

# REAL ESTATE APPLICATION

## DevOps Project



International Institute of Information Technology, Bangalore  
Under the guidance of Dr Thangaraju and Jacon Mathew.

Report by:

1. Akshat Pandey(MT2023116)
2. Ziya ur raheman(MT2023026)

# Contents

1. Abstract
2. Technology Stack
3. Introduction to the DevOps tools used
4. System Configuration
5. Software Development Lifecycle
6. Architecture and Basic information about tools used
  - a. Installation
  - b. CI/CD pipeline
  - c. Building image
  - d. Pushing image to Dockerhub
  - e. Deployment using Ansible
  - f. Pipeline script
7. Real estate Application
  - a. API calls
  - b. User Interface
8. Conclusions
9. References and Acknowledgements

## **Abstract**

Using a Real estate full stack web application in concern to be effective and ease down searching a real estate where everyone can add posts of their houses for buying or renting them.

The main goal of this project is to implement a typical Devops pipeline which enhances the frequent deliverables and then updations by user feedback and responsiveness. User effectiveness will be captured from front-end React Js Framework. Backend Logic will be securely handled by node (Express.js) in and Database management by the famous MangoDB.

# Technology Stack

- Technology Stack Used
  - Front End: React JS
  - Back End: node, express
  - Database: mangoDB
  - DevOps Tools: Git, Jenkins, Docker, Ansible

To incorporate the DevOps methodology, multiple DevOps tools have been used and successfully created an automated continuous integration, and continuous deployment of our application.

# DevOps Feature

## 1. **Continuous Integration and Continuous Deployment (CI/CD):**

- a. **Continuous Integration (CI)** involves the practice of frequently merging code changes into a shared repository. Automated builds and tests are run to ensure that new code integrates smoothly with the existing codebase, catching bugs early in the development process.
- b. **Continuous Deployment (CD)** extends CI by automatically deploying code changes to production or staging environments after passing automated tests. This practice ensures that new features, improvements, and bug fixes reach users quickly and reliably.

## 2. **Infrastructure as Code (IaC):** IaC allows the management and provisioning of computing infrastructure through machine-readable definition files, rather than through physical hardware configuration or interactive configuration tools. Popular IaC tools include Terraform, Ansible, and CloudFormation. This approach ensures that environments are consistent, easily reproducible, and scalable.

## 3. **Automated Testing:**

Automated testing is a cornerstone of DevOps, providing quick feedback on the quality and performance of the code. It includes unit tests, integration tests, and end-to-end tests, which are executed automatically during the CI/CD pipeline. Automated tests help in identifying issues early, reducing the risk of bugs reaching production.

## 4. **Code quality:**

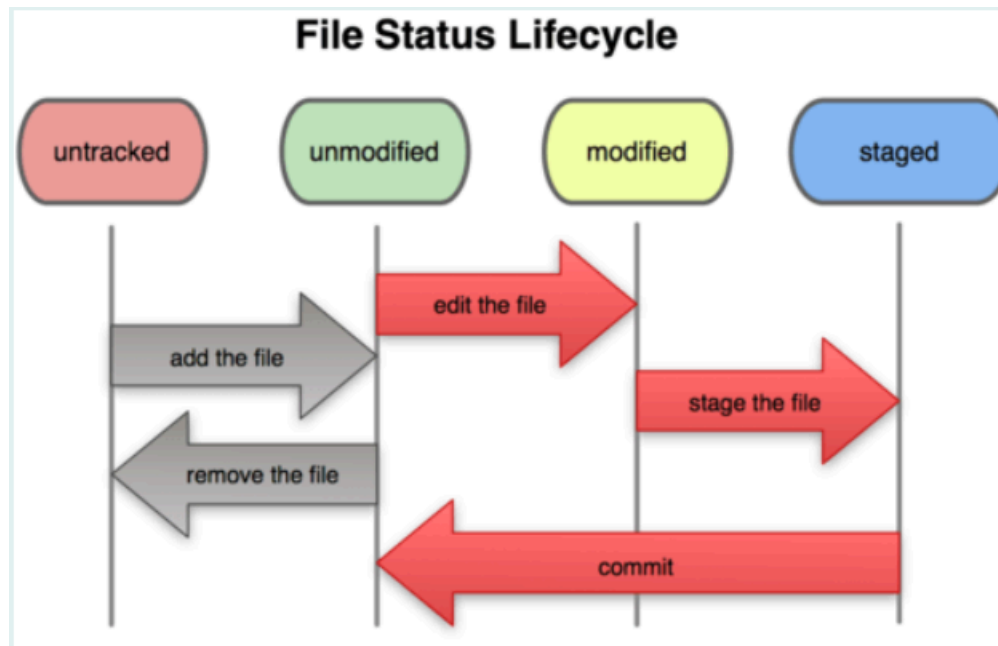
SonarQube is a comprehensive tool for maintaining and improving code quality and security. By integrating it into your development workflow, you can ensure that your codebase remains clean, efficient, and secure, ultimately leading to better software quality and developer productivity

## **5. Collaboration and Communication:**

- a. DevOps fosters a culture of collaboration and communication between development, operations, and other stakeholders. Tools like Slack, Jira, and Confluence facilitate effective communication and collaboration. Practices such as Agile methodologies and Scrum are often integrated with DevOps to enhance project management and streamline workflows, ensuring that teams work together seamlessly towards common goals.

# Introduction to DevOps tools

## Git and Github



## Important Features of Git

### Branching and Merging:

- Git's powerful branching model allows developers to create, manage, and merge branches easily. This enables multiple developers to work on different features, bug fixes, or experiments simultaneously without interfering with the main codebase.
- Feature branches can be merged into the main branch after they have been reviewed and tested, ensuring that the main branch is always stable.

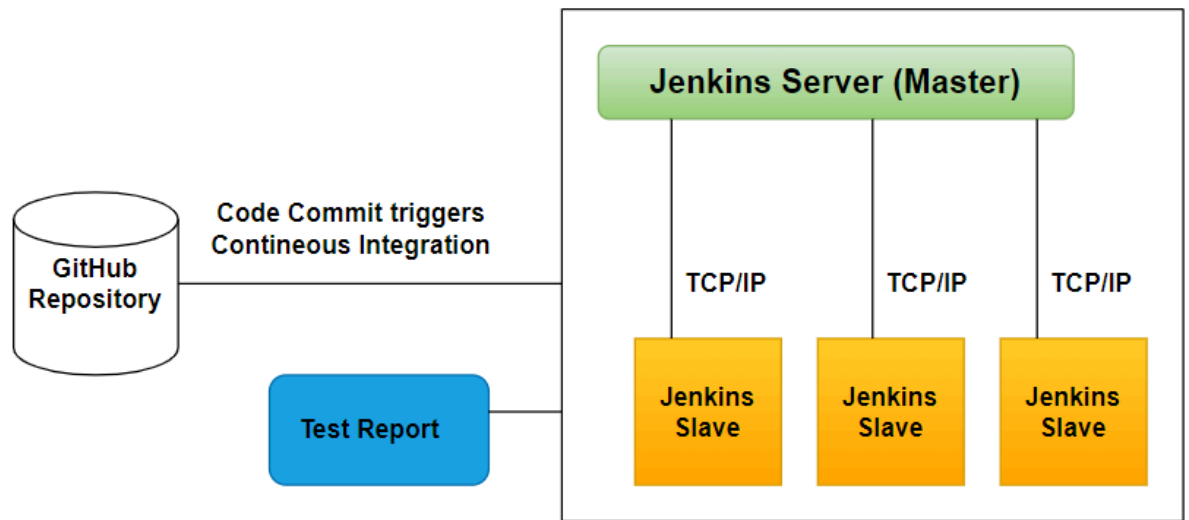
### Version Control:

- Git provides a comprehensive history of all changes made to the codebase, allowing teams to track and revert changes if necessary.

This is crucial for maintaining a stable codebase and understanding the evolution of the project.

- Every commit in Git is uniquely identified.

## Jenkins:



## Pipeline as Code:

- **Declarative and Scripted Pipelines:** Jenkins supports defining build pipelines through code, allowing complex build workflows to be versioned and maintained as part of the project's source code. Using the Jenkinsfile, you can define a pipeline's stages, steps, and conditions in either a declarative or scripted syntax, enabling clear, reusable, and manageable CI/CD processes.

## Extensible with Plugins:

- **Rich Plugin Ecosystem:** Jenkins has a vast library of plugins that extend its functionality to integrate with various tools and technologies. These plugins support a wide range of activities, such as version control (e.g., Git), build tools (e.g., Maven, Gradle), testing frameworks, deployment tools, and cloud platforms. This extensibility

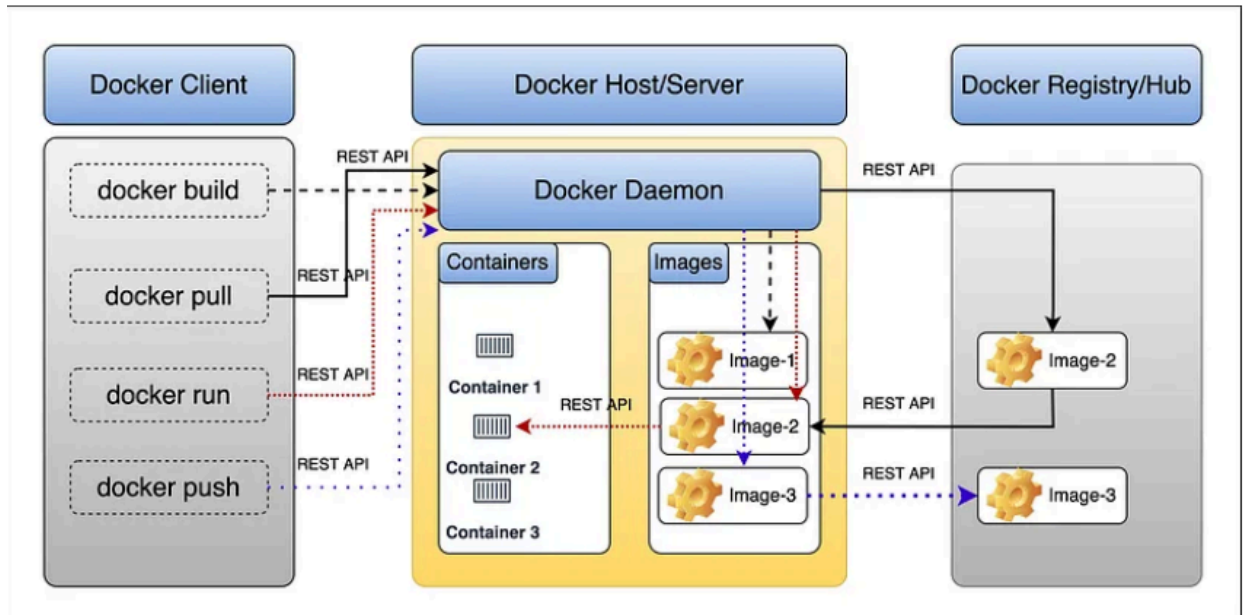


allows Jenkins to adapt to almost any development and deployment scenario.

### **Distributed Builds:**

- **Master-Slave Architecture:** Jenkins supports distributed builds, enabling the workload to be spread across multiple machines or nodes. The master node manages the build pipeline, while slave nodes (or agents) execute the build tasks. This setup improves performance, enables parallel execution, and makes better use of available resources, allowing for faster and more efficient CI/CD processes.

# Docker



## Containerization:

- ❖ **Lightweight and Efficient:** Docker containers package applications along with their dependencies, libraries, and configuration files into a single, lightweight, and portable unit. This ensures that the application runs consistently across different environments.
- ❖ **Isolation:** Each Docker container operates in isolation from others, providing separate environments for each application. This prevents conflicts and improves security by isolating processes and resources.

## Portability:

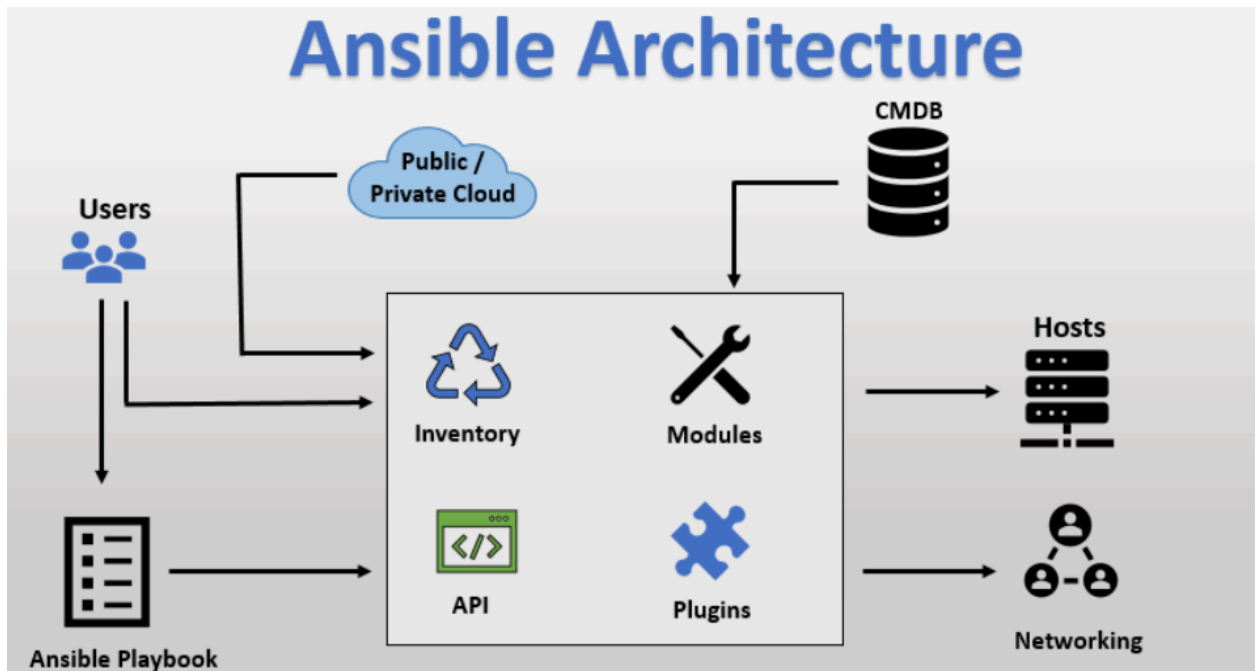
- ❖ **Cross-Platform Compatibility:** Docker containers can run on any system that supports Docker, whether it's a developer's local machine, on-premise data center, or any cloud provider. This portability ensures that applications run consistently across

different environments, simplifying deployment and reducing issues related to environment differences.

### **Version Control and Image Management:**

- ❖ **Docker Images:** Docker uses images as the basis for containers. These images can be versioned, stored, and shared through repositories like Docker Hub. Version control allows for tracking changes and reverting to previous versions if necessary.
- ❖ **Layered Architecture:** Docker images are built in layers, with each layer representing a step in the image build process (e.g., installing a package). This layered architecture makes images efficient and allows for reusable components, reducing build times and storage requirements.

# Ansible



- ❖ **Agentless Architecture:** Ansible operates in an agentless manner, meaning it doesn't require any agents or daemons to be installed and running on managed hosts. Instead, Ansible communicates with remote systems via SSH (Secure Shell) and utilizes existing SSH connections to execute tasks remotely. This agentless architecture simplifies setup and management, reduces overhead, and ensures that there's no need to maintain software on the managed hosts.
- ❖ **Idempotent Execution:** Ansible follows the principle of idempotent execution, meaning that applying the same configuration repeatedly yields the same result, regardless of the system's current state. This ensures that Ansible playbooks can be run multiple times without causing unintended changes or disruptions. Idempotent execution promotes predictability, reliability, and safety in configuration management and infrastructure automation tasks.

## ❖ Infrastructure as Code (IaC) and Configuration

**Management:** Ansible enables Infrastructure as Code (IaC) by allowing infrastructure configurations to be expressed as code using simple, human-readable YAML syntax. With Ansible playbooks, administrators can define desired states for systems and applications, specifying configurations, packages, services, and more. Ansible then automates the provisioning, configuration, and management of infrastructure based on these playbooks, promoting consistency, repeatability, and scalability.

### **System Configuration:**

1. Operating System: ubuntu 20.04 (LOCAL)
2. CPU & RAM: 8-core Processor, 8GB RAM
3. IDE's / Platforms: VS Code, Postman
4. Language: React+vite- npm runtime environment, JS
5. Database: MangoDB

### **Devops Tools:**

1. Source Control Management - GitHub
2. Continuous Integration - Jenkins
3. Containerization - Docker
4. Continuous Deployment - Ansible
5. Monitoring - ELK stack

# Basic information about tools used

## Installation

To install Node.js, execute the following command: `sudo apt install nodejs`

Node.js provides a package manager called npm for managing project dependencies. To install npm, use the following command: `sudo apt install npm`

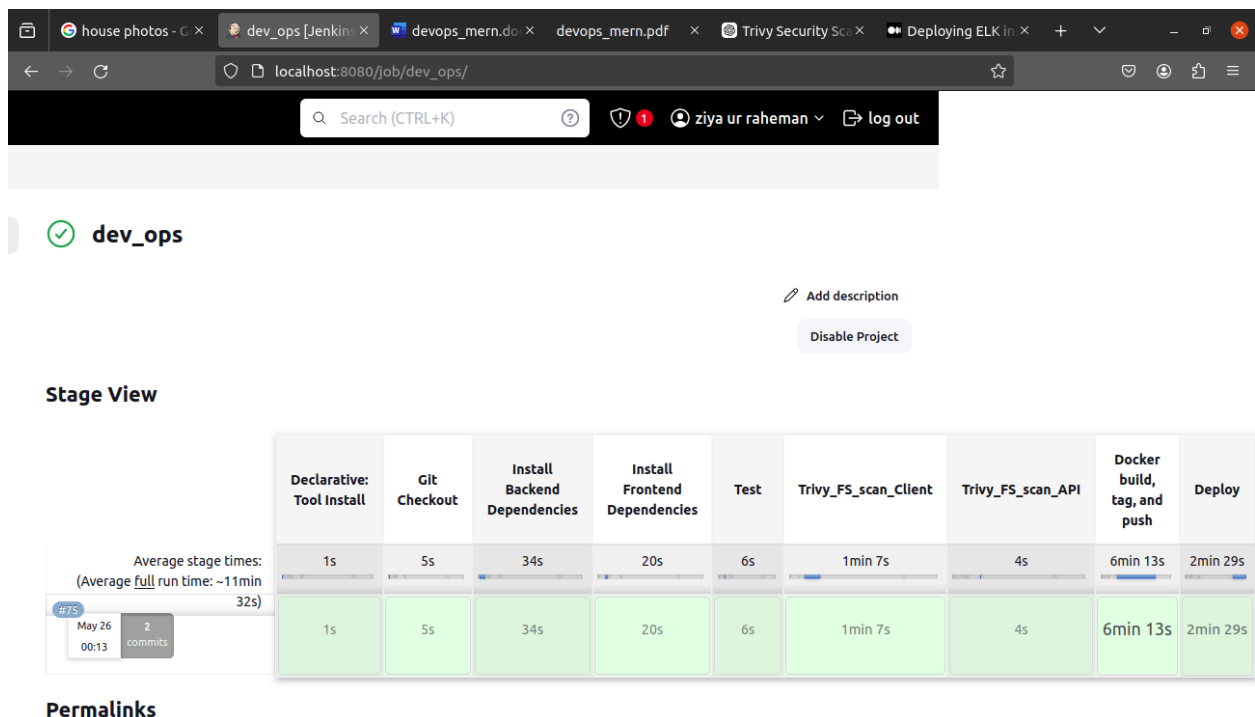
Vue.js is a JavaScript library used for developing user interfaces, especially for single-page applications. To install use the following command: `sudo npm install`

This will create a folder named **client** in your present working directory. Navigate to the folder: **`cd client`**

Start the VITE.js app using: **`npm run dev`**

# CI/CD Pipeline

For Continuous Integration we used Jenkins, which is free and an open-source automation server. Once we are finished with the code development, building the code on local repository, pushing the changes to remote repository. We can implement continuous Integration using Jenkins. In Jenkins, we create a new pipeline job through which we can run all the automation steps. To create a new pipeline, click on “New Item”. Once we click on “New Item”, it redirects to a page where we need to provide the Item name and the type of Item. Since we are attempting to create a pipeline job, we shall select the pipeline option. After creating the Jenkins pipeline job, we need to provide a description reasoning the pipeline work and a pipeline script based on which the automation job runs. Below is a screenshot of the pipeline script. Once the script is ready, we can run the job and test whether our repository is being cloned.



# Console Output after executing pipeline

```
← → ↻ localhost:8080/job/dev_ops/74/consoleText
Started by user ziya ur raheman
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/dev_ops
[Pipeline] {
[Pipeline] tool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Git Checkout)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/dev_ops/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/ziyak8856/full-stack-estate-main.git # timeout=10
Fetching upstream changes from https://github.com/ziyak8856/full-stack-estate-main.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
> git fetch --tags --force --progress -- https://github.com/ziyak8856/full-stack-estate-main.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 1fedca597ab6405f7e2dce363166e9ce3671cb09 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 1fedca597ab6405f7e2dce363166e9ce3671cb09 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master 1fedca597ab6405f7e2dce363166e9ce3671cb09 # timeout=10
Commit message: "elk"
> git rev-list --no-walk b400416e7addcc191952af0d78f4be1f66eec2de # timeout=10
[Pipeline] }
```



# Building Image using Docker

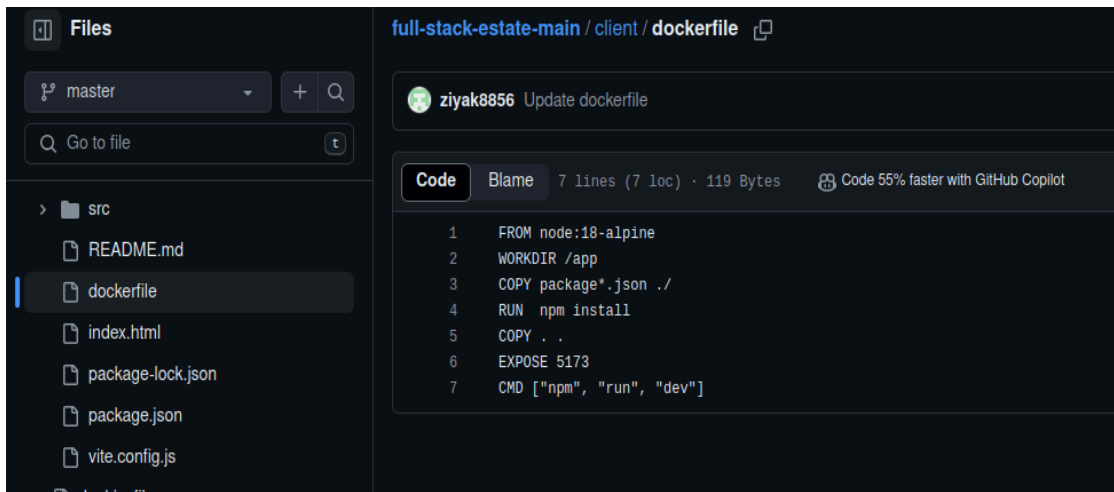
Docker is a tool that can package an application and its dependencies in a virtual container that can run on any linux server. This will enable flexibility and portability on where the application can run either on premise, public/private cloud, base metal etc. Docker is a platform which provides OS 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 automatically 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:

1. `sudo apt install docker.io`
2. `sudo systemctl start docker`
3. `sudo systemctl status docker`

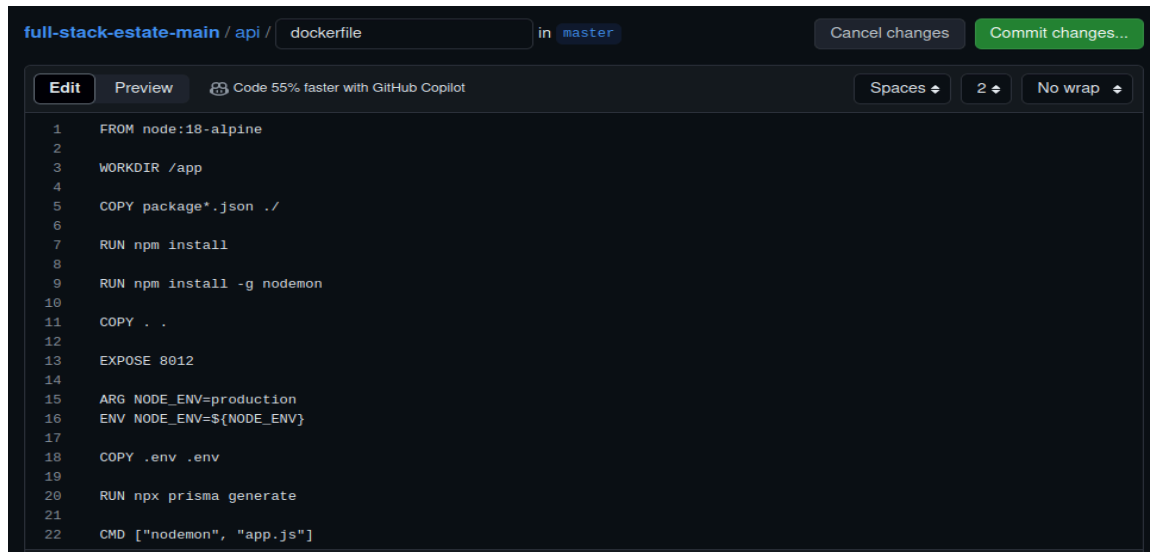
Frontend Docker file:



The screenshot shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like README.md, dockerfile, index.html, package-lock.json, package.json, vite.config.js, and .dockerignore. The code editor displays the content of the dockerfile, which is a 7-line Dockerfile for a Node.js application. The code is as follows:

```
1 FROM node:18-alpine
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 5173
7 CMD ["npm", "run", "dev"]
```

Backend Docker file:

A screenshot of a code editor showing a Dockerfile. The editor has a dark theme. At the top, there's a breadcrumb navigation: 'full-stack-estate-main / api / dockerfile' and a status 'in master'. To the right are buttons for 'Cancel changes' and 'Commit changes...'. Below the breadcrumb is a toolbar with 'Edit', 'Preview', and a GitHub Copilot icon with the text 'Code 55% faster with GitHub Copilot'. Further right are 'Spaces' (set to 2), and 'No wrap'. The main area contains a Dockerfile with 22 lines of code, numbered on the left. The code starts with 'FROM node:18-alpine' and ends with 'CMD ["nodemon", "app.js"]'.

```
1 FROM node:18-alpine
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 RUN npm install -g nodemon
10
11 COPY . .
12
13 EXPOSE 8012
14
15 ARG NODE_ENV=production
16 ENV NODE_ENV=${NODE_ENV}
17
18 COPY .env .env
19
20 RUN npx prisma generate
21
22 CMD ["nodemon", "app.js"]
```

**Docker Build Image:** `docker build -t imagename .` (Assuming current directory has dockerfile)

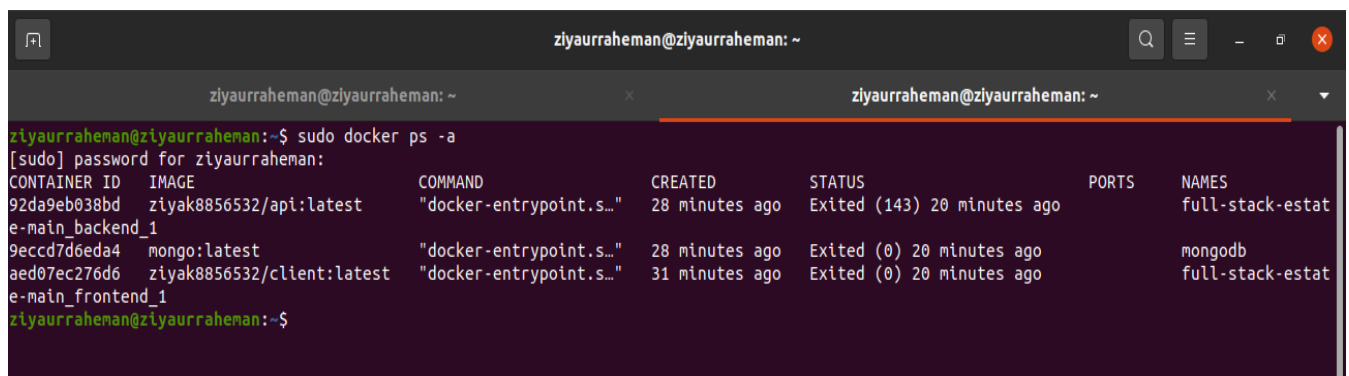
After the building stage, we create stages for building docker images. To build the backend docker image we create a stage in which we mention the directory from where we can access the dockerfile. `docker.build` is used to create the image.

Push Docker Image to Dockerhub Once the docker images are built we can push them to the docker hub.

# Deployment using Ansible

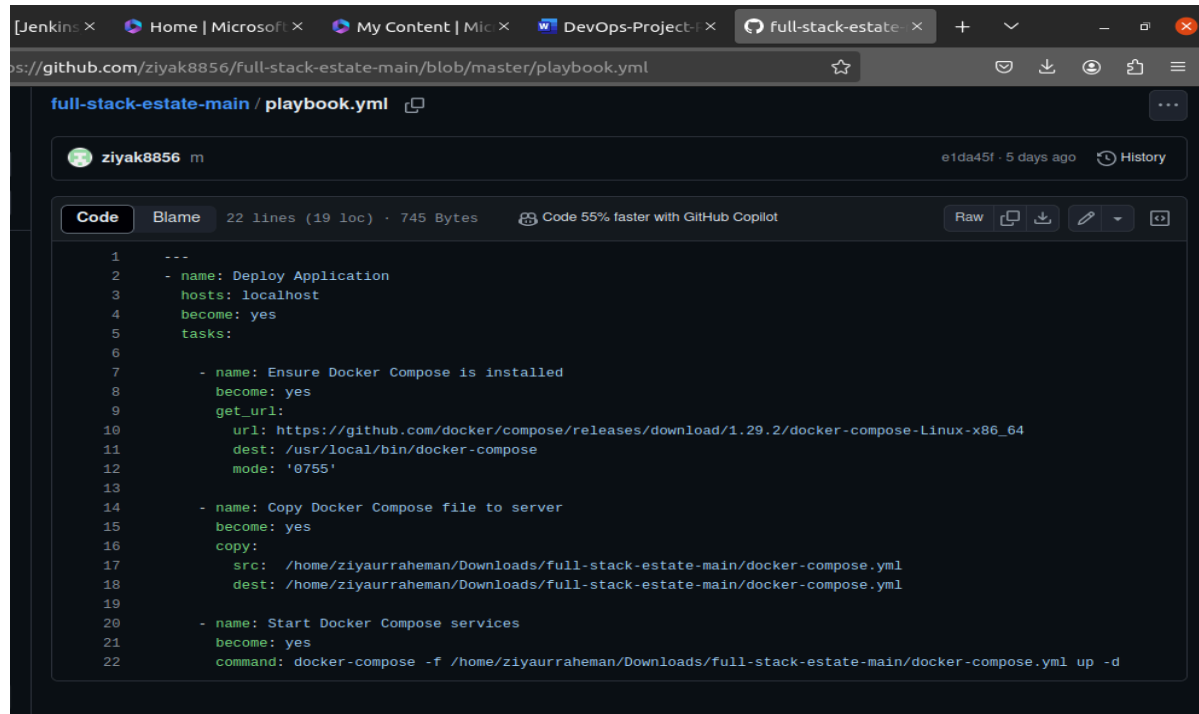
## Key Steps in Deployment using Ansible:

- **Inventory Setup:** Define an inventory file listing all the target machines where you want to deploy your application.
- **Playbook Creation:** Write YAML playbooks defining the tasks required for deployment. These tasks typically include tasks like installing dependencies, copying files, configuring services, and starting applications.
- **Task Execution:** Run the playbook using the **ansible-playbook** command, specifying the playbook file and the target inventory. Ansible connects to the target machines via SSH (or other supported connection methods) and executes the tasks defined in the playbook.
- **Idempotent Operations:** Ansible ensures idempotency, meaning running the playbook multiple times will result in the same desired state, regardless of the current state of the system. This reduces the risk of unintended changes during deployment.
- **Error Handling and Reporting:** Ansible provides detailed output during playbook execution, indicating which tasks succeeded and which failed. This allows for easy troubleshooting and error handling.

A terminal window with a dark background and light green text. The prompt is 'ziyaurraheman@ziyaurraheman: ~'. The user has entered 'sudo docker ps -a'. The output shows a table of Docker containers. The first container is 'full-stack-estat' with ID '92da9eb038bd', image 'ziyak8856532/api:latest', and command '"docker-entrypoint.s..."'. It was created 28 minutes ago and exited with status (143) 20 minutes ago. The second container is 'mongodb' with ID '9eccd7d6eda4', image 'mongo:latest', and command '"docker-entrypoint.s..."'. It was created 28 minutes ago and exited with status (0) 20 minutes ago. The third container is 'full-stack-estat' with ID 'aed07ec276d6', image 'ziyak8856532/client:latest', and command '"docker-entrypoint.s..."'. It was created 31 minutes ago and exited with status (0) 20 minutes ago. The prompt is now 'ziyaurraheman@ziyaurraheman:~\$'.

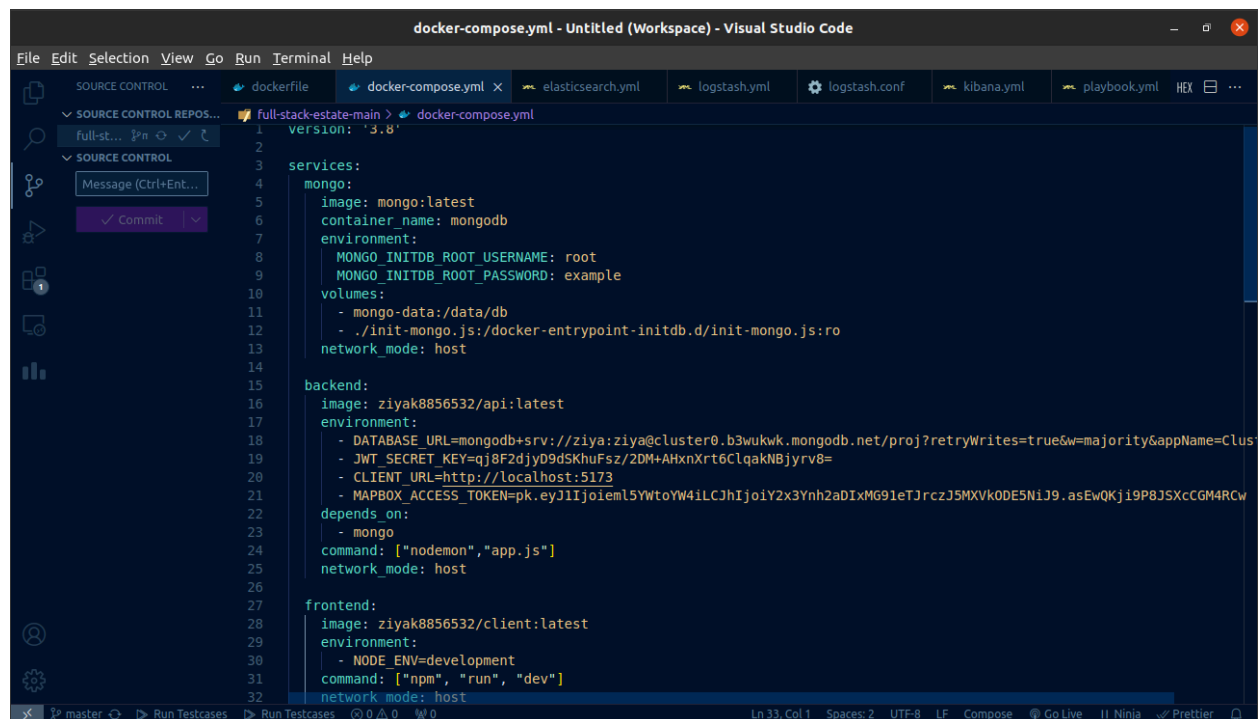
```
ziyaurraheman@ziyaurraheman: ~  
ziyaurraheman@ziyaurraheman:~$ sudo docker ps -a  
[sudo] password for ziyaurraheman:  
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS          NAMES  
92da9eb038bd   ziyak8856532/api:latest             "docker-entrypoint.s..." 28 minutes ago Exited (143) 20 minutes ago          full-stack-estat  
e-main_backend_1  
9eccd7d6eda4   mongo:latest                         "docker-entrypoint.s..." 28 minutes ago Exited (0) 20 minutes ago          mongodb  
aed07ec276d6   ziyak8856532/client:latest           "docker-entrypoint.s..." 31 minutes ago Exited (0) 20 minutes ago          full-stack-estat  
e-main_frontend_1  
ziyaurraheman@ziyaurraheman:~$
```

## YAML File:



```
1 ---
2 - name: Deploy Application
3   hosts: localhost
4   become: yes
5   tasks:
6
7     - name: Ensure Docker Compose is installed
8       become: yes
9       get_url:
10        url: https://github.com/docker/compose/releases/download/1.29.2/docker-compose-Linux-x86_64
11        dest: /usr/local/bin/docker-compose
12        mode: '0755'
13
14     - name: Copy Docker Compose file to server
15       become: yes
16       copy:
17        src: /home/ziyaurraheman/Downloads/full-stack-estate-main/docker-compose.yml
18        dest: /home/ziyaurraheman/Downloads/full-stack-estate-main/docker-compose.yml
19
20     - name: Start Docker Compose services
21       become: yes
22       command: docker-compose -f /home/ziyaurraheman/Downloads/full-stack-estate-main/docker-compose.yml up -d
```

## Docker-compose:



```
1 version: '3.8'
2
3 services:
4   mongo:
5     image: mongo:latest
6     container_name: mongoddb
7     environment:
8       MONGO_INITDB_ROOT_USERNAME: root
9       MONGO_INITDB_ROOT_PASSWORD: example
10    volumes:
11      - mongo-data:/data/db
12      - ./init-mongo.js:/docker-entrypoint-initdb.d/init-mongo.js:ro
13    network_mode: host
14
15   backend:
16     image: ziyak8856532/api:latest
17     environment:
18       - DATABASE_URL=mongodb+srv://ziya:ziya@cluster0.b3wukwk.mongodb.net/proj?retryWrites=true&w=majority&appName=Cluster0
19       - JWT_SECRET_KEY=qj8F2dJyD9dSKhuFsz/2DM+AHxnXrt6CLqakNBjyrv8=
20       - CLIENT_URL=http://localhost:5173
21       - MAPBOX_ACCESS_TOKEN=pk.eyJ1Ijoieml5YWtoYW4iLCJhIjoieY2x3Ynh2aD1xMG91eTJrczJ5MXVxK0DE5NiJ9.asEwQKj19P8JSxcCGM4RCw
22     depends_on:
23       - mongo
24     command: ["nodemon", "app.js"]
25     network_mode: host
26
27   frontend:
28     image: ziyak8856532/client:latest
29     environment:
30       - NODE_ENV=development
31     command: ["npm", "run", "dev"]
32     network_mode: host
```

# Pipeline Script

```
stage('Git Checkout') {  
  steps {  
    git 'https://github.com/ziyak8856/full-stack-estate-main.git'  
  }  
}
```

Explanation: "Stage 1: Git Checkout" fetches code from a GitHub repository. It specifically clones the 'main' branch of the repository ['https://github.com/ziyak8856/full-stack-estate-main.git'](https://github.com/ziyak8856/full-stack-estate-main.git). This step is crucial in CI/CD pipelines, initiating the process by retrieving the source code for subsequent actions like building, testing, and deployment

```
stage('Install Backend Dependencies') {  
  steps {  
    dir('api') {  
      script {  
        sh 'npm install'  
        sh 'npm install -g nodemon'  
      }  
    }  
  }  
}  
  
stage('Install Frontend Dependencies') {  
  steps {  
    dir('client') {  
      script {  
        sh 'npm install'  
      }  
    }  
  }  
}
```

Explanation: we installed front end and backend dependency using **npm install** and **npm install nodemone -g**

```
stage('Test') {
    steps {
        dir('api') {
            script {
                // Run the tests (assumes tests are defined in package.json
scripts)
                sh 'npm test'
            }
        }
    }
}
```

Explanation:

Stage 3: we are using test controller for testing in which we are testing jwt token testing

```
stage('Trivy_FS_scan_Client') {
    steps {
        dir('client') {
            sh "trivy fs --format table -o client-fs-report.html ."
        }
    }
}
stage('Trivy_FS_scan_API') {
    steps {
        dir('api') {
            sh "trivy fs --format table -o api-fs-report.html ."
        }
    }
}
```

Explanation:

Purpose: The pipeline has two stages that perform security scans on different parts of a project (client and api) using Trivy.

Trivy: It is a security scanner for containers and other artifacts, capable of detecting vulnerabilities, misconfigurations, and secrets. Output: The scan results are saved as HTML reports (client-fs-report.html and api-fs-report.html) in their respective directories.

```
stage('Sonarqube') {  
    steps {  
        withSonarQubeEnv('sonar') {  
            sh "$SCANNER_H/bin/sonar-scanner -Dsonar.projectKey=projEstate  
-Dsonar.projectName=projEstate"  
        }  
    }  
}
```

Explanation:

Purpose: This pipeline stage is responsible for running a SonarQube analysis on the codebase.

SonarQube: A tool for static code analysis to ensure code quality, detecting bugs, vulnerabilities, and code smells.

Configuration: withSonarQubeEnv('sonar') sets up the environment using the specified SonarQube server configuration (sonar). The 'sh' command runs the SonarQube scanner with parameters to set the project key and name.

```
stage('Docker build, tag, and push') {  
    steps {  
        script {  
            // Docker credentials  
            def DOCKER_USERNAME = 'ziyak8856532'  
            def DOCKER_PASSWORD = 'Jiyak8856'
```

```

        // Authenticate with Docker registry
        // Authenticate with Docker registry
        sh "echo ${DOCKER_PASSWORD} | sudo docker login -u
${DOCKER_USERNAME} --password-stdin https://index.docker.io/v1/"

        sh "docker build -t ${DOCKER_USERNAME}/api:latest ./api"
        sh "docker push ${DOCKER_USERNAME}/api:latest"

        // Build, tag, and push frontend image
        sh "docker build -t ${DOCKER_USERNAME}/client:latest
./client"
        sh "docker push ${DOCKER_USERNAME}/client:latest"
    }
}
}

```

Explanation:

**Purpose:** This stage automates the process of building, tagging, and pushing Docker images for both the backend and frontend components of an application to Docker Hub.

**Steps:**

- Define Docker Hub credentials
- Log in to Docker Hub using the provided credentials
- Build and push the backend (api) image
- Build and push the frontend (client) image

```

stage('Stage 8: Clean Docker Images') {
    steps {
        script {
            sh 'docker container prune -f'
            sh 'docker image prune -f'
        }
    }
}

```



```
}
```

Explanation: The **docker container prune -f** command is utilized to forcibly remove all stopped containers from the Docker host, freeing up disk space and resources. Subsequently, the **docker image prune -f** command is employed to forcefully remove all dangling images, which are images that are not associated with any containers, thus reclaiming additional disk space.

```
stage('Deploy') {  
    steps {  
        script {  
            sh 'ansible-playbook -i inventory.ini playbook.yml'  
        }  
    }  
}
```

Explanation: Ansible Deployment" orchestrates the deployment process using Ansible, a configuration management tool. Within this stage, a script block is executed to trigger the deployment playbook. The command **ansible-playbook -i inventory.yaml docker-compose-ansible.yaml** instructs Ansible to execute the playbook named 'docker-compose-ansible.yaml' using the inventory file 'inventory.yaml'. This playbook likely contains instructions for deploying the application components using Docker Compose or similar orchestration tools specified in the playbook. By leveraging Ansible, the deployment process can be automated and managed consistently across different environments, ensuring efficient and reliable deployment of the application.

# REAL ESTATE APP

## APIs Used

### User Authentication:

#### **register Function:**

Purpose: Handles user registration.

Process:

- Extracts username, email, and password from the request body.
- Hashes the password using bcrypt.
- Creates a new user in the database with the hashed password.
- Responds with a success message if the user is created successfully or an error message if the operation fails.

#### **login Function:**

Process:

- Extracts username and password from the request body.
- Checks if the user exists in the database.
- Validates the password using bcrypt.
- Generates a JWT token with a validity of one week.
- Sends the token as an HTTP-only cookie and returns the user information (excluding the password) if the credentials are valid.
- Responds with an error message if the user does not exist or the password is incorrect

#### **login Function:**

**Process:**

- Clears the JWT token cookie.
- Responds with a success message indicating the user has logged out.

## **User Post (display, delete, add):**

### **getPosts Function:**

Purpose: Retrieves a list of posts filtered by query parameters.

- Extracts query parameters from the request.
- Uses Prisma to find posts that match the query criteria and Returns the filtered posts or an error message if the operation fails

Process: Retrieves a specific post by its ID, including detailed information and user details.

- Extracts the post ID from the request parameters.
- Uses Prisma to find the post and include related postDetail and user information.
- Checks if the post is saved by the user (if the user is authenticated via JWT).
- Returns the post details and whether it is saved or an error message if the operation fails.

### **addpost Function:**

Process: Creates a new post with detailed information

- Extracts the post data and user ID from the request body and token respectively.
- Uses Prisma to create a new post and associated post details in the database.
- Returns the newly created post or an error message if the operation fails.

## **User Controller:**

### **getUsers Function:**

Purpose: Retrieves a list of all users from the database.

Process:

- Uses Prisma to fetch all users.
- Returns the list of users if successful or an error message if the operation fails.

### **getUser Function:**

Purpose: Retrieves a specific user by their ID from the database.

Process:

- Extracts the user ID from the request parameters.
- Uses Prisma to find the user with the given ID.
- Returns the user object if found or an error message if the operation fails.

### **updateuser Function:**

Purpose: Updates user details in the database.

Process:

- Extracts the user ID from the request parameters and the authenticated user ID from the token.
- Checks if the user is authorized to update the profile.
- Hashes the new password if provided.
- Updates user details in the database with the new information.
- Returns the updated user object (without the password) if successful or an error message if the operation fails.

### **savePost Function:**

Purpose: Allows a user to save or remove a post from their saved list.

Process:

- Extracts the post ID and the user ID from the request body and token respectively.
- Checks if the post is already saved by the user.
- If the post is saved, it is removed from the saved list; otherwise, it is saved.
- Returns a success message indicating whether the post was saved or removed, or an error message if the operation fails.

## profilePosts Function:

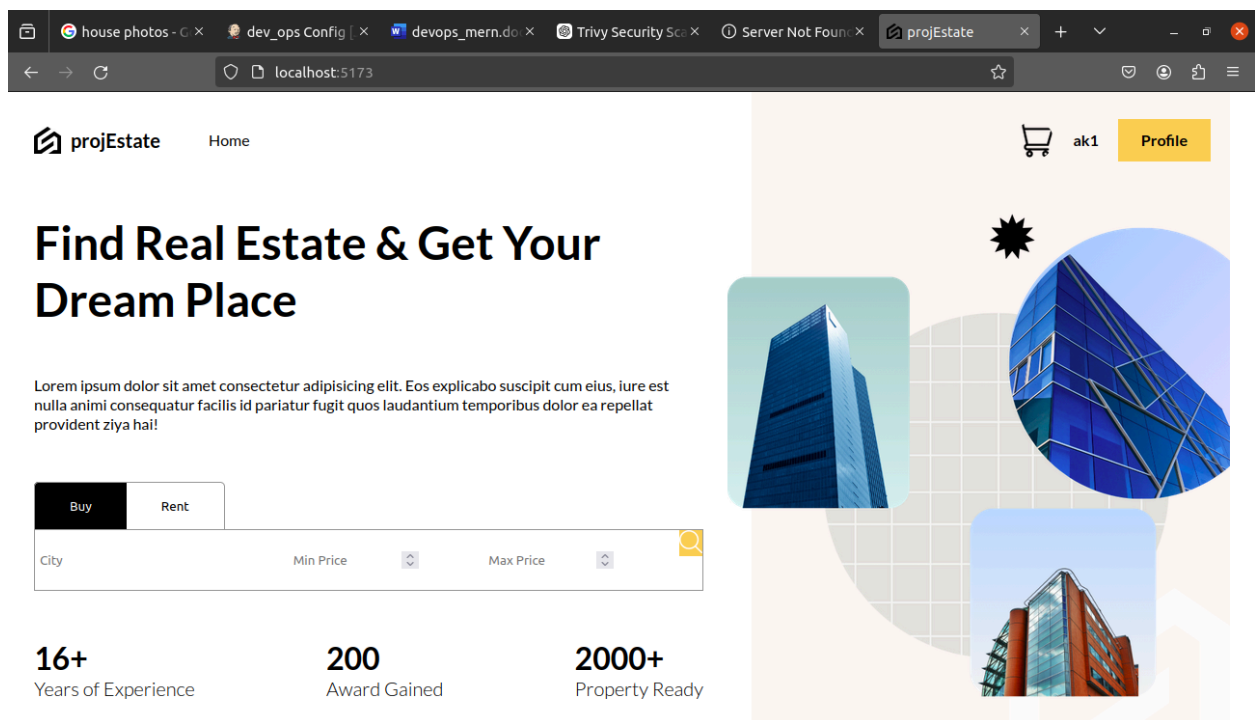
Purpose: Retrieves posts created by the user and posts saved by the user.

Process:

- Extracts the user ID from the token.
- Uses Prisma to retrieve posts created by the user and posts saved by the user.
- Extracts the saved posts from the response.
- Returns the user's posts and saved posts or an error message if the operation fails

## User Interface

Home Page:




## Post Add:

## Profile Update page:

house photo: × dev\_ops Coni × devops\_mern × Trivy Security × Server Not F × projEstate × Data | Cloud: × + -

localhost:5173/profile/update

 projEstate Home

Update Profile

Username

ak1


Email


ak1@gmail.com

Password

\*\*\*\*

Update

 ak1 Profile




Upload

## Profile page:


projEstate Home

Username: ak1  
E-mail: ak1@gmail.com  
Logout



My List





luxury apartment

 near jalgaon road


\$ 25000

 2 bedroom  1 bathroom

Create New Post

Messages

 John Doe X

Lorem ipsum dolor sit amet  
1 hour ago

Lorem ipsum dolor sit amet  
1 hour ago

Lorem ipsum dolor sit amet  
1 hour ago

Lorem ipsum dolor sit amet  
1 hour ago

Send

## **Conclusions**

In this project, we successfully developed a Real Estate app a DevOps approach. The integration of Mango Express Node.js for the backend, React.js for the frontend, MongoDB for the database, Jenkins for continuous integration, and Docker for containerization allowed us to build a robust and scalable system. Through the implementation process, we encountered various challenges, such as ensuring seamless integration between different technologies and frameworks, managing version control and code deployment, and handling database synchronization. However, with careful planning, collaboration, and the utilization of industry-standard tools, we were able to overcome these challenges and deliver a functional system.



The adoption of DevOps practices, including continuous integration and containerization, proved to be highly beneficial. Jenkins facilitated the automation of build, test, and deployment processes, ensuring efficient and reliable software delivery. Docker enabled us to package the application and its dependencies into containers, ensuring consistency across different environments and simplifying deployment.

By leveraging the power of React.js, we achieved a responsive and user-friendly frontend, enabling users to search for properties, view listings, and contact agents conveniently. The backend, developed using Mango Express Node.js, provided robust APIs for property management, user authentication, and interaction with the database. MongoDB served as a reliable and efficient data storage solution, ensuring data integrity and security.

## References

1. "React - A JavaScript library for building user interfaces." React.js Documentation. Available at: <https://reactjs.org/docs/getting-started.htm>
2. "Spring Boot Reference Guide." Spring Boot Documentation. Available at: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
3. "MySQL Documentation." MySQL. Available at: <https://dev.mysql.com/doc/>
4. "Jenkins Documentation." Jenkins. Available at: <https://www.jenkins.io/doc/>
5. "Docker Documentation." Docker. Available at: <https://docs.docker.com/>
6. Ansible Documentation: <https://docs.ansible.com/>
7. ELK STACK:  
<https://medium.com/@lopchannabeen138/deploying-elk-inside-docker-cont>

[ainer-docker-compose-4a88682c7643](#)