**Smart Attendance System - Project SOP (Standard Operating Procedure)**

**1. Introduction**

The **Smart Attendance System** leverages a combination of technologies for real-time attendance management. The system consists of a frontend web application, backend server, Redis for caching, and MongoDB for database storage. The backend server is developed using **Node.js**, and Redis is used for caching the attendance data. The project involves deploying the server on **Windows Subsystem for Linux (WSL)** to ensure smooth integration and a seamless development environment.

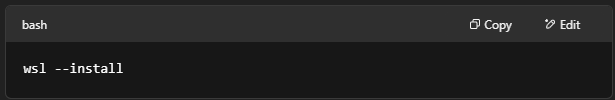
**2. WSL Selection & Setup**

**Why WSL?**

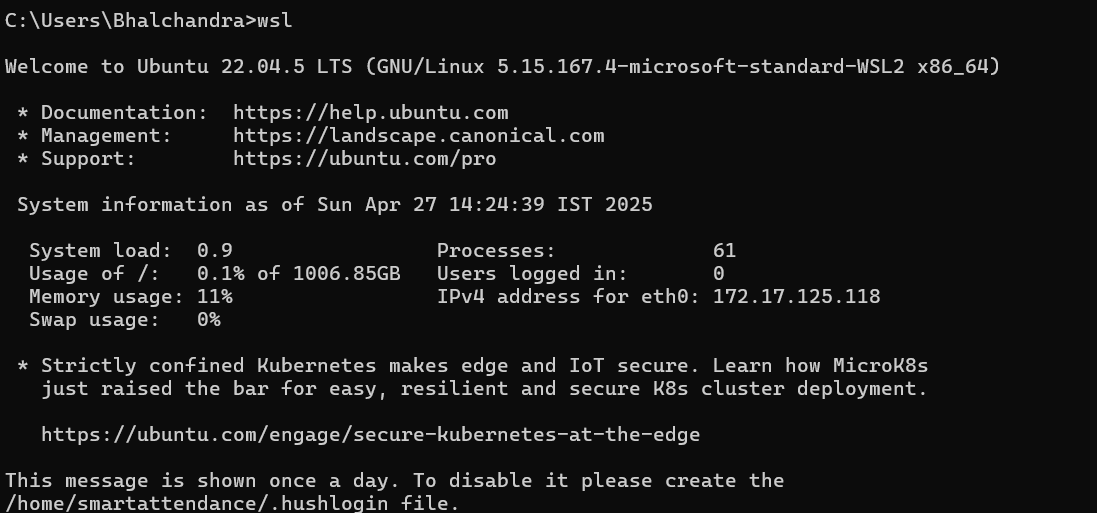
Initially, the development environment was set up on a Windows machine. However, during the development process, it became clear that Linux-based environments provide better compatibility with tools like Redis, MongoDB, and Node.js. Therefore, **Windows Subsystem for Linux (WSL)** was chosen as it provides a lightweight virtualized Linux environment that integrates seamlessly with Windows.

**Setting Up WSL:**

1. **Install WSL**: To set up WSL on Windows, I used the command:



This installed Ubuntu as the default WSL distribution on my machine.



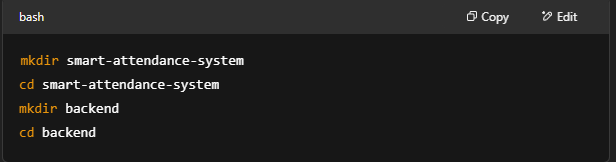
1. **Set up the necessary packages**: After installing WSL, I installed all required dependencies such as **Node.js**, **Redis**, and **MongoDB** (through their respective Linux packages).
2. **Access the WSL terminal**: I accessed WSL via the terminal, where I installed the tools required for my backend development.

**3. Backend Server Setup**

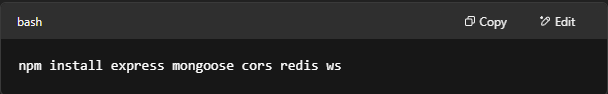
**Creating the Server:**

The backend server was created using **Node.js** and **Express**. Here’s the process for setting up the server:

1. **Set up the backend folder**: I created a directory for the backend server:



1. **Initializing Node.js Project**: I ran npm init -y to initialize the project and created a server.js file for the backend logic.
2. **Install necessary packages**: I installed all the necessary packages:



1. **MongoDB Setup**: I created a MongoDB database and configured a connection using **Mongoose**.
2. **Redis Setup**: Redis was set up as the cache layer for optimized access to attendance data. Redis helps to reduce repeated database calls by caching frequently accessed data.

**4. Username & Password Setup**

**Security Measures:**

For authentication and security, I decided to keep sensitive data such as usernames, passwords, and MongoDB credentials secure. Here's what I did:

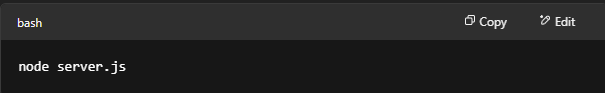
* I used **dotenv** for storing sensitive credentials in a .env file. This ensures credentials are not hardcoded directly into the codebase.

**5. Old Process for Running the Backend Server**

**Initial Setup (Before NVM & WSL):**

Earlier, I did not use **WSL**. My backend was directly running on **Windows**. The process for running the backend was as follows:

1. Open a terminal in Windows.
2. Navigate to the backend folder.
3. Run redis-server to start Redis.
4. Run the command to start the backend server:



1. This process was **manual** and required running both Redis and the Node.js server each time the laptop was restarted.

**6. Choosing NVM & Node.js Installation**

**Why I Chose NVM?**

As part of my effort to streamline the environment and reduce manual steps, I decided to use **Node Version Manager (NVM)** for managing Node.js versions. The reason for this decision was:

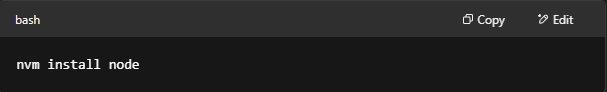
* **Version Control**: NVM allows switching between different versions of Node.js easily. This is useful for managing version compatibility issues between different projects.
* **Reusability**: It enables easy upgrading and downgrading of Node.js versions, without affecting the global installation.

**Process to Install NVM:**

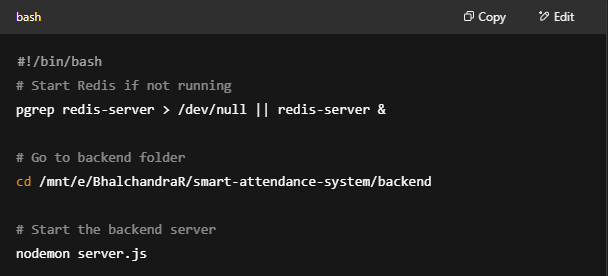
1. Installed **NVM** by running the following command in the WSL terminal:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

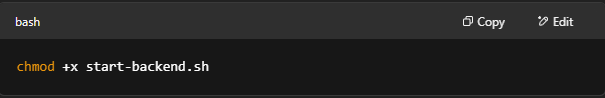
1. Installed the latest version of **Node.js** via NVM:



1. **Setting up a shell script**: I created a shell script start-backend.sh to automate the entire process of starting Redis and the backend server:



1. **Make script executable**:



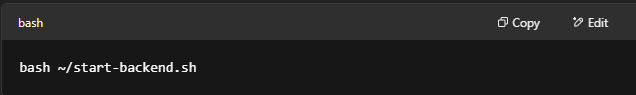
**7. Current Process for Running the Backend**

**Steps for Running the Backend with NVM:**

1. **Start Redis**: Redis is started automatically if not running using the pgrep command.
2. **Run Backend Server**: The backend is started using nodemon, which automatically restarts the server when changes are detected in the codebase.

**Running the Server:**

To start the server, I run the following command from the WSL terminal:



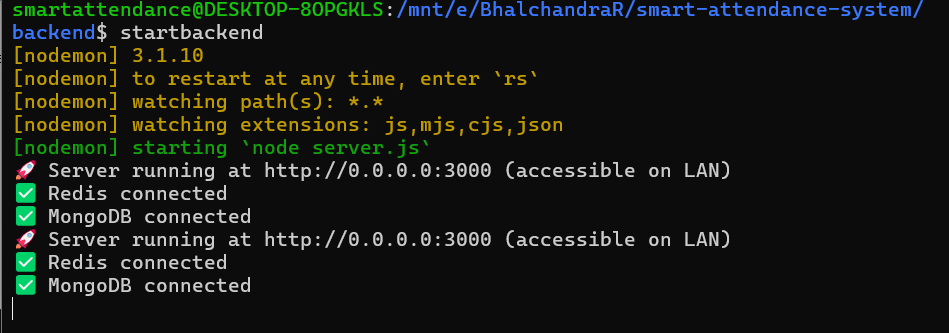
This command starts Redis (if not already running) and then automatically starts the backend server using nodemon.

**8. Conclusion**

With the **WSL-based setup**, **NVM for Node.js version management**, and **Redis caching**, the process of starting and running the backend is **simplified** and more **efficient**. The initial process was manual, but with these changes, I can now easily start my backend by running just one command.

This setup ensures that:

* **Redis** and **MongoDB** run smoothly on the Linux environment.
* The backend is **easily manageable** through nodemon and **automated scripts**.
* Future updates to **Node.js versions** are seamless via **NVM**.

Now, whenever I restart my laptop, I can simply open WSL, type startbackend, and my server is up and running.  
  


**9. Future Considerations**

1. **Backup and Restore**: I will look into creating automated backups for both **MongoDB** and **Redis** data.
2. **Deployment**: The project is ready to be deployed on a cloud server for production use.

**End of SOP**