Project Report: Configuration Management Microservice

1. Project Overview

This project involves developing a Configuration Management Microservice that manages network device configurations, including backup, restore, and change management. The key functionalities include:

1. **Configuration Backup and Restore:** Automating the backup and restore of device configurations.
2. **Secure Configuration Push:** Ensuring secure transmission of configurations to network devices.
3. **Change Tracking:** Monitoring and tracking configuration changes.
4. **User Interface:** A simple and intuitive UI for managing configurations.

2. Environment Setup

For the development of this project, the following tools and environments were set up:

1. **Java Development Kit (JDK):**
   - **Version:** JDK 21
   - **Download Link:** JDK 21
2. **Apache Maven:**
   - **Version:** Maven 3.9
   - **Download Link:** Apache Maven 3.9
3. **MySQL:**
   - **Version:** MySQL 8 CE (Community Edition)
   - **Download Link:** MySQL 8 CE
4. **Integrated Development Environment (IDE):**
   - **Version:** Spring Tool Suite IDE 4.0
   - **Download Link:** Spring Tool Suite 4.0

3. Directory Structure

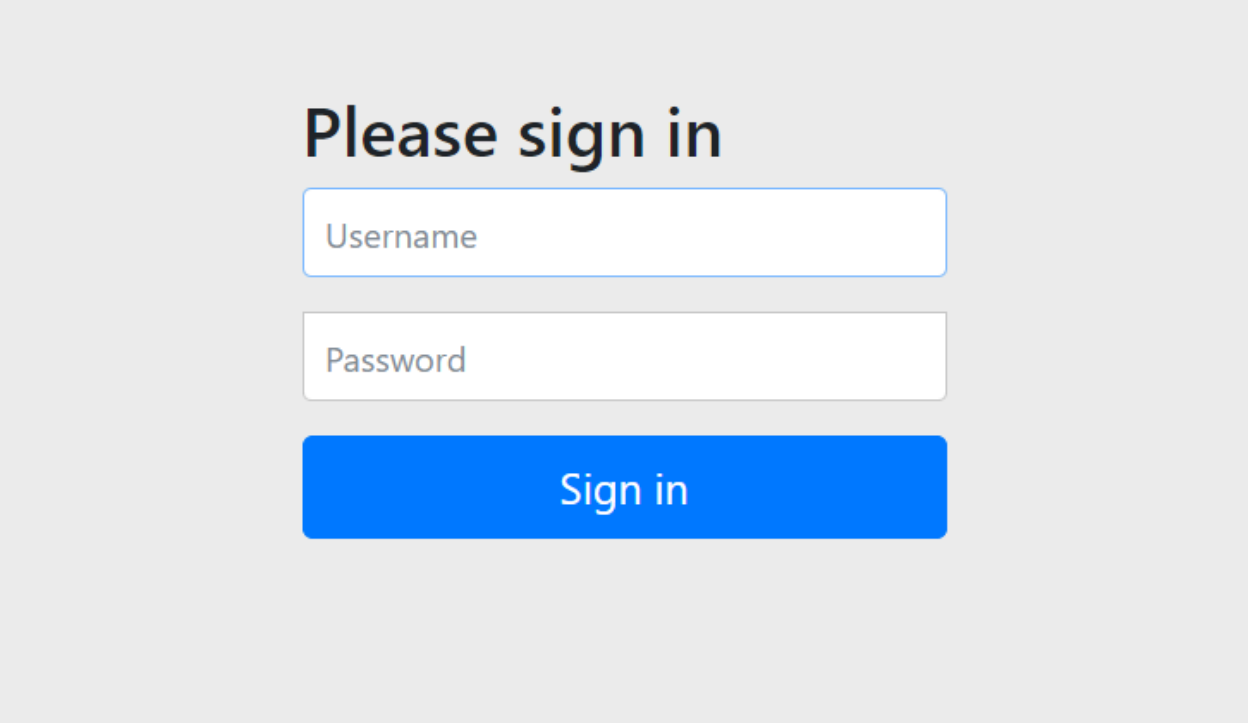
The project directory structure is organized as follows:

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── example/
│   │           └── nms/
│   │               ├── ConfigurationManagementServiceApplication.java
│   │               ├── config/
│   │               │   ├── ConfigBackup.java
│   │               │   ├── ConfigChangeTracker.java
│   │               │   ├── ConfigPush.java
│   │               │   └── DeviceService.java
│   │               ├── model/
│   │               │   ├── Device.java
│   │               │   └── ... other models ...
│   │               ├── repository/
│   │               │   ├── DeviceRepository.java
│   │               │   └── ... other repositories ...
│   │               ├── service/
│   │               │   ├── DeviceService.java
│   │               │   └── ... other services ...
│   │               ├── controller/
│   │               │   ├── ConfigController.java
│   │               │   └── ... other controllers ...
│   │               ├── util/
│   │               │   ├── ... utility classes ...
│   │               └── exception/
│   │                   └── ... custom exceptions ...
│   ├── resources/
│   │   ├── application.properties
│   │   ├── messages.properties
│   │   └── templates/
│   │       ├── dashboard.html
│   │       ├── devices.html
│   │       ├── givefilename.html
│   │       └── submitcommands.html
│   └── test/
│       └── java/
│           └── com/
│               └── example/
│                   └── nms/
│                       └── ... test classes ...
```

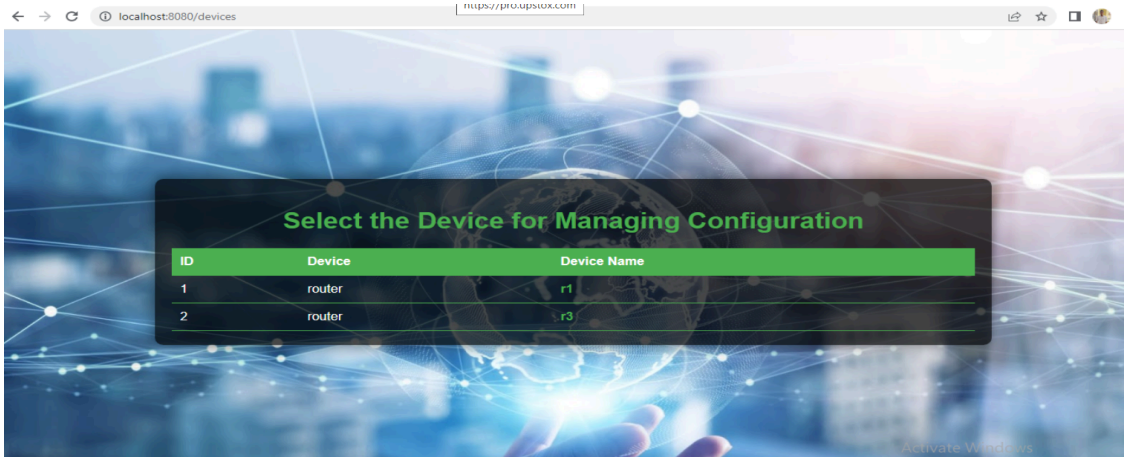
3.2 Project Flow

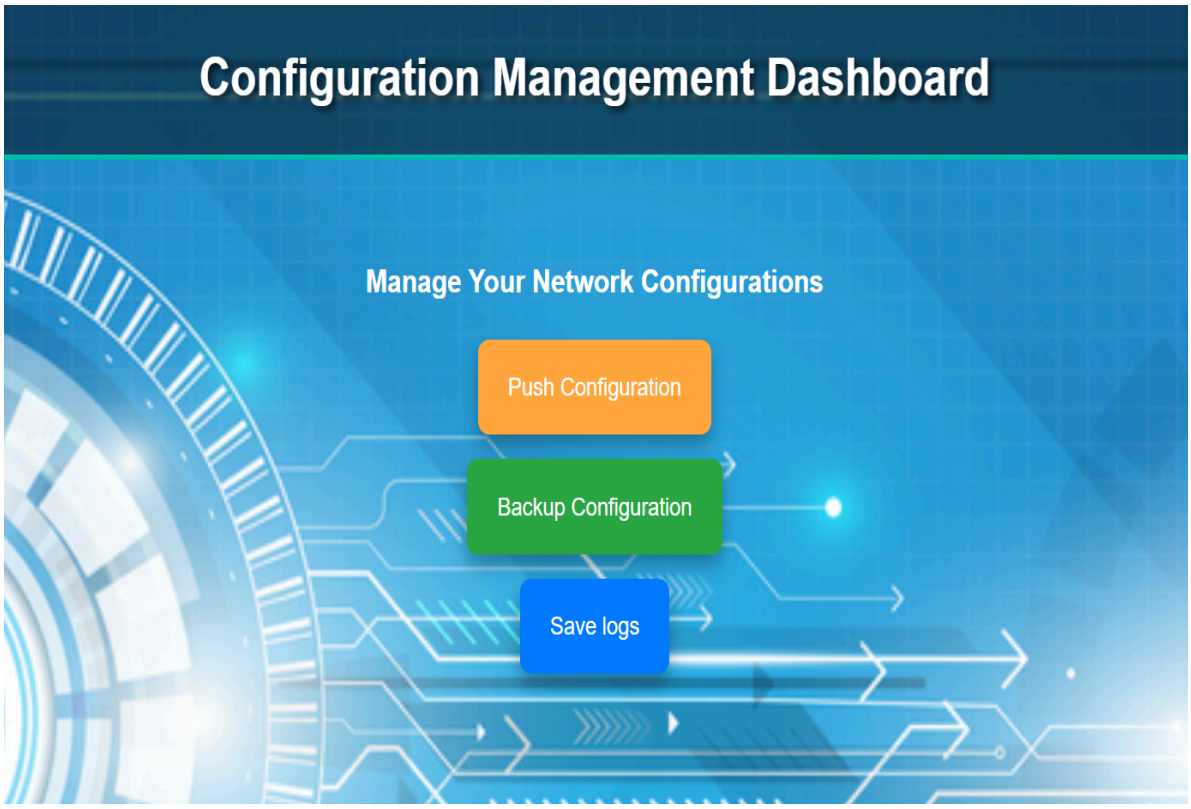
The project flow is as follows:

1. Accessing the URL redirects to the login page provided by Spring Boot Security.
2. After entering the username and password specified in the application file, the user is redirected to the dashboard page.

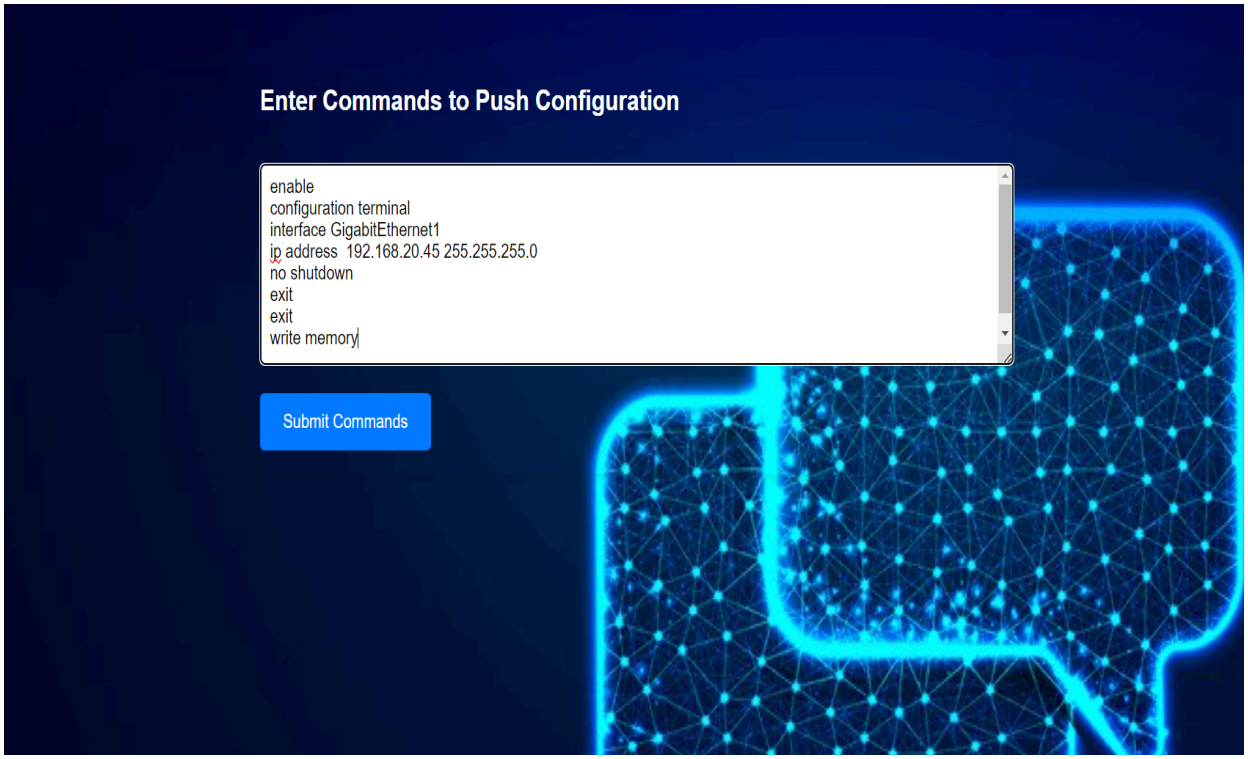


3. The dashboard page lists devices with their names, IP addresses, and ports.

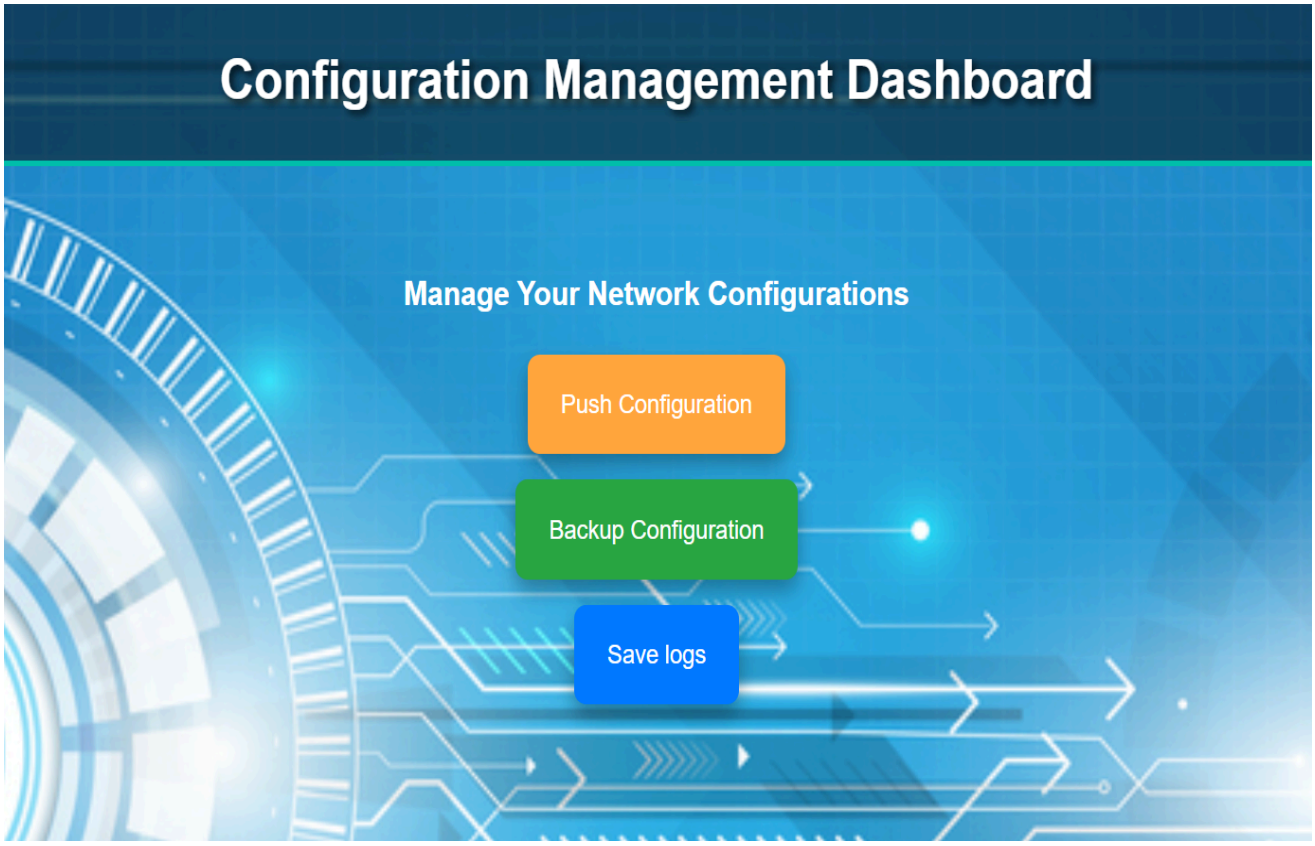




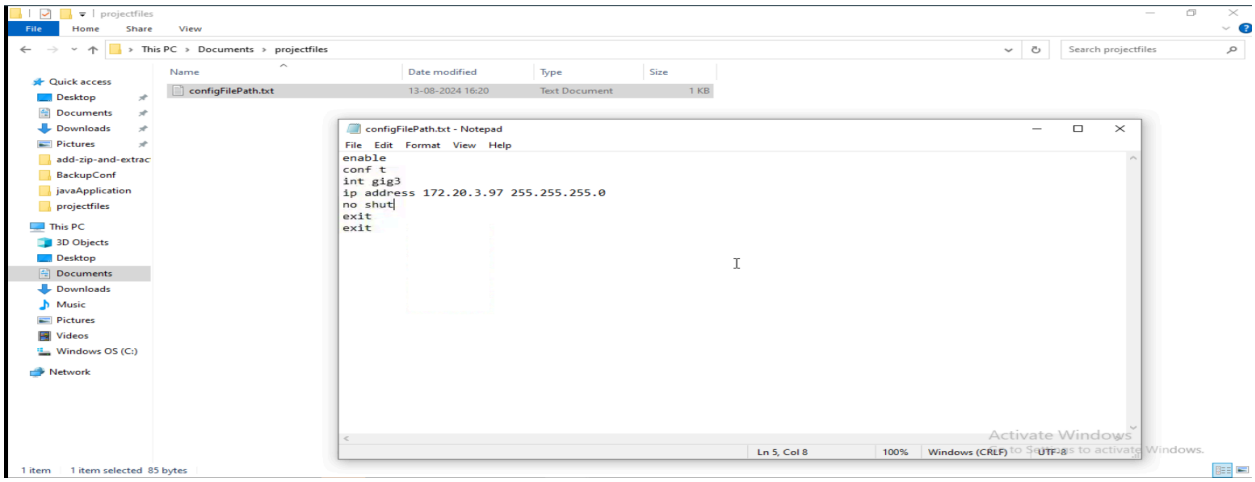
4. After selecting a device from the dashboard, the user is redirected to a new page to input commands for router configuration.



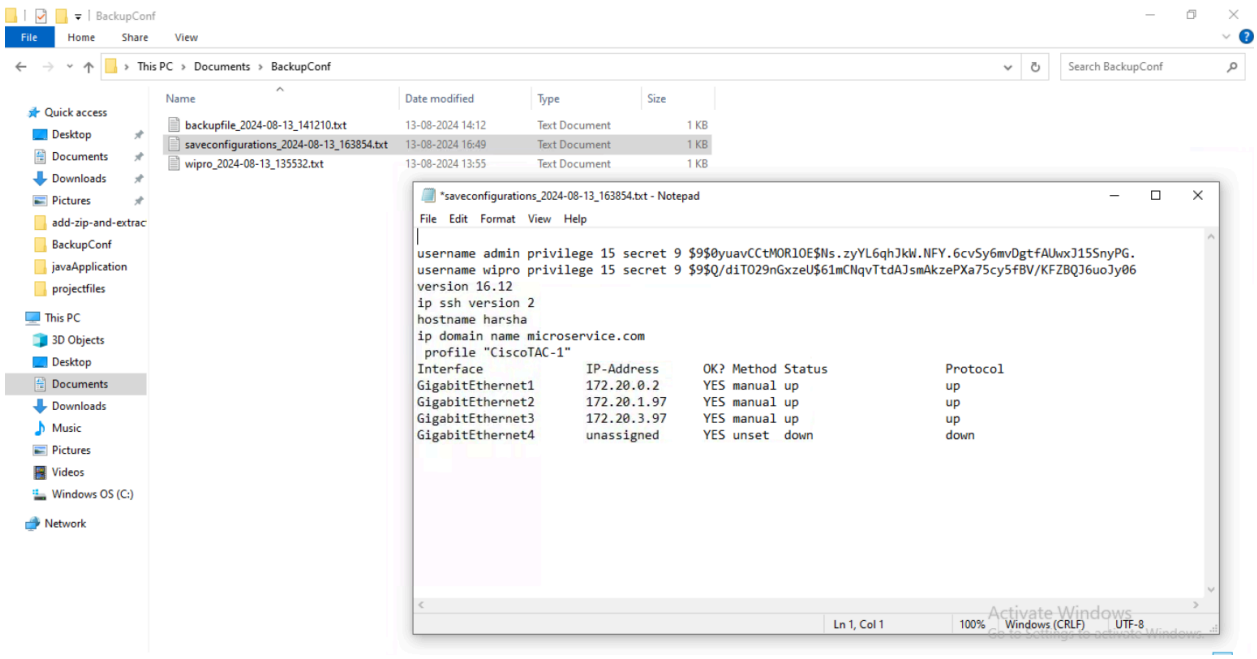
5. After clicking the 'Send' button, the user is redirected to a new page with three buttons: 'PushConfig,' 'BackupConfig,' and 'SaveLogs.'



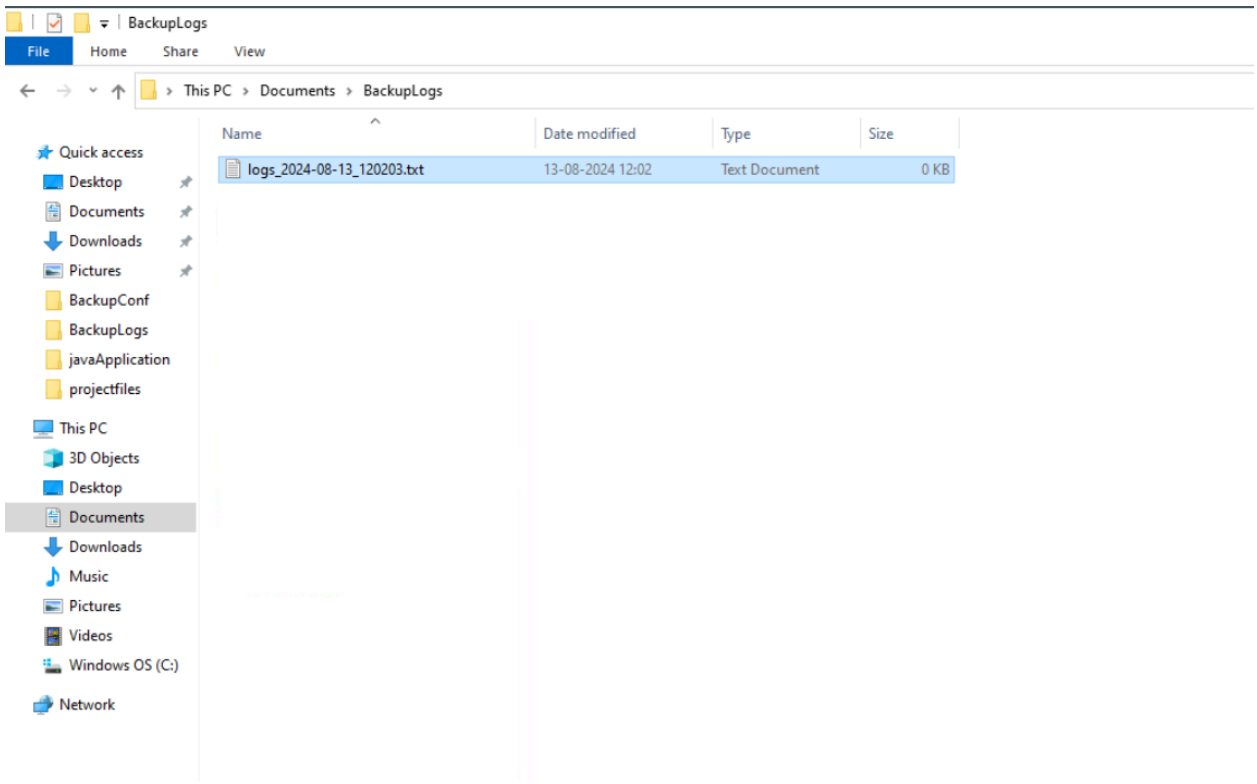
6. Clicking 'PushConfig' sends the commands to the router, executing them in the background while staying on the same page. The command we can give through the file which is useful for bulk input with a single push.



7. After pushing the config, clicking 'BackupConfig' saves the output of the config changes in a file.



8. 'SaveLogs' shows all config logs and exceptions raised since the project started, saved in a separate file.



4. Key Features and Functionality

4.1 Command Execution via SSH

This feature allows users to execute commands on a router via SSH. The commands are read from a text file located in the `commands` directory. The project uses the **JSch** library for SSH communication.

4.2 Backup and Restore

The microservice supports backing up the current running configuration of a network device and restoring it when needed. This feature ensures that device configurations can be restored in case of a failure.

4.3 Logging and Monitoring

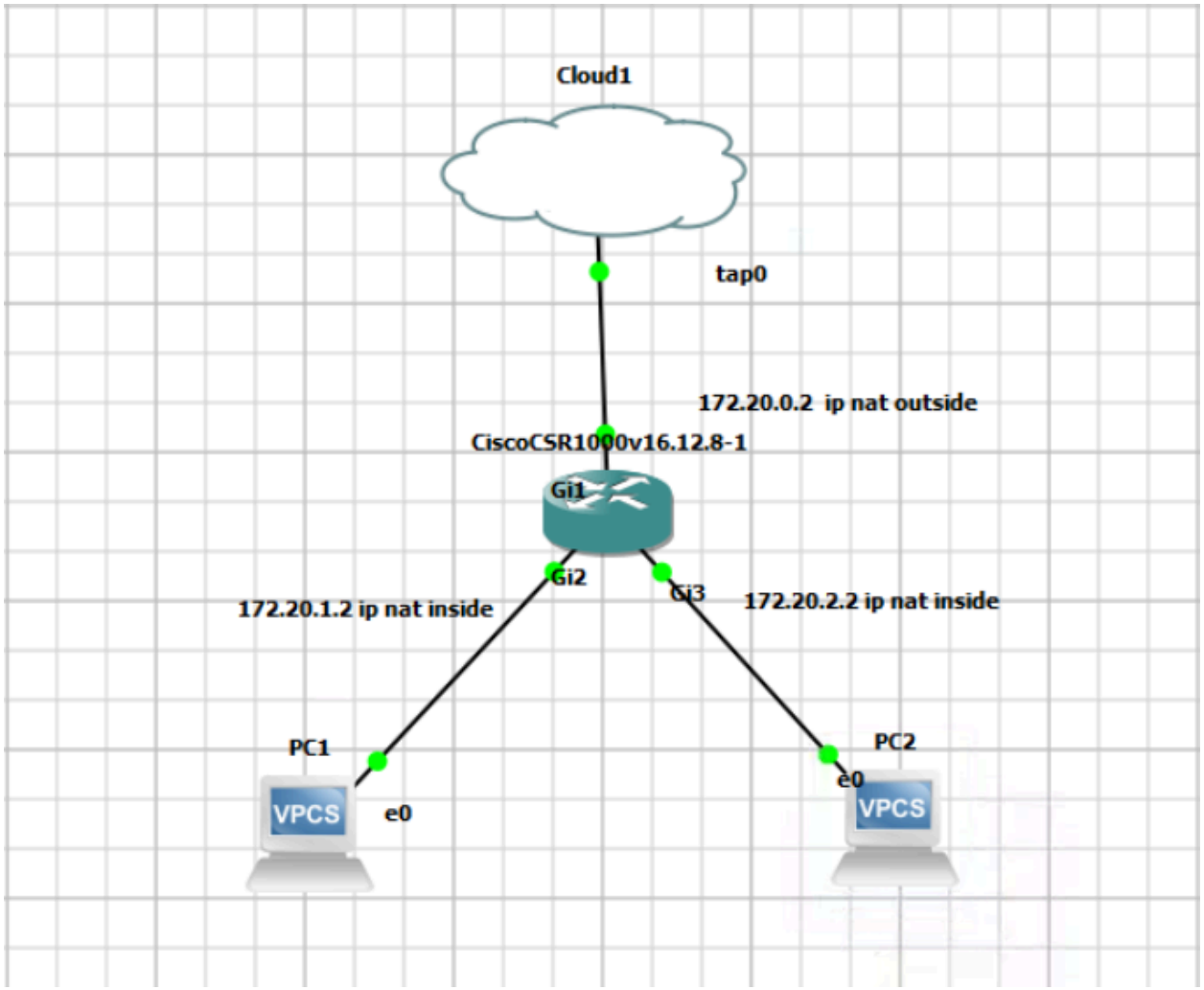
All configuration changes and actions performed by the microservice are logged with timestamps. Logs are stored in the `logs` directory, and any exceptions encountered are also logged for troubleshooting purposes.

5. Technology Stack

- **Backend:** Spring Boot
- **SSH Communication:** JSch library
- **Database:** MySQL 8 CE
- **Build Tool:** Maven 3.9
- **Java Version:** JDK 21
- **IDE:** Spring Tool Suite IDE 4.0

Network Configuration On GNS3

To enable network-enabled NAT on the router, the following configuration was applied:



Router Configuration for NAT

1. Enter Global Configuration Mode:

```
enable

configure terminal
```

2. Interface Configuration:

Interface GigabitEhternate1

Ip address 172.20.0.2 255.255.255.0

```
ip nat outside
```

```
No shutdown
```

```
Exit
```

Interface GigabitEhternate2

Ip address 172.20.1.2 255.255.255.0

```
ip nat inside
```

```
No shutdown
```

```
exit
```

3. Access Control List (ACL) Configuration:

```
access-list 1 permit 172.20.1.2 0.0.0.255
```

4. Configure NAT:

```
ip nat inside source list 1 interface GigabitEthernate1 overload
```

5. Exit Configuration Mode:

```
end
```

Router Configuration for SSH Access

1. Set Hostname and Domain Name

```
Router(config)# hostname CiscoCSR1000v
```

```
CiscoCSR1000v(config)# ip domain-name example.com
```

- Sets the router's hostname and domain name, necessary for SSH key generation.

2. Generate RSA Keys

```
CiscoCSR1000v(config)# crypto key generate rsa
```

- Generates RSA keys for SSH encryption. Choose a key modulus size (e.g., 2048 bits) for stronger encryption.

3. Create Local User Account

```
CiscoCSR1000v(config)# username admin privilege 15 secret
YourPassword
```

- Creates a user account with administrative privileges and an encrypted password.

4. Configure VTY Lines for SSH Access

```
CiscoCSR1000v(config)# line vty 0 4

CiscoCSR1000v(config-line)# transport input ssh

CiscoCSR1000v(config-line)# login local

CiscoCSR1000v(config-line)# exit
```

- Configures VTY lines for SSH access, allowing up to 5 simultaneous connections using local user credentials.

5. Enable SSH Version 2

```
CiscoCSR1000v(config)# ip ssh version 2
```

- Forces the router to use SSH version 2 for improved security.

6. Save the Configuration

```
CiscoCSR1000v# write memory
```

- Saves the current configuration to ensure persistence across reboots.

7. Verify the SSH Configuration

```
CiscoCSR1000v# show ip ssh
```

- Displays SSH configuration details, confirming correct setup.

8. Test SSH Access

```
ssh admin@172.20.0.2
```

- Initiates an SSH connection to the router. Replace with your username and router's IP address.

6. Conclusion

This configuration ensures secure SSH access for remote management and interaction with your Java application.

The Configuration Management Microservice provides a robust solution for managing network device configurations with secure SSH communication, change tracking, and a user-friendly interface. The project leverages modern tools and technologies to ensure scalability, reliability, and ease of use.