# User Manual: Server

## 1. Introduction

This server is designed for JWT-based authentication and role-based access authorization. It supports various user roles, including user, supervisor, and admin, with specific functionalities for each role.

## 2. Setting Up HTTPS

This server can be configured to run over HTTPS for secure communication. Below are the steps to set up HTTPS with self-signed certificates:

Prerequisites:1. OpenSSL installed on your machine (for generating self-signed certificates).2. Node.js installed.

1. Generate Self-Signed Certificates:

- Run the following command in your terminal to generate a self-signed certificate and key:

“bash openssl req -nodes -new -x509 -keyout server.key -out server.cert”

- You will be prompted to fill in some information. Use "localhost" as the Common Name (CN) for local development.

2. Place the Certificates in the Project:

- Move the generated `server.key` and `server.cert` files to a secure directory within your project (`certs/`).

## 3. How to Turn On the Server

**Prerequisites:**

1. Node.js installed on your machine.
2. MongoDB running either locally or remotely.
3. Create a `.env` file in the root directory with the following content:

SECRET=your\_jwt\_secret\_key  
MONGO\_URI=your\_mongodb\_connection\_string

**Steps:**

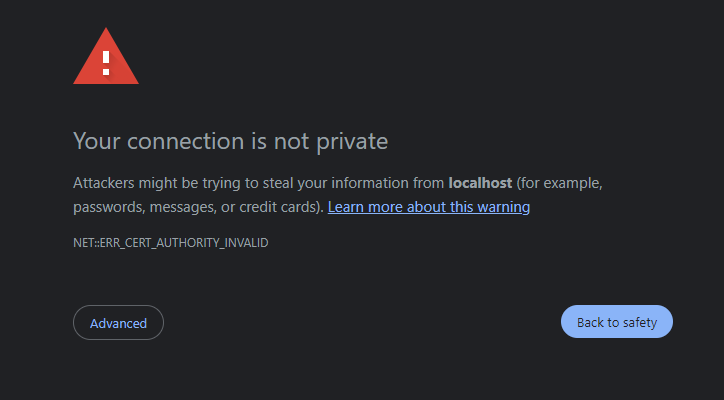
1. Install dependencies:  
   npm install
2. Start the server:

npm run dev

This will start the server using `nodemon` for automatic restarts on code changes.  
The server will be listening on `https://localhost:8443`.

**Accessing the Server:**

When you access the server through a web browser, you may encounter a security warning like this one:



This is because the browser does not automatically trust self-signed SSL certificates (which are used for localhost).

You have two options to proceed:

1. **Add the certificate to your system's trusted certificate store:**
   * On **Windows**:
     + Open certmgr.msc.
     + Navigate to "Trusted Root Certification Authorities."
     + Import the self-signed certificate (server.cert) into this store.
   * On **macOS**:
     + Open "Keychain Access."
     + Drag the certificate (server.cert) into the "System" keychain under "Certificates."
     + Set the certificate to "Always Trust."
   * On **Linux**:
     + Copy your certificate to /usr/local/share/ca-certificates/.
     + Update the certificate store with: sudo update-ca-certificates
2. **Bypass the warning in your browser:**
   * Click on "Advanced."
   * Select "Proceed to localhost (unsafe)."

This allows you to continue using HTTPS for local development while acknowledging the security warning.

## 4. How to Use

**Available Endpoints:**

**Sign-up:**

**URL:** */api/signUp* **Method:** POST **Description:** Registers a new user. **Body Parameters:**

{

'email': 'user@example.com',   
'password': 'Password123'

}

**Sign-in:**

**URL:** */api/signIn* **Method:** POST **Description:** Authenticates a user and provides a JWT token. **Body Parameters:**

{

'email': 'user@example.com',  
 'password': 'Password123'

}

**Get Homepage:**

**URL:** */api/homeBoard* **Method:** GET **Description:** Returns home page content.

**Get User Homepage:**

**URL:** */api/userBoard* **Method:** GET **Description:** Returns user board content. **Headers:**

{

'x-access-token': '<JWT\_TOKEN>'

}

**Get Supervisor Homepage:**

**URL:** */api/supervisorBoard* **Method:** GET **Description:** Returns supervisor board content. **Headers:**

{

'x-access-token': '<JWT\_TOKEN>'

}

**Get Admin Homepage:**

**URL:** */api/adminBoard* **Method:** GET **Description:** Returns admin board content. **Headers:**

{

'x-access-token': '<JWT\_TOKEN>'

}

**Add Supervisor:**

**URL:** */api/addSupervisor* **Method:** PUT **Description:** Adds supervisor role to a user. **Headers:**

{

'x-access-token': '<JWT\_TOKEN>'

}

**Body Parameters:**

{

'email': 'user@example.com'

}

**Remove Supervisor:**

**URL:** */api/removeSupervisor* **Method:** PUT **Description:** Removes supervisor role from a user. **Headers:**

{

'x-access-token': '<JWT\_TOKEN>'

}

**Body Parameters:**

{

'email': 'user@example.com'

}

## 5. Functionalities Overview

1. User Registration and Authentication: Allows users to sign up and sign in using their email and password.
2. Passwords are hashed using bcrypt for security.
3. Role-Based Access: Provides different access levels for users, supervisors, and admins.
4. JWT Authentication: Uses JSON Web Tokens for session management and secure API access.
5. Middleware for Validation: Includes middleware functions for validating passwords, checking if an email is already used, and verifying JWT tokens.
6. Admin Capabilities: Admins can promote users to supervisors and demote supervisors to regular users.
7. Rate Limiting: Limits the number of requests from a single IP to prevent abuse.
8. Logging: Uses `morgan` to log HTTP requests to a file for monitoring and debugging.
9. HTTPS Support: The server is configured to run over HTTPS with self-signed certificates for secure communication, ensuring encrypted data transfer.