# Documentation for Setting Up the Socket

### Overview

This documentation provides a comprehensive guide to setting up and running the server and client applications. Follow these instructions to ensure both components are correctly configured and operational.

---

## 1. Server Setup

### Prerequisites

- **Python**: Ensure Python 3.6 or higher is installed on your system.
- **Virtualenv**: Install `virtualenv` (if not already installed**)**:

  ```
  pip install virtualenv
  ```

  **Activate the Virtual Environment**:

  ```
  source venv/bin/activate
  ```

- **Dependencies**: Install the necessary Python packages. You can use `requirements.txt` for automated installation or install the packages manually as needed.

### Configuration

- **Configuration File**: The server's configuration details are specified in the `config.ini` file. You can modify this file as needed to update configuration settings.
- **File Paths**:You can modify this file as needed to update configuration settings but using paths in the docstring(already configured testing paths) will work.Read through the config.ini file thoroughly before starting the server and client and configure the paths correctly.

### Running the Server

1. **Check Port Availability**: The server runs on `localhost` and port `44445`. Ensure that this port is free or update the port number in the server.py and client.py file if necessary.
2. **Run the Server**:

To start the server, navigate to the parent directory containing `server.py` and execute:

```
python3 server.py
```

If you run the server with SSL disabled (`USE_SSL=False`), there is no need to provide a PEM passphrase.

If you enable SSL (`USE_SSL=True`), you will need to provide the configured PEM passphrase.The passphrase is “ **socket** ”

You can exit server by pressing Ctrl + c

## 2. Client Setup

### Running the Client

1. **Prepare the Client**:

   Navigate to the terminal with the parent directory containing the `client.py` file.

   To start the client, execute:

   ```
   python3 client.py
   ```

   The client will connect to the server running on `localhost`. You will need to input the port number used by the server, which is displayed in the server's TCP output.By default,it’s set to port 44445

2. **Searching Strings**:

   The client will allow you to search for strings in the `200k.txt` file. Enter the string you want to search for, and the client will display the results.

---

By following these instructions, you will have both the server and client applications properly set up and running, ready for testing and operation. If you encounter any issues, ensure all prerequisites are met and verify configuration settings as needed.

# Project Testing and Configuration Guide

### Installing Pytest

To ensure the tests run correctly, the project uses Pytest. If Pytest is not already installed, you can install it using the following command:

```
pip install pytest
```

### Test File Structure

The tests for this project are located in the `tests` folder. The pytest test file is called:

`test_server_client.py`.

### Running the Tests

To run the tests from the parent folder, use the following command:

```
pytest tests/test_server_client.py
```

## Speed Testing

The project also includes speed testing files which are still in the test folder and are called `speed_test_client.py` and `speed_test_server.py`, which are configured to test the speeds of the algorithms when used on different file sizes and measure queries per second.They cannot and aren't configured for pytest but for individual custom speed and query tests.

The test files containing strings to be searched for are located in the `tests` folder under a subfolder named `test_files`.File called file_10000.txt contains 10,000 rows, the file called file_500000.txt contains 500,000 rows and the file called file_1000000.txt contains 1,000,000 rows.

### Speed and Query Test Server Configuration

In the `speed_test_server.py` server code located in the tests folder, there is a variable `FILE_PATH` which by default points to:

```
FILE_PATH = "/path/to/folder/with/file/containing/strings"
```

But can be configured to point to:

```
FILE_PATH = "test_files/file_10000.txt"
```

Which is the path to the test files containing strings.(see in code docstring).

To run the speed_test_server.py speed testing file while still in the parent directory,run the

command `python3 tests/speed_test_server.py`

**Speed and Query Test  Client Configuration**

In the client code, there are several important variables:

**CONCURRENT_QUERIES**: Determines how many concurrent clients can query the server. By default, it is set to:

```
CONCURRENT_QUERIES = 50
```

**SEARCH_STRING**: The string that is being searched for. By default, it is set to:

```
SEARCH_STRING = "Row 9999"
```

To run the speed_test_client.py speed testing file while still in the parent directory,run the

command `python3 tests/speed_test_client.py`.You will see the test output in the client side after running the speed_test_client.py file.

**Running the Server and Client**

1. **Start the Server**: Ensure the server and client ports are the same.Always run the server first using the appropriate and available port and server ip.
2. **Start the Client**: After the server is running, you can then run the client script. If you attempt to run the client before the server, it will not work.
3. **Warning**: Make sure the ports and the server ips are not being used for another process

---

## Managing Your Server Script with Supervisor(Daemonizing Your Software)

This guide explains how to manage your server script using Supervisor, a process management tool.

## Prerequisites

- **Make sure SSL is set to False before running supervisor otherwise it won't work.**
- **Supervisor installed:** If not installed, run:

```
sudo apt-get update

sudo apt-get install supervisor
```

## Setting Up Supervisor

1. **Create a Configuration File:**
    - Choose a location where Supervisor looks for configuration files (e.g., `/etc/supervisor/conf.d/`).
    - Create a file named `server.conf` in the chosen location using an editor like `nano`:

```
sudo nano /etc/supervisor/conf.d/server.conf
```

**Configure the File:**

- Paste the following configuration into `server.conf` with the paths configured to match the paths in your pc.Please find a detailed explanation after this code block.

```
[program:server]

command=/path/to/your/project/venv/bin/python /path/to/your/project/server.py

directory=/path/to/your/project

autostart=true

autorestart=true

stderr_logfile=/path/to/your/project/error.log

stdout_logfile=/path/to/your/project/output.log

environment=PATH="/path/to/your/project/venv/bin:%(ENV_PATH)s"
```

The following is a detailed explanation of each line in the Supervisor configuration file:

**[program]**

- This line defines the section for a program named `server`. This is a label that Supervisor uses to identify the configuration block for the server process.

**command=/path/to/your/project/venv/bin/python /path/to/your/project/server.py**

- **command**: Specifies the command Supervisor will use to start your server.
    - `/path/to/your/project/venv/bin/python`: This is the path to the Python interpreter inside your virtual environment. Replace `/path/to/your/project` with the actual path to your project directory.
    - `/path/to/your/project/server.py`: This is the path to your server script. Replace `/path/to/your/project` with the actual path to your project directory.

**directory=/path/to/your/project**

- **directory**: Sets the working directory for the process. Supervisor will change to this directory before running the command. Replace `/path/to/your/project` with the actual path to your project directory.

**autostart=true**

- **autostart**: Configures the process to start automatically when Supervisor starts. Set to `true` to enable this feature.

**autorestart=true**

- **autorestart**: Configures the process to automatically restart if it exits unexpectedly. Set to `true` to enable this feature.

**stderr_logfile=/path/to/your/project/error.log**

- **stderr_logfile**: Specifies the path to the file where Supervisor will log the standard error output of the process. Replace `/path/to/your/project` with the actual path to your project directory.

**stdout_logfile=/path/to/your/project/output.log**

- **stdout_logfile**: Specifies the path to the file where Supervisor will log the standard output of the process. Replace `/path/to/your/project` with the actual path to your project directory.

**environment=PATH="/path/to/your/project/venv/bin:%(ENV_PATH)s"**

- **environment**: Sets environment variables for the process.
    - `PATH="/path/to/your/project/venv/bin:%(ENV_PATH)s"`: This ensures that the path to the virtual environment's `bin` directory is included in the `PATH` environment variable, followed by the existing `PATH` (`%(ENV_PATH)s`). Replace `/path/to/your/project` with the actual path to your project directory.

```
Replace all occurrences of /path/to/your/project with the actual
path to your project directory on your system.

Ensure that the paths and filenames used in the configuration
match the structure and names of your project files and
directories.
```

**This is an example of my configuration**

**[program:server]**

**command=/home/powell/Projects/Sockets/venv/bin/python /home/powell/Projects/Sockets/server.py**

**directory=/home/powell/Projects/Sockets**

**autostart=true**

**autorestart=true**

**stderr_logfile=/home/powell/Projects/Sockets/error.log**

**stdout_logfile=/home/powell/Projects/Sockets/output.log**

**environment=PATH="/home/powell/Projects/Sockets/venv/bin:%(ENV_PATH)s"**

**Update Supervisor:**

- Reload Supervisor to recognize the new configuration:

```
sudo supervisorctl reread

sudo supervisorctl update
```

## Using Supervisor Commands

1. **Starting the Script:**

   Start the server using Supervisor:

   ```
   sudo supervisorctl start server
   ```

**Checking Status:**

- Check the status of all programs managed by Supervisor:

  ```
  sudo supervisorctl status server
  ```

**Verifying the Running Process** (Use one of these commands):

- Check running processes with `ps`:

  ```
  ps aux | grep server.py
  ```

**Stopping the Script:**

- Stop the specific program:

  ```
  sudo supervisorctl stop server
  ```

## Additional Notes

- Should you make changes to the script or configuration file, then restart Supervisor to apply them:

  ```
  sudo supervisorctl restart server
  ```

### Enable Supervisor Service

To ensure that the Supervisor service is enabled to start on boot. You can use `systemctl` to enable it:

```
sudo systemctl enable supervisor
```

After running this command, you can verify that Supervisor is enabled by using:

```
sudo systemctl is-enabled supervisor
```

### Generating a Self-Signed Certificate

To, generate a self-signed certificate using OpenSSL run then fill the necessary info(there is already a generated certificate for you to use in the project for testing):

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

## Troubleshooting

1. **Process won't start:**
   - Check the error log: `cat /path/to/your/project/error.log`
   - Ensure paths in the configuration are correct
   - Verify the user has necessary permissions
2. **Supervisor won't reload configuration:**
   - Check Supervisor's main log: `sudo cat /var/log/supervisor/supervisord.log`
   - Ensure the configuration file syntax is correct
3. **Process keeps restarting:**
   - Check your server script for errors
   - Increase the `startretries` value in your configuration