.
   - The container exposes port 8086 for communication.
   - It depends on the WorkWize-db container.
   - Environment variables are configured for the database service name, database name, user, password, and backend internal port.

   iii. **WorkWize-frontend:**

   - This container runs the frontend application.
   - It uses a custom image (aj/WorkWize-frontend:1.0).
   - The working directory inside the container is set to /WorkWize/frontend.
   - The container exposes port 8087 for communication.
   - It depends on the WorkWize-backend container.

3. **Network:**

   Both the WorkWize-backend and WorkWize-frontend containers are part of the same network named docker_net.

4. **Data Persistence:**

   The MySQL container (WorkWize-db) uses a volume (WorkWize-db-vol) for data persistence.
   This ensures that user data is retained even when the server is down.

5.4.5 CONTINUOUS DEPLOYMENT USING ANSIBLE

- Ansible is used for deployment of all services onto the VM via docker-compose.
- Basically, Ansible uses a playbook to do these tasks.

```yaml
- name: Run Docker Compose file on VM
  hosts: all
  become: false

  vars_files:
    - ./env_vars.yaml

  tasks:
    - name: Copy Docker Compose file to VM
      copy:
        src: /var/lib/jenkins/workspace/Frontend/docker-compose.yaml
        dest: ./docker-compose.yaml

    - name: stop all the services
      shell:
        docker-compose down
      environment:
        MYSQL_ROOT_PASSWORD: "{{ MYSQL_ROOT_PASSWORD }}"
        MYSQL_USER: "{{ MYSQL_USER }}"
        MYSQL_PASSWORD: "{{ MYSQL_PASSWORD }}"
        MYSQL_DATABASE: "{{ MYSQL_DATABASE }}"
        BACKEND_EXPOSE_PORT: "{{ BACKEND_EXPOSE_PORT }}"
        BACKEND_INTERNAL_PORT: "{{ BACKEND_INTERNAL_PORT }}"
        FRONTEND_EXPOSE_PORT: "{{ FRONTEND_EXPOSE_PORT }}"
        FRONTEND_INTERNAL_PORT: "{{ FRONTEND_INTERNAL_PORT }}"
        DB_SERVICE_NAME: "{{ DB_SERVICE_NAME }}"
        BACKEND_SERVICE_NAME: "{{ BACKEND_SERVICE_NAME }}"
        FRONTEND_SERVICE_NAME: "{{ FRONTEND_SERVICE_NAME }}"

    - name: Remove images of frontend if any
      shell:
        docker rmi arjun201/frontend:1.0||ls

    - name: Remove images of backend if any
      shell:
        docker rmi arjun201/backend:1.0||ls

    - name: Start the containers on VM
      shell:
        docker-compose up -d
      environment:
        MYSQL_ROOT_PASSWORD: "{{ MYSQL_ROOT_PASSWORD }}"
        MYSQL_USER: "{{ MYSQL_USER }}"
        MYSQL_PASSWORD: "{{ MYSQL_PASSWORD }}"
        MYSQL_DATABASE: "{{ MYSQL_DATABASE }}"
        BACKEND_EXPOSE_PORT: "{{ BACKEND_EXPOSE_PORT }}"
        BACKEND_INTERNAL_PORT: "{{ BACKEND_INTERNAL_PORT }}"
        FRONTEND_EXPOSE_PORT: "{{ FRONTEND_EXPOSE_PORT }}"
        FRONTEND_INTERNAL_PORT: "{{ FRONTEND_INTERNAL_PORT }}"
        DB_SERVICE_NAME: "{{ DB_SERVICE_NAME }}"
        BACKEND_SERVICE_NAME: "{{ BACKEND_SERVICE_NAME }}"
        FRONTEND_SERVICE_NAME: "{{ FRONTEND_SERVICE_NAME }}"
```

Figure 5-36 Ansible Playbook

1. **Copy Docker Compose File to VM**
   - The playbook starts by copying the docker-compose.yaml file from the Jenkins workspace to the VM.
   - The source path is /var/lib/jenkins/workspace/WorkWize-Frontend/docker-compose.yaml.
   - The destination path on the VM is ./docker-compose.yaml.

2. **Stop All Services**
    - The playbook executes the command docker-compose down to stop all services defined in the Docker Compose file.
    - Environment variables are used for configuration, such as MySQL credentials, backend and frontend ports, and service names.

3. **Remove Images of Frontend (if any)**
    - The playbook attempts to remove the Docker image aj961/workwize-frontend:1.0.
    - If the image does not exist, it lists the available images.

4. **Remove Images of Backend (if any)**

    - Similar to the frontend, the playbook tries to remove the Docker image aj961/workwize-backend:1.0.
    - If the image is not found, it lists the existing images.

5. **Start Containers on VM**
    - Finally, the playbook starts the containers using docker-compose up -d.
    - Environment variables are again used for configuration, including MySQL credentials, backend and frontend ports, and service names.

5.4.6 MOTINORING AND LOGGING USING ELK STACK

The proactive method of observing systems with the goal of preventing downtime and outages, is monitoring. It involves measuring current behavior against predetermined baselines. Some of the commonly observed devices are CPU usage, storage capacities, request hits i.e network traffic, etc. by which we can find the root cause of the failure.

Elasticsearch is a modern analytics and search engine which is based on Apache Lucene. Using this we can draw meaningful information from index logs created by applications. We can use elastic to manage, store and search data which can come in handy in Logs, Metrics, Application monitoring, Endpoint security, and Search backend.
We can use ELK stack locally or kibana & elasticsearch can be run on the cloud and we can send logs using logstash installed on the host machine. Here, we have used cloud approaches as for installing ELK stack RAM requirements were high.

Using ELK we will be doing continuous monitoring. ELK uses three open-source projects namely Elastic search, Logstash, Kibana respectively.

- Elasticsearch is a distributed, free and open search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch is the central component of the ELK stack Logstash is a free and open

server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and sends it to your favorite “stash”.

- Kibana is a free and open user interface that lets you visualize your elasticsearch data and navigate the elastic stack.
  Do anything from tracking query load to understanding the way requests flow through apps.
- For logging we have used log4j2, a library useful for logging.

5.4.6.1 ELASTIC SEARCH

- First, we need to create an account on Elasticsearch cloud.

i. Go to https://www.elastic.co/ URL and signup.

ii. Create the first deployment.

iii. Go with the default configuration.

iv. Save username & password after the creation of deployment as we will need them hereafter and they will not be shown again.

6. USER INTERFACE & USER EXPERIENCE

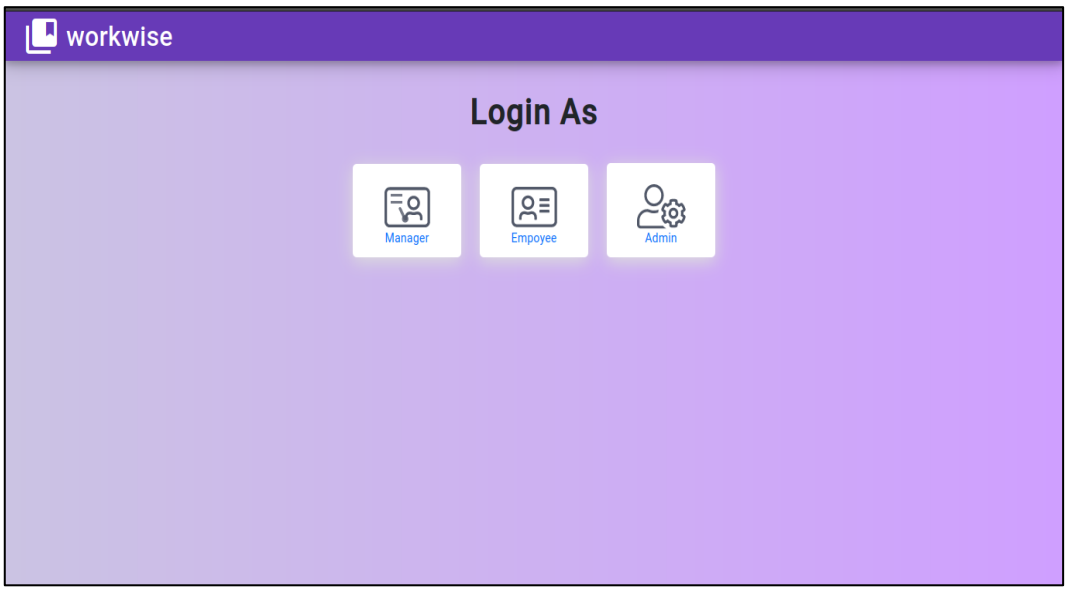

Figure 6-1 Welcome Page

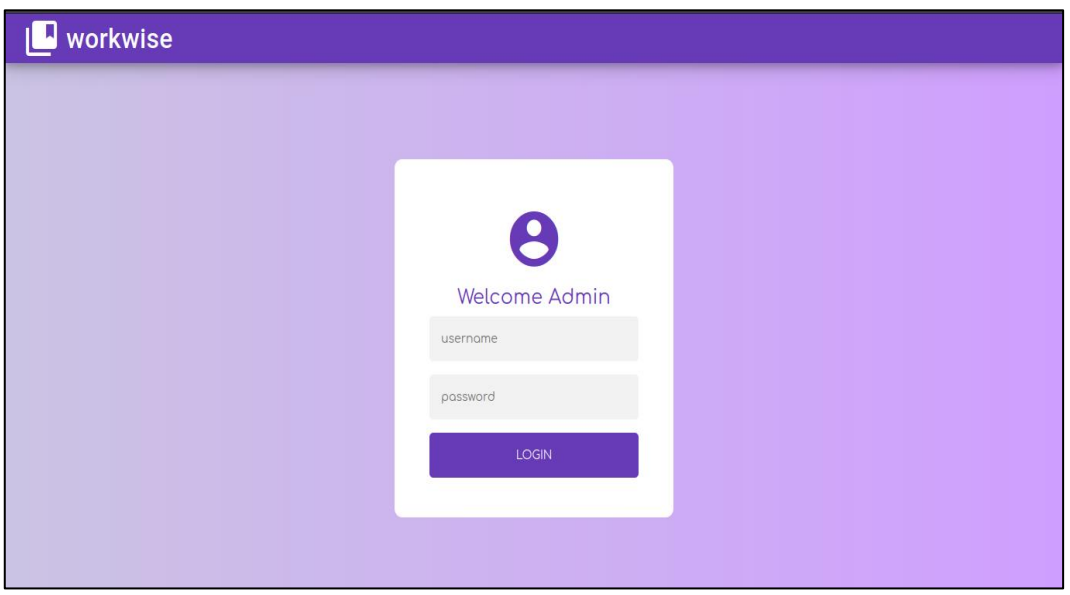

Figure 6-2 Log-In Page

6.1 ADMIN

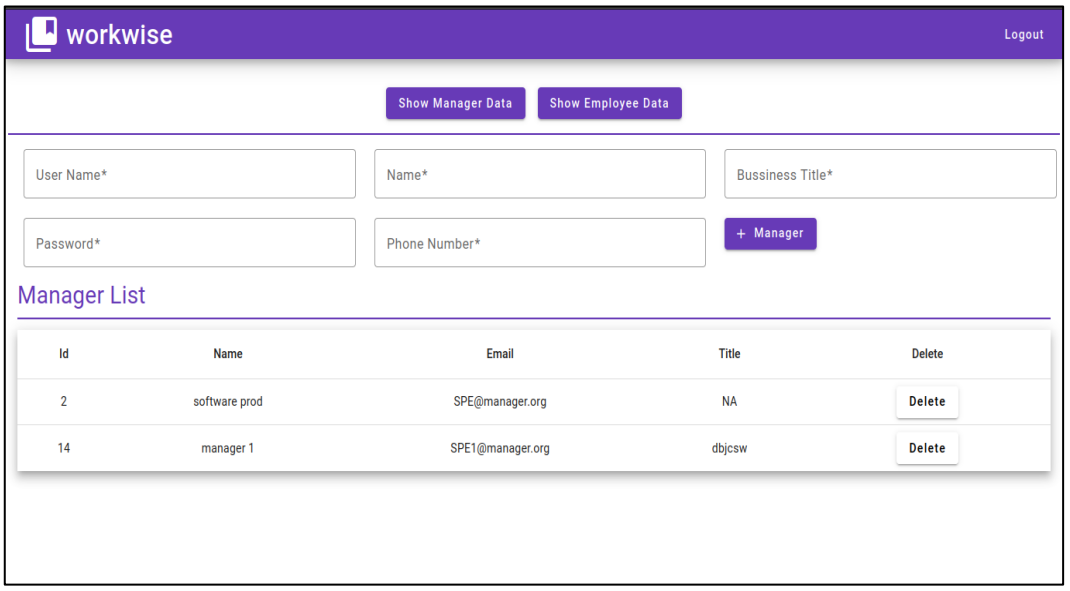

Figure 6-3 Admin – Dashboard0

6.2 MANAGER

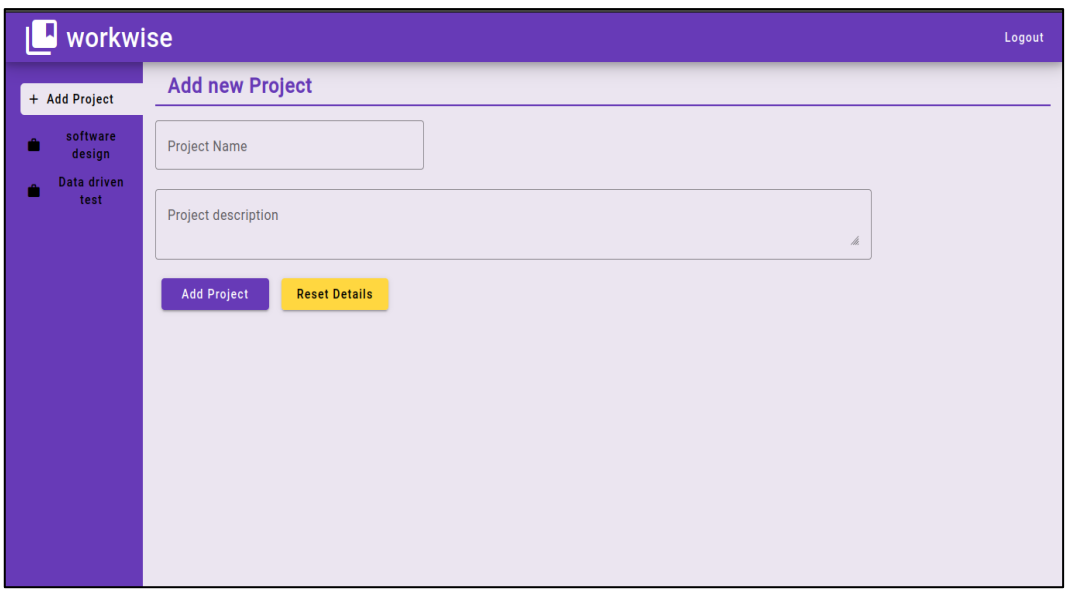

Figure 6-4 Manager - Add Project

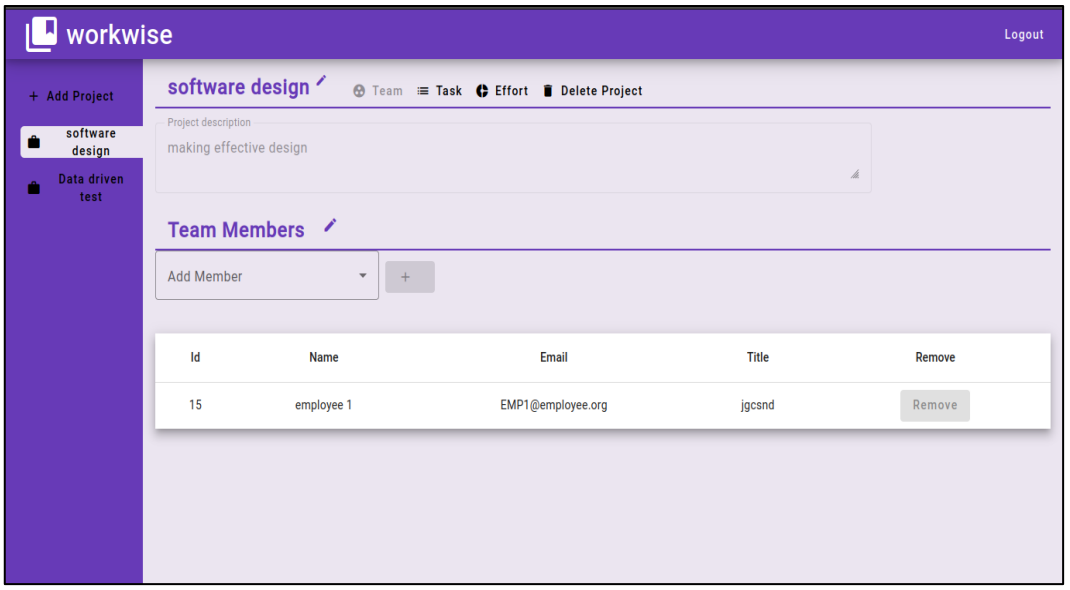

Figure 6-5 Manager – Project Dashboard

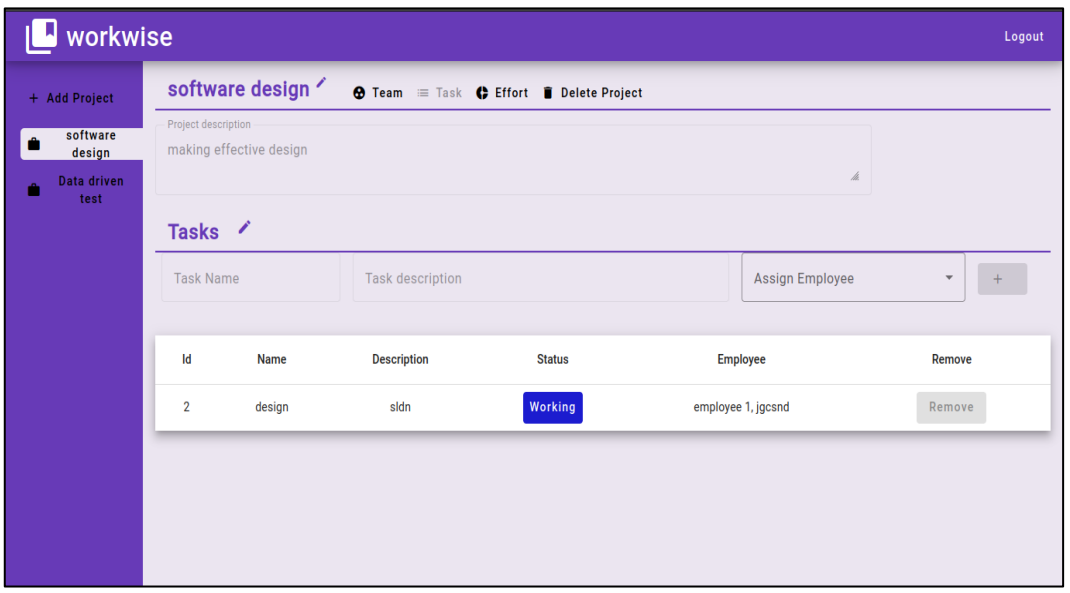

Figure 6-6 Manager - List of Tasks

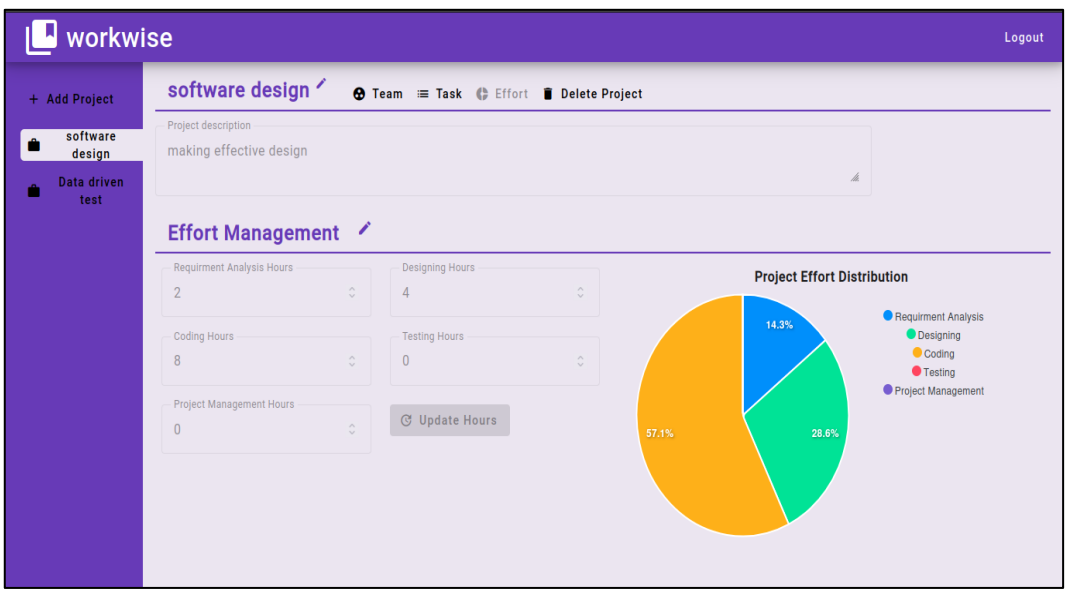

Figure 6-7 Manager - Update Efforts

6.3 EMPLOYEE

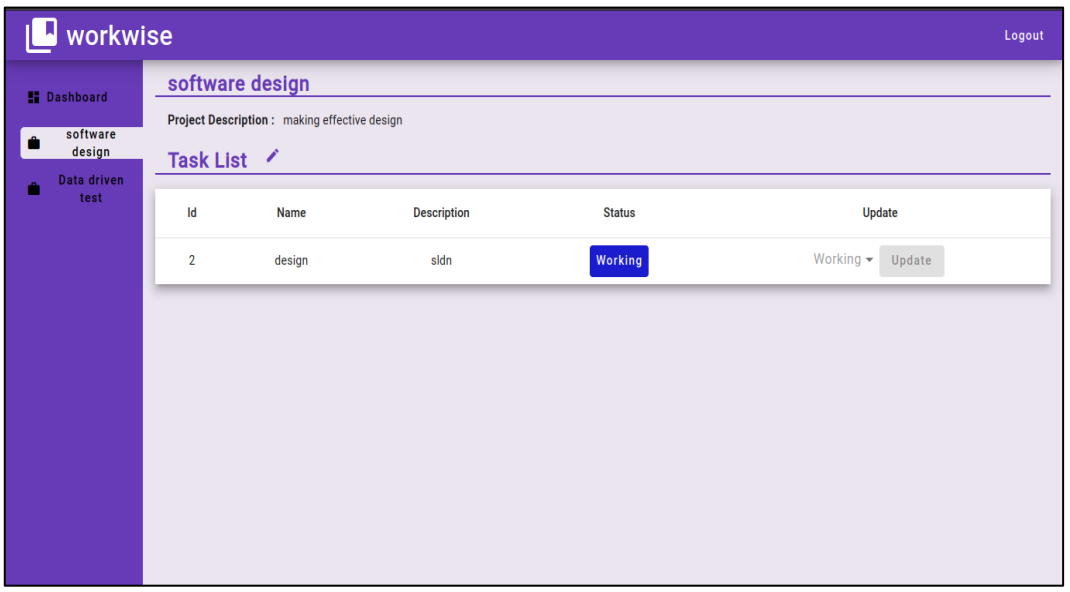

Figure 6-8 Employee - Update Task - I

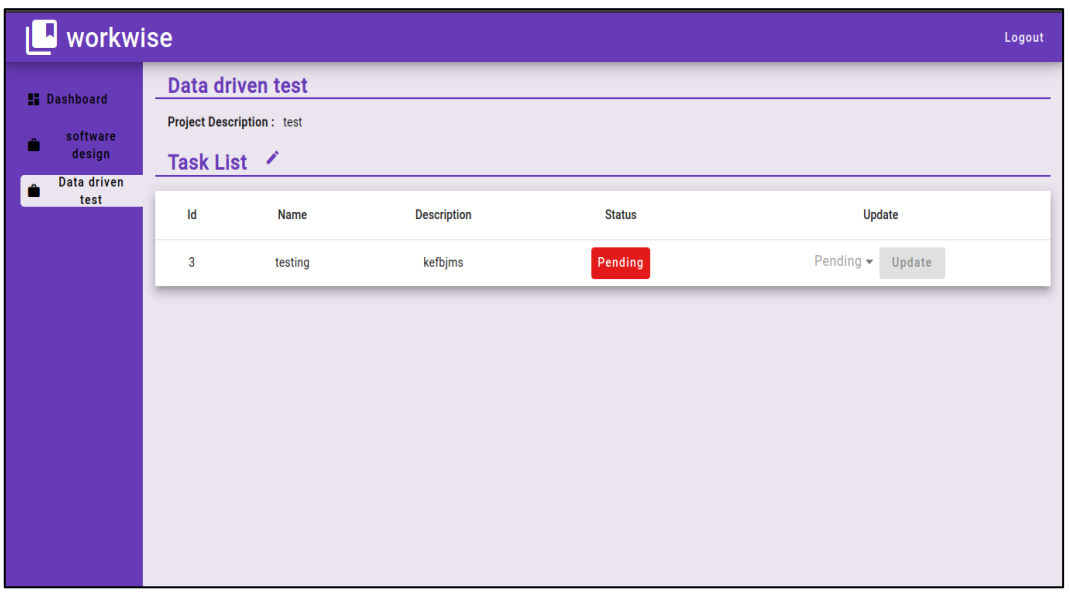

Figure 6-9 Employee - Update Task - II

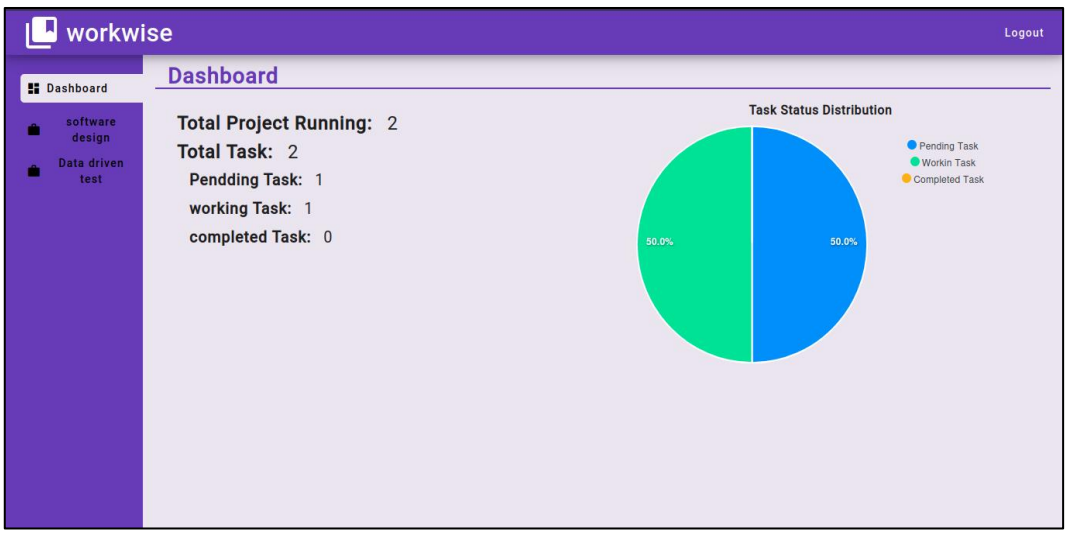

Figure 6-10 - Employee Dashboard

7. CHALLENGES

1. **CORS Policy During Preflight Requests:**

- **Problem:**

  Our application encountered issues with Cross-Origin Resource Sharing (CORS) during preflight requests. Preflight requests, which precede actual requests, do not carry headers such as JWT tokens.
  Initially, our authentication mechanism was applied to these preflight requests, causing the authentication to fail and subsequent requests to break.

- **Solution:**

  The solution was to configure the server to permit preflight requests without authentication, thereby ensuring that actual requests could proceed successfully.

2. **Securing Environmental/Configuration Variables:**

- **Problem:**

  Initially, configuration settings were hardcoded directly into our code, posing significant security risks.

- **Solution:**

  We transitioned to using Jenkins to securely store these configuration variables. Jenkins provides robust mechanisms for managing sensitive data, which allowed us to enhance the security of our configuration management process.

3. **Passing Environment Variables to Docker-Compose:**

- **Problem:**

  We needed to securely pass environment variables from Ansible to the Docker-Compose environment for service startup. Previously, these variables were passed as plain text, which was not secure.

- **Solution:**

  After researching, we adopted Ansible Vault to encrypt our data. This encrypted data could then be used in playbooks, ensuring secure transfer of environment variables to the appropriate services based on the hosts configured in the inventory file.

4. **Backend Container Exiting Prematurely:**

- **Problem:**

  The backend container was unexpectedly exiting shortly after launch. The root cause was its dependency on the MySQL container, which required additional time to initialize and create a user.
  During this period, the backend container couldn't connect to MySQL on port 3306, leading to its premature exit.

- **Solution:**

  The solution involved adding a dependency configuration in the Docker-Compose file, ensuring the backend service started only after the MySQL service was fully operational.
  Additionally, configuring the backend container to always restart on connection failures ensured resilience against transient issues.

5. **Privileges Issue When Running on Host:**

- **Problem:**

  While running the playbook on the host machine, some commands failed due to insufficient permissions, as they required sudor privileges.

- **Solution:**

  To address this, we utilized the **become: yes,** directive in the playbook, enabling commands to run with elevated privileges.
  Furthermore, we provided the necessary host credentials in the inventory file, ensuring that the playbook could execute all required commands without permission issues.

8. LIMITATION & FUTURE WORK

LIMITATIONS

- **Admin Management:** The application requires an admin to manage and oversee operations, limiting scalability and self-service capabilities.
- **Communication Gap:** There is no built-in functionality for communication between managers and employees, hindering efficient collaboration.
- **Task Subdivision:** The current system does not support dividing tasks into smaller, manageable sub-tasks, reducing task management flexibility.
- **Ticketing System:** There is no mechanism for users to raise tickets or queries, impacting support and issue resolution processes.

FUTURE WORK

- **Admin Independence:** Develop features to minimize or eliminate the dependency on an admin for managing the application, promoting self-service.
- **User Information Management:** Implement a user information tab with update functionality to maintain and modify user details efficiently.
- **Enhanced Authentication:** Introduce email and OTP-based authentication to improve security and ease of access.
- **Communication Tools:** Integrate a chat facility to enable quick and efficient communication between employees and managers.
- **User Experience Enhancement:** Focus on improving the overall user interface and experience to make the application more intuitive and user-friendly.
- **Performance Metrics:** Add features to track and display user performance metrics, providing insights into productivity and areas for improvement.

9. CONCLUSION

In the "WorkWize" project, we successfully developed a comprehensive task and project management system utilizing a DevOps approach. The integration of various tools and technologies allowed us to build a robust and scalable system that meets the diverse needs of users.

Functionalities Implemented:

- User authentication through login and registration
- Management of employees and managers
- Project and task management
- Task assignment and status updates
- Project effort analysis
- Display of employee statistics
- These functionalities were carefully implemented after thoroughly understanding the system requirements and module dependencies.

DevOps Practices and Tools:

- **Continuous Integration:** GitHub Actions facilitated automated builds and testing, ensuring that new code changes were consistently integrated and tested.
- **Configuration Management and Deployment:** Ansible was used to automate the configuration and deployment processes, enhancing efficiency and reducing manual errors.
- **Containerization:** Docker was employed to containerize the application, ensuring consistency across different environments and simplifying the deployment process.
- **Testing:** JUnit and Jasmine were utilized for rigorous unit and integration testing, ensuring the reliability and robustness of the system.
- **Monitoring:** The ELK stack (Elasticsearch, Logstash, and Kibana) provided comprehensive monitoring and logging capabilities, enabling us to track system performance and quickly address any issues.
- **Deployment:** The application was deployed on a Microsoft Azure Ubuntu-server VM, providing a scalable and reliable hosting environment.

Comprehensive testing was performed to determine and resolve potential flaws, ensuring a high-quality and reliable system. The automation of the entire application using DevOps tools streamlined the development and deployment processes, resulting in efficient and effective software delivery.

In conclusion, the "WorkWize" project exemplifies the successful integration of various technologies and frameworks within a DevOps environment. It showcases the power of collaboration, automation, and containerization in delivering a robust and efficient task and project management system. The system developed serves as a strong foundation for further enhancements and can potentially be deployed in real-world scenarios, providing significant value to its users.

REFERENCES

- **Apache Maven:** https://maven.apache.org/
- **Spring Framework:** https://spring.io/projects/spring-framework
- **Angular Framework**: https://angular.io/
- **Angular Material:** https://material.angular.io/
- **Docker:** https://www.docker.com/
- **Ansible:** https://www.ansible.com/
- **ELK Stack:** https://www.elastic.co/cloud/
- **Stack Overflow:** https://stackoverflow.com/
- **Chat-GPT:** https://chatgpt.com/
- **Past Year Report:**
    - **Venkatesh Boppana (MT2022140):** https://rb.gy/5yhfdx

https://stackoverflow.com/questions/64199452/logstash-not-showing-any-output-in-ubuntu

https://www.reddit.com/r/angular/comments/ha4kek/404_not_found_nginx_angular_docker/

https://blog.bitsrc.io/https-medium-com-adhasmana-how-to-do-ci-and-cd-of-node-js-application-using-github-actions-860007bebae6

https://stackoverflow.com/questions/66989383/could-not-resolve-dependency-npm-err-peerangular-compiler11-2-8

https://stackoverflow.com/questions/71014098/docker-compose-fails-with-copy-failed-stat-usr-src-app-dist-file-does-not-exi

https://blog.devgenius.io/how-to-build-and-run-a-nodejs-app-with-docker-github-actions-59eb264dfef5