

# Travel Planning App

CS 816 - Software Production Engineering



## International Institute of Information Technology, Bangalore

Under the Guidance of  
Prof. B Thangaraju

### Team - 77

(MT2023091)  
(MT2023172)

Abhishek Sharma  
Somesh Awasthi

# Index

S.No	Content	Page no
1	Problem Statement	2
2	Introduction	3
3	DevOps tool chain	4
4	DevOps Introduction	5
5	Steps Followed	7
6	DevOps Pipeline Execution Flow	11
7	Application Screenshots	29
8	Project Demo & GitHub Links	33

# Problem Statement

In today's fast-paced world, efficient travel planning is paramount for both leisure and business travelers. Our project focuses on the development of a sophisticated travel planning application tailored to meet the diverse needs of modern travelers. The objective is to create a user-centric platform that simplifies the entire travel planning process, from initial booking of transportation and accommodation to managing tour packages.

To achieve this, we leverage cutting-edge DevOps tools to automate development processes and implement robust monitoring tools for continuous performance optimization. By embracing these methodologies, we ensure rapid and agile development within our designated timeframe, without compromising on quality or reliability.

We aim to build our application into a comprehensive travel companion, integrating features such as creating tour packages, booking packages, and adding agents who will receive user package requests and handle all necessary bookings. By consolidating these diverse services into a single, cohesive platform, we aspire to provide users with a seamless and hassle-free experience, eliminating the need to juggle multiple apps or websites for their travel needs.

# Introduction

Travel planning involves multiple components, including transportation, accommodation, and activities, which often require users to navigate several platforms. This fragmentation can lead to increased stress and time consumption. Our project addresses this by developing an integrated application that provides end-to-end travel planning solutions.

## Objectives:

1. Destination Selection: Users kick start their journey by specifying their desired travel destination.
2. Attraction Selection: Present users with a curated selection of must-visit attractions, customized based on their trip duration and preferences.
3. Itinerary Planning: Arrange selected attractions into a cohesive itinerary, meticulously organized by day for optimal exploration.

## Project Technology Stack:

1. Front End - ReactJS
2. Back End - Spring Boot
3. Database - MySql
4. Version Control System - Git
5. CI/CD – Jenkins
6. Containerisation – Docker
7. Container Orchestration - Docker Compose
8. Configuration Management tool - Ansible
9. Monitoring and Alerting tool - ELK Stack

## Important Links:

1. GitHub repository - [click here](#)
2. DockerHub Registry Link:
  - Frontend Image Repository Link: [click here](#)
  - Backend Image Repository Link: [click here](#)

## DevOps tool chain

You can use any set of DevOps tool chains you want, but the pipeline would be the same. The pipeline includes,

1. Source Control Management:
  - Tools: GitHub, GitLab, BitBucket, etc.
  - Purpose: To manage and track changes in the source code, facilitate collaboration, and ensure version control.
2. Testing:
  - Tools: JUnit, Selenium, PyUnit, etc.
  - Purpose: To validate the functionality of individual components and ensure code quality through automated testing.
3. Build:
  - Tools: Maven, Gradle, Ant, etc.
  - Purpose: To automate the process of compiling the code, packaging it into executable formats, and managing dependencies.
4. Continuous Integration:
  - Tools: Jenkins, GitLab CI, Travis CI, etc.
  - Purpose: To automate the integration of code changes from multiple contributors, run automated tests, and build the application continuously.
5. Containerization:
  - Tool: Docker

- Purpose: To create lightweight, portable containers for the application, ensuring consistency across different environments. Push the created Docker image to Docker Hub.

## 6. Deployment and Configuration Management:

- Tools: Chef, Puppet, Ansible, Rundeck
- Purpose: To automate the configuration of environments, deploy Docker containers, and manage application settings. These tools are used to pull Docker images and run them on managed hosts.

## 7. Container Orchestration:

- Tools: Docker Compose, Kubernetes, OpenStack
- Purpose: To manage container deployment, scaling, and operations. Deployment can be done on a local machine or using cloud services such as Amazon AWS, Google Cloud, or other third-party clouds.

## 8. Monitoring:

- Tool: ELK Stack (Elasticsearch, Logstash, Kibana)
- Purpose: To monitor application performance, collect and analyze logs, and visualize data for proactive maintenance and troubleshooting.

# DevOps Introduction:

## What's DevOps?

DevOps derives its name from the fusion of development (Dev) and operations (Ops).

DevOps integrates development and operations to enhance software delivery efficiency, speed, and security, surpassing traditional methods. It provides a competitive advantage through a swifter and more iterative software development process rooted in Agile principles.

Additionally, DevOps promotes collaboration across the development cycle, leading to improved application flow and value delivery. It symbolizes a cultural transformation within IT, prioritizing incremental development, rapid deployment, and collective

responsibility for business results, drawing from Agile, lean methodologies, and systems theory.

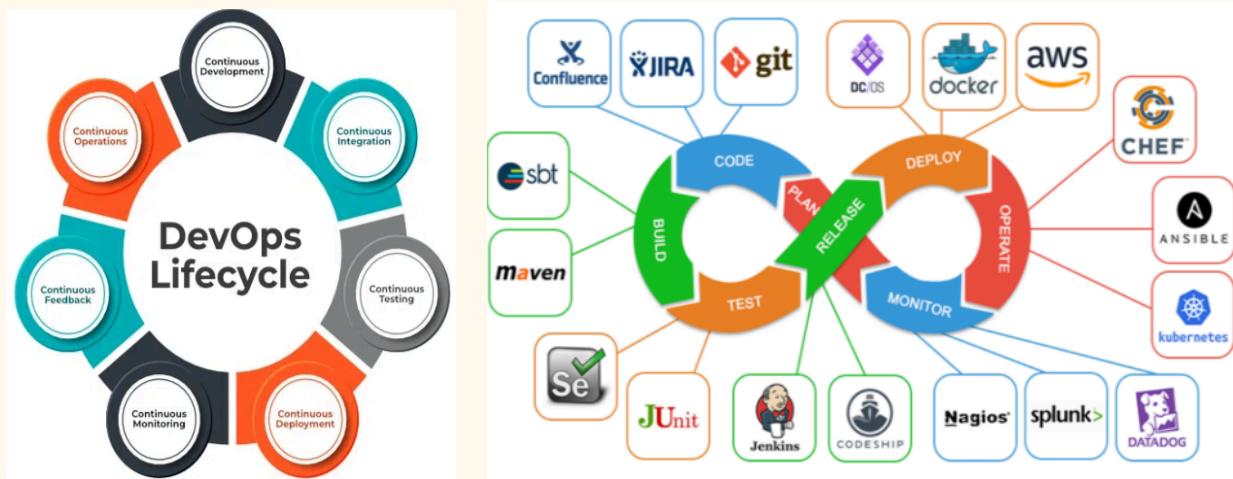


Fig. DevOps Lifecycle Phases

## Key Goals and Benefits of DevOps :

### Goals of DevOps

1. Ensures effective collaboration between teams
2. Creates scalable infrastructure platforms
3. Builds on-demand release capabilities
4. Provides faster feedback

### Benefits of DevOps

1. Smarter work and faster release
2. Quick resolution of issues
3. Better collaboration between teams
4. Fostering innovative mindsets
5. Faster threat detection
6. Increased customer satisfaction

## Steps Followed

### 1. Pipeline Declaration

- Git Installation:
  - Open a terminal window.
  - Update the package index to ensure you have the latest information about available packages by running the following command:  
`sudo apt update`
  - Once the update is complete, install Git by running the following command:  
`sudo apt install git`
  - Once the installation is complete, verify that Git was installed successfully by checking its version. You can do this by running the following command:  
`git --version`

```

Feb 25 7:16 PM
somesh@somesh-Predator-PH315-51:~ git:(main) ~ sudo apt update
[sudo] password for somesh:
Hit:2 https://brave-browser-apt-release.s3.brave.com stable InRelease
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:3 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:4 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:5 http://in.archive.ubuntu.com/ubuntu mantic InRelease
Hit:6 http://security.ubuntu.com/ubuntu mantic-security InRelease
Ign:8 https://ppa.launchpadcontent.net/ansible/ubuntu mantic InRelease
Get:9 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:10 http://in.archive.ubuntu.com/ubuntu mantic-updates InRelease [109 kB]
Err:11 https://ppa.launchpadcontent.net/ansible/ubuntu mantic Release
  404 Not Found [IP: 185.125.190.80 443]
Get:12 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:13 http://in.archive.ubuntu.com/ubuntu mantic-backports InRelease
Get:14 http://in.archive.ubuntu.com/ubuntu mantic-updates/main i386 Packages [160 kB]
Get:15 http://in.archive.ubuntu.com/ubuntu mantic-updates/main amd64 Packages [311 kB]
Reading package lists... Done
N: Skipping acquire of configured file 'main/binary-i386/Packages' as repository 'https://brave-browser-apt-release.s3.brave.com stable InRelease' doesn't support architecture 'i386'
E: The repository 'https://ppa.launchpadcontent.net/ansible/ubuntu mantic Release' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8)' manpage for repository creation and user configuration details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.40.1-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
git version 2.40.1

```

- Jenkins Installation:

- Install Java Development Kit (JDK):

```
sudo apt install default-jdk
```

- Install Jenkins

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

- Install ca-certificates

```
sudo apt install ca-certificates
```

- Install Jenkins

```
sudo apt-get update
```

```
sudo apt-get install jenkins
```

- To check Jenkins version

```
vim /var/lib/jenkins/config.xml
```

- To copy admin password

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

- Once you have installed it, we can start it and check its status by typing the following commands:

```
sudo systemctl start jenkins
```

```
sudo systemctl status jenkins
```

```
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Jenkins Continuous Integration Server
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; preset: enabled)
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Active: active (running) since Sun 2024-02-25 07:49:57 IST; 3min 5s ago
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Main PID: 32879 (java)
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Tasks: 62 (limit: 18913)
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: Memory: 2.0G
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: CPU: 36.397s
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: CGroup: /system.slice/jenkins.service
Feb 25 07:49:46 somesh-Predator-PH315-51 jenkins[32879]: [32879] /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.017+0000 [id=35] INFO Jenkins.InitReactorRunner$1#onAttained:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.405+0000 [id=44] INFO h.p.b.q.GlobalTimeOutConfiguration#load:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.469+0000 [id=44] WARNING o.j.p.d.DockerBuilder$DescriptorImpl:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.815+0000 [id=33] INFO jenkins.InitReactorRunner$1#onAttained:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.846+0000 [id=49] INFO jenkins.InitReactorRunner$1#onAttained:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.879+0000 [id=34] INFO jenkins.InitReactorRunner$1#onAttained:>
Feb 25 07:49:56 somesh-Predator-PH315-51 jenkins[32879]: 2024-02-25 07:49:56.891+0000 [id=34] INFO jenkins.InitReactorRunner$1#onAttained:>
```

After Jenkins is installed and operational, configuration is necessary. Begin by accessing port 8080 on the localhost. The URL format is: <https://localhost:8080>. Upon visiting this URL for the first time, Jenkins will prompt for the password, which can be found in the log file.

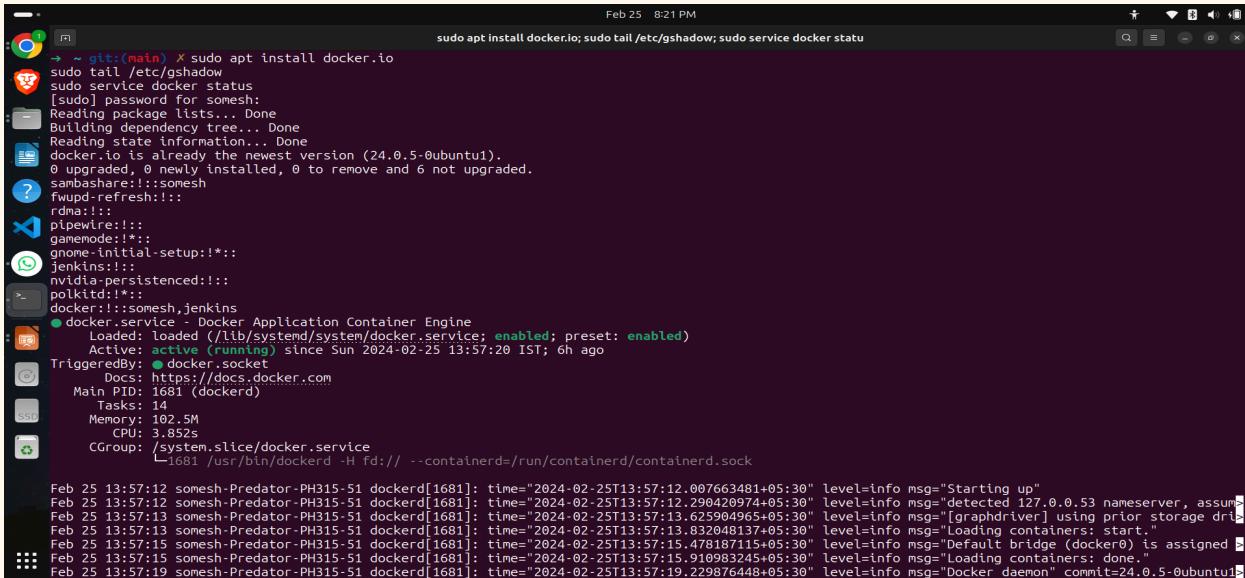
```
somesh@somesh-Predator-PH315-51:~
```

```
→ ~ git:(main) ✘ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
40969a616dae4cebaab643b0bcbb6eb4
→ ~ git:(main) ✘
```

- Install Recommended Plugins: After unlocking Jenkins, you'll be prompted to install the recommended plugins or select specific plugins to install.
- Create Admin User: Once the plugin installation is complete, you'll need to set up an admin user account.
- Configure Jenkins: After creating the admin user, Jenkins will ask you to set up the URL for accessing Jenkins and optionally install additional plugins.

- **Docker Installation:**

- Install Docker  
sudo apt install docker.io
- After Docker has been installed, it's important to ensure that Docker is within the same user group as Jenkins. This can be verified by executing the following command.
- Check Docker Service Status  
sudo service docker status
- Pull Docker Images  
docker pull ubuntu
- List Docker Images  
docker images
- Explore Docker Hub: Visit Docker Hub (<https://hub.docker.com/>) to discover and explore available Docker images and repositories



```
Feb 25 08:21 PM
git:(main) ~ sudo apt install docker.io; sudo tail /etc/gshadow; sudo service docker status
[sudo] password for somesh:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker.io is already the newest version (24.0.5-0ubuntu1).
Upgraded: 0 newly installed, 0 to remove and 6 not upgraded.
Same file:::somesh
fwupd-refresh:::
rda:::
pipewire:::
gamed:::
gnome-initial-setup:::
jenkins:::
nvidia-persistenced:::
polkitd:::
docker:::somesh,jenkins
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-02-25 13:57:20 IST; 6h ago
     TriggeredBy: ● docker.socket
      Docs: https://docs.docker.com
     Main PID: 1681 (dockerd)
       Tasks: 14
      Memory: 102.5M
        CPU: 3.852s
      CGroup: /system.slice/docker.service
              └─1681 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Feb 25 13:57:12 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:12.007663481+05:30" level=info msg="Starting up"
Feb 25 13:57:12 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:12.290420974+05:30" level=info msg="detected 127.0.0.53 nameserver, assuming it is a hostnameserver"
Feb 25 13:57:13 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:13.625904965+05:30" level=info msg="graphdriver[graphdriver] using prior storage driver"
Feb 25 13:57:13 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:13.832048137+05:30" level=info msg="Loading containers: start."
Feb 25 13:57:15 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:15.478187115+05:30" level=info msg="Default bridge (docker0) is assigned to the host interface eth0"
Feb 25 13:57:15 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:15.910983245+05:30" level=info msg="Loading containers: done."
Feb 25 13:57:19 somesh-Predator-PH315-51 dockerd[1681]: time="2024-02-25T13:57:19.229876448+05:30" level=info msg="Docker daemon" commit=24.0.5-0ubuntu1
```

- **Ansible Installation:**

- Install Ansible  
sudo apt install ansible
- Check Ansible Version  
ansible --version
- Verify Installation  
ansible localhost -m ping

```
somesh@somesh-Predator-PH315-51:~ 
→ ~ git:(main) ✘ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ansible is already the newest version (7.7.0+dfsg-1).
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
→ ~ git:(main) ✘ ansible --version
ansible [core 2.14.9]
  config file = None
  configured module search path = ['~/home/somesh/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = ~/home/somesh/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.11.6 (main, Oct 8 2023, 05:06:43) [GCC 13.2.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
→ ~ git:(main) ✘ ansible localhost -m ping
[WARNING]: No inventory was parsed, only implicit localhost is available
localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

- However, a common error that pops up is the following

```
# /opt/freeware/bin/ansible --version
ERROR: Ansible requires the locale encoding to be UTF-8; Detected None.
```

- To address this, we are going to change the locale encoding using the following command.

Type this command

```
sudo nano /etc/default/locale
```

It will show

```
LANG=en_IN
LC_CTYPE=en_IN:en
```

change it to

```
LANG=en_US.UTF-8
LC_CTYPE=en_US.UTF-8
```

```
somesh@somesh-Predator-PH315-51:~ 
→ ~ git:(main) ✘ sudo cat /etc/default/locale
# File generated by update-locale
LANG=en_US.UTF-8
LC_CTYPE=en_US.UTF-8
→ ~ git:(main) ✘
```

# DevOps Pipeline Execution Flow

## Execution Flow

### 1. CI/CD Pipeline

Our Jenkins pipeline is designed to automate the entire build, test, and deployment process of our travel planning application. Below is an explanation of each stage in the pipeline, along with the execution flow:

## I. Pipeline Declaration

The pipeline is declared using the `pipeline` block, which defines the entire CI/CD process. We use the `agent any` directive to run the pipeline on any available Jenkins agent.



The screenshot shows the Jenkins Pipeline configuration interface. The title bar says "Pipeline". Under "Definition", there is a "Pipeline script" section with a dropdown menu. Below it is a "Script" editor containing Groovy code for the pipeline. The code defines environment variables and stages. A checkbox for "Use Groovy Sandbox" is checked. At the bottom are "Save" and "Apply" buttons.

```

Pipeline
Definition
Pipeline script
Script ? 
2  pipeline {
3    agent any
4
5    environment {
6      M2_HOME = "/opt/apache-maven-3.9.6"
7      PATH = "${M2_HOME}/bin:${env PATH}"
8      DOCKER_HOST = "unix:///var/run/docker.sock"
9      DOCKERHUB_USERNAME = "someshawasthi"
10     GITHUB_REPO_URL = 'https://github.com/abhisheksharma402/SPEMajorProject'
11
12     // KUBECONFIG = credentials('kube-cred-main')
13   }
14
15   stages {
16     ...
17   }
18 }
19
20
Use Groovy Sandbox ?
Pipeline Syntax
Save Apply

```

## II. Environment Variables

Environment variables are defined to store values that will be used across the pipeline stages, such as Docker Hub username and GitHub repository URL.

```

environment {

  M2_HOME = "/opt/apache-maven-3.9.6"

  PATH = "${M2_HOME}/bin:${env PATH}"

  DOCKER_HOST = "unix:///var/run/docker.sock"

  DOCKERHUB_USERNAME = "someshawasthi"

  GITHUB_REPO_URL = 'https://github.com/abhisheksharma402/SPEMajorProject'

  // KUBECONFIG = credentials('kube-cred-main')

}

```

### III. Stages

The pipeline consists of multiple stages, each representing a step in the CI/CD process.

#### A. Checkout

In this stage, we clean the workspace and checkout the code from the GitHub repository.

```
19
20
21 stage('Checkout') {
22   steps {
23     script {
24       cleanWs()
25       git branch: 'main', url: "${GITHUB_REPO_URL}"
26     }
27   }
28 }
29
```

#### B. Making Port Available

We stop all running Docker containers, remove them, and delete all Docker images to free up ports and resources.

```
29
30
31
32 stage('Making Port Available') {
33   steps {
34     script {
35       // Stop all containers
36       sh 'docker ps -aq | xargs -r docker stop'
37       sh 'docker ps -aq | xargs -r docker rm'
38       sh 'docker images -q | xargs -r docker rmi -f'
39     }
40   }
41 }
42
43
```

## C. Maven Build

We build the backend project using Maven, skipping tests for now

```
39
40    stage('Maven Build') {
41        steps {
42            dir('SPE-Project-Backend/main') {
43                script {
44                    sh 'mvn clean install -DskipTests'
45                }
46            }
47        }
48    }
49
```

## D. Build Docker Images

We build Docker images for the backend and frontend services using the respective Dockerfiles.

```
52
53    stage('Build Docker Images') {
54        steps {
55            script {
56                sh "docker build -t ${DOCKERHUB_USERNAME}/travelguide-frontend -f Dockerfiles/FrontendDockerfile ."
57                sh "docker build -t ${DOCKERHUB_USERNAME}/travelguide-backend -f Dockerfiles/BackendDockerfile ."
58            }
59        }
60    }
61
62
```

## E. List Docker Images

We list all Docker images to verify that the images have been built successfully.

```
65
66    stage('List Docker Images') {
67        steps {
68            script {
69                // Use Docker CLI to list images
70                sh 'docker images'
71            }
72        }
73    }
74
75
```

## F. Push Docker Images to Registry Stage

Docker images are tagged and pushed to Docker Hub in this stage.

```

78
79      stage('Push Docker Images to Registry') {
80        steps {
81          script {
82            // Tag and push backend image
83            sh 'docker image tag someshawasthi/travelguide-backend:latest ${DOCKERHUB_USERNAME}/travelguide-backend:version1.0'
84            docker.withRegistry('', 'someshawasthi-dockerhub') {
85              sh "docker push ${DOCKERHUB_USERNAME}/travelguide-backend:version1.0"
86            }
87          }
88        }
89      }
90      // Tag and push frontend image
91      sh 'docker image tag someshawasthi/travelguide-frontend:latest ${DOCKERHUB_USERNAME}/travelguide-frontend:version1.0'
92      docker.withRegistry('', 'someshawasthi-dockerhub') {
93        sh "docker push ${DOCKERHUB_USERNAME}/travelguide-frontend:version1.0"
94      }
95    }
96  }
97

```

## G. Run Ansible Inventory and Playbook Stage

This stage runs the Ansible playbook to deploy the application using the inventory file provided.

```

100
101      stage('Run Ansible Inventory and Playbook') {
102        steps {
103          script {
104            ansiblePlaybook (
105              playbook: 'playbook.yml',
106              inventory: 'inventory.txt',
107              extras: '-K'
108            )
109          }
110        }
111      }
112

```

## H. Testing Frontend Stage

In the final stage, the frontend application is tested using npm and Jest.

```

112
113      stage("Testing Frontend"){
114        steps {
115          dir('SPE-Project-Frontend/SPE-Major-Project/SPEMajorProject'){
116            script{
117              sh 'npm install --save-dev jest-environment-jdom --force'
118              sh 'npm install --save-dev jest@latest ts-jest@latest --force'
119              // sh 'npm install -g ts-jest --force'
120              sh 'npm run test:login'
121            }
122          }
123        }
124      }

```

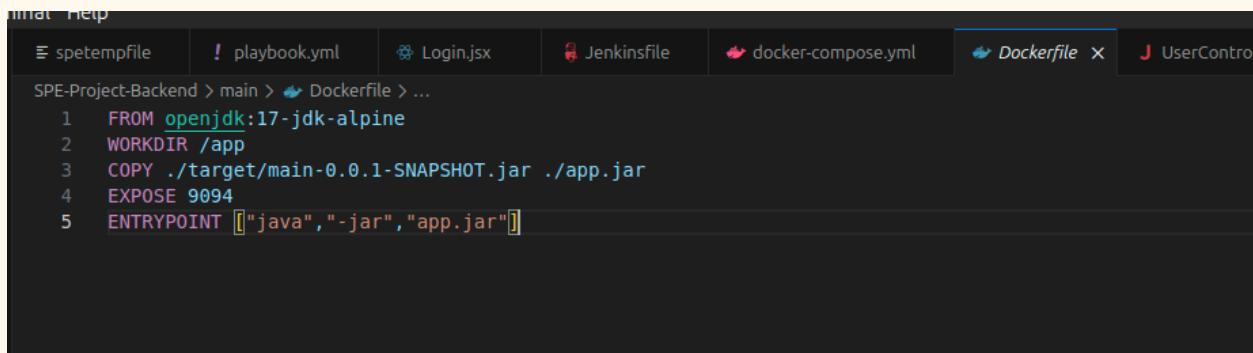
## 2. Containerization

In our travel planning application, we employ containerization to ensure consistent and reliable deployment across various environments.

Containerization encapsulates the application and its dependencies into a container, enabling smooth transitions from development to production without compatibility issues. We utilize Docker to containerize both the backend and frontend and database components of our application. Below is a detailed explanation of the Dockerfiles used for each component.

### a. SPE-Project-Backend

- i. For the backend component, we use a Dockerfile that sets up an environment for running a Java Spring Boot application. Here is the Dockerfile for the backend:



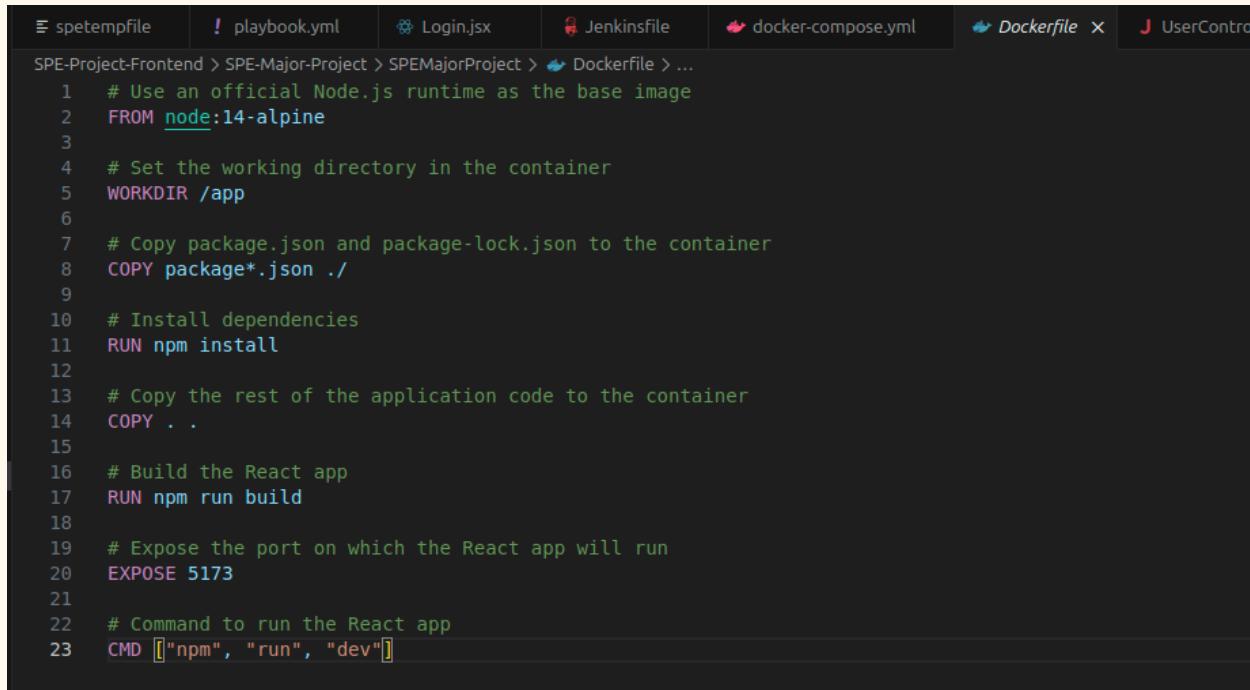
The screenshot shows a code editor interface with a dark theme. The tab bar at the top includes 'spetempfile', 'playbook.yml', 'Login.jsx', 'Jenkinsfile', 'docker-compose.yml', 'Dockerfile' (which is currently selected), and 'UserController'. The main editor area displays the following Dockerfile content:

```
SPE-Project-Backend > main > Dockerfile > ...
1 FROM openjdk:17-jdk-alpine
2 WORKDIR /app
3 COPY ./target/main-0.0.1-SNAPSHOT.jar ./app.jar
4 EXPOSE 9094
5 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Explanation:

- FROM openjdk:17-jdk-alpine: This line specifies the base image, which is a lightweight Alpine Linux distribution with OpenJDK 17. This choice ensures a minimal footprint and a secure environment for running Java applications.
- WORKDIR /app: Sets the working directory inside the container to /app. All subsequent instructions will be executed in this directory.

- COPY ./target/main-0.0.1-SNAPSHOT.jar ./app.jar: Copies the built JAR file from the host machine into the container. This JAR file contains the compiled Spring Boot application.
  - EXPOSE 9094: Exposes port 9094 to the host, allowing external access to the application running inside the container. This is the port on which the backend service will be accessible.
  - ENTRYPOINT ["java","-jar","app.jar"]: Defines the command to run when the container starts. It uses the java -jar command to execute the JAR file.
- b. SPE-Project-Frontend
- i. For the frontend component, we use a Dockerfile that sets up an environment for running a React application. Here is the Dockerfile for the frontend:



```

spetempfile playbook.yml Login.jsx Jenkinsfile docker-compose.yml Dockerfile UserContro
SPE-Project-Frontend > SPE-Major-Project > SPEMajorProject > Dockerfile > ...
1 # Use an official Node.js runtime as the base image
2 FROM node:14-alpine
3
4 # Set the working directory in the container
5 WORKDIR /app
6
7 # Copy package.json and package-lock.json to the container
8 COPY package*.json .
9
10 # Install dependencies
11 RUN npm install
12
13 # Copy the rest of the application code to the container
14 COPY .
15
16 # Build the React app
17 RUN npm run build
18
19 # Expose the port on which the React app will run
20 EXPOSE 5173
21
22 # Command to run the React app
23 CMD [ "npm", "run", "dev" ]

```

## Explanation:

- FROM node:14-alpine: Specifies the base image, which is a lightweight Alpine Linux distribution with Node.js 14. This choice ensures a minimal and efficient environment for running Node.js applications.

- WORKDIR /app: Sets the working directory inside the container to /app. This is where the application files will be stored and executed.
- “COPY package.json .\*/”: Copies package.json and package-lock.json files to the working directory. These files contain the metadata and dependencies for the Node.js application.
- RUN npm install: Installs the Node.js dependencies defined in the package.json file. This ensures that all necessary libraries and modules are available in the container.
- COPY . ..: Copies the entire application code from the host machine to the container. This includes all source files and assets needed to build and run the application.
- RUN npm run build: Builds the React application. This step compiles the React code and optimizes it for production.
- EXPOSE 5173: Exposes port 5173 to the host, allowing external access to the React application running inside the container. This is the port on which the frontend service will be accessible.
- CMD ["npm", "run", "dev"]: Defines the command to run when the container starts. It uses the npm run dev command to start the React development server.

### 3. Ansible Playbook

The deployment of our travel planning application leverages Ansible to ensure an automated, consistent, and efficient deployment process. Ansible is a powerful automation tool that allows us to manage and configure systems, deploy applications, and orchestrate advanced workflows. By using Ansible, we can streamline our deployment process and minimize manual intervention, thereby reducing the risk of human errors. Below is the Ansible playbook used for deploying our Dockerized application.

## Ansible Playbook for Deployment

The following Ansible playbook defines the tasks required to deploy the application using Docker Compose.



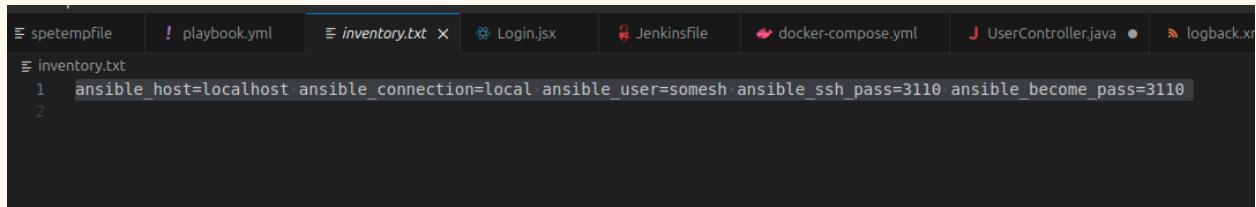
```
! playbook.yml
1 ---
2   - name: Run Docker Compose
3     hosts: localhost
4     remote_user: somesh
5     connection: local
6     gather_facts: yes
7     tasks:
8       - name: Run Docker Compose
9         command: docker compose up -d
```

### Explanation:

- **name: Run Docker Compose:** This is the name of the playbook, providing a brief description of its purpose, which is to run Docker Compose for our application.
- **hosts: localhost:** Specifies that the playbook will be run on the local machine. This is suitable for our development and initial deployment purposes.
- **remote\_user: somesh:** Defines the user under which the playbook will be executed. In this case, it is the user somesh.
- **connection: local:** Indicates that the playbook will use a local connection method, meaning it will run on the local machine rather than connecting to a remote host.
- **gather\_facts: yes:** Enables fact gathering, which collects information about the system (e.g., OS details, network interfaces). This can be useful for conditional tasks and configurations within the playbook.

## Ansible Inventory

Before diving into the playbook, we need to set up an Ansible inventory file. This file defines the hosts and groups of hosts that Ansible will manage. For our purposes, we are targeting the local machine for deployment.



```
! playbook.yml      inventory.txt      Login.jsx      Jenkinsfile      docker-compose.yml      UserController.java      logback.xml
inventory.txt
1 ansible_host=localhost ansible_connection=local ansible_user=somesh ansible_ssh_pass=3110 ansible_become_pass=3110
2
```

## 4. Container Orchestration

Container orchestration is a crucial aspect of deploying and managing containerized applications in a production environment. It involves automating the deployment, scaling, and management of containerized applications. In our project, we use Docker Compose for orchestrating the services that make up our travel planning application. This section details the Docker Compose setup used to manage the various containers involved in our project.

## Docker Compose File

The docker-compose.yml file is used to define and run multi-container Docker applications. Below is the configuration file used for orchestrating our application's services.

```
! docker-compose.yml
1 version: '3'
2 services:
3   mysql:
4     image: mysql:8
5     container_name: mysql-database
6     restart: always
7     ports:
8       - '3306:3306'
9     environment:
10      - MYSQL_ROOT_PASSWORD=Hope@402
11      # - MYSQL_ROOT_PASSWORD=root
12      - MYSQL_DATABASE=travelguide
13     volumes:
14       - mysql_data:/var/lib/mysql
15     networks:
16       - networkmysql
17     healthcheck:
18       test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
19       interval: 30s
20       timeout: 10s
21       retries: 5
22
23   frontend:
24     image: abhisheksharma402/frontendservice:latest
25     container_name: myfrontend
26     ports:
27       - "5173:5173"
28     networks:
29       - networkmysql
30
31   backend:
32     image: abhisheksharma402/backendservice:latest
33     container_name: mybackend
34     ports:
35       - "9092:9094"
36     environment:
37       - MYSQL_HOST=mysql
38       - MYSQL_PORT=3306
39       - MYSQL_DB_NAME=travelguide
40       - MYSQL_USER=root
41       - MYSQL_PASSWORD=Hope@402
42       - LOGGING_LOGSTASH_HOST=logstash
43       - LOGGING_LOGSTASH_PORT=5001
44       # - MYSQL_PASSWORD=root
```

```

30
31     backend:
32         image: abhisheksharma402/backendservice:latest
33         container_name: mybackend
34         ports:
35             - "9092:9094"
36         environment:
37             - MYSQL_HOST=mysql
38             - MYSQL_PORT=3306
39             - MYSQL_DB_NAME=travelguide
40             - MYSQL_USER=root
41             - MYSQL_PASSWORD=Hope@402
42             - LOGGING_LOGSTASH_HOST=logstash
43             - LOGGING_LOGSTASH_PORT=5001
44             # - MYSQL_PASSWORD=root
45         networks:
46             - networkmysql
47         depends_on:
48             - mysql
49         restart: on-failure:3
50         healthcheck:
51             test: ["CMD", "curl", "-f", "http://localhost:9092/actuator/health"]
52             interval: 30s
53             timeout: 10s
54             retries: 3
55
56     elasticsearch:
57         image: docker.elastic.co/elasticsearch/elasticsearch:8.3.3
58         container_name: elastic
59         environment:
60             - bootstrap.memory_lock=true
61             - ES_JAVA_OPTS=-Xms512m -Xmx512m
62             - "discovery.type=single-node"
63             - xpack.security.enabled=false
64         ports:
65             - "9200:9200"
66         # volumes:
67         #   - elastic_data:/usr/share/elasticsearch/data
68         networks:
69             - networkmysql
70

```

```

71
72     kibana:
73         image: docker.elastic.co/kibana/kibana:8.3.3
74         container_name: kibana
75         ports:
76             - "5601:5601"
77         environment:
78             - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
79         depends_on:
80             - elasticsearch
81         networks:
82             - networkmysql
83
84
85     logstash:
86         image: docker.elastic.co/logstash/logstash:8.3.3
87         container_name: logstash
88         volumes:
89             - /home/somesh/iiitb_files/SPE/SPEMajorProject/logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
90             - /home/somesh/iiitb_files/SPE/SPEMajorProject/logstash/pipeline:/usr/share/logstash/pipeline:ro
91         ports:
92             - "5001:5001/tcp"
93             - "5001:5001/udp"
94             - "9600:9600"
95         environment:
96             - LS_JAVA_OPTS=-Xmx512m -Xms512m
97         networks:
98             - networkmysql
99         depends_on:
100            - elasticsearch
101
102    networks:
103        networkmysql:
104    volumes:
105        mysql_data:
106

```

## Explanation:

- MySQL Database:
  - Image: Uses the official MySQL 8 image.
  - Container Name: Named `mysql-database`.
  - Restart Policy: Set to always restart the container if it stops.
  - Ports: Exposes port 3306 for database access.
  - Environment Variables: Sets the root password and initializes the `travelguide` database.
  - Volumes: Persists data using a named volume `mysql_data`.
  - Networks: Connects to the `networkmysql` network.
  - Health Check: Ensures the MySQL service is running and responsive.
- Frontend Service:
  - Image: Uses the latest image of the frontend service.
  - Container Name: Named `myfrontend`.
  - Ports: Exposes port 5173 for the React application.
  - Networks: Connects to the `networkmysql` network.
- Backend Service:
  - Image: Uses the latest image of the backend service.
  - Container Name: Named `mybackend`.
  - Ports: Exposes port 9094 for the backend service.

- Environment Variables: Configures database connection and logging settings.
- Networks: Connects to the networkmysql network.
- Depends On: Ensures the MySQL service is available before starting.
- Restart Policy: Restarts the container on failure, up to three times.
- Health Check: Ensures the backend service is running and responsive.
- Elasticsearch:
  - Image: Uses the Elasticsearch 8.3.3 image.
  - Container Name: Named elastic.
  - Environment Variables: Configures memory and node settings.
  - Ports: Exposes port 9200 for Elasticsearch.
  - Networks: Connects to the networkmysql network.
- Kibana:
  - Image: Uses the Kibana 8.3.3 image.
  - Container Name: Named kibana.
  - Ports: Exposes port 5601 for Kibana.
  - Environment Variables: Connects Kibana to Elasticsearch.
  - Depends On: Ensures Elasticsearch is available before starting.
  - Networks: Connects to the networkmysql network.

- Logstash:
  - Image: Uses the Logstash 8.3.3 image.
  - Container Name: Named logstash.
  - Volumes: Mounts local configuration and pipeline files.
  - Ports: Exposes ports 5001 (TCP/UDP) and 9600 for Logstash.
  - Environment Variables: Configures memory settings for Logstash.
  - Networks: Connects to the networkmysql network.
  - Depends On: Ensures Elasticsearch is available before starting.

## Networks and Volumes

- networks:
  - networkmysql: Defines a custom bridge network for inter-service communication.
- volumes:
  - mysql\_data: Defines a named volume for persisting MySQL data.

## 5. Unit Testing

Unit testing ensures that individual components of the application function correctly. This section describes the unit testing setup for both the frontend (React) and backend (Spring Boot) components of the travel planning application.

### Frontend Unit Testing

The frontend unit tests focus on the `Login` component using `React Testing Library` and `Jest`. The tests verify that the component renders correctly, responds to user interactions, and handles login errors appropriately.

## Key Tests:

1. Render Login Component: Ensures that the login component and its initial elements are rendered correctly.

```

22      it('renders the Login component with initial elements', () => {
23        render(
24          <BrowserRouter>
25            <Login />
26          </BrowserRouter>
27        );
28      );
29
30      // Check if the buttons for roles are rendered
31      expect(screen.getByText('Customer')).toBeInTheDocument();
32      expect(screen.getByText('Agency')).toBeInTheDocument();
33      expect(screen.getByText('Agent')).toBeInTheDocument();
34
35      // Check if the input fields are rendered
36      // expect(screen.getByPlaceholderText('Username')).toBeInTheDocument();
37      // expect(screen.getByPlaceholderText('Password')).toBeInTheDocument();
38
39      // Check if the Sign in button is rendered
40      expect(screen.getByText('Sign in')).toBeInTheDocument();
41    );
42  );

```

2. Role Button Color Change: Tests the role selection button color change upon clicking

```

43      test('changes role button color on click', () => {
44        render(
45          <BrowserRouter>
46            <Login />
47          </BrowserRouter>
48        );
49
50        const customerButton = screen.getByText('Customer');
51        const agencyButton = screen.getByText('Agency');
52        const agentButton = screen.getByText('Agent');
53
54        // Click on the 'Agency' button
55        fireEvent.click(agencyButton);
56
57        // Check if the 'Agency' button has the active background color
58        expect(agencyButton).toHaveStyle('background-color: #4FA786');
59
60        // Check if the other buttons have the default background color
61        expect(customerButton).toHaveStyle('background-color: #efefef');
62        expect(agentButton).toHaveStyle('background-color: #efefef');
63      );
64

```

3. Login Failure Handling: Mocks a failed login attempt and verifies that the error message is displayed.

```
05
66     test('shows error message on login failure', async () => [
67       axios.post.mockRejectedValueOnce(new Error('Login failed'));
68
69       render(
70         <BrowserRouter>
71           <Login />
72         </BrowserRouter>
73       );
74
75       // Select role
76       fireEvent.click(screen.getByText('Agency'));
77
78       // Enter email and password
79       fireEvent.change(screen.getByTestId('signin-username'), { target: { name: 'email', value: 'sd@xyz.com' } });
80       fireEvent.change(screen.getByTestId('signin-password'), { target: { name: 'password', value: '1234' } });
81
82       // Submit form
83       fireEvent.click(screen.getByTestId('signin-btn'));
84
85       await waitFor(() => {
86         expect(axios.post).toHaveBeenCalledWith(
87           'http://localhost:9094/agency/login',
88           { email: 'sd@xyz.com', password: '1234' },
89           expect.any(Object)
90         );
91       });
92
93       // Check if the error message is displayed
94       await waitFor(() => {
95         expect(screen.getByText('Error while Signing Up')).toBeInTheDocument();
96       });
97     ]);
98   
```

## Backend Unit Testing

The backend unit tests focus on the UserController using JUnit and Spring Boot Test. These tests check the functionality of user registration, authentication, and other user-related operations.

## Key Tests:

1. User Registration: Tests user registration by sending a POST request with user details and verifying the response status.

```
42  
43     @Test // abhisheksharma402  
44     @WithMockUser  
45     void register() throws Exception{  
46         RegistrationRequest registrationRequest = new RegistrationRequest();  
47         registrationRequest.setEmail("test@example.com");  
48         registrationRequest.setFirstName("test");  
49         registrationRequest.setLastName("example");  
50         registrationRequest.setPassword("1234");  
51  
52         UserEntity userEntity = new UserEntity();  
53         userEntity.setEmail("test@example.com");  
54         userEntity.setFirstName("test");  
55         userEntity.setLastName("example");  
56  
57         when(userService.register(registrationRequest)).thenReturn(userEntity);  
58  
59         mockMvc.perform(post(urlTemplate: "/user/register")  
60             .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token  
61             .contentType(MediaType.APPLICATION_JSON)  
62             .content("{\"firstName\": \"test\", \"lastName\": \"example\", \"password\": \"1234\", \"email\": \"test@example.com\"}")  
63             .andExpect(status().isAccepted());  
64     }  
65 }
```

2. User Registration: Tests user registration by sending a POST request with user details and verifying the response status.

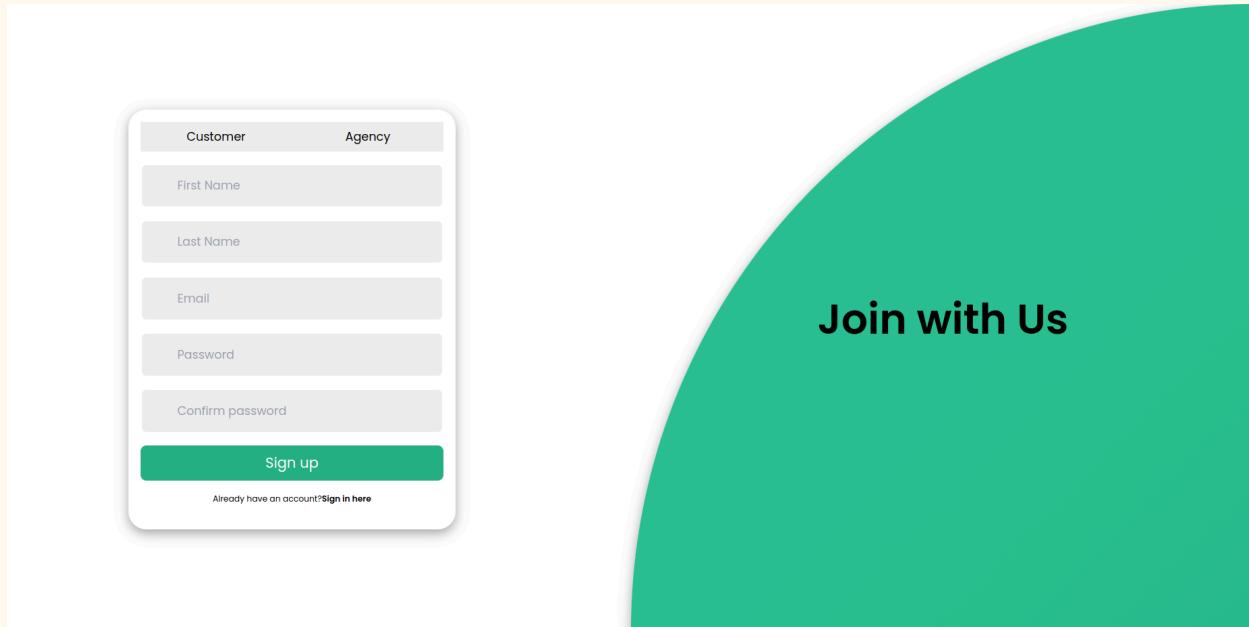
```
66  
67     @Test // abhisheksharma402  
68     @WithMockUser  
69     void authenticate() throws Exception {  
70  
71         AuthenticationRequest authenticationRequest = new AuthenticationRequest();  
72         AuthenticationResponse authenticationResponse = new AuthenticationResponse();  
73         when(userService.authenticate(any(AuthenticationRequest.class))).thenReturn(authenticationResponse);  
74  
75         mockMvc.perform(post(urlTemplate: "/user/login")  
76             .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token  
77             .contentType(MediaType.APPLICATION_JSON)  
78             .content("{\"username\": \"test@example.com\", \"password\": \"password\"}")  
79             .andExpect(status().isOk());  
80     }  
81 }
```

### 3. User Registration: Tests user registration by sending a POST request with user details and verifying the response status.

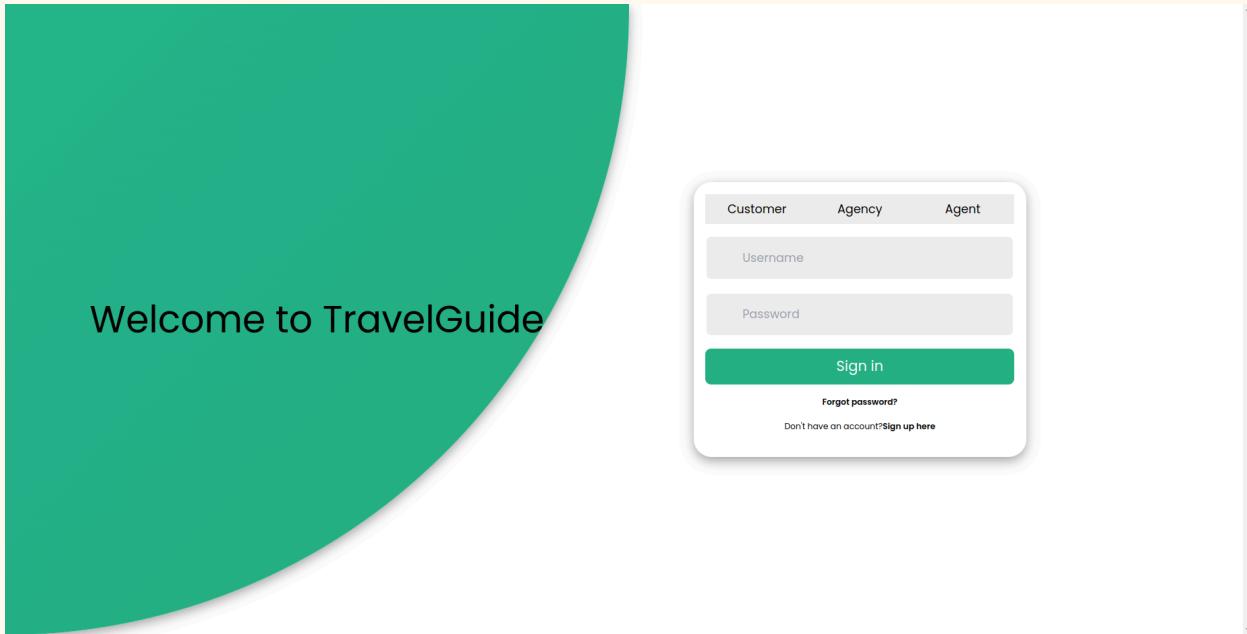
```
94     @Test // abhisheksharma402
95     @WithMockUser
96     void getPackages() throws Exception {
97         PackageEntity packageEntity = new PackageEntity();
98         packageEntity.setId(30);
99         packageEntity.setName("Test Package");
100        packageEntity.setDestination("port blair");
101
102        when(packageRepository.findByDestination(any(String.class))).thenReturn(List.of(packageEntity));
103
104        mockMvc.perform(get( UriTemplate: "/user/packages"
105                            .with(SecurityMockMvcRequestPostProcessors.csrf())
106                            .param( name: "destination", ...values: "port blair")
107                            .andExpect(status().isOk())
108                            .andExpect(jsonPath( expression: "$[0].name").value( expectedValue: "Test Package")));
109    }
110 }
```

## Application Screenshots

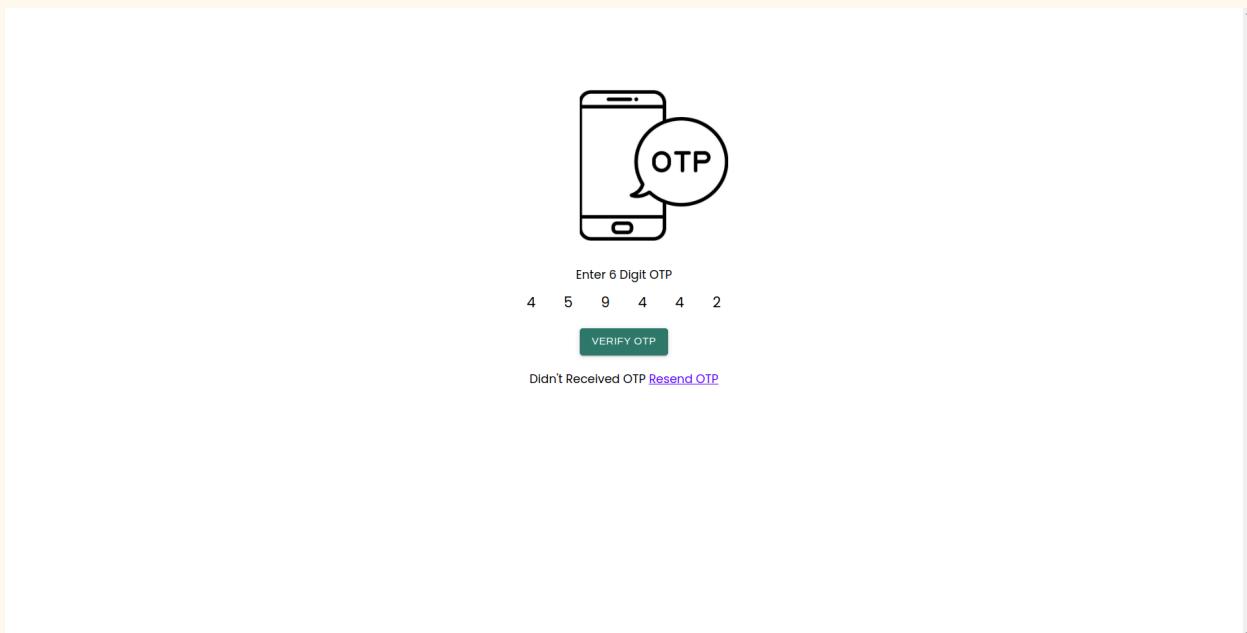
Register Component:



## Login Component



## Activate Account Component



## Customer Dashboard

The screenshot shows a customer dashboard with a sidebar on the left containing icons for Home, Trips, and Profile. The main area is titled "Your Travel History" and displays a list of five trips:

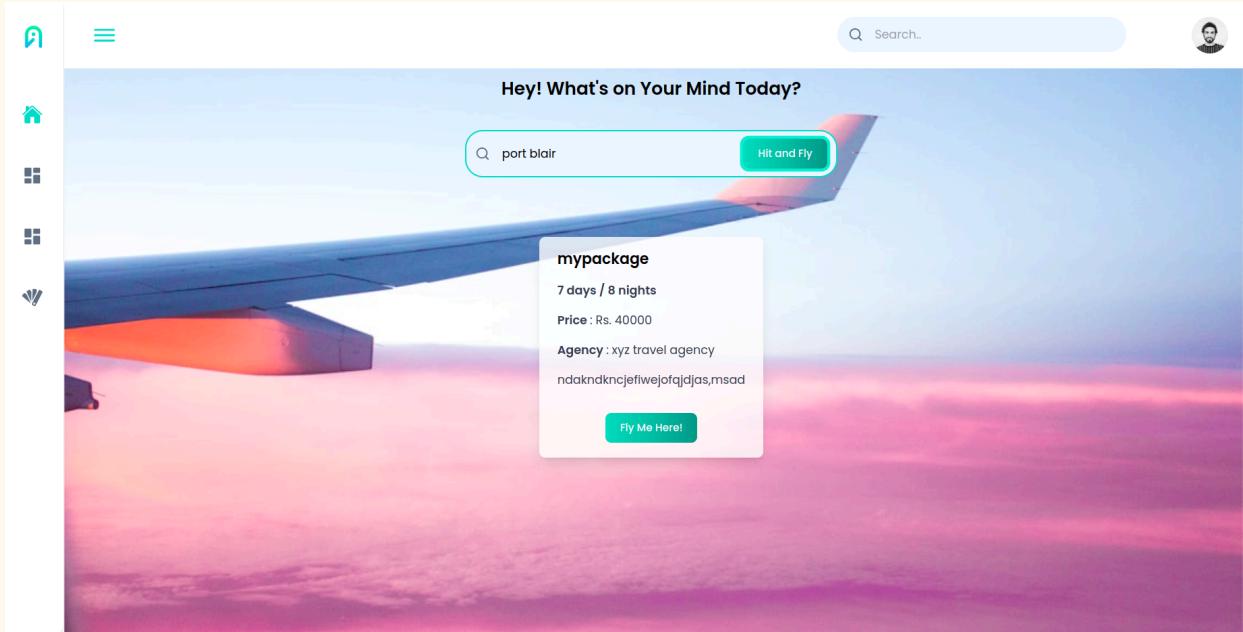
Location	Nights	Check-in Date	Check-out Date	Adults	Total Cost
The great wall of China China	5 Night	10Jan, 2023	16Jan, 2023	2 Adult	\$124
Taj Mahal India	4 Night	8Dec, 2023	21Dec, 2023	2 Adult	\$140
Niagara Falls Canada	12 Night	11 Dec, 2023	11 Dec, 2023	5 Adult	\$560
Great Barrier Reef Italy	3 Night	22 Dec, 2023	25 Dec, 2023	2 Adult	\$200
Piramid Greek	4 Night	24 Dec, 2023	28 Dec, 2023	3 Adult	\$900

At the bottom, there is a pagination bar showing "1-10 Of 10 Entries" and a "Next" button.

## Plan a Trip Component

The screenshot shows a "Plan a Trip" component with a sidebar on the left containing icons for Home, Trips, and Profile. The main area features a large image of an airplane wing against a sunset sky with the text "Hey! What's on Your Mind Today?". Below the image is a search bar with the placeholder "Where do you want to fly to?" and a green "Hit and Fly" button.

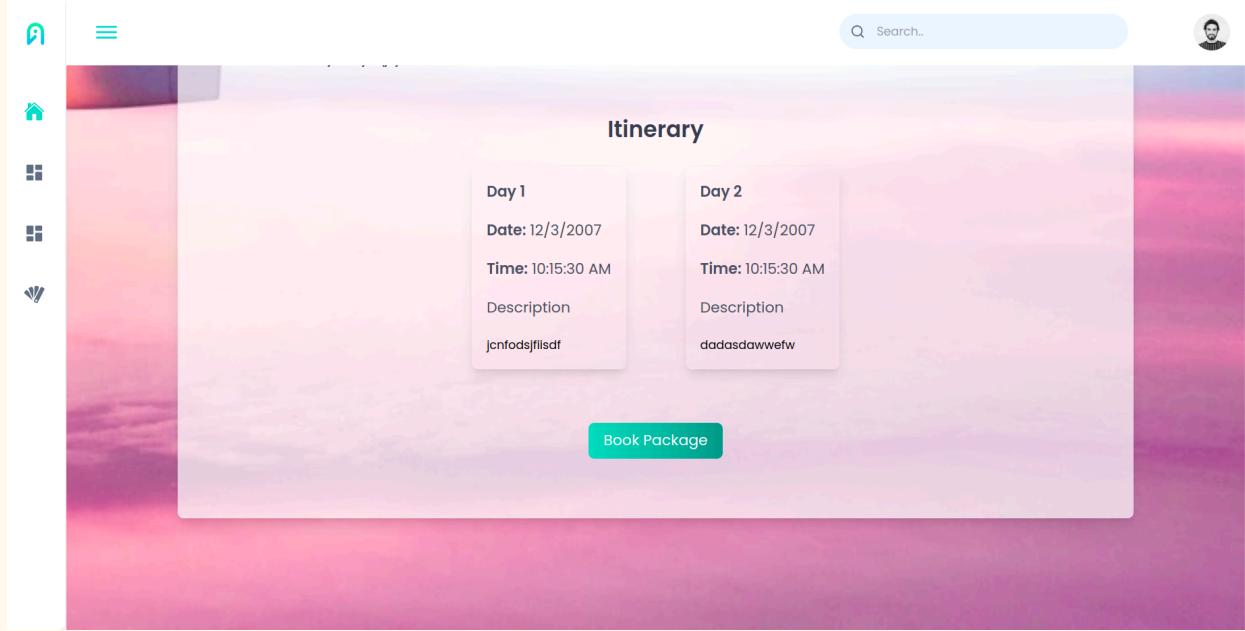
Displaying all the packages with the destination search query



## Individual Package Details

The screenshot shows the same mobile application interface as the previous one, but now focusing on a specific package. The header "Package Details" is visible. The package is identified as "mypackage". The "Description" section contains the placeholder text "ndakndkncjefiwejofqjdjas,msad".

my package	Agency Details	Agent Details
Destination: port blair	Name: xyz travel agency	Name: abhishek sharma
Number of Days: 7	Email: support@xyz.com	Email: abhishek@xyz.com
Number of Nights: 8	Phone Number: 1234567890	Phone Number: 0987654321
Price: Rs. 40000	Address: ncmznhf1	
Places to Visit:	License Number: BJH3121JEKDAASD33	
<ul style="list-style-type: none"> <li>• havlock island</li> <li>• ross island</li> <li>• cellular jail</li> </ul>		
Description		



## Project Demo & GitHub Links

GitHub Link:

- Link: [click here](#)

DockerHub Registry Link:

- Frontend Image Repository Link: [click here](#)
- Backend Image Repository Link: [click here](#)