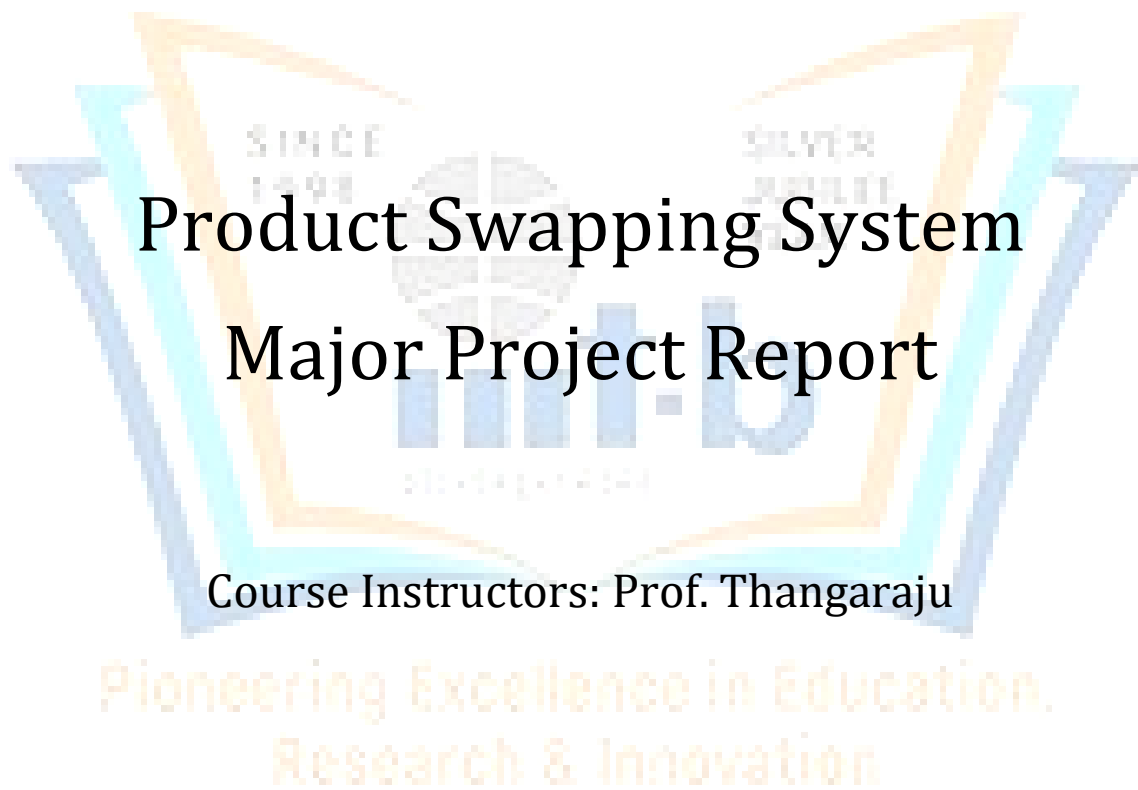


CS-816 Software Production Engineering



Course Instructors: Prof. Thangaraju

Done By:

IMT2020127 Dharmin Mehta

IMT2020066 Hasanabbas Momin

Introduction	4
Project Overview	4
Objectives	4
About The Project	5
Project Features	5
User Authentication	5
Product Management	5
Handling Swap Requests	6
Project Structure	8
Backend Development	8
Frontend Development	8
Database	8
API Integration	8
Overview	8
API Documentation	9
DevOps Practice	10
Version Control (Git)	10
Snapshots	11
Testing	11
Tools Used	11
Testing Description	12
User Case	12
Swap Requests Case	12
Product Case	13
Jenkins Pipeline Build	13
Back-End	14
Explanation	14
Script	14
Front-End	16
Explanation	16
Script	16
Snapshots	18
Building/Packaging	18

Maven Build	19
Spring Boot:	19
Containerization – Docker	19
Explanation	19
Usage in the project	19
Docker Compose File(docker-compose.yml):	19
Dockerfiles:	19
Snapshots	20
Ansible	21
Explanation	21
Ansible Playbook	21
Integration in the Project	22
Snapshots	22
Logging	23
Explanation	23
Elasticsearch	23
Logstash	23
Kibana	24
Spring Boot Application Logging Configuration	24
Viewing Logs	24
Snapshots	24
Challenges Faced	25
How To Run	25
Direct Running This Project	25
For Complete Setup	27
Links And References	28
Links	28
GitHub Repos	28
Dockerhub	28
References	28

Introduction

Project Overview

In SWAPSIE, users have the freedom to explore a variety of items listed on the platform. It's a bit like window shopping, but with the added excitement of being able to trade. If you spot something you're interested in, you can propose a swap to the owner of that item. On the flip side, you might receive swap requests from other users who are interested in what you have to offer.

Think of it as a virtual barter system, where the value of items is determined through mutual agreement between users. It's a fun and interactive way to discover new things, get rid of items you no longer need, and engage in a community of swappers. Swapsie is designed to bring back the charm of trading and swapping goods in a modern, digital setting.

In this project, we leverage DevOps practices, including Ansible for configuration management, Docker for containerization, and Jenkins for continuous integration, to enhance the development, deployment, and maintenance processes.

Objectives

- **Emulating Barter Online** : Create a user-friendly platform where individuals can virtually experience the essence of bartering. Utilize Ansible for automated configuration management, Docker for efficient containerization, and Jenkins for continuous integration to ensure a seamless development, deployment, and maintenance process.
- **User-Friendly Item Management** : Enable users to effortlessly manage their listed items, showcasing them to potential swappers. Implement Docker for containerization to enhance scalability and streamline deployment.
- **Swap Request Management**: Facilitate the process of sending and receiving swap requests, providing a clear and efficient communication channel. Employ Ansible for automated configuration management to enhance the efficiency of request management.

- **Item Exchange Workflow** : Develop a robust workflow for item exchange upon mutual agreement between users. Implement Jenkins for continuous integration to ensure a reliable and automated workflow.

About The Project

Project Features

User Authentication

Description: Swapsie ensures secure and personalized user experiences through robust authentication mechanisms.

Key Functionalities:

- User Registration: Users can create new accounts with unique credentials.
- Login and Logout: Secure authentication for accessing and leaving the system.
- User Profiles: Personalized profiles with details like name, avatar, and contact information.

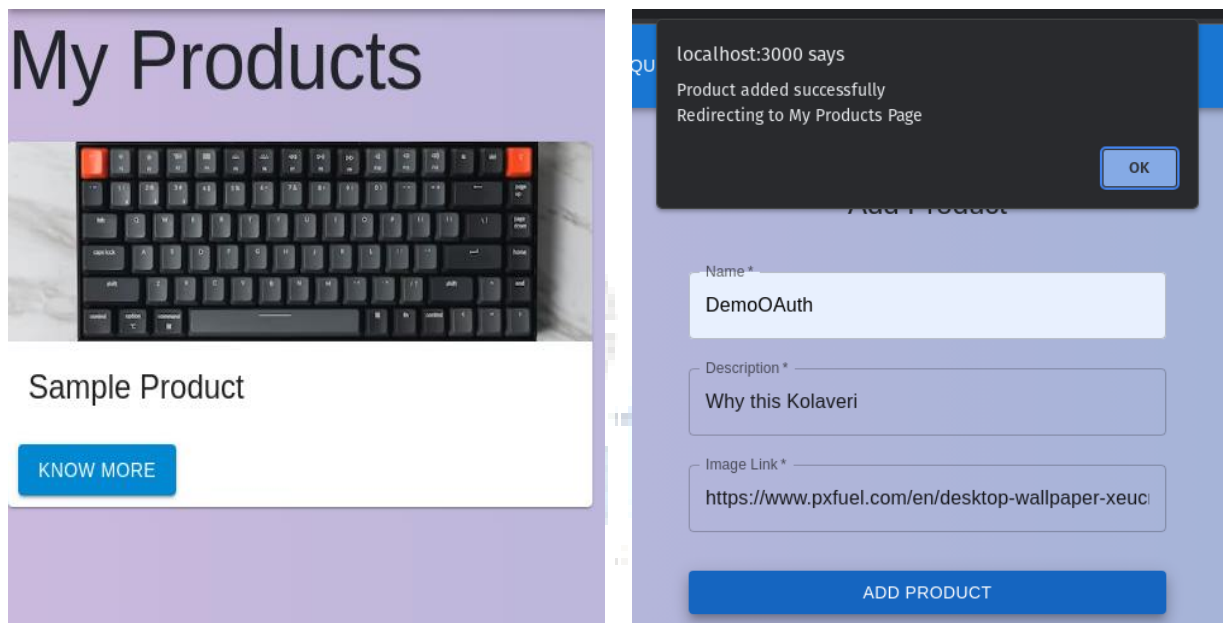
The image displays two side-by-side mockups of user authentication forms. The left form is titled 'Sign in' and features a purple lock icon at the top. It contains input fields for 'Email Address *' and 'Password *', a 'Remember me' checkbox, and a blue 'LOGIN' button. A link at the bottom reads 'Don't have an account? Sign Up'. The right form is titled 'Sign up' and also features a purple lock icon. It includes input fields for 'First Name *' (filled with 'Dharmin'), 'Last Name *' (filled with 'Mehta'), 'Email Address *' (filled with 'dharmin23072002@gmail.com'), and 'Password *' (filled with seven dots). It has a blue 'SIGN UP' button and a link at the bottom that reads 'Already have an account? Sign in'.

Product Management

Description: The platform facilitates effective management of user products, including adding, editing, and deleting items.

Key Functionalities:

- Product Addition: Users can add details about the products they wish to showcase.
- Product Editing: Allows users to modify product information as needed.
- Product Deletion: Users can remove products that are no longer available for swapping.
- Product Visualization: Clear presentation of product details, including images and descriptions.



Handling Swap Requests

Description: Swapsie's core feature revolves around facilitating the exchange of products between users through a simple and efficient process.

Key Functionalities:

- Product Swapping: Users can select a product from another user and initiate a swap request.
- Incoming Swap Requests: Users receive notifications about incoming swap requests.
- Acceptance and Rejection: Options for users to accept or decline incoming swap requests.
- Swap Status Tracking: Users can track the status of their swap requests.

Why this Kolaveri Details



Name: Why this Kolaveri




Product Id: 6

Owner: GAJENDRA MEHTA

Description: Ayein

[Request Swap](#)

Create SwapRequest Page

Your Product		Swap with
		
Random Name KNOW MORE		Why this Kolaveri KNOW MORE

Choose one of your products to swap with:

Random Name

[SEND REQUEST](#)

Your product



Why this
Kolaveri

[KNOW MORE](#)

Swap with



Random
Name

[KNOW MORE](#)

Status :- Pending

[Delete](#)

Your product



Random
Name

[KNOW MORE](#)

Swap with



Why this
Kolaveri

[KNOW MORE](#)

Status :- Pending

[Accept](#)

[Decline](#)

[Delete](#)

Project Structure

Backend Development

- **Java Spring Boot:** Utilized for building robust and scalable backend services, providing a foundation for efficient server-side functionalities.
- **Maven:** Employed for automated project build management and dependency resolution.

Frontend Development

- **React.js:** Chosen as the frontend library for building interactive and dynamic user interfaces.
- **Material-UI:** Used for styling and theming components, ensuring a modern and consistent design.

The following pages were created

Database

- **MySQL:** Selected as the relational database management system to store and manage data efficiently.

API Integration

Overview

- **RESTful API Design :** Designing and developing a set of RESTful APIs to facilitate communication between the frontend and backend components. Ensuring adherence to REST principles, including stateless communication and resource-based endpoints.
- **Authentication Endpoints :** Implementation of secure authentication endpoints, allowing users to register, log in, and maintain session information securely.
- **Product Management API :** Creation of APIs for managing products, including functionalities for adding, updating, and deleting products. Endpoints to retrieve product details and lists for display on the frontend.

- **Swap Request Handling** : Designing APIs for handling swap requests, enabling users to initiate, accept, decline, and manage swap requests.
- **User Profile APIs** : Implementation of APIs to manage user profiles, allowing users to view and update their profile information.
- **Error Handling and Validation** : Integration of robust error handling mechanisms to provide meaningful responses and ensure a smooth user experience. Validation of incoming data to maintain data integrity and prevent erroneous requests.
- **Testing and Debugging** : Incorporation of testing mechanisms, including unit tests and integration tests,

API Documentation

Endpoints	HTTP Method	Input	Description
Swap Service			
createSwapRequest	POST	-	Creates Swap Request
getAllSwapRequests	GET	-	Gets All Swap Requests iwith status
deleteSwapRequest	DELETE	-	Deletes a outgoing Swap Request
updateSwapReques t	PUT	product_id, swapRequest	Updates the the request
acceptSwapReques t	PUT	product_id	Accept an Incomin Swap Request
declineSwapReques t	PUT	product_id	Decline an Incoming Swap Request
getSwapRequestByP roduct1	GET	user_id(1)	Get All Swap Requests for a particular product. Used to handle multiple swap requests for a single item and handle race condnions
getSwapRequestByP roduct2	GET	user_id(2)	Get All Swap Requests for a particular product. Used to handle multiple swap requests for a single item and handle race condnions

getSwapRequestByUser1	GET	user_id(1)	Get all swap requests by User1
getSwapRequestByUser2	GET	user_id(2)	Get all swap requests by User2
Product Service			
addProduct	POST	product	Add a product
getAllProducts	GET	-	Gets list of all products
getProductById	GET	product_id	Search for product by ID
getAllProductsByUserId	GET	user_id	Gives all products of a single user
deleteProduct	DELETE	product_id	Delete a Product
updateProduct	PUT	product_id, Product	Update product details
User Service			
createUser	POST	User	Creates a new user(register)
getUsers	GET	-	Gets all users
updateUser	PUT	user_id, User	Update user details
deleteUser	DELETE	user_id	Delete user
login	POST	email, password	Login User

DevOps Practice

Version Control (Git)

Git was used as the version control system during development to track code changes and enable collaboration between team members.

We created 2 repositories for Front-End and Backend respectively. Adding commit messages helps in maintaining the versions as we had to revert back to older ones multiple times. There was only one branch **main** created.

All the links can be found at the end of the document.

The screenshot displays the GitHub interface for the repository 'SPE Major backend'. The repository is owned by 'HasanAbbasMumtaz' and has 1 commit. The file browser shows a directory structure with folders like 'src', 'logs', and 'README.md'. The 'README' tab is selected, showing the repository's purpose: 'This is the back-end of the SPE Major project Swapsie, which provides a platform to swap and route products'. The page also includes sections for 'Releases', 'Packages', 'Contributors', and 'Languages'.

1. **Unit Testing:** Unit testing involves testing individual components or functions of a software application in isolation, typically at the method or class level. The purpose is to ensure that each unit of code works as expected and to identify and fix bugs at an early stage of development. JUnit, a widely used testing framework for Java, provides annotations and assertions to facilitate the creation and execution of unit tests.
2. **Integration Testing:** Integration testing focuses on verifying the interactions and interfaces between different components or modules of a system to ensure they

work seamlessly together. It tests the integration points where units come together. In the context of JUnit, integration tests may involve testing the collaboration of multiple classes or components to validate the overall behavior of the system when these units are integrated. Integration testing ensures that the components, when combined, function correctly.

Testing Description

User Case

- Create User:
 - Method under test: `UserServiceImpl#createUser(User)`.
 - Test scenarios include successful user creation and handling exceptions during the creation process.
- Get Users:
 - Method under test: `UserServiceImpl#getUser()`.
 - Test scenarios cover the correct retrieval of user lists and handling exceptions during retrieval.
- Update User:
 - Method under test: `UserServiceImpl#updateUser(long, User)`.
 - Test scenarios cover successful user updates, exceptions during updates, and handling attribute-specific exceptions.
- Delete User:
 - Method under test: `UserServiceImpl#deleteUser(long)`.
 - Test scenarios include successful user deletion, exceptions during deletion, attribute-specific exceptions, and user not found scenarios.
- Login Functionality:
 - Method under test: `UserServiceImpl#login(LoginRequest)`.
 - Test scenarios cover successful logins, user not found during login, exceptions during login, and a disabled incomplete test.
- Incomplete Test:
 - Test Name: `testUpdateUser5 (Disabled)`.
 - An incomplete test marked for future completion, involving a `NullPointerException` scenario.

Swap Requests Case

- Create SwapRequest Tests: Two test methods validate the `createSwapRequest` functionality.
 - Test 1 asserts successful creation by mocking the repository's `save` method and comparing the returned object.
 - Test 2 verifies that a `ResourceNotFoundException` is thrown when an error occurs during the creation process.
- Get All SwapRequests Tests: Two test methods validate the `getAllSwapRequests` functionality.
 - Test 1 ensures correct retrieval of swap requests by mocking the repository's `findAll` method.

- Test 2 verifies that a `ResourceNotFoundException` is thrown when an error occurs during retrieval.
- Delete SwapRequest Tests: Five test methods cover various scenarios for the `deleteSwapRequest` functionality.
 - Test 1 focuses on successful deletion, mocking the repository's `delete` and `findById` methods.
 - Test 2 asserts that a `ResourceNotFoundException` is thrown when the delete operation encounters an error.
 - Test 3 and Test 5 are marked as incomplete (disabled) with explanations for future completion.
 - Test 4 simulates a scenario where the swap request contains null attributes, leading to a `NullPointerException`.
- Accept and Decline SwapRequest Tests: Two test methods each for `acceptSwapRequest` and `declineSwapRequest` are outlined.
 - Tests verify the behavior when accepting or declining swap requests, covering scenarios involving null attributes and successful operations.
- Incomplete tests (disabled) are marked with explanations for future completion.

Product Case

- Add Product:
 - Verifies that adding a product returns the same product instance saved.
 - Throws a `ResourceNotFoundException` if an error occurs during the save operation.
- Get All Products:
 - Verifies that retrieving all products returns an empty list.
 - Throws a `ResourceNotFoundException` if an error occurs during the retrieval operation.
- Get Product by ID:
 - Verifies that retrieving a product by ID returns the expected product instance.
 - Throws a `ResourceNotFoundException` if the specified product ID is not found or if an error occurs during retrieval.
- Delete Product:
 - Verifies that deleting a product returns a success message.
 - Throws a `ResourceNotFoundException` if an error occurs during delete or if the specified product ID is not found.
- Update Product:
 - Verifies that updating a product returns the updated product instance.
 - Throws a `ResourceNotFoundException` if an error occurs during save, update, or if the specified product ID is not found.

Jenkins Pipeline Build

- An end-to-end Jenkins pipeline was created for continuous integration and delivery. Steps in the pipeline include:
- Checking out code from Git
- Running unit and integration tests

- Building and packaging the app (JAR, WAR files)
- Pushing Docker images
- Deploying containers to staging/production

We had built 2 separate pipelines for Front-End and Back-End

Back-End

Explanation

1. Git Clone: Clones the Git repository from the specified URL (https://github.com/Dharmin-23/SPE_Major_backend) and branch (main) using provided credentials (Github-credentials).
2. Running Test Cases: Executes Maven command (mvn clean test) to clean the project, compile the source code, and run test cases.
3. Maven Build: Performs a Maven build (mvn clean install), which includes cleaning the project, compiling the source code, running tests, and packaging the application into a JAR or WAR file.
4. Docker Build Image: Builds a Docker image named dharmin23/swapsie-backend using the Dockerfile in the project. The image is constructed based on the application artifacts generated in the previous Maven build stage.
5. Push Docker Image: Pushes the built Docker image to a Docker registry (assumed to be Docker Hub in this case) using credentials (DockerHubCred). This step makes the Docker image available for deployment or distribution.
6. Clean Docker Images: Removes unnecessary Docker containers and images to free up resources, ensuring a clean environment.

Script

```
pipeline {
    environment{
        dockerimage=""
    }
    agent any
    stages {
        stage('1: Git clone') {
            steps {
                git branch: 'main',credentialsId:'Github-credentials',
                url: 'https://github.com/Dharmin-23/SPE_Major_backend'
            }
        }
    }
}
```

```
stage("2: Running Test cases"){  
  steps{  
    sh "mvn clean test"  
  }  
}
```

```
stage("3: Maven Build"){  
  steps{  
    sh "mvn clean install"  
  }  
}
```

```
stage('4: Docker Build Image') {  
  steps {  
    script{  
      dockerimage=docker.build "dharmin23/swapsie-backend"  
    }  
  }  
}
```

```
stage('5: Push Docker Image') {  
  steps {  
    script{  
      docker.withRegistry("",'DockerHubCred'){  
        dockerimage.push()  
      }  
    }  
  }  
}
```

```

stage('6 : Clean docker images'){
    steps{
        script{
            sh 'docker container prune -f'
            sh 'docker image prune -f'
        }
    }
}
}
}

```

Front-End

Explanation

1. Git Clone: Clones the Git repository from the specified URL (https://github.com/Dharmin-23/SPE_Major_frontend) and branch (main) using provided credentials (Github-credentials).
2. Docker Build Image: Builds a Docker image named dharmin23/swapsie-frontend:latest using the Dockerfile in the frontend project. The image is constructed based on the frontend application artifacts.
3. Push Docker Image: Pushes the built Docker image to a Docker registry (assumed to be Docker Hub in this case) using credentials (DockerHubCred). This step makes the Docker image available for deployment or distribution.
4. Ansible Pull Docker Image: Uses Ansible to pull the Docker image on the target hosts. The Ansible playbook (ansible-playbook.yml) is executed with specified configurations, such as inventory and credentials (shubham).

Script

```

pipeline {
    environment{
        dockerimage=""
    }
    agent any
    stages {

```



```
stage('Git clone') {
    steps {
        git branch: 'main',credentialsId:'Github-credentials',url: 'https://github.com/Dharmin-23/SPE_Major_frontend'
    }
}

stage('Docker Build Image') {
    steps {
        script{
            dockerimage=docker.build "dharmin23/swapsie-frontend:latest"
        }
    }
}

stage('Push Docker Image') {
    steps {
        script{
            docker.withRegistry("",'DockerHubCred'){
                dockerimage.push()
            }
        }
    }
}
```

A large, semi-transparent watermark is centered over the code. It features a circular logo with a grid pattern and the text 'MIT-B' in a bold, sans-serif font. Below the logo, the text 'Pioneering Excellence in Education. Research & Innovation.' is written in a smaller, sans-serif font. The watermark is oriented diagonally.

```
stage('Ansible pull docker image') {
    steps {
        ansiblePlaybook colorized: true,
        credentialsId: 'shubham',
        disableHostKeyChecking: true,
        inventory: 'inventory',
```

```
playbook: 'ansible-playbook.yml'
```

Snapshots

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

Build History **trend** v

Filter builds...

✓ #5	7 Dec 2023, 15:05
✗ #4	7 Dec 2023, 15:00
✓ #3	7 Dec 2023, 13:44
✗ #2	7 Dec 2023, 13:33
✗ #1	7 Dec 2023, 13:04

Atom feed for all Atom feed for failures

Stage View

Average stage times:
(Average full run time: ~3min 5s)

	Success Git clone	Running Test cases	Maven Build	Docker Build Image	Push Docker Image	Stage 6 : Clean docker images
#5 Dec 07 15:05 No Changes	1s	50s	56s	6s	40s	932ms
#4 Dec 07 15:01 No Changes	3s	52s	56s	9s	44s	516ms failed
#3 Dec 07 13:44 No Changes	4s	1min 7s	1min 17s	11s	50s	
#2 Dec 07 13:33 No Changes	7s	2min 28s	1min 47s	17s	2s failed	
#1 Dec 07 13:04 No Changes						

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

Build History **trend** v

Filter builds...

✓ #19	11 Dec 2023, 20:25
✗ #18	11 Dec 2023, 20:22
✗ #17	11 Dec 2023, 20:15
✗ #16	11 Dec 2023, 20:11
✗ #15	11 Dec 2023, 20:10
✓ #14	11 Dec 2023, 18:21
✗ #13	11 Dec 2023, 18:15

Stage View

Average stage times:
(Average full run time: ~2min 54s)

	Git clone	Docker Build Image	Push Docker Image	Ansible pull docker image
#19 Dec 11 20:25 No Changes	1s	1min 32s	40s	6s
#18 Dec 11 20:22 No Changes	777ms	47s	1min 4s	15s
#17 Dec 11 20:15 No Changes	850ms	49s failed	35ms failed	34ms failed
#16 Dec 11 20:11 No Changes	1s	4min 31s aborted	25ms aborted	30ms aborted
#15 Dec 11 20:10 No Changes	802ms	1min 28s failed	71ms failed	43ms failed

Building/Packaging

- The Java Spring Boot backend app is packaged as a JAR file using Maven builds

- The React frontend is built using Webpack and Create React App scripts
- Build outputs are versioned and uploaded as artifacts in Jenkins

Maven Build

- Maven is a Java build automation and project management tool.
- It uses a Project Object Model (POM) file for configuration.
- Maven manages dependencies, automates builds, and follows convention over configuration.
- Build phases (e.g., compile, test) are defined in the build lifecycle.

Spring Boot:

- Spring Boot simplifies Java application development with opinionated defaults.
- It includes an embedded web server and starters for various tasks.
- Spring Boot Actuator provides production-ready features for monitoring.
- Applications are typically packaged as executable JAR files.
- Building and Packaging:
- Developers define project configurations and dependencies in the Maven POM file.

The `mvn clean install` command triggers Maven to compile, test, and package Spring Boot applications into executable JARs.

Containerization – Docker

Explanation

- Docker is used to containerize the Spring Boot backend, React frontend, and MySQL database
- Dockerfiles define the environment and dependencies
- Docker Compose spins up linked containers for local testing
- Containers are pushed to Docker Hub in pipeline

Usage in the project

Docker Compose File(`docker-compose.yml`):

- Defines services: MySQL, backend, and frontend.
- Configures MySQL service with a specific database, root password, and port.
- Maps container ports to host ports for backend and frontend.
- Establishes dependencies, ensuring backend starts after MySQL.

Dockerfiles:

1. Backend:
 - ☐ Uses OpenJDK 17.
 - ☐ Sets the working directory, copies the JAR file, exposes port 8081, and starts the application.
2. Frontend:
 - ☐ Uses Node Alpine.

- ☐ Sets the working directory, copies package files, installs dependencies, exposes port 3000, and starts the application.
- 3. Networks and Volumes:
 - ☐ Creates a custom bridge network named "swapsie-network" for communication.
 - ☐ Defines a volume named "swapsie-dp" to persist MySQL data.
- 4. Integration: Docker Compose integrates services, ensuring the backend communicates with MySQL.
 - ☐ Networking allows containers to interact within the defined "swapsie-network."
 - ☐ Volumes persist MySQL data across container restarts.

Overall, Docker and Docker Compose streamline deployment and orchestration in the project.

Snapshots

Front-End Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package.json ./
COPY package-lock.json ./
COPY ./ ./
RUN npm i
EXPOSE 3000
CMD ["npm", "run", "start"]
```

Back-End Dockerfile

```
# Fetching latest version of Java
FROM openjdk:17

# Setting up work directory
WORKDIR /app

# Copy the jar file into our app
COPY ./target/swapsie-0.0.1-SNAPSHOT.jar /app

# Exposing port 8080
EXPOSE 8081

# Starting the application
CMD ["java", "-jar", "swapsie-0.0.1-SNAPSHOT.jar"]
```

Docker-Compose

```

SPE_Major_frontend > docker-compose.yml
1  version: '3'
2  services:
3    mysql-swapsie-service:
4      image: mysql
5      container_name: mysql-swapsie-service
6      restart: always
7      networks:
8        - swapsie-network
9      environment:
10       MYSQL_DATABASE: "swapsieDevops"
11       MYSQL_ROOT_PASSWORD: "Password"
12     ports:
13       - "3307:3306"
14     volumes:
15       - swapsie-dp:/var/lib/mysql
16

```

```

swapsie-backend-container:
  image: "dharmin23/swapsie-backend"
  container_name: swapsie-backend-container
  restart: always
  networks:
    - swapsie-network
  # environment:
  #   # MYSQL_URL: jdbc:mysql://mysql-swapsie-service:3306/swa
  #   # MYSQL_NAME: root
  #   # MYSQL_PASSWORD: Password
  ports:
    - "8081:8081"
  depends_on:
    - mysql-swapsie-service

swapsie-frontend-container:
  image: "dharmin23/swapsie-frontend"
  container_name: swapsie-frontend-container
  restart: always
  networks:
    - swapsie-network
  ports:
    - "3000:3000"
  depends_on:
    - swapsie-backend-container

```

```

networks:
  swapsie-network:
    driver: bridge

volumes:
  swapsie-dp:

```

Ansible Explanation

Ansible is an open-source automation tool for configuration management, application deployment, and task automation.

It uses YAML-based playbooks to describe automation tasks and executes them on remote hosts.

Ansible provides idempotent tasks, ensuring the desired state is achieved regardless of the initial conditions.

It uses SSH for communication and requires no agent installation on managed nodes.

Ansible simplifies infrastructure management, making it scalable and reproducible.

Ansible Playbook

- Name: "Deploy Docker Image to Container"
- Hosts: "all" - Targets all hosts defined in the Ansible inventory.
- Tasks:
 - Copy Compose File
 - Copies the local "docker-compose.yml" file to the remote host.
 - Uses the "copy" Ansible module.

Executes the "docker-compose up -d" command on the remote host.

Uses the "command" Ansible module.

<!--3. **Prune Docker Images:**

- (Commented Out) Prunes dangling Docker images using "docker image prune --force."
- Uses the "command" Ansible module. -->

Integration in the Project

1. Deployment Process:
 - a. Ansible automates the deployment of Docker containers.
 - b. Copies the Docker Compose file to the target host.
 - c. Executes "docker-compose up -d" to start containers in the background.
2. Orchestration:
 - a. Coordinates the deployment process across multiple hosts defined in the Ansible inventory.
 - b. Ensures consistency in deploying the Dockerized application.
3. Extensibility:
 - a. Additional tasks, like pruning Docker images (commented out), can be easily integrated and executed.
 - b. Consistency and Reproducibility:
 - c. Ansible ensures consistent deployment regardless of the host's initial state.
 - d. Playbook tasks are idempotent, making it safe to run multiple times without unintended side effects.

Snapshots

```
---
- name: Deploy Docker Image to Container
  hosts: all
  tasks:
    - name: Copy compose file to remote host
      copy:
        src: ./docker-compose.yml
        dest: ./

    - name: run docker-compose file
      command: docker-compose up -d

    # - name: Prune the dangling Docker images
    #   command: docker image prune --force
```

Logging

Explanation

The ELK Stack, which stands for Elasticsearch, Logstash, and Kibana, is a powerful set of tools that can be used for centralized logging and log analysis. When combined with logs generated from a Spring Boot application, it provides a robust solution for managing and visualizing log data. Here's a step-by-step explanation of how you can use the ELK Stack with Spring Boot logs:

Elasticsearch

- Role: Elasticsearch is a distributed search and analytics engine. It will store and index the log data.
- Setup:
 - Install and configure Elasticsearch.
 - Ensure Elasticsearch is running on the specified host and port (default is localhost:9200).

Logstash

- Role: Logstash is a data processing pipeline that ingests, processes, and forwards log data to Elasticsearch.
- Setup:
 - Install and configure Logstash.
 - Create a Logstash configuration file (logstash.conf) that specifies how to parse and forward logs.
- Heres how logstash.conf looks like

```
conf Copy code

input {
  file {
    path => "/path/to/spring-boot-app.log"
    start_position => "beginning"
    sincedb_path => "/dev/null"
  }
}

filter {
  grok {
    match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:level} %{message}" }
  }
  date {
    match => [ "timestamp", "ISO8601" ]
  }
}
```

```
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "spring-boot-logs-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug }
}
```

Kibana

- Role: Kibana is a web-based user interface for visualizing and exploring log data stored in Elasticsearch.
- Setup:
 - Install and configure Kibana.
 - Access Kibana through a web browser (default is <http://localhost:5601>).
 - Set up an index pattern in Kibana to match the index pattern specified in the Logstash configuration (spring-boot-logs-*).

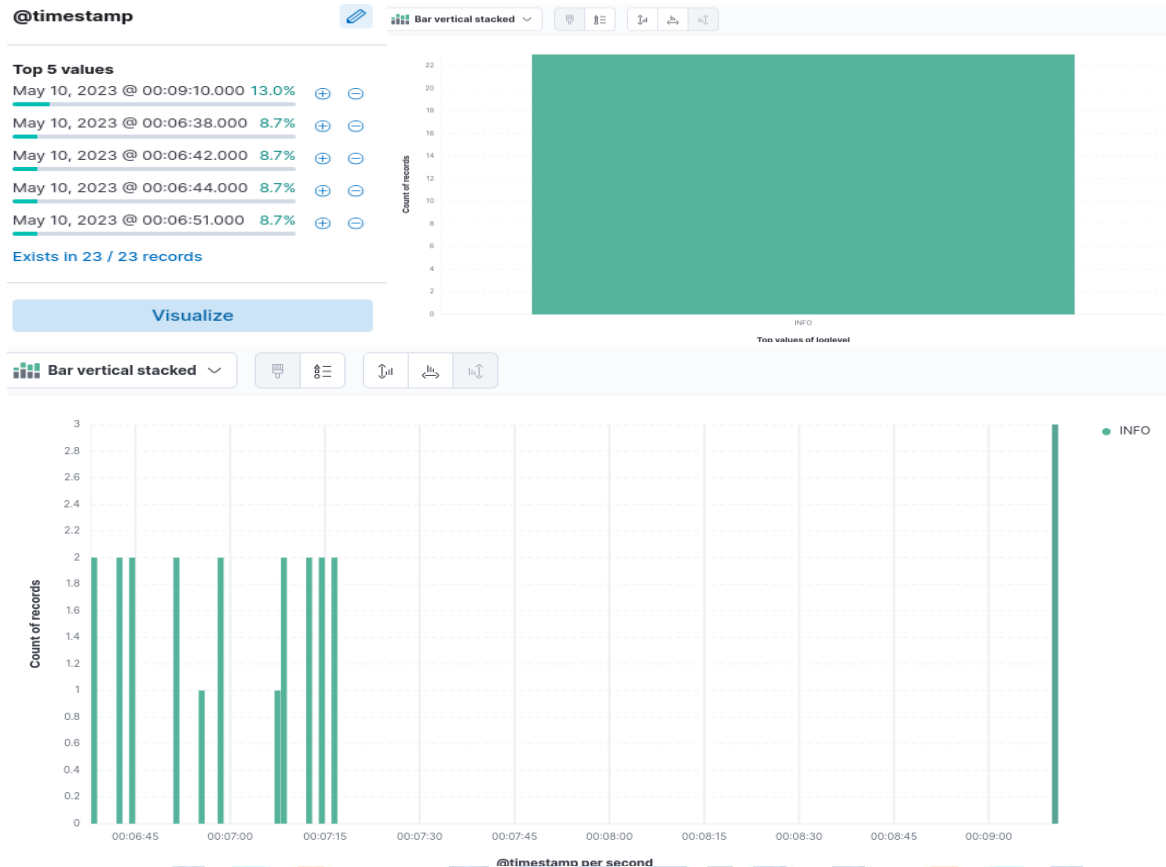
Spring Boot Application Logging Configuration

- Ensure that your Spring Boot application is configured to generate logs in a format that Logstash can parse. Spring Boot uses a default logging configuration that should work with the provided Logstash configuration.
- Common logging frameworks used with Spring Boot include Logback and Log4j2.

Viewing Logs

- Start your Spring Boot application.
- Log data will be collected by Logstash, processed, and indexed in Elasticsearch.
- Use Kibana to explore and visualize your logs. Create dashboards, visualizations, and searches to gain insights into your log data.

Snapshots



Challenges Faced

Integrating the various technologies into a cohesive platform was challenging at times. Getting components like React, Spring Boot, MySQL, Docker, Ansible, and Jenkins to work together seamlessly involved troubleshooting configuration issues.

Prioritizing features and scope was tricky with a small team. We had to focus on delivering core functionality first before expanding into nice-to-have items.

How To Run

Direct Running This Project

Install docker and docker-compose on your system.

Create a "docker-compose.yml" file in a directory and paste the following code in it:

version: '3'

services:

mysql-swapsie-service:

image: mysql

container_name: mysql-swapsie-service

restart: always

networks:

- swapsie-network

environment:

MYSQL_DATABASE: "swapsieDevops"

MYSQL_ROOT_PASSWORD: "Password"

ports:

- "3307:3306"

volumes:

- swapsie-dp:/var/lib/mysql

swapsie-backend-container:

image: "dharmin23/swapsie-backend"

container_name: swapsie-backend-container

restart: always

networks:

- swapsie-network

environment:

MYSQL_URL: jdbc:mysql://mysql-swapsie-service:3306/swapsieDevops?useSSL=false&useUnicode=yes&characterEncoding=UTF-8&allowPublicKeyRetrieval=true&serverTimezone=UTC

MYSQL_NAME: root

MYSQL_PASSWORD: Password

ports:

- "8081:8081"

depends_on:

- mysql-swapsie-service

swapsie-frontend-container:

image: "dharmin23/swapsie-frontend"

container_name: swapsie-frontend-container

restart: always

networks:

- swapsie-network

ports:

- "3000:3000"

depends_on:

- swapsie-backend-container

networks:

swapsie-network:

driver: bridge

volumes:

swapsie-dp:

Then run the following command:

```
sudo docker-compose up -d
```

For Complete Setup

The application can be run locally using Docker Compose. This will spin up containers for the React frontend, Spring Boot backend, and MySQL database.

Create 2 Jenkins pipelines, and paste the script given in the doc above. Jenkins can be accessed at

<http://localhost:8080/>

If its properly installed to have a detailed doc for steps on **Jenkins setup** download the doc from

<https://drive.google.com/drive/folders/1kUPyiOHaPkcRaf9s066mZcDKjhXtM2Sz?usp=sharing>

3 images will be created for front-end, back-end and the database repectively and automatically deployed. You can access it through by typing below url in your browser. The sessions for each system is stored permanently hence you can easily view all your previous swaps, and added products.

<http://localhost:3000/>

Links And References

Links

GitHub Repos

1. Front-End : https://github.com/Dharmin-23/SPE_Major_frontend
2. Back-End : https://github.com/Dharmin-23/SPE_Major_backend

Dockerhub

<https://hub.docker.com/repositories/dharmin23>

References

Docker Documentation - <https://docs.docker.com>

Ansible Documentation - <https://docs.ansible.com>

React Documentation - <https://reactjs.org/docs/>

Spring Boot Documentation - <https://spring.io/projects/spring-boot>

MySQL Documentation - <https://dev.mysql.com/doc/>



The logo for MIT-B Silver Jubilee is a large, stylized 'M' composed of multiple overlapping lines in blue, orange, and teal. Inside the 'M' is a circular emblem with a gear-like border. Text within the emblem includes 'SINCE 1977' at the top, 'MIT-B' in large letters in the center, and 'SILVER JUBILEE YEAR' at the bottom. Below the 'M' logo, the text 'Pioneering Excellence in Education. Research & Innovation.' is written in a serif font.

Pioneering Excellence in Education.
Research & Innovation.