



**CS 816 - Software Production Engineering**  
Major Project - CitizenConnect

*Under guidance of*  
**Prof. B. Thangaraju**

REPORT BY -

Pracheti Bhale

MT2023155

Manasi Purkar

MT2023158

<https://github.com/ManasiPurkar/CitizenConnect.git>

Video Link:

[https://drive.google.com/drive/folders/1\\_18Uy0WpGFb8SJZulo4VtUMN7dO2i65R?usp=sharing](https://drive.google.com/drive/folders/1_18Uy0WpGFb8SJZulo4VtUMN7dO2i65R?usp=sharing)

## TABLE OF CONTENTS

<b>TABLE OF CONTENTS.....</b>	<b>2</b>
<b>DEVOPS INTRODUCTION.....</b>	<b>5</b>
What is DevOps?.....	5
Why DevOps?.....	5
What is CI/CD?.....	6
<b>TOOLS AVAILABLE.....</b>	<b>9</b>
<b>CONFIGURATIONS.....</b>	<b>12</b>
FRONTEND.....	12
BACKEND.....	17
<b>MICROSERVICE ARCHITECTURE.....</b>	<b>21</b>
<b>ELK STACK.....</b>	<b>26</b>
<b>TESTING USING MOKITO.....</b>	<b>27</b>
<b>SECURITY USING JWT.....</b>	<b>29</b>
<b>SCREENSHOTS OF ALL REQUIRED FILES AND SERVICES.....</b>	<b>32</b>
<b>STEPS TO BUILD.....</b>	<b>48</b>
<b>SCREENSHOTS OF THE APPLICATION.....</b>	<b>49</b>
<b>LIST OF ALL API's.....</b>	<b>60</b>
<b>CONCLUSION AND FUTURE WORK.....</b>	<b>61</b>

## ABSTRACT

This project aims to develop Citizen Connect, a comprehensive application designed to streamline the registration and management of government complaints submitted by citizens. The current process for reporting government-related issues often suffers from inefficiency and lack of transparency, leading to delays in complaint resolution and citizen dissatisfaction. To address these challenges, Citizen Connect will introduce several key features:

- *User-Friendly Complaint Registration Interface*: An intuitive and accessible interface will enable citizens to easily register complaints, providing essential details such as complaint type, location, and description.
- *Real-Time Status Updates*: Citizen Connect will offer real-time updates on the status of registered complaints, allowing citizens to track the progress of resolution efforts and stay informed about the actions being taken.
- *Centralized Issue Monitoring Dashboard*: A feature designed for a central authority, such as a Nagar-sevak, to monitor and manage complaints effectively. This dashboard will provide a comprehensive view of all complaint and performance metrics for unresolved issues, categorized by geographic areas.

By implementing these features, Citizen Connect aims to enhance the efficiency and transparency of the complaint resolution process, ultimately improving citizen satisfaction and fostering better governance.

### Functionalities -

1. Registration of all citizens.
2. Admin will register a nagarsevak for a specific area.
3. Citizens can register complaints.
4. Citizens can view complaints registered by them and in their area separately.
5. Citizens can comment on any complaint.
6. A nagarsevak can view complaints registered in his/her area.
7. A nagarsevak can comment on a complaint to provide a status update of the work.

8. Nagarsevak can change the status of the complaint. There are three status possibilities namely *pending*, *ongoing* and *resolved*.

Technical Stack:

- Frontend: React Native
- Backend: Java Spring Boot
- Database: MySQL
- Version Control System (VCS): Git
- Containerization: Docker
- Container Orchestration : Docker compose
- CI/CD Tool: Jenkins
- Logging and Monitoring: ELK Stack

## DEVOPS INTRODUCTION

### What is DevOps?

DevOps is a software development methodology or cultural approach that emphasizes collaboration, communication, integration, and automation between software development (Dev) and IT operations (Ops) teams. The primary goal of DevOps is to improve the efficiency, speed, and quality of software development and delivery processes.

- Dev – People involved in developing products.
- Ops – System engineers, administrators, operations staff, release engineers, DBAs, network engineers and Security professionals.
- Agile Software Development – collaboration of customers, product management, developers and QA to fill in the gaps and rapidly iterate towards a better product.
- DevOps – extending Agile principles beyond the boundaries of “the code” to the entire delivered service.
- Goal of DevOps – span the entire delivery pipeline.
  - Improved deployment frequency
  - faster time to market
  - lower failure rate of new release
  - shortened lead time between fixes
  - faster mean time to recovery in the event of a new release crashing.

### Why DevOps?

Goal is to make this journey smooth, quick and without hiccups. As the development team reaches code complete, then deploy the code to the test cluster where the QA team goes through a test pass. After meeting the test pass exit criteria, roll the code into an intermediary staging environment, which is called “Pre-production”. This staging environment matches our production hardware as closely as possible and use it to do final acceptance testing. This model helps ensure the highest quality release when finally deployed to Production.

Each department defines its goals based on the division of work. Development and

operations are two distinct departments. Often, these departments act like silos because they are independent of each other. The development department may be measured by its speed in creating new features, whereas the operations department may be judged by server uptime and application response time.

Development teams struggle for change, whereas operations teams struggle for stability. The conflict between development and operations is caused by a combination of conflicting motivations, processes and tooling, as a result, isolated silos evolve.

In a nutshell, the conflict between development and operations is :

1. Need for change: Development produces changes (e.g: new features, bug fixes and work based on change requests). They want their changes rolled out to production.
2. Fear of change: Once the software is delivered, the operations department wants to avoid making changes to the software to ensure stable conditions for the production systems.

DevOps helps development teams increase the frequency of application updates. Agile adoption traditionally omits operations, obstructing the delivery pace. DevOps removes this obstruction by applying agile values to the deployment, environment configuration, monitoring and maintenance tasks.

### What is CI/CD?

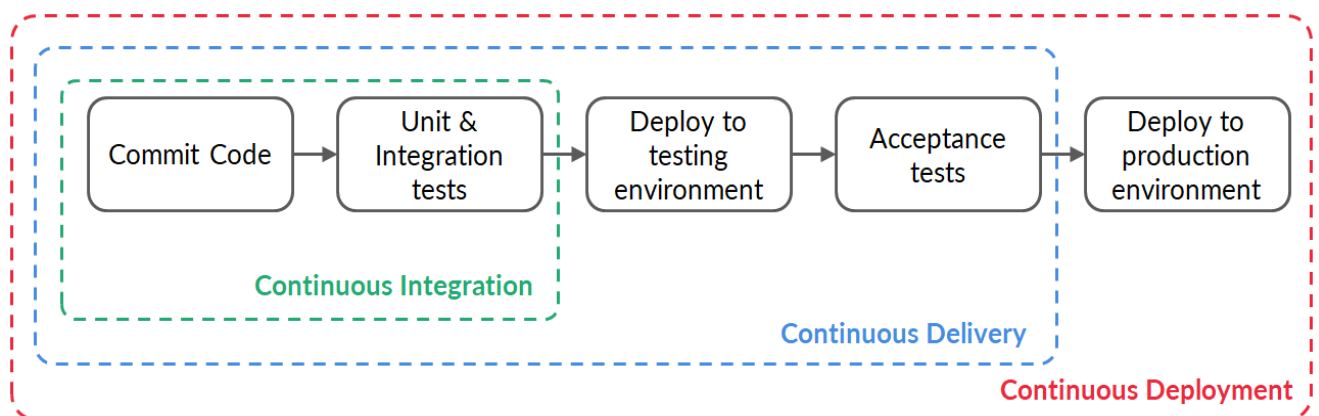
In software engineering, Continuous Integration (CI) is defined as a process of integrating all developers' code to a shared repository in a version-control system (VCS) frequently. CI is supposed to be used in conjunction with automated build, test and QA.

A build server compiles the code and reports the results to the developers on a regular basis or even after each commit. Unit testing in the developer's local environment before committing to the mainline helps avoid one developer's

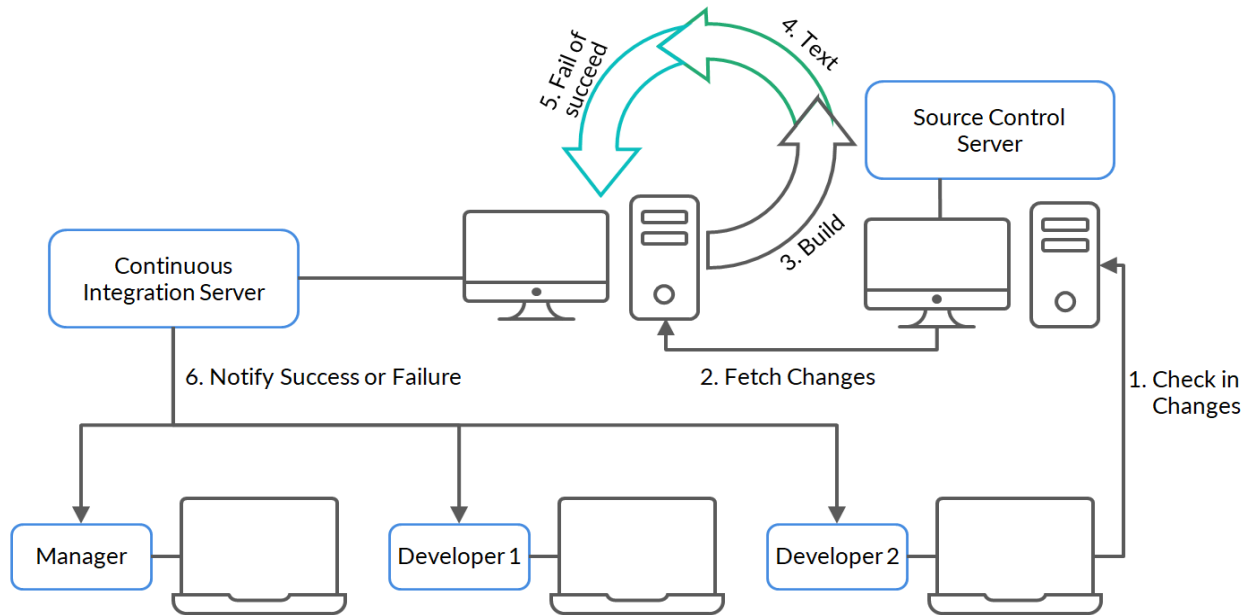
work-in-progress while breaking another developer's copy. Along with running the unit and integration checks, you can check the quality of your code and profile performance and format the documentation from the source code, thereby facilitating QA processes.

This quality check application mainly aims to enhance software quality and reduce delivery time continuously. CI is connected closely with Continuous Delivery or Continuous Deployment which is also called the CI/CD pipeline. Continuous Delivery ensures that an incremental feature of a software product checked in on the mainline is ready to deliver to the end user. Continuous Deployment makes the deployment process fully automated.

CI in its simplest form, involves a tool that continuously monitors VCS for any new changes. Whenever any new changes are pushed, this tool starts compiling and testing your application. If a bug is found or the code fails, the developers are immediately notified to fix the issue. In DevOps the typical ideology is that “If you are going to fail, then fail fast so that we can immediately fix it”, so catching bugs or failures as fast as possible is the game here. If you integrate it with automated end-to-end acceptance checks, CI can serve as a feedback method, offering a straightforward image of the current state of development efforts. It will allow you to deploy the latest version of your application either automatically or as a one-click process.



## CONTINUOUS INTEGRATION





## TOOLS AVAILABLE

1. *Git* : Git is a distributed version control system (DVCS) used for tracking changes in source code during software development. It is a widely used tool for managing codebases, collaborating with team members, and tracking changes across multiple contributors. Key features of Git include - Version Control, Branching and Merging, Distributed Development, Commit at any point, Local Repository, Pull Request and Conflict Resolution.
2. *GitHub* : GitHub is a web-based platform and hosting service for version control using Git. It enables developers to collaborate on projects, track changes, and manage codebases effectively. GitHub provides features such as pull requests, issues tracking, and project wikis to facilitate collaboration and communication among team members. It supports both public and private repositories, offering visibility and security options. GitHub fosters a vibrant community of developers, promoting open-source collaboration and knowledge sharing.
3. *Jenkins* : Jenkins is an open-source automation server used primarily for continuous integration (CI) and continuous delivery (CD) pipelines. It automates various stages of the software development lifecycle, including building, testing, and deploying applications. Jenkins supports defining CI/CD pipelines as code using Jenkinsfile, allowing for reproducibility and traceability. It offers extensibility through a vast ecosystem of plugins, enabling integration with version control systems, testing frameworks, and cloud providers. Jenkins helps teams deliver high-quality software faster and more reliably.
4. *Docker* : Docker is a platform that enables developers to package, deploy, and run applications in lightweight, portable containers. These containers encapsulate applications and their dependencies, ensuring consistency across different environments, such as development, testing, and production. Docker containers provide isolation and resource management, allowing applications to run reliably on any infrastructure, from local machines to cloud environments. Docker simplifies the process of software deployment by eliminating compatibility issues and streamlining the deployment pipeline. It accelerates development cycles,

improves scalability, and promotes the adoption of microservices architecture and cloud-native technologies.

5. *Ansible* : Ansible is an open-source automation tool used for configuration management, application deployment, and orchestration. It simplifies IT operations by automating repetitive tasks, such as provisioning servers, managing configurations, and deploying applications. Ansible uses YAML-based playbooks to define tasks and workflows, allowing for easy automation of complex infrastructure tasks. It's agentless, secure, and scalable, making it ideal for managing modern IT environments.

6. *Kubernetes* : Kubernetes, commonly abbreviated as K8s (numeronym), is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. Kubernetes is a powerful container orchestration platform with a rich set of features that facilitate the deployment, scaling, and management of containerized applications. It's features include container orchestration, auto scaling, service discovery and load balancing, rolling updates and rollbacks, self-healing declarative configurations, multi cloud and hybrid deployments, portability, extensibility, resource management and logging.

7. *Docker Compose* : Docker Compose is a tool for defining and managing multi-container Docker applications. Using a docker-compose.yml file, you can specify how different services in your application should interact, including their configurations, networks, and volumes. This YAML file allows you to define each container with details like the image to use, environment variables, and ports to expose. Docker Compose simplifies starting and stopping all services with simple commands like docker-compose up and docker-compose down. It also supports scaling services, making it easy to run multiple instances of a container. Ideal for development, testing, and microservices, Docker Compose streamlines the orchestration of containerized applications, ensuring consistency and reducing complexity.

8. *ELK Stack* : The ELK Stack is a powerful set of open-source tools for searching, analyzing, and visualizing log data in real-time. It consists of Elasticsearch, Logstash, and Kibana:

- Elasticsearch: A distributed search and analytics engine that indexes and stores data, enabling fast searches and complex queries.
- Logstash: A data processing pipeline that ingests data from various sources, transforms it, and sends it to Elasticsearch. It supports a wide range of input, filter, and output plugins.
- Kibana: A visualization tool that integrates with Elasticsearch to create interactive dashboards and reports. It allows users to explore data through charts, graphs, and maps.

The ELK Stack is commonly used for monitoring and observability, log and event data analysis, and security information and event management (SIEM).

9. *Ngrok* : Ngrok is a tool that provides secure tunnels from a public endpoint to a local service. It allows you to expose a local server to the internet for testing, demos, or remote access.

## CONFIGURATIONS

### FRONTEND

#### Prerequisites:

##### *1. Node.js and npm*

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows you to run JavaScript code outside of a browser. npm (Node Package Manager) is a package manager for Node.js, used to install libraries and packages.

#### Installation Steps:

##### *1. Download Node.js:*

- Visit the [Node.js official site](#).
- Choose the LTS (Long-Term Support) version for stability. This version is recommended for most users.
- Download the installer for your operating system (Windows, macOS, or Linux).

##### *2. Install Node.js:*

- Run the downloaded installer.
- Follow the prompts in the installation wizard.
- On macOS and Linux, you might need to use sudo to install Node.js globally.

##### *3. Verify Installation:*

- Open a terminal or command prompt.
- Run the following commands to check if Node.js and npm are installed correctly: 'node -v' and 'npm -v'.
- These commands should print the installed versions of Node.js and npm.

## 2. *Java Development Kit (JDK)*

The Java Development Kit (JDK) is necessary for Android development. React Native uses the JDK to build and run Android applications.

### *Installation Steps:*

#### 1. *Download JDK:*

- Visit the Oracle JDK download page.
- Download the latest version of the JDK suitable for your operating system.

#### 2. *Install JDK:*

- Run the downloaded installer.
- Follow the installation instructions.
- On macOS and Linux, you might need to use `sudo` to install the JDK globally.

#### 3. *Set Up Environment Variables (if not automatically set):*

- You need to add the JDK's bin directory to your system's PATH environment variable.
- On Windows:
  1. Open the Start menu, search for "Environment Variables," and select "Edit the system environment variables."
  2. Click on "Environment Variables."
  3. Under "System variables," find the Path variable and click "Edit."
  4. Add the path to the JDK's bin directory (e.g., C:\Program Files\Java\jdk-14\bin).

#### 4. *Verify Installation:*

- Open a terminal or command prompt.
- Run the following command to check if the JDK is installed correctly: `javac -version`.
- This should print the installed version of the JDK.

## Install React Native CLI

React Native CLI is the official command-line interface for React Native. Run following command for installation:

```
npm install -g react-native-cli
```

## Setup Android Development Environment

1. Android Studio: Necessary for the Android emulator.
  - Download and install from [Android Studio official site](#).
2. Android SDK:
  - a. During Android Studio installation, ensure that the Android SDK, SDK Platform, and Android Virtual Device are selected.
  - b. Open Android Studio and navigate to Preferences -> Appearance & Behavior -> System Settings -> Android SDK.
  - c. Ensure the following packages are installed:
    - i. SDK Platforms: Install the latest version of Android.
    - ii. SDK Tools: Install the latest versions of Android SDK Build-Tools, Android Emulator, Android SDK Tools, and Android SDK Platform-Tools.
3. Configure Environment Variables:

Add the following lines to your ~/.bash\_profile or ~/.zshrc file, depending on your shell:

```
export ANDROID_HOME=$HOME/Library/Android/sdk
```

```
export PATH=$PATH:$ANDROID_HOME/emulator
```

```
export PATH=$PATH:$ANDROID_HOME/tools
```

```
export PATH=$PATH:$ANDROID_HOME/tools/bin
```

```
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

### Create and Start an Android Emulator

Create an AVD (Android Virtual Device):

1. Open Android Studio.
2. Navigate to Tools -> AVD Manager.
3. Click Create Virtual Device.
4. Choose a device definition and select a system image (recommended: latest stable release).
5. Follow the prompts to complete the AVD setup.

Start the Emulator: In AVD Manager, click the play button next to your newly created virtual device.

### Install Expo CLI

Expo simplifies development by providing a set of tools and services built around React Native. Following is the command to install Expo CLI:

```
npm install -g expo-cli
```

### Create a new react native project with Expo

1. Initialize a new project: Use following command

```
expo init <Project_Name>
```

2. Navigate to the project directory:

```
cd <Project_Name>
```

3. Start the development server: This will start the Metro bundler, and you will see a QR code in your terminal or browser.

`expo start`

### Testing with Expo Go App

1. Install Expo Go on your mobile phone: Available on Google play store.
2. Run the App: Open the Expo Go on your device and scan the QR code displayed by the 'expo start'.

### Running on Android Emulator

Ensure the emulator is running. Now start the emulator using AVD Manager. Use below command to build and run the app:

`expo run:android`



## BACKEND

1. Setup spring project for every microservice.
2. Here we have done following microservices -
  - Service Registry
  - API Gateway
  - User Service
  - Complaint Service
  - Auth Service
3. Add respective required dependencies in each project

### 1. Service Registry -

- *Spring Cloud Starter* - This dependency includes core Spring Cloud components and configurations necessary for a Spring Cloud application.
- *Spring Cloud Netflix Eureka Server* - Purpose: Provides the Eureka server functionality. Eureka is a service registry that enables microservices to register and discover each other.
- *Spring Boot Starter Test* - Includes dependencies for testing Spring Boot applications, such as JUnit, Hamcrest, and Mockito. It's used to write and run tests for the application.
- *Spring Cloud OpenFeign* - Provides support for Feign, a declarative web service client. It simplifies HTTP API clients by using declarative annotations and integrating with Spring Cloud.
- *Spring Cloud Dependencies (Dependency Management)* - Manages the versions of Spring Cloud dependencies to ensure compatibility and prevent version conflicts. This simplifies the management of dependency versions across the project.

- *Maven Compiler Plugin (Build Section)* - The Spring Boot Maven plugin provides Spring Boot support in Maven, allowing you to package the application as an executable JAR or WAR.

## 2. Auth Service -

- *Spring Boot Starter Security*- Provides core security features for Spring applications, such as authentication and authorization.
- *Lombok* - Simplifies Java code by providing annotations to reduce boilerplate code, such as getters, setters, constructors, and more.
- *Spring Boot Starter Test*
- *Spring Security Test* - Provides testing support for Spring Security, enabling tests for security-related functionality.
- *Spring Boot Starter Web* - Provides necessary components for building web applications, including Spring MVC, RESTful services, and embedded Tomcat.
- *Spring Cloud Netflix Eureka Client* - Enables the application to register with and discover services from a Eureka server
- *Spring Cloud OpenFeign*
- *Spring Cloud Starter Bootstrap* - Ensures the application loads properties from a centralized configuration server at the bootstrap phase.
- *JSON Web Token (JWT)* - Provides support for creating and verifying JSON Web Tokens (JWTs) for authentication and authorization.
- *Spring Cloud Dependencies* (Dependency Management)

- *Maven Compiler Plugin* (Build Section)

### 3. API Gateway -

- Spring Cloud Starter
- Spring Cloud Netflix Eureka Client
- Lombok
- Spring Boot Starter Test
- Spring Boot Starter WebFlux - Provides components for building reactive web applications with Spring WebFlux.
- JSON Web Token (JWT)
- Spring Cloud Gateway - Implements the API Gateway pattern with Spring Cloud Gateway, providing features like routing, filtering, and load balancing.

### 4. UserService and ComplaintService-

- *Spring Boot Starter Data JPA*: This dependency provides the necessary dependencies to work with Spring Data JPA, which simplifies the implementation of data access layers.
- *Spring Boot Starter Web*
- *MySQL Connector/J*: This dependency provides the MySQL JDBC driver for connecting to a MySQL database.
- Lombok
- Spring Boot Starter Test
- *Spring Boot Starter Validation*: This starter includes dependencies for validation support in Spring Boot applications.
- *Jakarta Validation API*: This is the Jakarta Bean Validation API, which provides a standard way to validate Java objects.

- *JavaMail API*: This dependency provides the JavaMail API for sending and receiving email messages.
- *Spring Security Crypto*: This dependency provides cryptographic utilities for Spring Security, such as password encoding.
- *Spring Cloud Starter Bootstrap*
- *Spring Cloud Starter Netflix Eureka Client*
- *Spring Cloud Starter OpenFeign*
- *Spring Boot Starter WebFlux*
- *SLF4J API*: This is the Simple Logging Facade for Java (SLF4J) API, which provides a flexible logging API for Java applications.
- *Logstash Logback Encoder*: This encoder formats logs according to the Logstash JSON format, which is commonly used for centralized logging solutions.
- *Mockito JUnit Jupiter*: This dependency provides integration between Mockito and JUnit Jupiter for testing.
- *Mockito Inline*: This dependency provides the Mockito Inline library for testing, which allows Mockito mocks to be created without explicitly calling MockitoAnnotations.initMocks().

#### 4. Set up application.yml files

## MICROSERVICE ARCHITECTURE

Microservice architecture is an architectural style that structures an application as a collection of loosely coupled, independently deployable services. Each service is fine-grained and focuses on a specific business capability. This approach contrasts with traditional monolithic architecture, where all components and functions are tightly integrated into a single, large application.

Microservice architecture provides a robust framework for building scalable, flexible, and maintainable applications. It addresses many of the limitations associated with monolithic architectures, making it a preferred choice for modern software development, especially in environments that require rapid innovation and agility.

### Key Characteristics:

1. *Independent Deployment:* Each microservice can be deployed independently without impacting other services. This enables continuous integration and continuous deployment (CI/CD).
2. *Decentralized Data Management:* Each service manages its own database. This encapsulates the service's data and logic, reducing dependencies between services.
3. *Technology Diversity:* Different services can use different programming languages, frameworks, and data storage technologies, allowing teams to choose the best tool for each job.
4. *Scalability:* Services can be scaled independently based on demand. For instance, a service experiencing high load can be scaled without scaling the entire application.
5. *Resilience and Fault Tolerance:* Each service is built around a specific business function, such as user management, payment processing, or inventory management.

## Need for Microservice Architecture:

### *1. Handling Complexity:*

- As applications grow in size and complexity, monolithic architectures become difficult to manage, understand, and modify. Microservices break down the complexity into manageable pieces.

### *2. Improved Agility and Speed:*

- Smaller, focused teams can develop, test, deploy, and scale their services independently, accelerating development cycles and time to market.

### *3. Scalability:*

- Individual services can be scaled independently, allowing for better resource utilization and cost-efficiency. This is particularly useful in scenarios where different parts of the application have different scaling needs.

### *4. Resilience and Reliability:*

- By isolating services, the failure of one service does not necessarily lead to the failure of the entire system. This makes the overall system more resilient and easier to troubleshoot and maintain.

### *5. Flexibility in Technology Stack:*

- Teams can select the most appropriate technology stack for each service, leveraging the latest innovations and avoiding the limitations of a single technology stack.

### *6. Continuous Deployment and Delivery:*

- Microservices support the rapid release of new features and updates. Each service can be developed, tested, and deployed independently, reducing the risk associated with large-scale deployments.

### *7. Enhanced Team Autonomy:*

- Development teams can work independently on different services, which aligns well with agile and DevOps practices. This promotes a culture of ownership and accountability.

### Real World Application:

Microservice architecture is widely adopted in various industries and by numerous tech giants:

- Amazon: Uses microservices to handle different parts of their vast e-commerce platform, ensuring scalability and reliability.
- Netflix: Employs microservices to manage their streaming service, allowing for rapid deployment of new features and better fault isolation.
- Uber: Utilizes microservices to support their complex, distributed system of ride-sharing, payments, notifications, and more.

### Microservice architecture implementation in CitizenConnect

CitizenConnect is designed using a microservice architecture to enhance scalability, maintainability, and deployment flexibility. The system is divided into several distinct services, each handling specific functionalities related to citizen interactions and administrative operations. The key services include User Service, Complaint Service, Auth Service, API Gateway, and Service Registry.

### *List of all Microservices:*

1. *User Service*
  - a. Functionality: Manages details of Citizens and Nagar Sevaks.
  - b. Responsibilities:
    - i. Storing and retrieving user profiles.
2. *Complaint Service*
  - a. Functionality: Manages details of complaints raised by citizens and comments on these complaints.
  - b. Responsibilities:
    - i. Recording complaints from citizens.
    - ii. Tracking the status of complaints.
    - iii. Handling comments and updates on complaints.
3. *Auth Service*

- a. Functionality: Handles authentication and JWT token verification.
  - b. Responsibilities:
    - i. Verifying JWT tokens for secure API access.
    - ii. Issuing tokens upon successful login.
    - iii. Ensuring secure communication between services.
4. *API Gateway*
- a. Functionality: Acts as an entry point for all requests to the system.
  - b. Responsibilities:
    - i. Routing requests to the appropriate microservice.
    - ii. Handling request aggregation and response transformation.
    - iii. Providing security and rate limiting.
5. *Service Registry*
- a. Functionality: Registers and discovers services within the system.
  - b. Responsibilities:
    - i. Registering each service instance.
    - ii. Enabling service discovery for inter-service communication.
    - iii. Utilizing Eureka for service registration and discovery.

### *Intercommunication -*

Inter-service communication is achieved using FeignClient for declarative REST clients, allowing seamless and type-safe HTTP requests between services.

#### *1. User and Complaint Service Communication*

- Scenario: Accessing complaints registered by a user or complaints within a user's area.
- Interaction: Complaint Service fetches user-related data from the User Service using FeignClient.

#### *2. Complaint Registration and Comments*

- Scenario: Registering new complaints or comments on existing complaints.
- Interaction: Complaint Service directly handles the recording of complaints and comments.

#### *3. User Authentication and Registration*

- Login:



- Interaction: Auth Service retrieves user credentials from the User Service to validate login details.
- Registration:
  - Interaction: During the registration of admins and citizens, the Auth Service interacts with the User Service. This operation does not require a token as it is public.

### Ngrok :

1. Create ngrok Account and Get Auth Token Create an Account: Go to ngrok website and sign up for a free account. Get Auth Token: After logging in, navigate to your dashboard and copy your ngrok authtoken.

2. Install ngrok and Expose a Port. Download and Install ngrok: Download ngrok from ngrok download page. Unzip the file and move the ngrok binary to a directory in your PATH (e.g., /usr/local/bin). Authenticate and Start ngrok:

3. Create Dockerfile to Automate Installation and Exposure

## ELK STACK

ELK Stack is a powerful combination of three open-source tools: Elasticsearch, Logstash, and Kibana. Each component of the ELK Stack plays a specific role in collecting, processing, storing, and visualizing log data:

1. *Elasticsearch*: At the core of the ELK Stack is Elasticsearch, which is a distributed, RESTful search and analytics engine. Elasticsearch is designed for horizontal scalability, allowing it to handle large volumes of data across multiple nodes. It's used for indexing and searching structured and unstructured data, making it ideal for log analysis.
2. *Logstash*: Logstash is a data processing pipeline that ingests, transforms, and sends log data to Elasticsearch. It's responsible for collecting logs from various sources, parsing them, enriching them, and formatting them into a common format that Elasticsearch can understand. Logstash supports a wide range of inputs, filters, and outputs, making it highly flexible and customizable.
3. *Kibana*: Kibana is a powerful visualization and analytics platform that sits on top of Elasticsearch. It provides a user-friendly interface for exploring, analyzing, and visualizing data stored in Elasticsearch. With Kibana, users can create custom dashboards, charts, graphs, and maps to gain insights from their log data. Kibana also supports features like time series analysis, anomaly detection, and machine learning.

Together, Elasticsearch, Logstash, and Kibana form a comprehensive log management solution known as the ELK Stack. This stack is widely used for real-time log analysis, monitoring, troubleshooting, and security analytics across a variety of use cases and industries.

## TESTING USING *MOKITO*

Mockito is a popular Java library used for mocking dependencies and writing unit tests. Here's a generalized theory about Mockito:

### *Mocking:*

- Mockito allows developers to create mock objects, which simulate the behavior of real objects in a controlled manner during testing.
- Mock objects are used to isolate the code being tested by replacing real dependencies with fake implementations.
- Mocking is particularly useful when testing code that relies on external systems, databases, or complex objects that are difficult to set up in a testing environment.

### *Key Concepts:*

1. *@Mock Annotation:*
  - Used to create a mock object of a class or interface.
  - Annotated fields are automatically initialized with mock instances.
2. *@InjectMocks Annotation:*
  - Used to inject mock objects into the class under test.
  - Automatically injects mocks into the target object's fields.
3. *MockitoAnnotations.openMocks():*
  - Initializes objects annotated with Mockito annotations like *@Mock*, *@InjectMocks*, etc.
  - Typically called in the setup phase of test methods or in a setup method annotated with *@BeforeEach*.
4. *Stubbing:*
  - Refers to defining the behavior of mock objects.
  - Developers specify the return values or behaviors of methods invoked on mock objects.
5. *Verification:*
  - Involves verifying that certain interactions with mock objects occurred during the test.
  - Mockito provides methods to verify method invocations, argument values, and the number of invocations.

### Usage:

#### *1. Unit Testing:*

- Mockito is commonly used for unit testing to isolate the code being tested and verify its behavior.

#### *2. Integration Testing:*

- Mockito can also be used in integration testing to mock external dependencies and focus on testing the integration between components.

#### *3. Behavior-Driven Development (BDD):*

- Mockito supports BDD-style testing with methods like `given()`, `when()`, `then()` for improved readability and expressiveness in tests.

### Benefits:

- *Simplicity:* Mockito provides a simple and easy-to-understand API for mocking.
- *Flexibility:* Allows developers to mock interfaces, classes, abstract classes, and even final classes and methods.
- *Maintainability:* Helps in writing clean and maintainable tests by isolating dependencies and focusing on specific behaviors.

### Limitations:

- Mockito is primarily designed for unit testing and may not be suitable for all testing scenarios, such as complex integration tests.
- It's important to use Mockito judiciously and avoid over-mocking, which can lead to brittle tests and decreased test readability.

### Best Practices:

- Mock only the dependencies that are necessary for the test scenario.
- Keep the test setup simple and avoid overly complex stubbing.
- Verify only the interactions that are relevant to the behavior being tested.
- Write clear and descriptive test cases to enhance readability and maintainability.

# SECURITY USING JWT

JSON Web Tokens (JWT) is a compact, URL-safe means of representing claims securely between two parties. It's widely used for authentication and authorization in web applications due to its simplicity, compactness, and self-contained nature. Here's a detailed report on JWT:

## 1. Introduction to JWT:

- Definition: JWT is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- Components:
  - Header: Contains metadata about the token such as the type (JWT) and the signing algorithm being used.
  - Payload: Contains the claims. Claims are statements about an entity (typically, the user) and additional data.
  - Signature: Used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

## 2. Structure of JWT:

- JWTs are composed of three base64-encoded sections separated by dots (.): Header.Payload.Signature.
- Example:  
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`

## 3. Working Principle:

- Authentication: After the user logs in, the server generates a JWT and sends it to the client. The client includes the JWT in subsequent requests to authenticate itself.
- Authorization: The server verifies the JWT to ensure that the client has permission to access the requested resources.

- Stateless: JWTs are stateless, meaning the server doesn't need to store session information. All necessary information is contained within the token itself.

#### 4. Use Cases:

- Single Sign-On (SSO): JWTs are commonly used for implementing SSO across multiple domains.
- Secure Communication: JWTs can be used to securely transmit information between different components of a distributed system.
- User Authentication: JWTs authenticate users by including user-specific information (claims) in the token.

#### 5. Security Considerations:

- Encryption: JWTs are not encrypted by default. Sensitive information should be avoided in the payload or encrypted separately.
- Token Expiry: JWTs can include an expiry time (exp claim) to mitigate the risk of token reuse.
- Signature Algorithm: The choice of signature algorithm affects the security of JWTs. Strong algorithms like RS256 or ES256 should be used.
- Token Size: Keep the JWT size small to reduce overhead and improve performance.
- Revocation: JWTs are stateless, so they cannot be revoked. Care should be taken when issuing long-lived tokens.

#### 6. Best Practices:

- Use HTTPS: Transmit JWTs over HTTPS to prevent interception and tampering.
- Validate Input: Validate all input to prevent injection attacks and ensure that the JWT is properly formed and signed.
- Avoid Storing Sensitive Data: Avoid storing sensitive information in JWTs to minimize security risks.

## 7. Libraries and Frameworks:

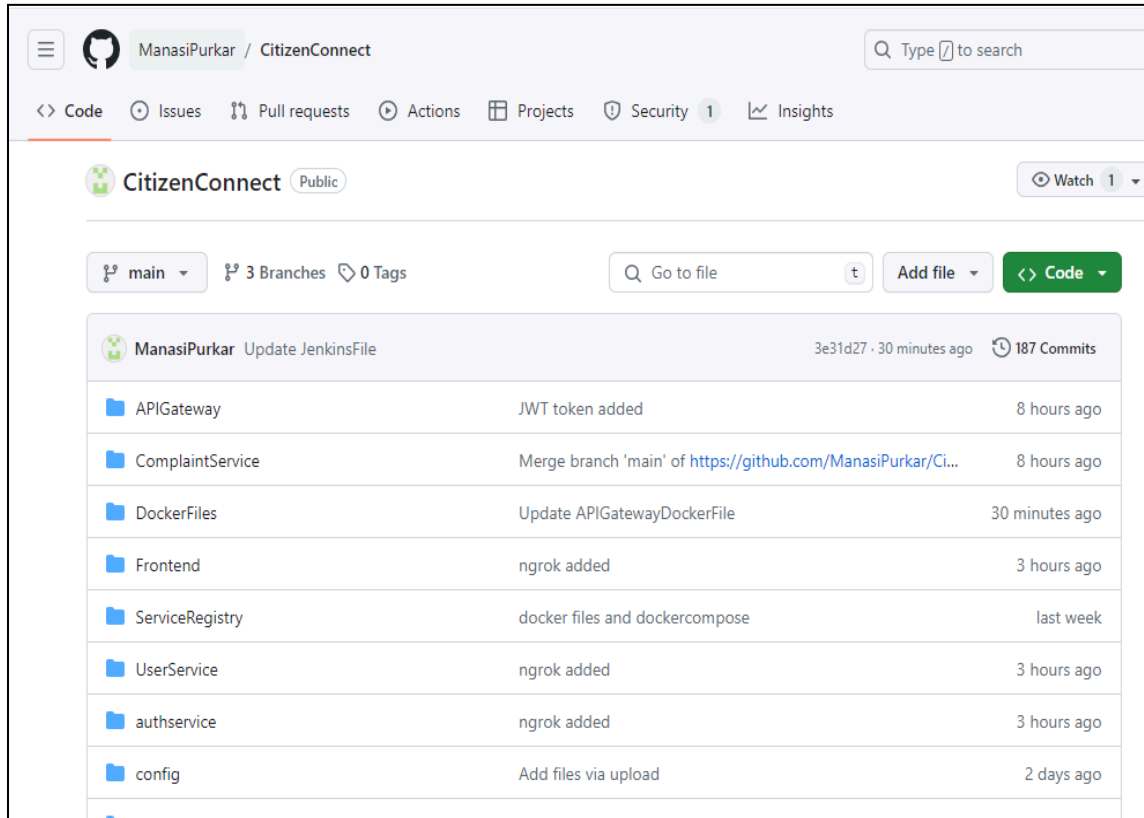
- Numerous libraries and frameworks provide support for JWT generation, parsing, and validation in various programming languages and platforms.
- Examples include *jsonwebtoken* in Node.js, *pyjwt* in Python, *java-jwt* in Java, and *Microsoft.IdentityModel.Tokens* in .NET.

## 8. Conclusion:

JWTs offer a flexible and secure method for authentication and authorization in web applications. By following best practices and understanding security considerations, developers can leverage JWTs effectively to enhance the security and scalability of their applications. However, it's essential to carefully design token payloads and choose appropriate security measures to mitigate potential risks associated with JWTs.

## SCREENSHOTS OF ALL REQUIRED FILES AND SERVICES

### 1. Version Control using Git



### 2. Docker Files

#### a. API Gateway Docker File:



```

FROM ubuntu:22.04

# Set working directory
WORKDIR /app

# Copy application files
COPY ./APIGateway /app
COPY ./APIGateway/target/APIGateway-0.0.1-SNAPSHOT.jar ./

# Install dependencies
RUN apt-get update && apt-get install -y wget unzip \
    && apt-get install -y software-properties-common \
    && add-apt-repository ppa:openjdk-r/ppa \
    && apt-get update && apt-get install -y openjdk-21-jdk

# Install ngrok
RUN wget https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-v3-stable-linux-amd64.zip \
    && unzip ngrok-v3-stable-linux-amd64.zip \
    && mv ngrok /usr/local/bin/ \
    && rm ngrok-v3-stable-linux-amd64.zip

```

```

# Set ngrok authToken
RUN ngrok authToken 2bexTFg5VNQNuyBAHXJBrPTwhOR_fJhk6PLTAZFisXKPTDdq

# Expose the port
EXPOSE 9093

# Create an entrypoint script
RUN echo '#!/bin/bash\n\
ngrok http --bind-tls=true 9093 &\n\
java -jar /app/APIGateway-0.0.1-SNAPSHOT.jar' > /app/entrypoint.sh

# Make the entrypoint script executable
RUN chmod +x /app/entrypoint.sh

# Use the entrypoint script
CMD ["/app/entrypoint.sh"]

```

b. Complain Service Docker File:

```
ComplaintServiceDockerFile x
1 FROM openjdk:21
2 WORKDIR /app
3 COPY ./ComplaintService /app
4 COPY ./ComplaintService/target/ComplaintService-0.0.1-SNAPSHOT.jar ./
5 EXPOSE 9092
6 CMD ["java","-jar","ComplaintService-0.0.1-SNAPSHOT.jar"]
7
```

c. User Service Docker File:

```
UserServiceDockerFile x
1 FROM openjdk:21
2 WORKDIR /app
3 COPY ./UserService /app
4 COPY ./UserService/target/UserService-0.0.1-SNAPSHOT.jar ./
5 EXPOSE 9091
6 CMD ["java","-jar","UserService-0.0.1-SNAPSHOT.jar"]
7
```

d. Service Registry Docker File

```
ServiceRegistryDockerFile x
1 FROM openjdk:21
2 WORKDIR /app
3 COPY ./ServiceRegistry /app
4 COPY ./ServiceRegistry/target/ServiceRegistry-0.0.1-SNAPSHOT.jar ./
5 EXPOSE 8761
6 CMD ["java","-jar","ServiceRegistry-0.0.1-SNAPSHOT.jar"]
7
```

e. Frontend Docker File

```

28  # Use the official Node.js image as the base image
29  FROM node:latest AS builder
30
31  # Set working directory
32  WORKDIR /app
33
34  # Copy package.json and package-lock.json to the working directory
35  COPY package.json package-lock.json ./
36
37  # Install dependencies
38  RUN npm install
39
40  # Copy the entire frontend directory to the working directory
41  COPY . .
42
43  # Expose the port the app runs on
44  EXPOSE 8081
45  #EXPOSE 19001
46  #EXPOSE 19002
47
48  # set environment variables
49  #ARG REACT_APP_SERVER_URL
50  #ENV REACT_APP_SERVER_URL $REACT_APP_SERVER_URL
51
52  # Command to run the app
53  CMD ["npm", "start"]
54
```

## 2. Jenkins File

```

pipeline {
  environment {
    DOCKERHUB_CREDENTIALS = credentials('DockerHubCred')
    MYSQL_CREDENTIALS = credentials('MysqlCred')
    DOCKERHUB_USER = 'manasip44'
    // MYSQL_CREDENTIALS = credentials('MySQL_Credentials')
  }
  agent any
  stages {
    stage('Clone repository') {
      steps {
        git branch: 'main', url: 'https://github.com/ManasiPurkar/CitizenConnect.git'
      }
    }

    stage('Maven Test User Service'){
      steps{
        echo 'Building Job'
        sh 'cd UserService; mvn test';
      }
    }
    stage('Maven Test Complaint Service'){
      steps{
        echo 'Building Job'
        sh 'cd ComplaintService; mvn test';
      }
    }
  }
}

```

```

    stage('Maven Test Auth Service'){
      steps{
        echo 'Building Job'
        sh 'cd authservice; mvn test';
      }
    }
    stage('Maven Build CitizenConnectServiceRegistry'){
      steps{
        echo 'Building Job'
        sh 'cd ServiceRegistry; mvn clean install';
      }
    }
    stage('Maven Build CitizenConnectAPIGateway'){
      steps{
        echo 'Building Job'
        sh 'cd APIGateway; mvn clean install';
      }
    }
    stage('Maven Build Auth Service'){
      steps{
        echo 'Building Job'
        sh 'cd authservice; mvn clean install';
      }
    }
  }
}

```

```

stage('Maven Build CitizenConnectUserService'){
  steps{
    echo 'Building Job'
    sh 'cd UserService; mvn clean install -DSPRING_DATASOURCE_USERNAME=$MYSQL_CREDENTIALS_USR
      -DSPRING_DATASOURCE_PASSWORD=$MYSQL_CREDENTIALS_PSW';
  }
}

stage('Maven Build CitizenConnectComplaintService'){
  steps{
    echo 'Building Job'
    sh 'cd ComplaintService; mvn clean install -DSPRING_DATASOURCE_USERNAME=$MYSQL_CREDENTIALS_USR
      -DSPRING_DATASOURCE_PASSWORD=$MYSQL_CREDENTIALS_PSW';
  }
}

stage('Build Image for Microservices'){
  steps{
    echo 'Building docker Image'
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:serviceregistry -f DockerFiles/ServiceRegistryDockerFile .";
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:apigateway -f DockerFiles/APIGatewayDockerFile .";
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:authservice -f DockerFiles/AuthServiceDockerFile .";
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:userservice -f DockerFiles/UserServiceDockerFile .";
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:complaintservice -f DockerFiles/ComplaintServiceDockerFile .";
    //sh "cd Frontend; docker build -t ${DOCKERHUB_USER}/citizenconnect:frontend -f Dockerfile .";
    sh "docker build -t $DOCKERHUB_USER/citizenconnect:frontend -f DockerFiles/FrontendDockerFile .";
  }
}

```

```

stage('Login into docker hub'){
  steps{
    echo 'Login into docker hub'
    sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin';
  }
}

stage('Push Image to DockerHub'){
  steps{
    echo 'Pushing Images into DockerHub'
    sh 'docker push $DOCKERHUB_USER/citizenconnect:serviceregistry';
    sh 'docker push $DOCKERHUB_USER/citizenconnect:apigateway';
    sh 'docker push $DOCKERHUB_USER/citizenconnect:authservice';
    sh 'docker push $DOCKERHUB_USER/citizenconnect:userservice';
    sh 'docker push $DOCKERHUB_USER/citizenconnect:complaintservice';
    sh 'docker push $DOCKERHUB_USER/citizenconnect:frontend';
  }
}

stage('Delete Image from localsystem'){
  steps{
    echo 'Deleting Docker Image in docker'
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:serviceregistry';
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:apigateway';
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:authservice';
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:userservice';
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:complaintservice';
    sh 'docker rmi $DOCKERHUB_USER/citizenconnect:frontend';
  }
}

```

```

stage('Run Ansible Playbook'){
    steps{
        script {
            ansiblePlaybook(
                playbook: 'playbook.yml',
                inventory: 'inventory'
            )
        }
    }
}

```

### 3. Docker-Compose File

```

version: '3'

services:
  serviceregistry:
    image: manasip44/citizenconnect:serviceregistry
    container_name: serviceregistry
    ports:
      - "8761:8761"
    networks:
      - citizenconnect_network
    healthcheck:
      test: ["CMD-SHELL", "curl -f http://localhost:8761/ || exit 1"]
      interval: 30s
      timeout: 10s
      retries: 5

  apigateway:

```

```

    image: manasip44/citizenconnect:apigateway
    container_name: apigateway
    ports:
      - "9093:9093"
    networks:
      - citizenconnect_network
    environment:
      - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://serviceregistry:8761/eureka/
    depends_on:
      - serviceregistry

```

```

    depends_on:
      - serviceregistry

userservice:
  image: manasip44/citizenconnect:userservice
  container_name: userservice
  ports:
    - "9091:9091"
  environment:
    - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://serviceregistry:8761/eureka/
    - SPRING_DATASOURCE_URL=jdbc:mysql://userservice_db/userservice?createDatabaseIfNotExist=true
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=root
    - LOGGING_LOGSTASH_HOST=logstash
    - LOGGING_LOGSTASH_PORT=5001
  networks:
    - citizenconnect_network
  depends_on:
    - serviceregistry
    - apigateway
    - userservice_db
  volumes:
    - /home:/logs
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:9091/actuator/health || exit 1"]
    interval: 30s

```

```

    timeout: 10s
    retries: 5

complaintservice:
  image: manasip44/citizenconnect:complaintservice
  container_name: complaintservice
  ports:
    - "9092:9092"
  environment:
    - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://serviceregistry:8761/eureka/
    - SPRING_DATASOURCE_URL=jdbc:mysql://complaintservice_db/complaintservice?createDatabaseIfNotExist=true
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=root
    - LOGGING_LOGSTASH_HOST=logstash
    - LOGGING_LOGSTASH_PORT=5001
  networks:
    - citizenconnect_network
  depends_on:
    - serviceregistry
    - apigateway
    - complaintservice_db
  volumes:
    - /home:/logs
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:9092/actuator/health || exit 1"]
    interval: 30s
    timeout: 10s
    retries: 5

```

```
authservice:
  image: manasip44/citizenconnect:authservice
  container_name: authservice
  ports:
    - "9898:9898"
  networks:
    - citizenconnect_network
  environment:
    - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://serviceregistry:8761/eureka/
  depends_on:
    - serviceregistry
```



```

81
82     userservice_db:
83         image: mysql:8
84         container_name: userservice_db
85         ports:
86             - "3308:3306"
87         environment:
88             - MYSQL_ROOT_PASSWORD=root
89             - MYSQL_DATABASE=userservice
90         volumes:
91             - userservice_database:/var/lib/mysql
92             #- ./init-userservice-db.sql:/docker-entrypoint-initdb.d/init.sql
93         command: --bind-address=0.0.0.0
94         networks:
95             - citizenconnect_network
96
97     complaintservice_db:
98         image: mysql:8
99         container_name: complaintservice_db
100        ports:
101            - "3307:3306"
102        environment:
103            - MYSQL_ROOT_PASSWORD=root
104            - MYSQL_DATABASE=complaintservice
105        volumes:
106            - complaintservice_database:/var/lib/mysql

```

```

108        command: --bind-address=0.0.0.0
109        networks:
110            - citizenconnect_network
111
112    citizenconnectfrontend:
113        image: manasip44/citizenconnect:frontend
114        container_name: citizenconnectfrontend
115        ports:
116            - "8081:8081"
117        networks:
118            - citizenconnect_network
119
120    elasticsearch:
121        image: docker.elastic.co/elasticsearch/elasticsearch:8.3.3
122        container_name: elasticsearch
123        restart: always
124        environment:
125            - bootstrap_memory_lock = true
126            - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
127            - xpack.security.enabled=false
128            - discovery.type=single-node
129            - network.host=0.0.0.0
130            - http.port=9200
131        ports:
132            - "9200:9200"
133        networks:
134            - citizenconnect_network

```

```
docker-compose.yml x
135     volumes:
136     - elastic_data:/usr/share/elasticsearch/data
137     - ./elasticsearch/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
138
139     kibana:
140     image: docker.elastic.co/kibana/kibana:8.3.3
141     container_name: kibana
142     restart: always
143     environment:
144     ELASTICSEARCH_URL: http://elasticsearch:9200
145     ELASTICSEARCH_HOSTS: '["http://elasticsearch:9200"]'
146     ports:
147     - "5601:5601"
148     networks:
149     - citizenconnect_network
150     depends_on:
151     - elasticsearch
152
153     logstash:
154     image: docker.elastic.co/logstash/logstash:8.3.3
155     container_name: logstash
156     restart: always
157     volumes:
158     - ./config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro
159     - ./pipeline:/usr/share/logstash/pipeline:re
160     environment:
161     LS_JAVA_OPTS: "-Xmx512m -Xms512m"
```

```
docker-compose.yml x
159     - ./pipeline:/usr/share/logstash/pipeline:re
160     environment:
161     LS_JAVA_OPTS: "-Xmx512m -Xms512m"
162     ports:
163     - "5001:5001/tcp"
164     - "5001:5001/udp"
165     - "9600:9600"
166     networks:
167     - citizenconnect_network
168     depends_on:
169     - elasticsearch
170
171     networks:
172     citizenconnect_network:
173     driver: bridge
174
175     volumes:
176     complaintservice_database:
177     userservice_database:
178     elastic_data:
179     driver: local
180
```

## 5. Playbook.yml

```
---
- name: Deploy application with Docker Compose
  hosts: all
  become: yes

  pre_tasks:
    - name: Ensure pip is installed
      apt:
        name: python3-pip
        state: present
      when: ansible_os_family == "Debian"

    - name: Ensure pip is installed
      yum:
        name: python3-pip
        state: present
      when: ansible_os_family == "RedHat"

    - name: Install Docker Compose using pip
      pip:
        name: docker-compose
        state: present

  tasks:
    - name: Verify Docker Compose installation
      command: docker-compose --version
      register: docker_compose_version

    - debug:
        msg: "Docker Compose version: {{ docker_compose_version.stdout }}"

    - name: Copy Docker Compose file
      copy:
```

```

tasks:
  - name: Verify Docker Compose installation
    command: docker-compose --version
    register: docker_compose_version

  - debug:
    msg: "Docker Compose version: {{ docker_compose_version.stdout }}"

  - name: Copy Docker Compose file
    copy:
      src: docker-compose.yml
      dest: ./

  - name: Run Docker Compose
    community.general.docker_compose:
      project_src: ./
      state: present

```

## 6. Inventory

```

docker.env  playbook.yml  inventory x
1  [host1]
2  client1 ansible_host=192.168.120.199 ansible_connection=ssh ansible_user=manasi
3  ansible_ssh_pass=Family@2001 ansible_sudo_pass=Family@2001
4  ansible_ssh_common_args='-o StrictHostKeyChecking=no'
5

```

## 7. Logstash.yml

```
logstash.yml x
1 http.host: "0.0.0.0"
2 path.config: /usr/share/logstash/pipeline
3 xpack.monitoring.enabled: false
4 xpack.monitoring.elasticsearch.hosts: ["http://elasticsearch:9200"]
5
```

## 8. Elasticsearch.yml

```
logstash.yml elasticsearch.yml x
1 cluster.name: "elasticsearch"
2 network.host: 0.0.0.0
3 xpack.security.enabled: false
4
```

## 9. Logstash.conf

```
logstash.yml elasticsearch.yml logstash.conf x
1 input {
2   tcp {
3     port => 5001
4     codec => json
5   }
6 }
7 output {
8   elasticsearch {
9     hosts => ["http://elasticsearch:9200"]
10    index => "springboot-logs"
11  }
12  stdout { codec => rubydebug }
13 }
14
```

## 10. Testing-

```

@Test
void registerNagarsevak_Success() {
    String generatedPassword = "generatedPassword";
    when(areaRepository.findById(nagarsevakRequest.getAreaCode())).thenReturn(Optional.of(area));
    when(passwordEncoder.encode(generatedPassword)).thenReturn("encodedPassword");
    when(userRepository.save(any(Users.class))).thenReturn(user);
    when(nagarsevakRepository.save(any(Nagarsevak.class))).thenAnswer(i -> i.getArgument(0));
    when(emailService.sendEmail(anyString(), anyString(), anyString())).thenReturn(true);

    try (MockedStatic<PasswordGeneratorService> mockedStatic = mockStatic(PasswordGeneratorService.class)) {
        mockedStatic.when(PasswordGeneratorService::generatePassword).thenReturn(generatedPassword);

        Nagarsevak savedNagarsevak = adminService.registerNagarsevak(nagarsevakRequest);

        assertNotNull(savedNagarsevak);
        assertEquals("John", savedNagarsevak.getFirstname());
        assertEquals("Doe", savedNagarsevak.getLastname());
        assertEquals("Area 1", savedNagarsevak.getArea().getName());
        assertTrue(savedNagarsevak.getActive());

        verify(userRepository).save(any(Users.class));
        verify(nagarsevakRepository).save(any(Nagarsevak.class));
        verify(emailService).sendEmail(anyString(), anyString(), eq("nagarsevak@example.com"));
    }
}

```

```

@Test
public void testCreateComplaint_ThrowsAPIRequestException() {
    // Arrange
    ComplaintDTO complaintDTO = ComplaintDTO.builder()
        .address("123 Main St")
        .title("Test Title")
        .description("Test Description")
        .citizenId(1)
        .areaCode("12345")
        .areaName("Test Area")
        .department_code(1)
        .build();

    when(departmentRepository.findById(1)).thenReturn(Optional.empty());

    // Act & Assert
    APIRequestException exception = assertThrows(APIRequestException.class, () -> {
        complaintService.createComplaint(complaintDTO);
    });

    assertEquals("Wrong Department !", exception.getMessage());

    verify(departmentRepository, times(1)).findById(1);
    verify(complaintsRepository, never()).save(any(Complaints.class));
    System.out.println("create complaint failure tested");
}

```

11. Eureka Service Registry

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">e5d80cb8a219:API-GATEWAY:9093</a>
AUTHSERVICE	n/a (1)	(1)	UP (1) - <a href="#">d481fec5d240:AuthService:9898</a>
COMPLAINTSERVICE	n/a (1)	(1)	UP (1) - <a href="#">645afc7860d7:ComplaintService:9092</a>
USERSERVICE	n/a (1)	(1)	UP (1) - <a href="#">05935653aabf:UserService:9091</a>

12. Pipeline in Jenkins

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

GitHub Hook Log

Build History

trend

✔ CitizenConnectSPE

Add description

Disable Project

Stage View

Average stage times:  
(Average full run time: ~46min 33s)

#109

May 24 22:49

2 commits

Declarative: Checkout SCM	Clone repository	Build Image for Microservices	Login into docker hub	Push Image to DockerHub	Delete Image from localsystem	Run Ansible Playbook
2s	1s	9min 20s	6s	25min 25s	3s	11min 25s
2s	1s	9min 20s	6s	25min 25s	3s	11min 25s

## STEPS TO BUILD

*Step 1:* Create a Frontend application in React Native and a Backend application in SpringBoot. After creation of these applications separately, integrate them both. Check if the application works fine after integration. Push all the changes into the Github repository whenever any new changes are made.

*Step 2:* Create the following files - Jenkinsfile, Ansible Inventory file, Ansible Playbook, DockerFiles for Frontend application, User service, Complaint service, Service registry and API Gateway. Create Docker Compose file for following services - serviceregistry, apigateway, userservice, complaintservice, userservice\_db, complaintservice\_db, citizenconnectfrontend, elasticsearch, kibana and logstash.

*Step 3:* Install Jenkins and Install following plugins : Docker, Ansible, Git and Pipeline.

*Step 4:* Setting Up Jenkins Pipeline Job

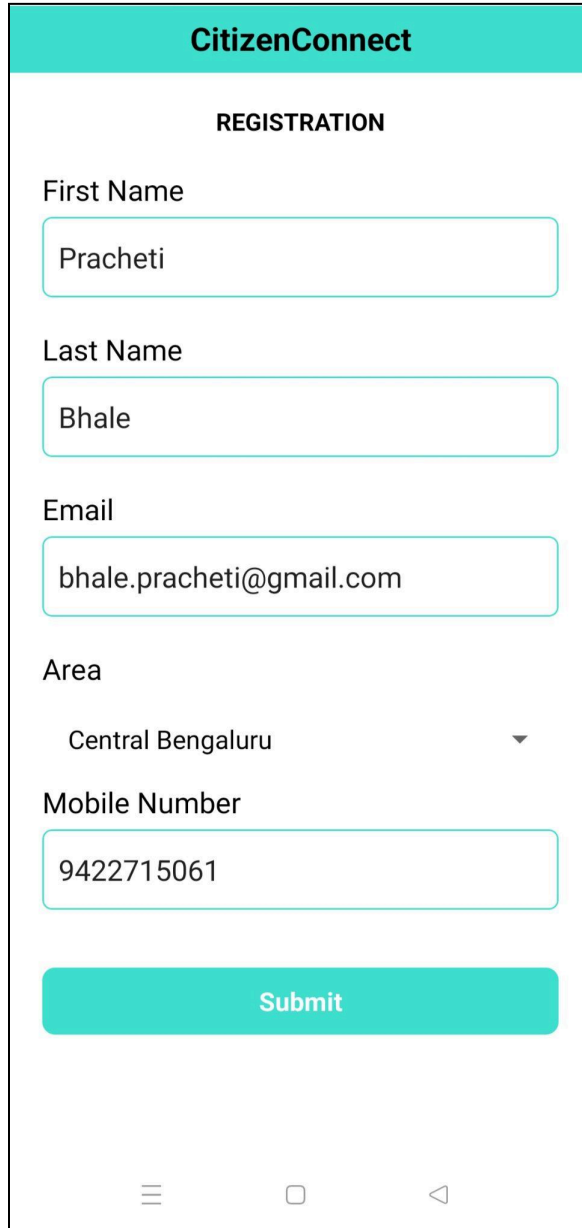
1. Create a New Pipeline Job: Go to Jenkins, create a new item, and select "Pipeline".
2. Configure Pipeline:
  - Under the "Pipeline" section, set "Definition" to "Pipeline script from SCM".
  - Select "Git" and provide the repository URL where your Jenkinsfile is stored.
  - Specify the script path if it's not in the root of the repository.
3. Add Credentials:
  - Add Docker Hub credentials in Jenkins under "Credentials" with the ID docker-hub-credentials-id.
  - Change credentials in inventory file according to you credentials.
4. Save and Build: Save the configuration and trigger a build. Jenkins will now pull the code, build Docker images, push them to Docker Hub, and deploy the services using Ansible.

*Step 5:* We can check on our localhost if all containers are running using command “docker ps -a”. Now check all logs on elastic search. Navigate to “localhost:5601” and verify logs as needed.



## SCREENSHOTS OF THE APPLICATION

### 1. Registration Page



The screenshot displays the registration interface of the CitizenConnect application. At the top, a teal header bar contains the text "CitizenConnect". Below this, the word "REGISTRATION" is centered. The form consists of several input fields: "First Name" with the value "Pracheti", "Last Name" with the value "Bhale", "Email" with the value "bhale.pracheti@gmail.com", and "Area" with a dropdown menu currently showing "Central Bengaluru". Below these is a "Mobile Number" field containing "9422715061". A large teal "Submit" button is positioned at the bottom of the form. The bottom of the screen shows three standard Android navigation icons: a hamburger menu, a home button, and a back arrow.

**CitizenConnect**

**REGISTRATION**

First Name  
Pracheti

Last Name  
Bhale

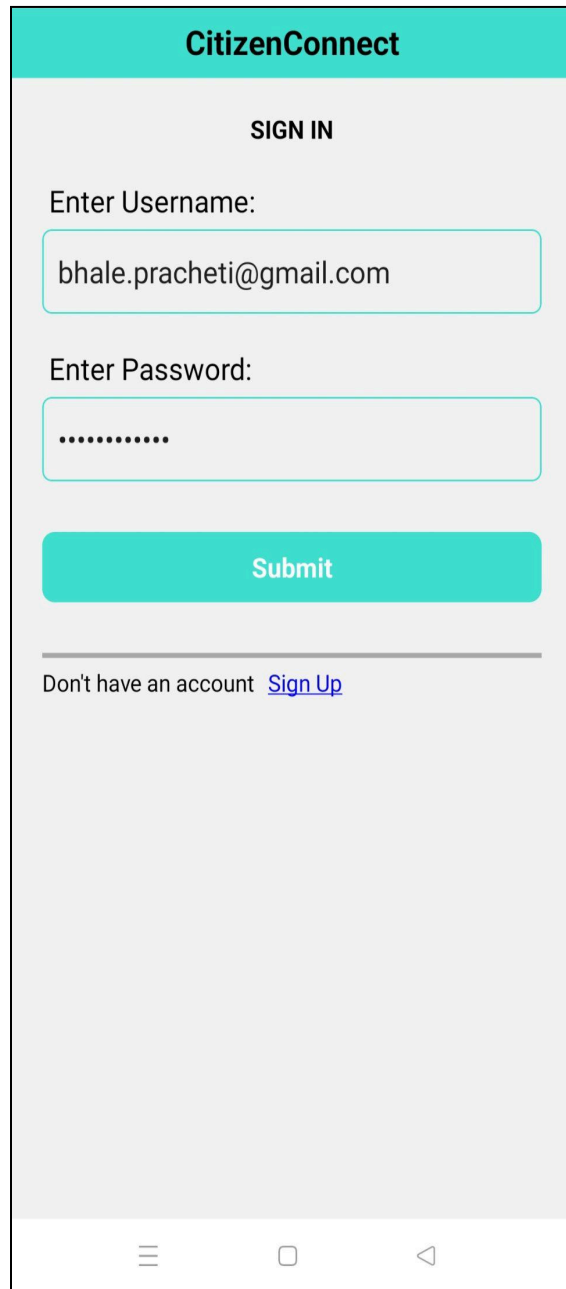
Email  
bhale.pracheti@gmail.com

Area  
Central Bengaluru ▼

Mobile Number  
9422715061

**Submit**

## 2. Sign In Page



The image is a mobile app mockup for the 'CitizenConnect' sign-in page. It features a teal header with the app name, a light gray main area with form fields, and a white bottom bar with Android navigation icons. The form includes labels for 'Enter Username:' and 'Enter Password:', a text input field containing an email address, a password input field with masked characters, a teal 'Submit' button, and a link to 'Sign Up' for new users.

**CitizenConnect**

**SIGN IN**

Enter Username:

bhale.pracheti@gmail.com

Enter Password:

.....

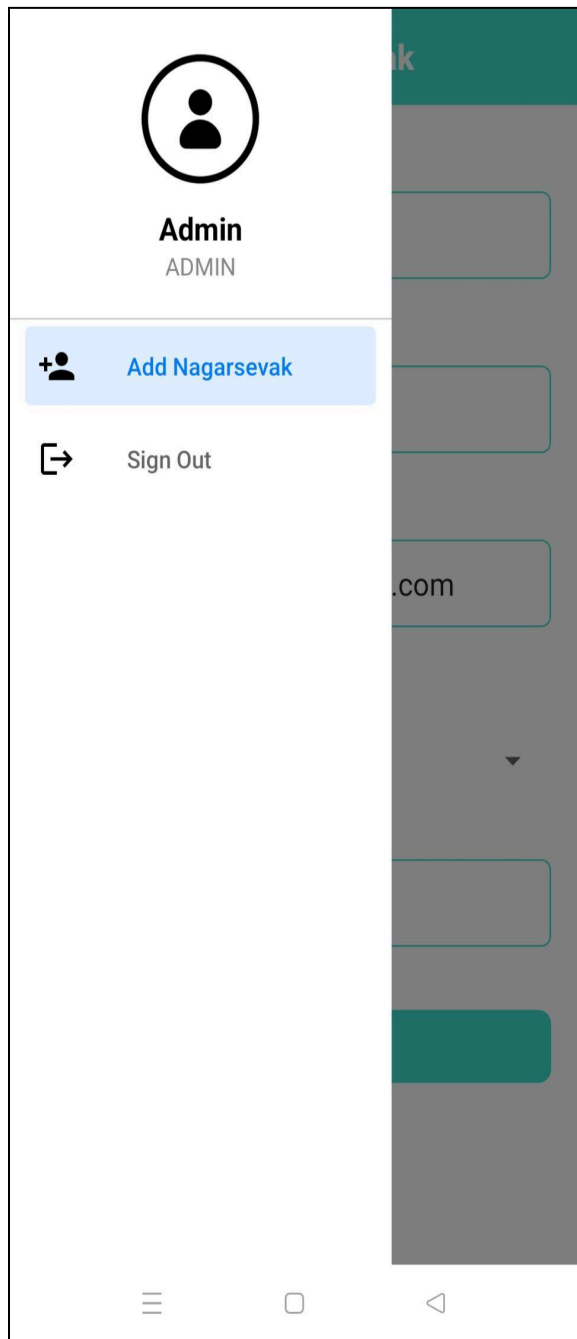
**Submit**

---


Don't have an account [Sign Up](#)

### 3. Admin Pages

#### a. Navigation View



*b. Admin's dashboard*

**Add Nagarsevak**

First Name

Last Name




Email

Area

Central Bengaluru ▼

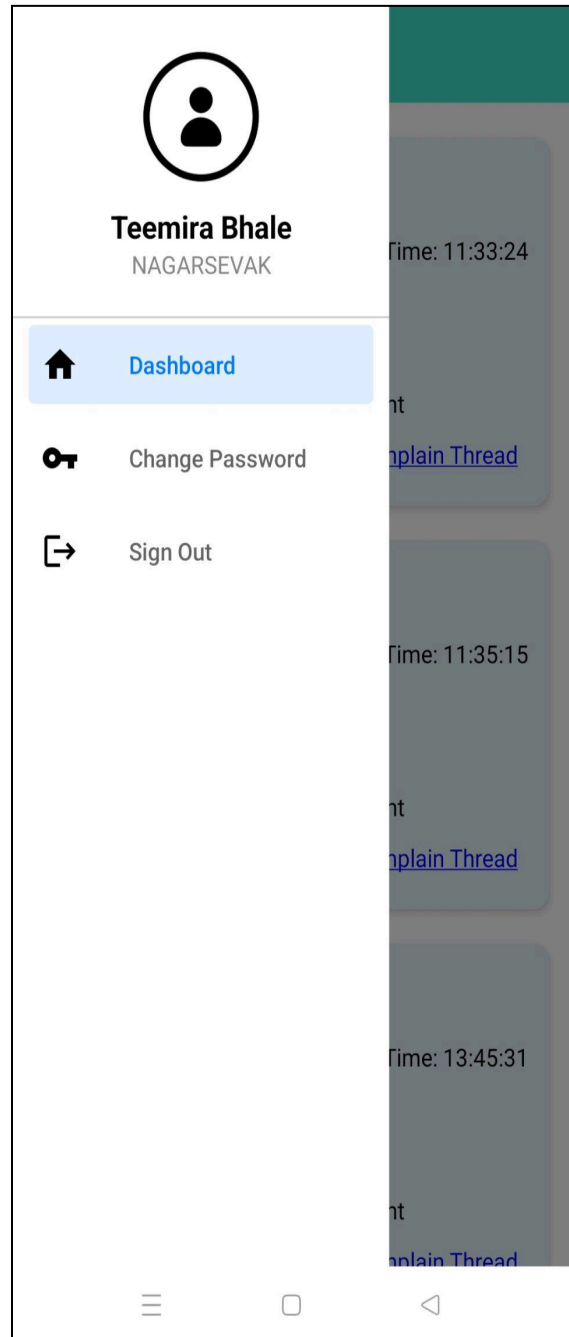
Mobile Number

**Register**

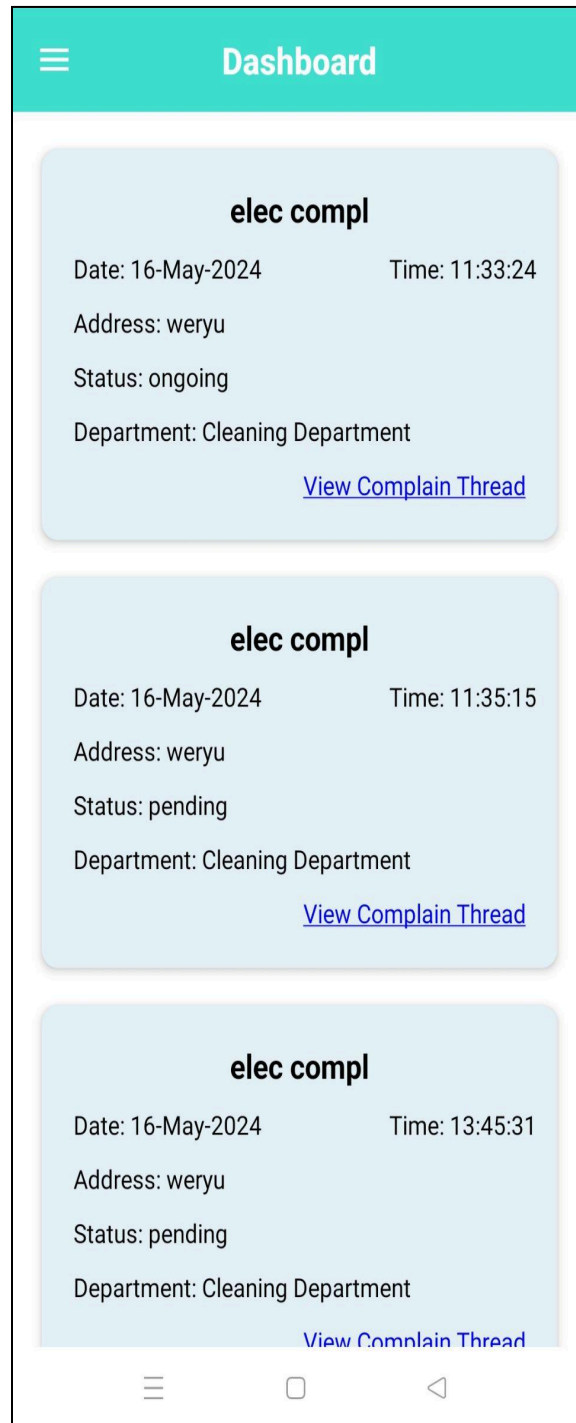


#### 4. Nagarsevak Pages

##### a. Navigation View

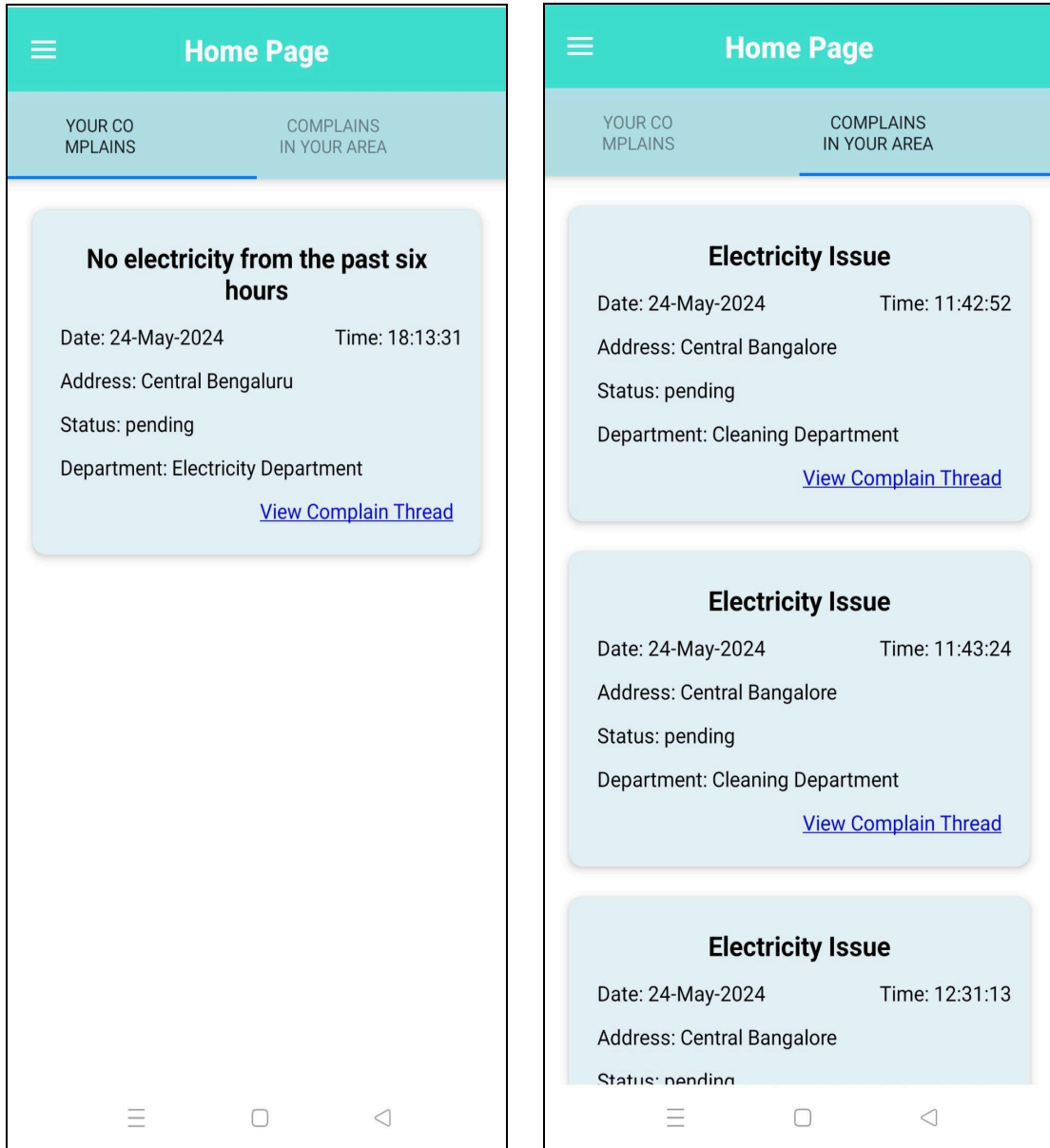


*b. Nagarsevak Dashboard*




## 5. Citizen Pages

### a. Navigation View



*b. Register Complain*

 **Register Complain**

Central Bengaluru

▼

Electricity Department


▼


Central Bengaluru


No electricity from the past six hours

There has been no electricity from the

Submit

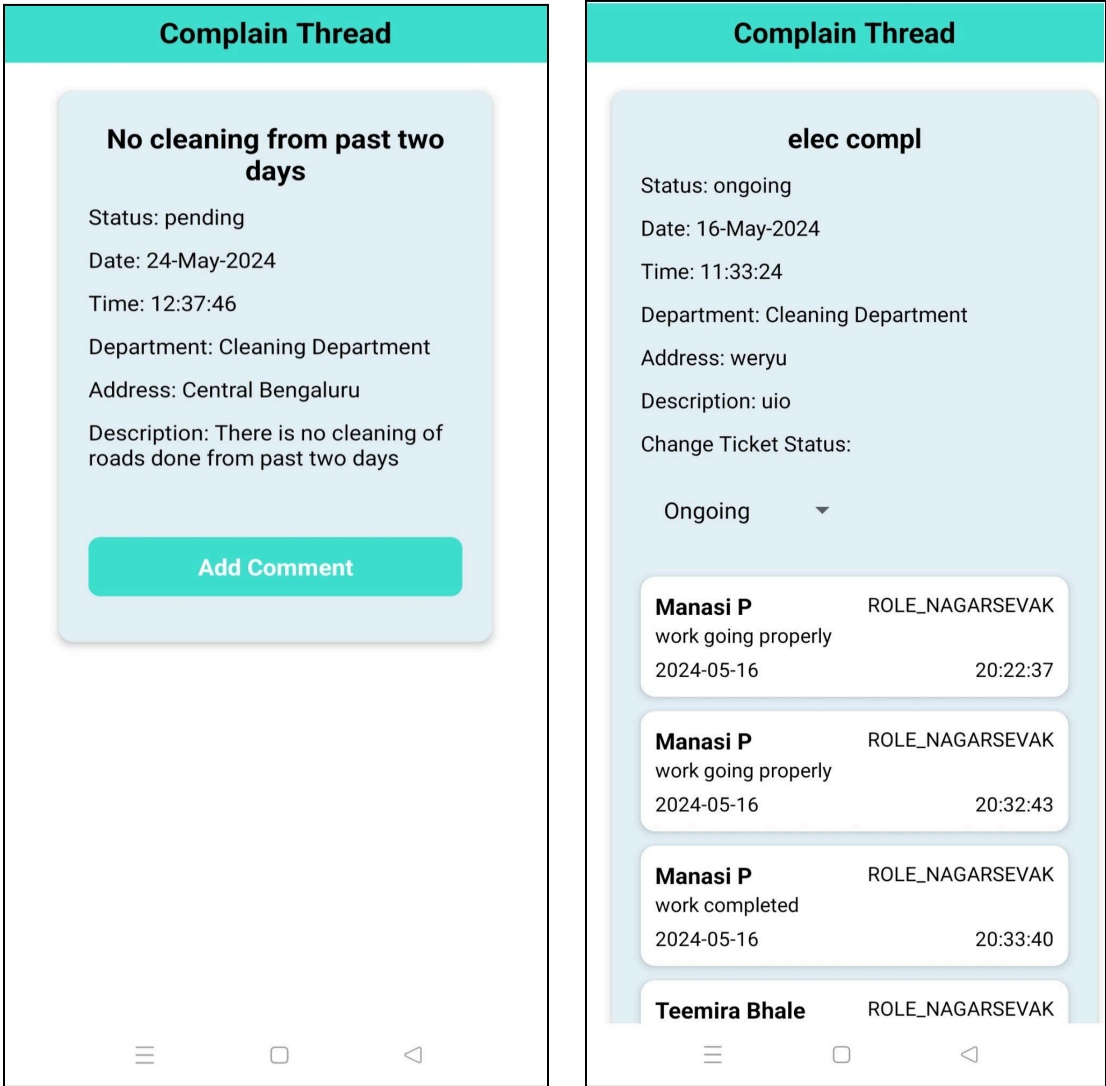








6. View Complain Thread



## 7. Add Comment

### Complain Thread

Department: Cleaning Department

Address: weryu

Description: uio

Change Ticket Status:

Ongoing ▼

#### Enter the comment

Please resume the work soon

OK

work completed

2024-05-16 20:33:40

**Teemira Bhale** ROLE\_NAGARSEVAK

Still issue exists

2024-05-24 14:45:14

Add Comment

## 8. Change Password

Change Password

Current Password:

.....

New Password:

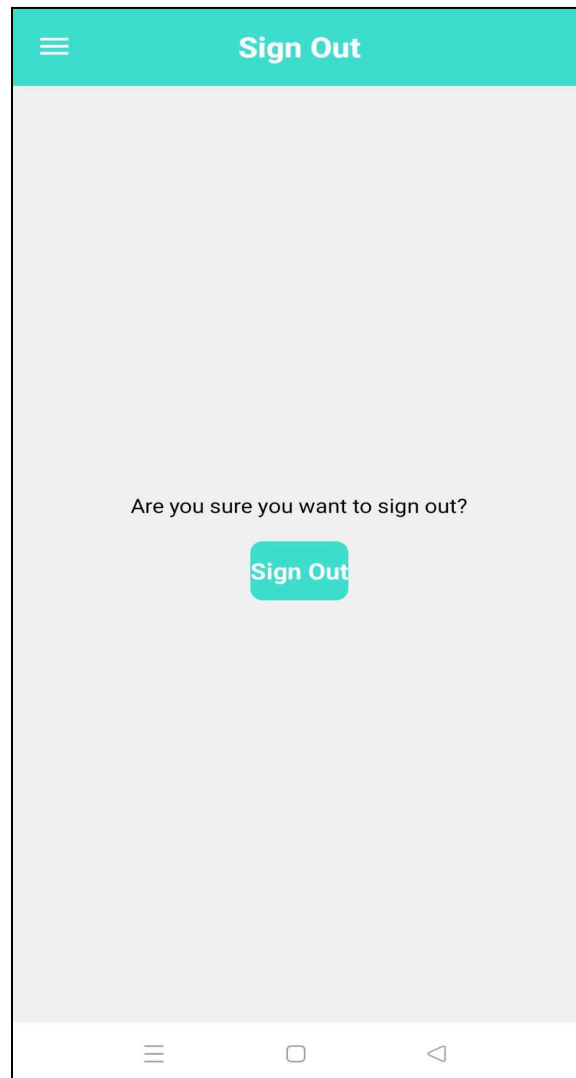
.....

Confirm Password:

.....

Change Password

## 9. Sign Out



## LIST OF ALL API's

Link : <https://documenter.getpostman.com/view/31255878/2sA3JRaKQp>

## **CONCLUSION AND FUTURE WORK**

The CitizenConnect project successfully implemented a robust and scalable platform for managing citizen complaints through a well-architected microservice backend using Spring Boot and a responsive mobile application developed with React Native. By leveraging a suite of DevOps tools such as Jenkins for CI/CD, Docker and Docker Compose for containerization, Ansible for configuration management, and the ELK stack for centralized logging, the project ensured efficient deployment, monitoring, and maintenance processes. Security was a key focus, with JWT (JSON Web Tokens) used for secure user authentication and authorization, ensuring that user data and interactions are protected. Additionally, the Eureka service registry enabled service discovery. This foundation not only addressed current needs but also positioned the platform for future enhancements. Future work will focus on scaling with Kubernetes, enhancing security, expanding features, improving user experience, and fostering community engagement, ensuring CitizenConnect remains a valuable tool for citizens and city officials.