PRODUCTION AND DEVELOPMENT

Both sections require certain configurations to establish the serial communication between the Flask API and the hardware device. Therefore a Configuration file located in the root directory of (.env) is used to set up the serial communication between the Flask API and the hardware device. The syntax of the configuration files should be as follows:

```
port = "/dev/ttyUSB0"
baud_rate = 9600
```

In the above syntax, `port` specifies the serial port used to connect to the hardware device, and `baud_rate` specifies the speed of data transfer in bits per second. Under Linux, you can find out the port used via

```
ls /dev
```

On Windows, the Device Manager can be used.

DEVELOPMENT

Microcontroller

In this section, we will guide you through the process of setting up a development environment for the NodeMCUv1 board, which forms a key component of the TRNG. Our primary tools will include the official Arduino IDE and a microcontroller code specific to our TRNG, both of which are downloadable from designated repositories.

Requirements: A personal computer with the Legacy Arduino IDE installed. You can download it from the official website. You'll also need a NodeMCU ESP8266 development board which can be purchased from any reputable electronics supplier. The microcontroller code for the TRNG should be downloaded from the current repository.

1. **Clone the Repository**

   Use the following command to clone the repository:

   ```
   git clone https://github.com/athena-security/cep.git
   ```

2. **Open the Sketch in the Arduino IDE**

   Navigate to the downloaded repository location and open the "nodemcu.ino" file.

3. **Configure the Arduino IDE to Support the NodeMCU Board**

   Begin by adding the ESP8266 board manager URL. Go to File > Preferences and in the "Additional Boards Manager URLs" field, add the following link: "http://arduino.esp8266.com/stable/package_esp8266com_index.json". Click OK to save changes.

   Next, install the ESP8266 board. Navigate to Tools > Board > Boards Manager, search for ESP8266, and proceed with the installation.

4. **Select the NodeMCU Board**

   Go to Tools > Board > ESP8266 Boards > NodeMCU 1.0 (ESP-12E Module).

5. **Select the Communication Port**

   Go to Tools > Port and select the COM Port that your NodeMCU is connected to.

6. **Upload the Sketch to the Board**

   Click the upload button (->) or select Sketch > Upload from the menu.

Upon successful completion of these steps, the LED and fan connected to your NodeMCU board should respond to "on" and "off" commands sent via the serial interface. The board should also transmit processed sensor data when active.

DEVELOPMENT

API

This section will walk you through setting up a development environment for your Flask API. We will be using a Python virtual environment to encapsulate our development dependencies.

Requirements: A Linux machine with Python installed (You can check the Python version by using the command: `python3 --version`). Pip, the Python package installer (Check if pip is installed by using the command: `pip3 --version`). If it's not installed, you can install it with `sudo apt install python3-pip`. Ensure you have the Flask application's source code in the `/home/dev/backend/` directory.

1. **Create and Activate a Virtual Environment**

   Python 3 comes with built-in support for virtual environments. You can create a virtual environment using the following command:

   ```
   python3 -m venv /home/dev/backend/backend_env
   ```

   Activate the virtual environment:

   ```
   source /home/dev/backend/backend_env/bin/activate
   ```

   Your prompt should change to indicate the activated environment.

2. **Install Flask and Other Dependencies**

   With the virtual environment activated, you're ready to install Flask and its dependencies. Assuming you have a requirements.txt file located in `/home/dev/backend/`, install these dependencies:

   ```
   pip3 install -r /home/dev/backend/requirements.txt
   ```

3. **Run the Flask Application**

   1. Run the Application with Flask's Built-in Server

      With the dependencies installed, you can now run the Flask application. Navigate to the application directory and run the Flask application:

      ```
      cd /home/dev/backend/
      flask run
      ```

      By default, the Flask application runs on port 5000. You can access it by opening http://127.0.0.1:5000 in your web browser.

2. Run the Application with Gunicorn

   Gunicorn is a WSGI HTTP server for Python web applications. If you installed Gunicorn as per the `requirements.txt` file, you can use it to run the Flask application:

   ```
   cd /home/dev/backend/
   gunicorn --bind 127.0.0.1:8000 api:app
   ```

   This command will start the Gunicorn server on `127.0.0.1` at port `5000`. Make sure to replace `api:app` with your application's WSGI entry point. You can now access your application by opening http://127.0.0.1:5000 in your web browser.

4. **Develop and Test the API**

   You can now start developing your API. Use a tool like Postman or curl to test the API endpoints. After making changes to your code, you need to restart the Flask server to see the changes take effect. Remember to deactivate the virtual environment when you're done:

   ```
   deactivate
   ```

   This setup ensures that your development dependencies are isolated from other Python projects on your machine.

DEVELOPMENT

Frontend

This section will walk you through setting up a development environment for your Angular application.

Requirements: A Linux machine with Node.js and npm (Node Package Manager) installed. You can check Node.js version using `node -v` and npm version using `npm -v`. If not installed, you can download Node.js and npm from here. Angular CLI installed globally on your system. You can install it using npm: `npm install -g @angular/cli`. You can check the Angular CLI version using `ng version`. Ensure you have the Angular application's source code in the `/home/dev/frontend/ directory`.

1. **Navigate to Your Application Directory**

   Navigate to the directory where your Angular application's source code is located:

   ```
   cd /home/dev/frontend/
   ```

2. **Install Dependencies**

   Angular project dependencies are listed in the `package.json` file located in the project root directory. You can install these dependencies using npm:

   ```
   npm install
   ```

3. **Run the Angular Application**

   You can now start the Angular development server using the Angular CLI:

   ```
   ng serve
   ```

   By default, the Angular application runs on port 4200. You can access it by opening `http://localhost:4200` in your web browser.

4. **Develop and Test the Application**

   You can now start developing your application. The Angular development server automatically watches for file changes and reloads the page. Hence, you can see the changes in real-time in your web browser. Remember to stop the Angular development server when you're done:

   ```
   # Press Ctrl+C in the terminal where the server is running
   ```

   This setup ensures that your development dependencies are isolated from other projects on your machine. Happy coding!

PRODUCTION

Static IP

In some network environments, it may be desirable or necessary to assign a static IP address to a machine, instead of relying on dynamic IP assignment via DHCP. A static IP address can be beneficial in cases where the machine needs to be consistently reachable at the same IP, such as for a server or when configuring network hardware.

In this section, we will guide you through the process of setting up a static IP address for your Linux machine. Specifically, we will be configuring the `eth0` network interface to use a static IP address. This setup will utilize the `dhcpcd` network configuration tool, which is pre-installed on most Linux distributions.

Requirements: A Linux machine with dhcpcd network configuration tool installed (most distributions have dhcpcd pre-installed). Root or sudo access to execute the necessary commands.

1. **Edit the dhcpcd configuration file**

   Open the dhcpcd configuration file located at `/etc/dhcpcd.conf` and paste the following lines at the end of the file:

   ```
   sudo vim /etc/dhcpcd.conf
   ```

   Add the following configuration for the `eth0` interface at the end of the file:

   ```
   # file: /etc/dhcpcd.conf
   interface eth0
   static ip_address=172.16.78.59/24
   ```

   Remember to replace `172.16.78.59/24` with your desired static IP address and CIDR notation. After adding these lines, save and close the file.

2. **Restart the dhcpcd service**

   To apply these changes, you need to restart the `dhcpcd` service:

   ```
   sudo systemctl restart dhcpcd
   ```

3. **Verify the IP address assignment**

   Check the IP address assignment to the `eth0` interface:

   ```
   ip addr show eth0
   ```

The output should show the static IP address assigned to the `eth0` interface. With this setup, your `eth0` interface will now have a static IP address which will persist across reboots.

PRODUCTION

Python Virtual Environment

A Python virtual environment is a self-contained directory tree that includes a Python installation and a number of additional packages. The use of a virtual environment is to prevent potential conflicts between system-wide Python packages and the packages used in your particular Python project. In simpler terms, it allows you to create an isolated environment for your Python project, so you can install any package you need without worrying about affecting other Python projects. This is especially useful when working with many different projects that may require different versions of packages.

In this section, we will guide you on setting up a Python virtual environment for your Flask application.

Requirements: A Linux machine with Python installed (You can check the Python version by using the command: `python3 --version`). Pip, the Python package installer (Check if pip is installed by using the command: `pip3 --version`). If it's not installed, you can install it with `sudo apt install python3-pip`. Ensure you have the Flask application's source code in the `/home/dev/backend/` directory

1. **Install Virtual Environment**

   Python 3 comes with built-in support for virtual environments. You can create a virtual environment using the following command:

   ```
   python3 -m venv /home/dev/backend/backend_env
   ```

2. **Activate the Virtual Environment**

   After creating a virtual environment, you need to activate it:

   ```
   source /home/dev/backend/backend_env/bin/activate
   ```

   Your prompt should change to show the name of the activated environment.

3. **Install Flask and Other Dependencies**

   With the virtual environment activated, you can now install Flask and its dependencies. Use pip to install these dependencies:

   ```
   pip3 install -r /home/dev/backend/requirements.txt
   ```

4. **Verify the Installdlations**

You can verify if the packages are installed successfully in your virtual environment by checking the list of installed packages: `pip3 freeze`. This command should output a list of installed packages along with their versions, including Flask and its dependencies.

Remember to deactivate the virtual environment when you're done:

```
deactivate
```

Now, you have a Python virtual environment set up at /home/dev/backend/backend_env with Flask and its dependencies installed. You're ready to start developing your Flask application within this environment.

PRODUCTION

API & Gunicorn

Gunicorn is a Python Web Server Gateway Interface (WSGI) HTTP server. It is a pre-fork worker model, which means that it forks multiple worker processes to handle incoming requests. This can provide your Flask application with better concurrency and load handling capabilities. Setting up Gunicorn as a systemd service allows Gunicorn to start on boot, ensuring your web application is always available. It also monitors the Gunicorn processes, automatically restarting them if they crash, thus enhancing the reliability of your application.

This section will guide you on setting up the Gunicorn application server to run as a systemd service.

Requirements: A Linux machine with systemd. Gunicorn and your application installed in a virtual environment at `/home/dev/backend/backend_env`. `dev` user and `www-data` group existing on the system. Ensure you have the Flask application's source code in the `/home/dev/backend/` directory and root or sudo access to execute the necessary commands.

1. **Create a systemd Service File**

   Systemd service files are located in the `/etc/systemd/system` directory. Create a new service file for Gunicorn:

   ```
   sudo vim /etc/systemd/system/gunicorn.service
   ```

2. **Add Configuration to the Service File**

   In the newly created file, add the following lines:

   ```
   # file: /etc/systemd/system/gunicorn.service
   [Unit]
   Description=Gunicorn server instance
   After=network.target

   [Service]
   User=dev
   Group=www-data
   WorkingDirectory=/home/dev/backend/
   Environment="PATH=/home/dev/backend/backend_env/bin"
   ExecStart=/home/dev/backend/backend_env/bin/gunicorn --access-logfile - --
   error-logfile - --log-level debug --timeout 900 --workers 1 --bind
   127.0.0.1:8000 api:app


   [Install]
   WantedBy=multi-user.target
   ```

This configuration specifies that we want to start the Gunicorn service after the network target has been reached during the boot process. It also sets the user and group that will run the service, sets the working directory, and specifies the command to start Gunicorn.

Ensure that the paths for `WorkingDirectory`, `Environment`, and `ExecStart` point to the correct locations for your setup. Save and close the file.

3. **Enable and Start the Gunicorn Service**

   Now that we have our systemd service file, we can tell systemd to start it during the boot sequence:

   ```
   sudo systemctl enable gunicorn
   ```

   You can also start the service manually without rebooting:

   ```
   sudo systemctl start gunicorn
   ```

4. **Check the Status of the Gunicorn Service**

   To verify that the service is running as expected, you can use:

   ```
   sudo systemctl status gunicorn
   ```

   The output should indicate that the service is active and running. Now, Gunicorn will start on boot and will be managed by systemd.

PRODUCTION

Frontend & Nginx

Nginx is a powerful, high-performance HTTP server and reverse proxy server. It can serve static files and can also proxy requests to other servers, making it a versatile tool for both serving and routing HTTP requests.

In the following steps, we will be setting up two Nginx server blocks. The first server block will serve as a web server listening on port 443, typically used for serving secure HTTP traffic. The second server block will act as a reverse proxy server listening on port 8443, forwarding requests to a backend service running on http://127.0.0.1:8000. This setup allows you to use Nginx to serve your web application and to route requests to your Flask API.

It's important to note that in a production setup, port 443 is typically used for serving secure HTTP traffic over SSL/TLS. As we're using a self-signed certificate in this setup, your browser may display a warning about the site's security. This is expected behavior. In a live, public-facing server environment, you'd want to use a certificate issued by a trusted certificate authority to avoid this warning. However, for development, testing, and internal use, a self-signed certificate can be sufficient.

Requirements: Nginx and a web application ready to be served at `/var/www/html/trng-frontend` and the gunicorn server with the backend up and running.

1. **Build the Angular Application**

   Navigate to your Angular application directory and build your Angular application using the Angular CLI. The `--prod` flag will cause Angular to build in production mode, optimizing the build for production:

   ```
   cd /home/dev/frontend/
   ng build --configuration production
   ```

   The built files will be stored in the `dist/` directory inside your project folder.

2. **Move the Built Files to the Nginx's Root Directory**

   Copy the built files to the root directory of the Nginx server. In the provided Nginx configuration, the root directory is `/var/www/html/trng-frontend`. You may need to create this directory if it does not exist:

   ```
   sudo mkdir -p /var/www/html/trng-frontend
   sudo cp -r dist/* /var/www/html/trng-frontend/
   ```

3. **Generate SSL Certificates**

Firstly, we need to generate a SSL certificate and key. We can use the OpenSSL toolkit to generate a self-signed certificate. Run the following command, and it will create a certificate and key in your current directory.

```
openssl req -x509 -newkey rsa:4096 -nodes -out trng.crt -keyout trng.key -
days 365
```

This will start an interactive script which will ask you for various bits of information. Fill it out as you see fit. These fields are not critical for a self-signed certificate. Make sure you move the `trng.crt` and `trng.key` files to the `/etc/nginx/ssl` directory, or adjust the nginx configuration file to point to the location where you've stored your certificates and keys.

After running this command, move the generated `trng.crt` and `trng.key` files to the `/etc/nginx/ssl` directory:

```
sudo mkdir -p /etc/nginx/ssl
mv trng.crt trng.key /etc/nginx/ssl/
```

4. **Create Nginx Config File**

   Navigate to the Nginx directory containing site configurations and create a new configuration file. This is usually `/etc/nginx/sites-available`.

   ```
   sudo vim /etc/nginx/sites-available/trng.conf
   ```

5. **Paste Configuration**

   Copy the given configuration into the file. Ensure that the SSL certificates and key files referenced in the configuration are at the specified location in your file system. Save and close the file.

   ```
   # file: /etc/nginx/sites-available/trng.conf
   server {
     listen 443 ssl;
     server_name $domain_name;

     ssl_certificate /etc/nginx/ssl/trng.crt;
     ssl_certificate_key /etc/nginx/ssl/trng.key;

     location / {
       root /var/www/html/trng-frontend;
           try_files $uri $uri/ /index.html;
     }
   }

   server {
   ```

```
    listen 8443 ssl;
        server_name $domain_name;

    ssl_certificate /etc/nginx/ssl/trng.crt;
    ssl_certificate_key /etc/nginx/ssl/trng.key;

    location /trng/ {
            proxy_pass http://127.0.0.1:8000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header X-Forwarded-Port $server_port;

    }
  }
```

Save and close the file.

6. **Enable the Site**

   Create a symbolic link to this file in the `/etc/nginx/sites-enabled` directory.

   ```
   sudo ln -s /etc/nginx/sites-available/trng.conf /etc/nginx/sites-enabled/
   ```

7. **Test Nginx Configuration**

   Before applying the changes, check if the Nginx configuration is correct. If the configuration is correct, you should see: `nginx: configuration file /etc/nginx/nginx.conf test is successful`

   ```
   sudo nginx -t
   ```

   If there are errors in your configuration, the `nginx -t` command will output them and highlight the lines in the files where they are located. Correct these errors before you proceed.

8. **Reload Nginx**

   Once the configuration test is successful, reload the Nginx service to apply the changes.

   ```
   sudo systemctl reload nginx
   ```

   The setup is now complete. You can test the setup by navigating to your domain on a web browser. Remember that you have configured two different ports, 443 and 8443, so you might want to test both ports to ensure your setup is working correctly.