

14. JUNE 2023

**TRUE RANDOM NUMBER GENERATOR**

**USER**

**MANUAL**

# AthenaSecurity

## PARTICLES IN A WIND TUNNEL

### TABLE OF CONTENTS

Introduction	2
System Architecture Documentation	3
Interface Specifications	6
Statistical Tests	9
Security Considerations	13
Environment Setup	14
User Interface	15
Conclusion	23
Appendix A - Endpoints	24
Appendix B - Environment Setup	25

**T**his user guide provides detailed insights into the functioning of a Python-based true random number generator (TRNG). The TRNG operates using a Flask An application programming interface (API) that facilitates communication over a serial connection with a dedicated hardware device. This device's primary role is the generation of random numbers, which are then served to an Angular-based frontend application.

The design and functionalities of this True Random Number Generator (TRNG) follow the specifications and guidelines presented in the Bundesamt für Sicherheit in der Informationstechnik (BSI) paper titled "[A proposal for: Functionality classes for random number generators - Version 2.0](#)". This reference document provides a comprehensive framework for designing, implementing, and evaluating random number generators. Our system is designed to comply with these standards, ensuring the production of high-quality random numbers suitable for various applications.

## SYSTEM ARCHITECTURE DOCUMENTATION

The system architecture of the application comprises three main components: a backend Flask application, a hardware device serving as a random number generator, and a frontend Angular application.

The Flask application acts as an intermediary, facilitating communication between the software and the hardware random number generator via the serial port. Utilizing the Python serial module, the Flask application can send specific commands to the hardware device, including initialization, shutdown, and restart operations.

Once the hardware device is initialized, it continuously generates random bits, which are buffered in the serial input buffer. When a request for random numbers is received, the Flask application reads the random bits from the buffer. These bits are then parsed and converted into hexadecimal values, resulting in an array of hexadecimal strings as the output. Additionally, the Flask application offers the ability to generate a file containing random test data based on user-defined parameters. Users can choose between binary or text format for the generated file.

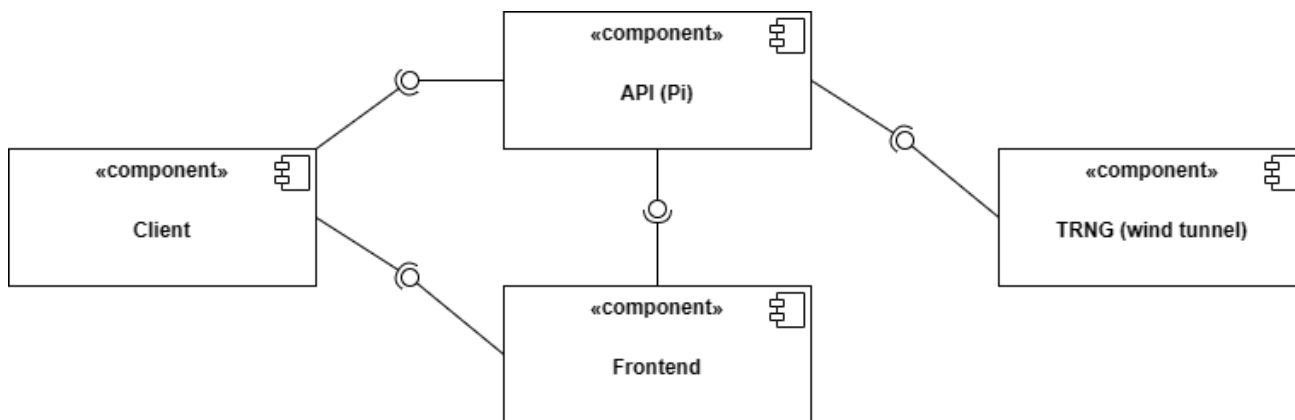
To serve the Flask API, a Gunicorn server is employed. This Python WSGI Hypertext Transfer Protocol (HTTP) server is known for its simplicity and high performance. The Flask API runs on the localhost at port 8000 and is reverse-proxied by an Nginx server block listening on port 8443. This reverse proxy configuration provides added flexibility and enhances security.

Furthermore, the system utilizes a Python virtual environment to manage dependencies. This practice ensures that the project's packages remain isolated and do not interfere with those of other Python projects. By keeping dependencies contained within the virtual environment, the system promotes a more robust and error-free development and deployment process.

On the frontend side, the application is developed using Angular, a powerful framework for building single-page applications (SPAs). The Angular frontend is hosted on an Nginx server listening on port 443. Nginx is recognized for its stability, rich feature set, and efficient resource consumption, making it an ideal choice for serving the Angular application.

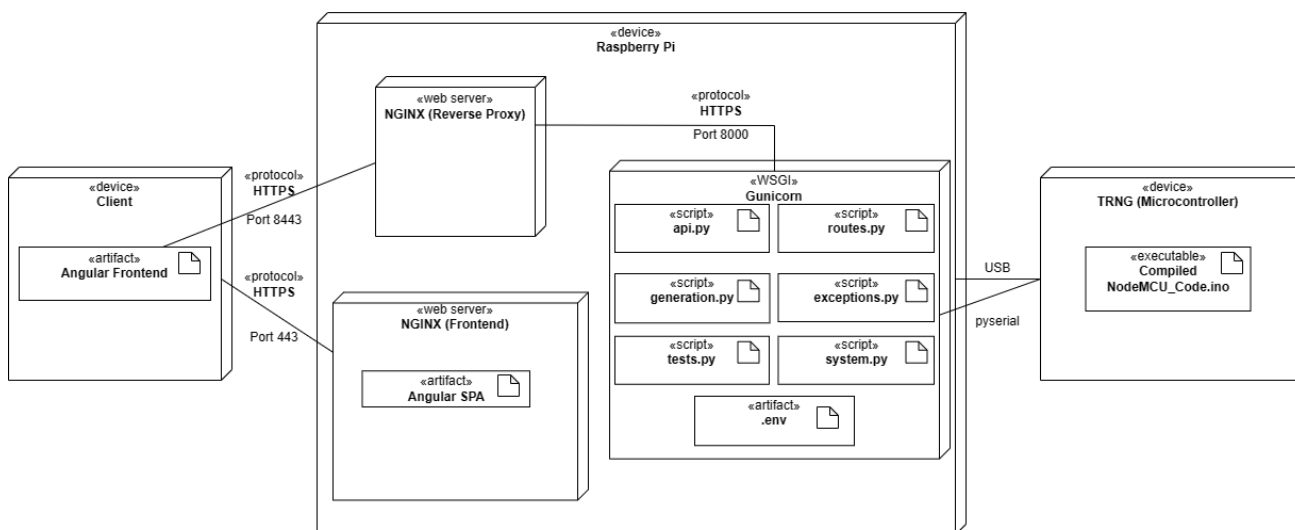
To ensure secure communication, SSL/TLS (Secure Sockets Layer/Transport Layer Security) encryption is implemented for all traffic to and from the servers. Self-signed

certificates generated using OpenSSL are used for this purpose. Additionally, the system is configured with a static IP address of 172.16.78.59 for the system network interface 'eth0', establishing a stable networking environment that enables reliable communication.



UML Component Diagram

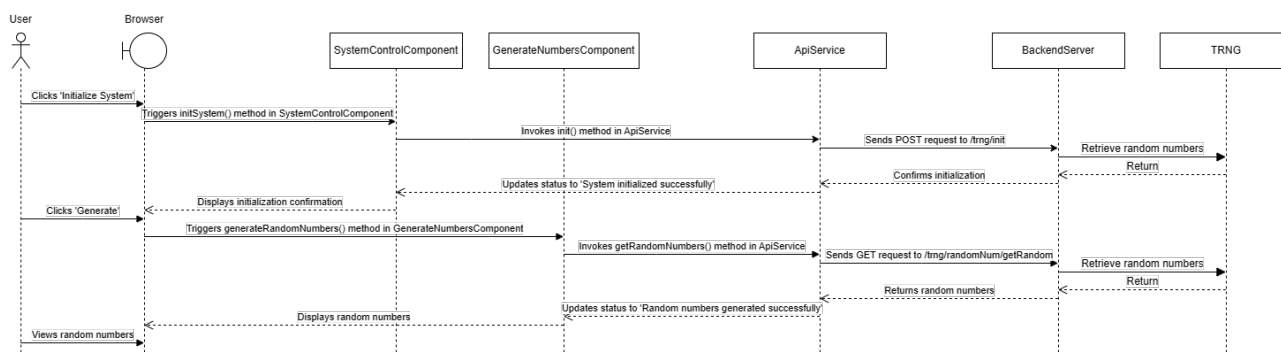
This diagram showcases the interaction between the system components. The Angular application communicates directly with the Flask API, which, in turn, communicates with the hardware device. A more detailed representation of this setup can be found in the accompanying UML deployment diagram:



UML Deployment Diagram

Here we can see, that the Raspberry Pi acts as the central node and deploys three servers. The first server is the Gunicorn WSGI server, responsible for handling all API requests. These requests are passed on by the second server, the NGINX reverse proxy

server, which therefore serves as an intermediary. The Angular Frontend submits requests to the NGINX reverse proxy, benefiting from its routing capabilities. The third server is dedicated to serving the Angular application independently. Additionally, the



UML Sequence Diagram

diagram depicts the serial USB connection between the Raspberry Pi and the TRNG hardware device.

The sequence diagram provided above outlines the primary workflow of the system. A user interacts with the Angular application, which subsequently sends requests to the Flask API. The API, in turn, communicates with the hardware device, performing actions such as generating random numbers based on the user's request.

In summary, this architecture provides a secure, scalable, and performant setup for serving the Angular application and Flask API to the end users, and ensures a smooth, efficient workflow for developers. It is designed to provide a flexible, reliable, and secure way of generating random numbers through a hardware device, allowing for a diverse range of uses in various applications.

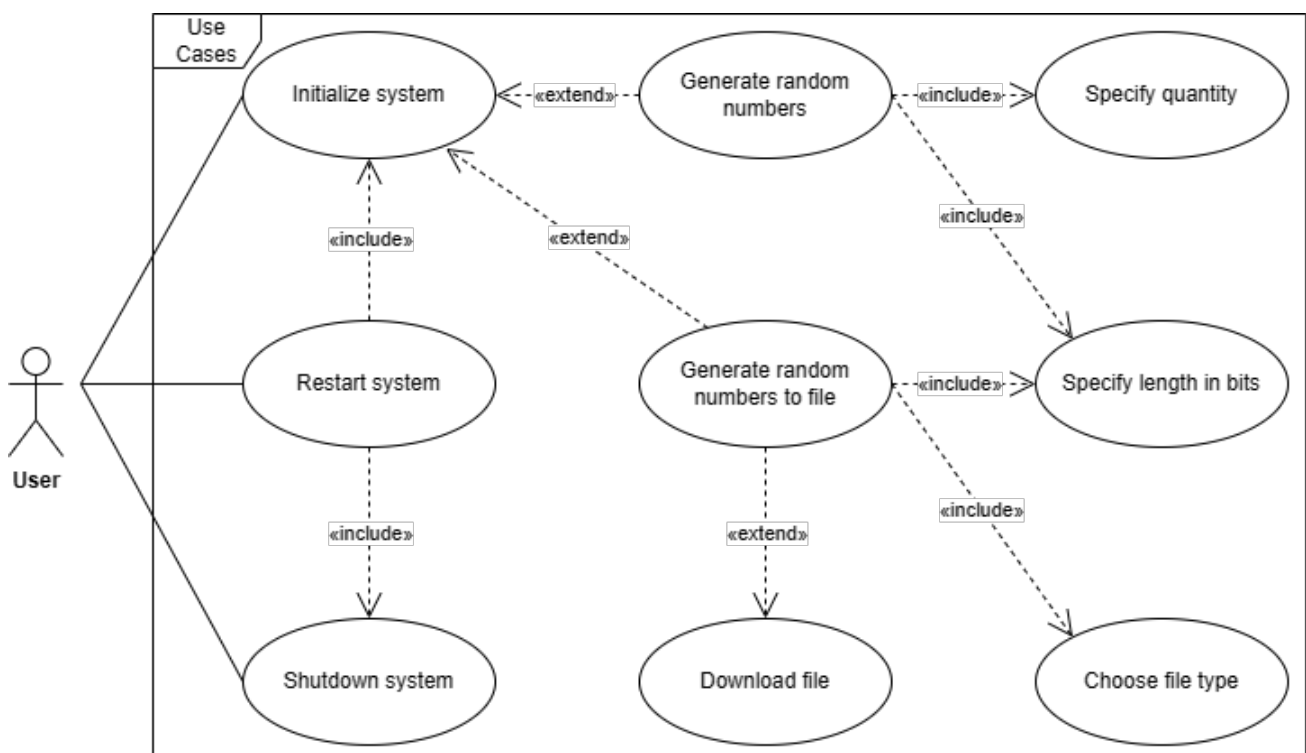
# INTERFACE SPECIFICATIONS

## API ENDPOINTS

The system architecture comprises of several key parts working together to offer a seamless user experience. At the heart of the backend is a Flask microservice written in Python. Flask's simplicity and flexibility allow us to develop a lightweight API for our application, where several routes and functionalities are defined. The Flask API exposes several endpoints for interaction.

- init: Initializes the hardware random number generator.
- getRandom: Returns randomly generated numbers.
- generateTestdata: Generates a test data file with random numbers.
- downloadFile: Downloads the generated test data file.
- shutdown: Shuts down the hardware random number generator.
- restart: Restarts the hardware random number generator.

Each of these endpoints can be mapped to a use case, which are visualized in the UML use case diagram below.



UML Use Case Diagram

The Flask API returns a HTTP status code of 200 for any successfully satisfied request. Other than that, the system can be in various states of errors. Below is a list of HTTP response codes used by the Flask API to signalize errors.

1. 404: The requested page was not found. Check the URL in the address bar of the Browser for typing errors.
2. 409: This response occurs if users send an initialization request, when the system has already been initialized.
3. 432: This response occurs if users try to generate random number, when the system is not initialized.
4. 555: This response can either occur if the Flask API loses connection to the serial device (TRNG) or if an error occurs during the generation of random numbers (e.g. detection of a total failure).

Comprehensive details regarding the individual endpoints of our system, covering both their functions and operations, can be located in Appendix A - Endpoints. This section delves into an extensive exploration of each endpoint, their potential responses, and scenarios of usage. It should be referred to for a thorough understanding of how to correctly interact with our system's endpoints.

## SERIAL INTERFACE

Serial communication between the Flask API and the hardware device is facilitated by a configuration file. This file aids in setting up the specific parameters required for the correct establishment of the connection. Here's an example of the syntax to be used within this file:

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

In this configuration:

- port: This line specifies the serial port used to establish the connection with the hardware device. On a Unix-based system like Linux, this is typically designated as "/dev/ttyUSB0" or similar, depending on the specific hardware setup. You can determine the exact port being used by executing the command "ls /dev" in the



terminal. This will list all device files, and your serial device will generally appear as `"/dev/ttyUSB*" or "/dev/ttyACM*".`

- `baud_rate`: This line designates the speed of the data transfer in bits per second. In this example, a baud rate of 9600 is used, which is a common default for many devices. However, the specific baud rate would depend on your device's specifications.

By using this configuration file, the Flask API can successfully establish a reliable serial communication channel with the hardware device, allowing for efficient data exchange and interaction.

# STATISTICAL TESTS

## TEST SUITES

The statistical validity of the numbers produced by the True Random Number Generator (TRNG) has been thoroughly examined, yielding promising results. Notably, they satisfy the stringent reference standards of the German Federal Office for Information Security, also known as the Bundesamt für Sicherheit in der Informationstechnik (BSI). This was confirmed by passing tests T0 through T8 as defined in the BSI's guidelines, as per the AIS\_31\_testsuite, the recognised yardstick for assessing the quality of random number generators.

## IMPLEMENTED TESTS

One of the obligatory tests that is implemented and prescribed by the PTG.2 standard is the total failure test. This test is meant to detect a total breakdown of the entropy source. In our case, a total breakdown of the entropy source is present, when no snippets are moving in front of the solar cell. This could happen if the fan stops blowing for example. A situation like this will produce patterns or constant values as an output of the TRNG.

The total failure test implemented works as follows: The generation of random numbers is done in chunks of 100 bytes. The total failure test checks how many bytes are identical. A total failure is detected, if the amount of identical bytes exceeds a fixed value. The value needs to be determined in a way that it effectively detects a total failure, while not turning in a false alarm too frequently.

After testing the TRNG by simulating a total failure with the LED and fan turned off, we observed that a fixed value of six identical bytes proves to be a good trade between having a good rate of detection, while keeping the probability of a false alarm low. The probability for the event that the total failure test detects a total failure, even though there is none (i.e. the probability for an ideal RNG to produce such a result), can be calculated with the help of the cumulative binomial distribution function:

$$F(k; n, p) = P(X \leq k) = \sum_{i=0}^k \binom{n}{i} \cdot p^i \cdot (1 - p)^{n-i}$$

For this case  $n = 100$ ,  $k = 6$  and  $p = 1/256$  as there are  $2^8 = 256$  different outcomes for a

$$1 - F(6; 100, \frac{1}{256}) = P(X \leq 6) = \sum_{i=0}^6 \binom{100}{i} \cdot p^i \cdot (1 - \frac{1}{256})^{100-i} \approx 1.617009 \times 10^{-7}$$

random byte. As this would be the counter event of more than 6 identical bytes in 100, we can subtract this from 1, which leads to:

With this probability and the time it takes to conduct a total failure in seconds which is

$$\frac{800}{60} \approx 13.33333333 \left( \frac{\text{seconds}}{\text{total failure test}} \right)$$

for a bit generation rate of 60 per second (as 800 bits = 100 bytes are needed for a total failure test; time for computation is neglected), we can calculate the expected number of false alarms in a year with

$$E(X) = n * p$$

where X: Number of false alarms, n is the number of total failure tests conducted in a year and

$$p = 1.617009 \times 10^{-7}$$

n can be calculated with

$$\frac{365 \text{ days}}{\text{year}} \cdot \frac{24 \text{ hours}}{\text{day}} \cdot \frac{60 \text{ minutes}}{\text{hour}} \cdot \frac{60 \text{ seconds}}{\text{minute}} \cdot \frac{1 \text{ total failure test}}{13.33333333 \text{ seconds}} \approx 2,365,200 \frac{\text{total failure tests}}{\text{year}}$$

which leads us to

$$E(\text{Number of false alarms}) = 2,365,200 \cdot 1.617009 \times 10^{-7} \approx 0.3825$$

If we take the reciprocal of this number, we get

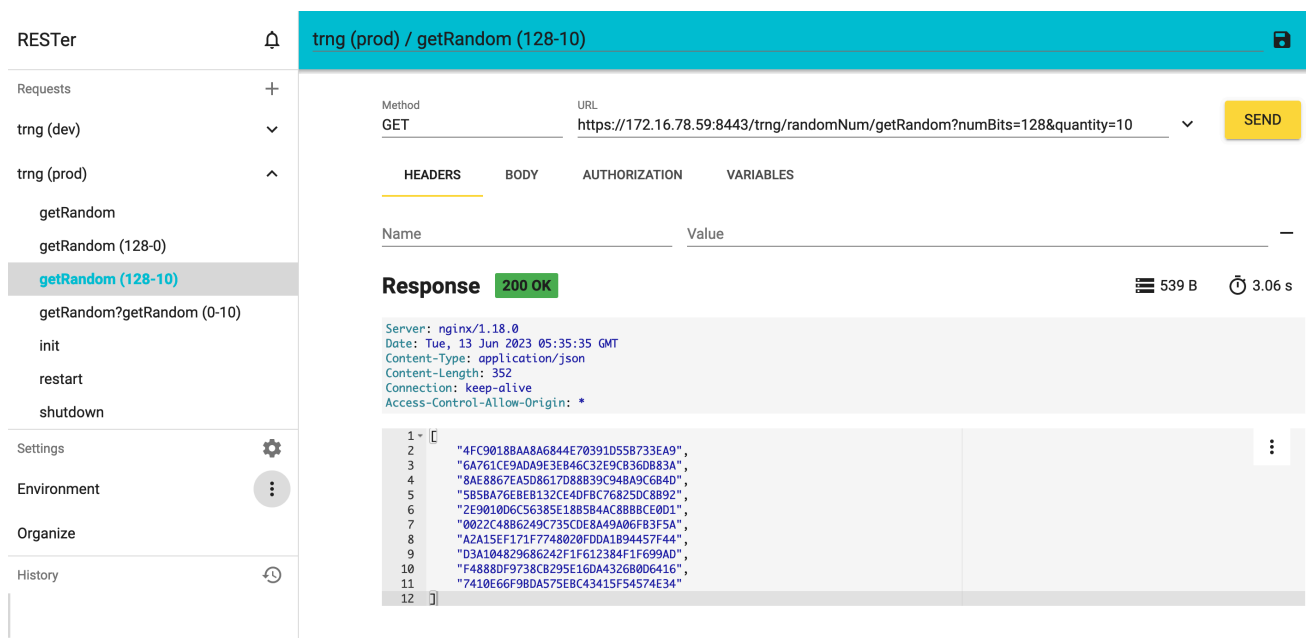
$$0.3825^{-1} \approx 2.6144$$

which means it would take on average 2.6144 years for this event to be expected once.

Furthermore, by calculating confidence intervals on the base of a random sample with 20,000 bytes, we could observe the probability of a total failure not being detected, even though there is one, to be between 3.0% to 3.8% with a confidence of 99.7%. This suggests that the determined fixed value allows the test to effectively detect a total failure.

## SOFTWARE TESTING

Our Representational State Transfer (REST) API has undergone manual testing procedures employing the flexible tool, "RESTER". This tool empowers users to dispatch a variety of HTTP requests to a REST interface, offering broad capabilities to manipulate HTTP headers and bodies.



Screenshot of the "RESTER" REST API testing tool

Additionally, "RESTER" offers a detailed perspective of responses obtained from the API, enabling a thorough investigation of how the server responds under different circumstances. This ensures our understanding of the robustness and effectiveness of the API under a range of operational conditions.

The robustness of the API's error handling capability was also scrutinized through manual testing. This included providing illegal or invalid inputs and intentionally interrupting the serial connection during operation. This approach guarantees the

system's resilience in handling unexpected circumstances and its capability to return appropriate responses during such events.

## SECURITY CONSIDERATIONS

To ensure a reliable operation of the TRNG, users should consider following things:

1. Make sure the solar cell inside the TRNG is protected from direct sunlight, as this causes the microcontroller to measure maximum voltage.
2. Make sure the TRNG is set on a flat surface, as this can cause the snippets inside the TRNG to change altitude thus stop blocking light to the solar cell.
3. Beware, that the padding of leading zero's when generating numbers which are not divisible by four, changes the outcomes and thus the probability of the first hexadecimal digit of the random number. This should be held in mind when handling such numbers.
4. Make sure to keep the TRNG away from conductive material to prevent the microcontroller from being damaged.
5. The TRNG was not tested in places with very low air humidity, in which the snippets may hold on to the surface of the tunnel.
6. A long-term test over several weeks and beyond was not performed. It is conceivable that the paper snippets wear off due to abrasion, or that the effect of the anti-static spray wears off.

## ENVIRONMENT SETUP

The following documentation is divided into two main sections: Development and Production.

### 1. **Development:**

The development section of this documentation focuses on setting up and running the system in a development environment. This is where the features are built and tested before they are deployed to the production environment.

### 2. **Production:**

The production section of this documentation focuses on preparing the system for a production environment. The features that have been built and tested in the development environment are made production-ready, and the system is set up to handle real-world traffic.

Both the "Development" and "Production" sections of this documentation, detailing the system setup and the preparation for a real-world environment, can be found in the appendix (Appendix B - Environment Setup).

## USER INTERFACE

The user interface developed with Angular provides a user-friendly and efficient means of interacting with the hardware-based random number generator. It offers a well-structured layout and intuitive design, facilitating seamless user engagement with the system's functionalities.

The interface incorporates responsive buttons that enable users to initiate requests to the designated API endpoints mentioned earlier. This simplifies the process of interacting with the random number generator, allowing users to trigger specific operations effortlessly. To enhance flexibility and customization, the user interface includes input fields that allow users to enter parameters for the `getRandom` and `generateTestdata` endpoints. This enables users to fine-tune their requests and customize the output based on specific requirements or preferences.

Moreover, the user interface includes a convenient feature for downloading the file generated by the `generateTestdata` endpoint. By clicking the designated "Download File" button, users can conveniently retrieve the generated file for further analysis or integration with other applications.

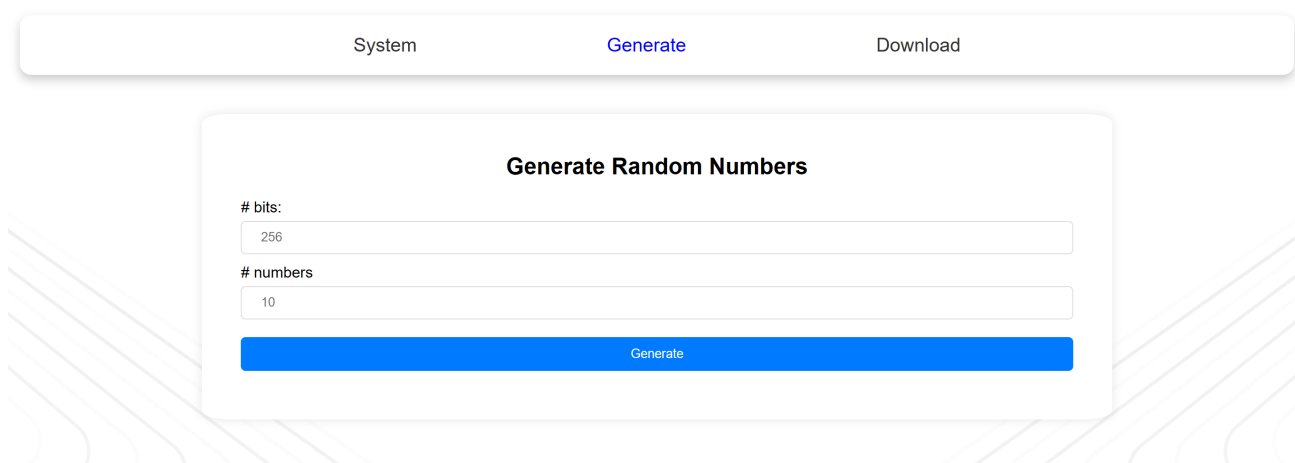
In the next section, we will walk through the step-by-step instructions on how to retrieve random numbers using the user interface.



## STEP BY STEP INSTRUCTIONS

### GENERATE RANDOM NUMBERS

The Angular SPA provides a clear and simple graphical user interface (GUI) to generate random numbers. Once you have set everything up correctly, you will be able to use the Angular frontend to generate numbers. You can access the GUI by typing in the static IP address of the Raspberry Pi in your Browser's adress bar, which will take you to the default page: Generate.



System Generate Download

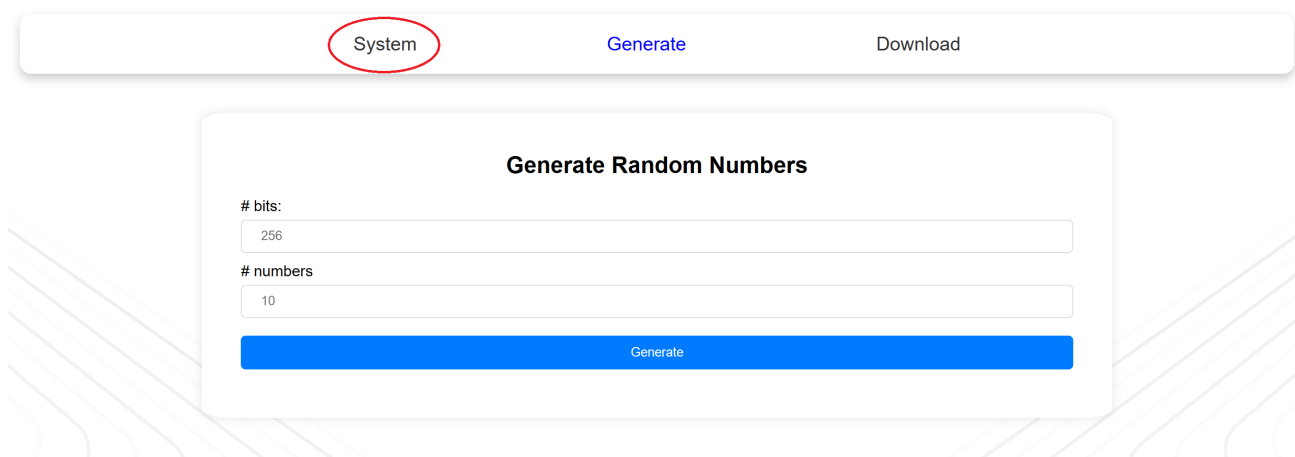
**Generate Random Numbers**

# bits:  
256

# numbers  
10

Generate

Before generating any numbers, initialize the system (TRNG) by navigating to the **System** tab.



System Generate Download

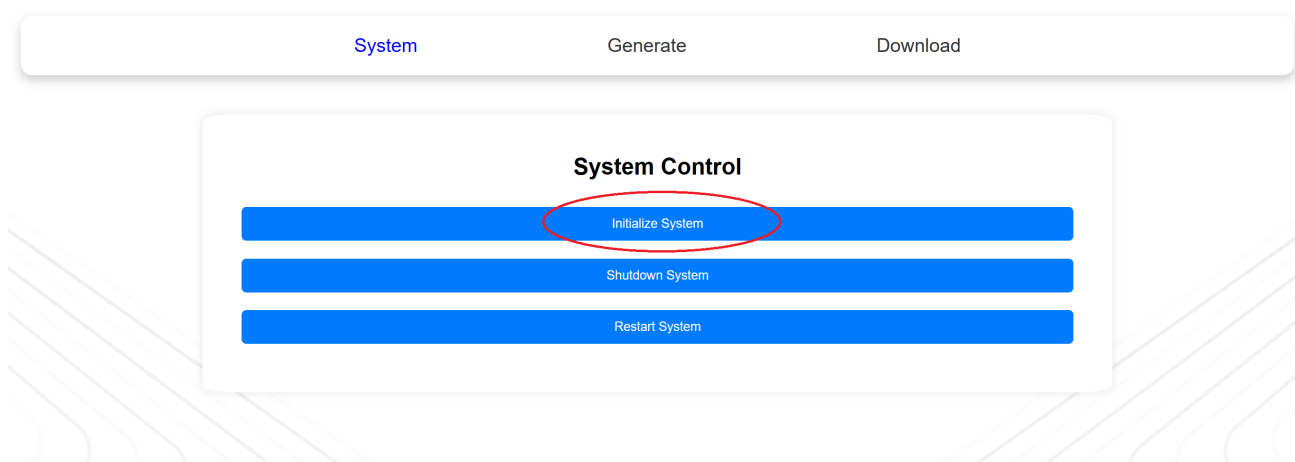
**Generate Random Numbers**

# bits:  
256

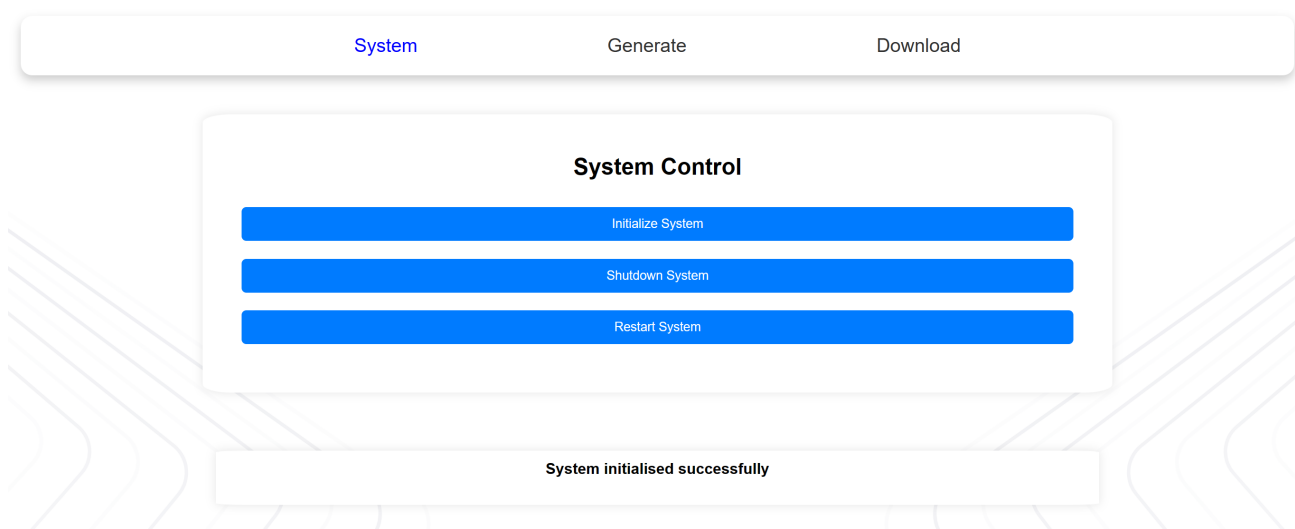
# numbers  
10

Generate

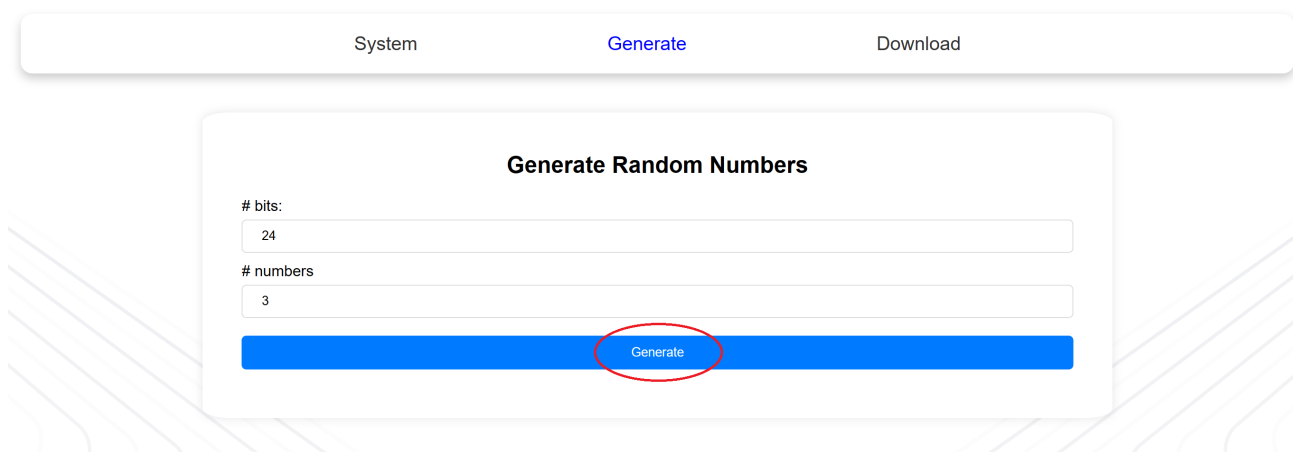
Once clicked, you arrive at the System page. Here you will find options to initialize, restart, and shut down the system. Click on the **Initialize** button to start up the TRNG. The shutdown button can be used to turn the TRNG off and the restart button shuts the system down and reinitializes it shortly after.



After clicking **Initialize**, a status bar will appear below the buttons, indicating the system is being initialized. Wait for a few seconds until a confirmation message appears, indicating a successful initialization.

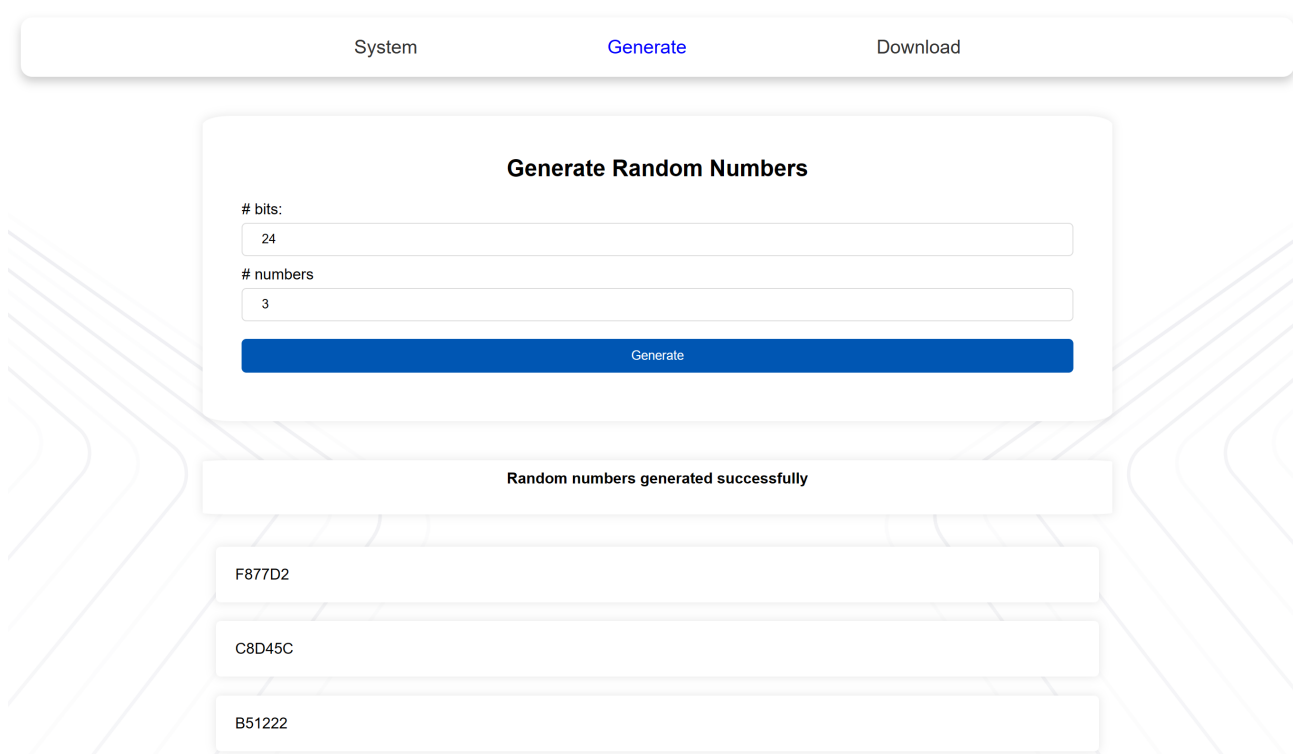


Return to the Generate page by clicking the **Generate** tab on the top navigation bar. To generate random numbers, you need to provide two parameters: **# bits** and **# numbers**. The default value for the # bits field is set to '128', representing the length of each random number in bits. For example, if you want to generate **24**-bit numbers, simply enter '24' in this field. The # numbers field specifies the quantity of numbers to generate, with a default value of '10'. In this example, we will generate '**3**' numbers.



The screenshot shows the 'Generate Random Numbers' interface. At the top, there is a navigation bar with three tabs: 'System', 'Generate' (highlighted in blue), and 'Download'. Below the navigation bar, the main content area is titled 'Generate Random Numbers'. It contains two input fields: '# bits:' with the value '24' and '# numbers' with the value '3'. Below these fields is a large blue button labeled 'Generate', which is circled in red.

A status bar will appear, indicating that the numbers are being generated. After the generation is complete, the status bar will confirm the successful generation, and the random numbers will be displayed below.



The screenshot shows the 'Generate Random Numbers' interface after successful generation. The navigation bar remains the same. The main content area is titled 'Generate Random Numbers'. Below the input fields, a blue status bar displays the text 'Random numbers generated successfully'. Below the status bar, three white boxes display the generated random numbers: 'F877D2', 'C8D45C', and 'B51222'.

*Note: The displayed numbers consist of six characters in hexadecimal format, as each four bits correspond to one hexadecimal digit. If the requested bit length is not divisible by four, it is padded with leading zeros.*

## DOWNLOAD RANDOM NUMBERS

If you wish to download the generated numbers as a file, navigate to the Download tab.

System Generate Download

**Download Random Numbers**

# bits:  
400

filetype:  
.bin

Generate

File successfully generated

Download

On the **Download** page, provide the desired parameters. The first field represents the length of the random number in bits, similar to the Generate page. The default value for this field is '5000'. In this example, we will use **400**.

The second field, filetype, offers a dropdown menu where you can choose between two filetypes: '**bin**' (binary) and '**txt**' (text). The default selection is '**bin**', where random numbers are written as bytes to the file. We will choose '**bin**' for this example.

Click the Generate button to initiate the file generation process. Once the generation is complete, the status bar will display the message "File successfully generated" and present a download button. Clicking the download button will prompt your browser to start the download of the file.

System Generate Download

### Generate Random Numbers

# bits: 24

# numbers 3

Generate

Random numbers generated successfully

F877D2

If you have finished generating random numbers and no longer need the system, you can shut it down. Navigate to the **System** tab on the top navigation bar.

On the System page, locate the **Shutdown System** button and click it. The status bar will display a confirmation message once the shutdown process is successful.

System Generate Download

### System Control

Initialize System

Shutdown System

Restart System

System successfully shutdown

By following these step-by-step instructions, you can effectively utilize the Angular SPA's graphical user interface (GUI) to generate random numbers. Remember to initialize the system, input the desired parameters, and navigate through the different tabs for generating numbers, downloading files, and shutting down the system as needed.

## CONCLUSION

This user guide provides the necessary information to successfully use the hardware-based random number generator. By following these instructions, users can interact with the hardware device via the Flask API or the Angular user interface to generate random numbers and files of random data.

Furthermore, the system's architecture has been designed with flexibility, robustness, and scalability in mind. The use of Python, Flask, and Angular for this project provides a robust and scalable solution for hardware-based random number generation. The hardware device can be controlled through the Flask API, which is a lightweight and flexible solution, making it easier to add or modify functionality. The use of Angular for the frontend allows for a dynamic and responsive user interface. The choice of Gunicorn and Nginx for serving the application offers stability and security for production deployments. However, they do require additional configuration and may not be necessary for smaller scale applications.



## APPENDIX A - ENDPOINTS

## ENDPOINTS

### Main Endpoint

**GET** `/trng/randomNum/getRandom`

This endpoint returns an array of sequences of bits of a specified length and quantity. The API guarantees that the sequences are randomly drawn. The length and quantity of the sequences can be specified using the following query parameters:

- **numBits** (optional, integer, minimum 1, default 1): The number of random bits in each bit sequence.
- **quantity** (optional, integer, minimum 1, default 1): The number of random sequences to generate.

#### Responses

- **200 OK:** The response is an array of strings, where each string is a randomly generated bit sequence of the specified length. The bit sequences are encoded as hexadecimal strings, with leading zeros if necessary. Example response:

```
[ "08c1f27b", "12d45f38", "9a6b7c5d" ]
```

- **432 Service Unavailable:** This error code is returned if the system is not ready to generate random numbers, such as when the random number source is in standby mode or is not reachable.
- **555 Internal Server Error:** This error code is returned if the system is unable to initialize within 60 seconds.

#### Example Usage

To generate 10 random bit sequences of length 8 using curl, you can run the following command:

```
curl "https://172.16.78.59:8443/trng/randomNum/getRandom?numBits=8&quantity=10"
```

## System Control

### GET /trng/randomNum/init

This endpoint initializes the random number generator. If the system is already initialized, a message indicating this fact will be returned.

#### Responses:

- **200 OK:** The system was initialized successfully. Example response: `{'message': 'System initialised successfully'}`
- **409 Conflict:** The system is already initialized. Example response: `{'message': 'System already initialized'}`
- **555 Internal Server Error:** There was an error during system initialization.

#### Example Usage:

```
curl "https://172.16.78.59:8443/trng/randomNum/init"
```

### GET /trng/randomNum/shutdown

This endpoint shuts down the random number generator. If the system is already in standby, a message indicating this fact will be returned.

#### Responses:

- **200 OK:** The system was shutdown successfully. Example response: `{'message': 'System successfully shutdown'}`
- **409 Conflict:** The system is already shut down. Example response: `{'message': 'System already shut down'}`
- **555 Internal Server Error:** There was an error during system shutdown.

#### Example Usage:

```
curl "https://172.16.78.59:8443/trng/randomNum/shutdown"
```

## Additional Endpoints

### GET /trng/randomNum/generateTestdata

This endpoint generates a set of test data. The length and type of the data can be specified using the following query parameters:

- numBits (optional, integer, minimum 1, default 5000): The length of the data to generate.
- filetype (optional, string, default 'bin'): The type of the file to generate.

#### Responses:

- **200 OK:** The data was generated successfully. Example response: `{ 'message': 'Generation successful' }`
- **432 Service Unavailable:** The system is not ready to generate test data.
- **555 Internal Server Error:** There was an error during data generation.

#### Example Usage:

```
curl "https://172.16.78.59:8443/trng/randomNum/generateTestdata?numBits=1000&filetype=txt"
```

### GET /trng/randomNum/downloadFile

After calling the `/trng/randomNum/generateTestdata` endpoint, you can use this endpoint to download a file that contains a set of test data.

#### Responses:

- **200 OK:** The file download begins.
- **555 Internal Server Error:** The requested file could not be found or there was an error during the file download.

#### Example Usage:

```
curl "https://172.16.78.59:8443/trng/randomNum/downloadFile"
```

### **GET /trng/randomNum/restart**

This endpoint restarts the random number generator. If the system is already in standby, a message indicating this fact will be returned.

#### **Responses:**

- **200 OK:** The system was restarted successfully. Example response: `{ 'message' : 'System successfully restarted' }`
- **555 Internal Server Error:** There was an error during system restart.

#### **Example Usage:**

```
curl "http://localhost:5000/trng/randomNum/restart"
```

*Please note that for each of these endpoints, the 555 error code is returned for any exceptions not explicitly caught by the system, indicating an unexpected server error.*

## APPENDIX B - ENVIRONMENT SETUP

### 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](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 Installations

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.