14. JUNE 2023

TRUE RANDOM NUMBER GENERATOR

USER MANUAL

AthenaSecurity

PARTICLES IN A WIND TUNNEL

TABLE OF CONTENTS

This 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.

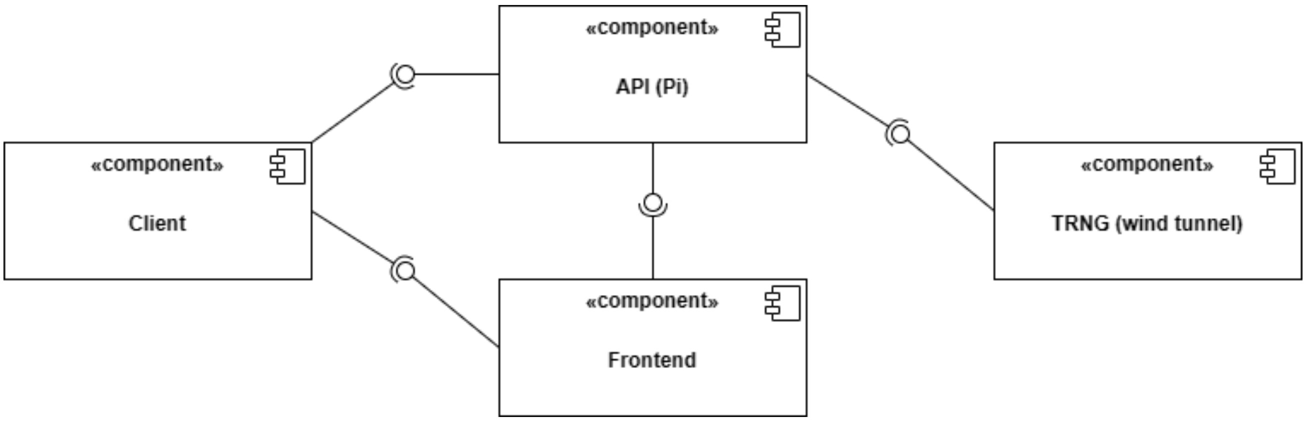
Once the hardware device is initialized, it continuously generates random bits, which are buffered in the serial input buffer. When a request for random numbers is received, the Flask application reads the random bits from the buffer. These bits are then parsed and converted into hexadecimal values, 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.

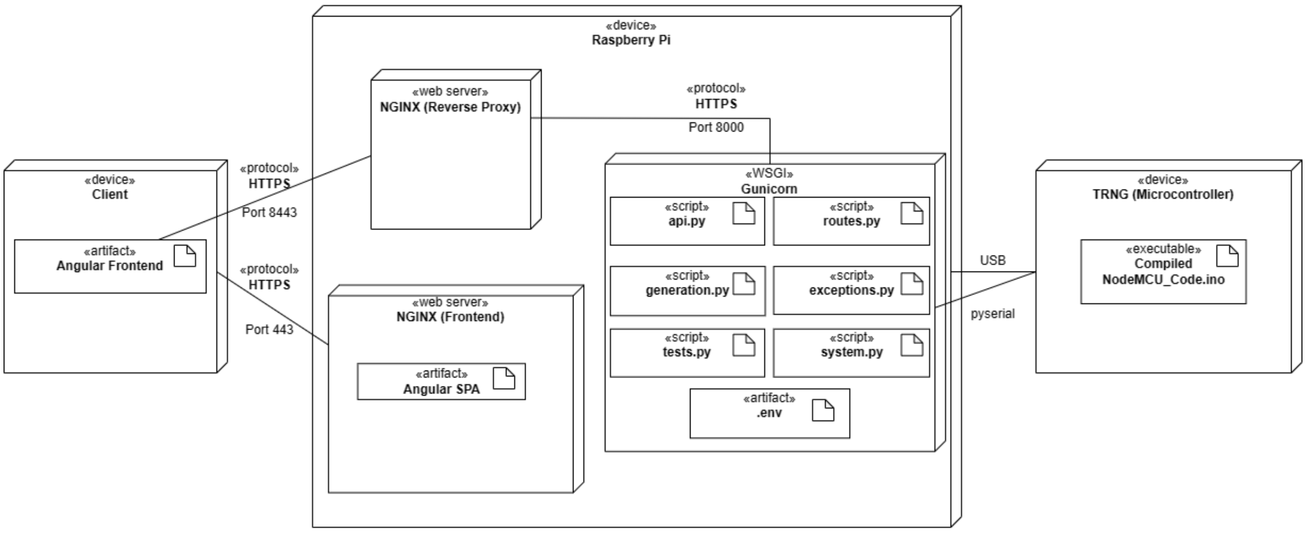
On the frontend side, the application is developed using Angular, a powerful framework for building single-page applications (SPAs). The Angular frontend is hosted on an Nginx server listening on port 443. Nginx is recognized for its stability, rich feature set, and efficient resource consumption, making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



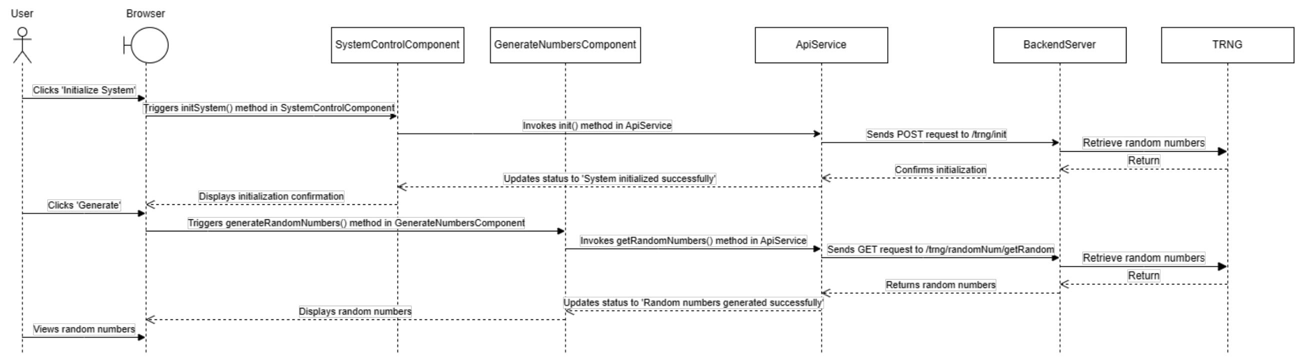
UML Component Diagram

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


User
Browser
SystemControlComponent
GenerateNumbersComponent
ApiService
BackendServer
TRNG
Clicks 'Initialize System'
Triggers initSystem() method in SystemControlComponent
Invokes init() method in ApiService
Sends POST request to /trng/init
Retrieve random numbers
Return
Confirms initialization
Updates status to 'System initialized successfully'
Displays initialization confirmation
Clicks 'Generate'
Triggers generateRandomNumbers() method in GenerateNumbersComponent
Invokes getRandomNumbers() method in ApiService
Sends GET request to /trng/randomNum/getRandom
Retrieve random numbers
Return
Returns random numbers
Updates status to 'Random numbers generated successfully'
Displays random numbers
Views random numbers

UML Sequence Diagram

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.

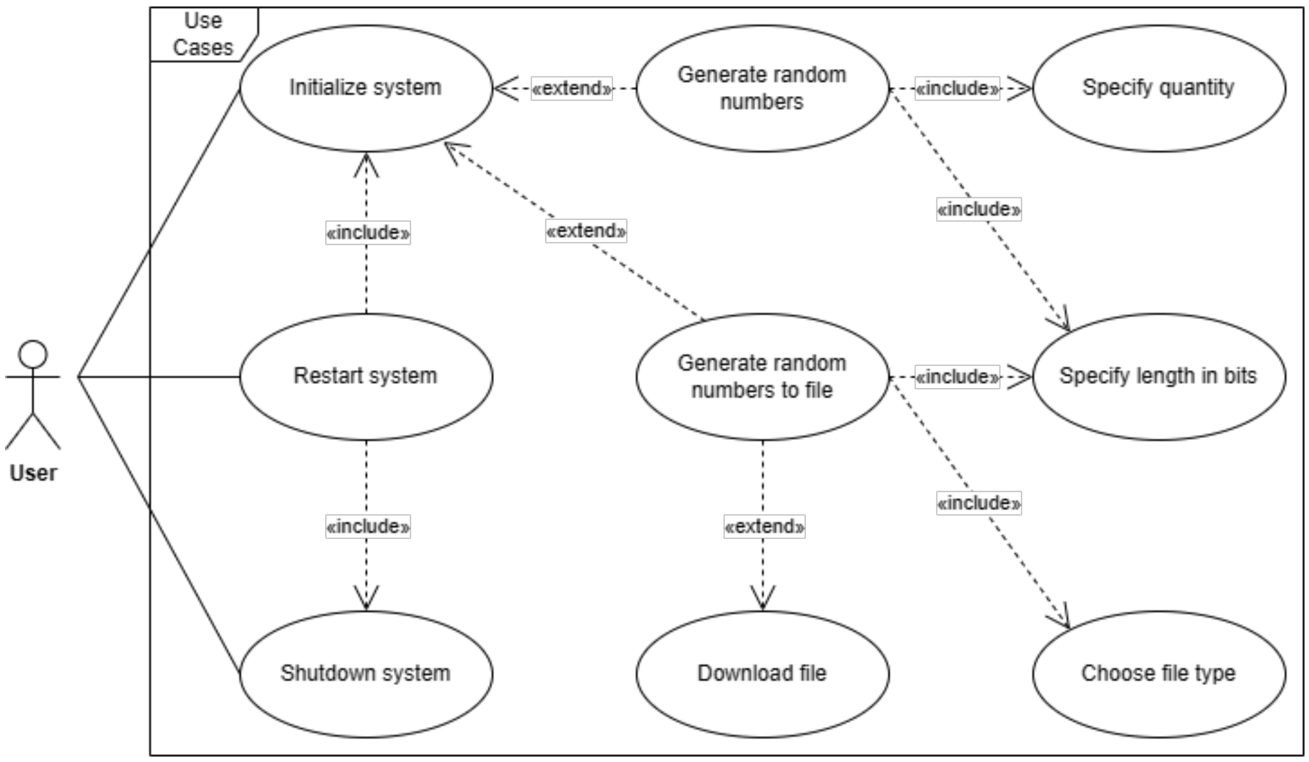
INTERFACE SPECIFICATIONS

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 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:

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

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

$$\frac{800}{60} \approx 13.33333333 \; (\frac{seconds}{total\ failure\ test})$$

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

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

which leads us to

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

If we take the reciprocal of this number, we get

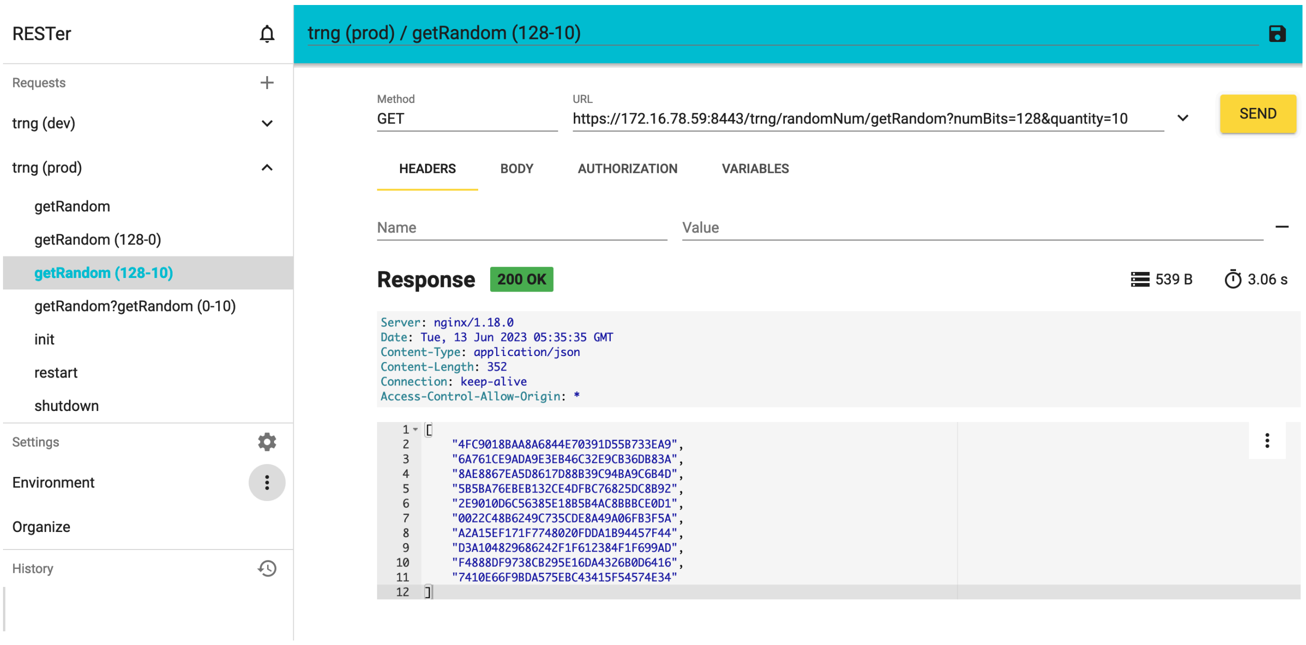
$$0.3825^{-1} \approx 2.6144$$

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.



Method: GET
URL: https://172.16.78.59:8443/trng/randomNum/getRandom?numBits=128&quantity=10

```
Server: nginx/1.18.0
Date: Tue, 13 Jun 2023 05:35:35 GMT
Content-Type: application/json
Content-Length: 352
Connection: keep-alive
Access-Control-Allow-Origin: *
```

```
[
    "4FC9018BAA8A6844E70391D55B733EA9",
    "6A761CE9ADA9E3EB46C32E9CB36DB83A",
    "8AE8867EA5D8617D88B39C94BA9C6B4D",
    "5B5BA76EBEB132CE4DFBC76825DC8B92",
    "2E9010D6C56385E18B5B4AC8BBBCE0D1",
    "0022C48B6249C735CDE8A49A06FB3F5A",
    "A2A15EF171F7748020FDDA1B94457F44",
    "D3A104829686242F1F612384F1F699AD",
    "F4888DF9738CB295E16DA4326B0D6416",
    "7410E66F9BDA575EBC43415F54574E34"
]
```

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 - Deployment).

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.

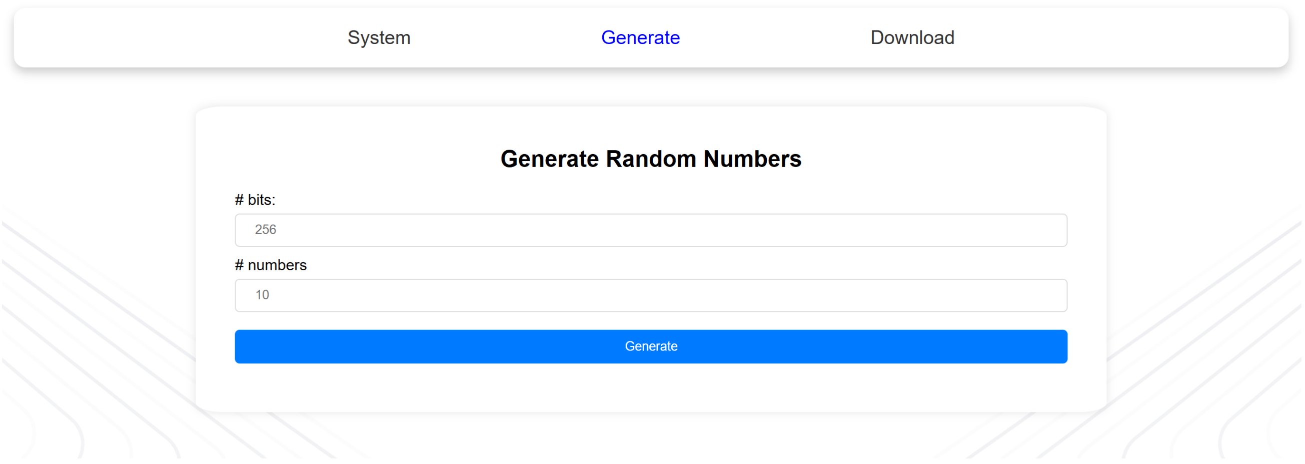
Moreover, the user interface includes a convenient feature for downloading the file generated by the generateTestdata endpoint. By clicking the designated "Download File" button, users can conveniently retrieve the generated file for further analysis or integration with other applications.

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

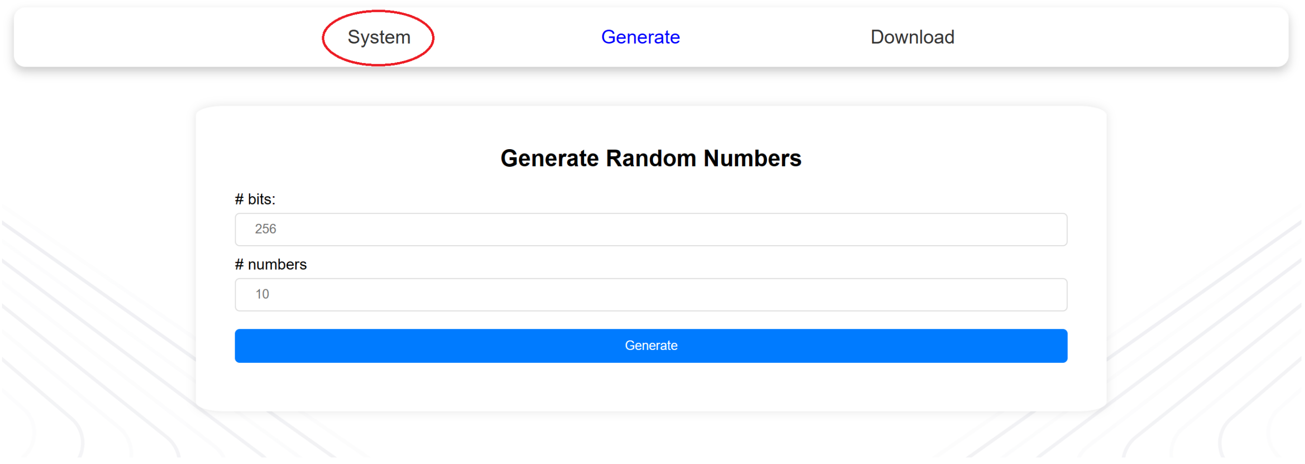
STEP BY STEP INSTRUCTIONS

GENERATE RANDOM NUMBERS

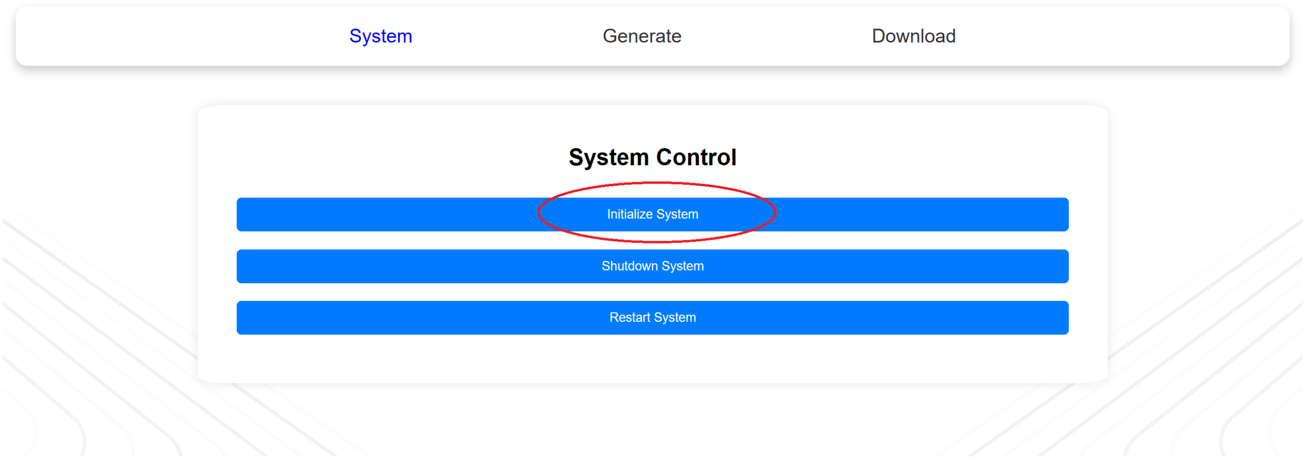
The Angular SPA provides a clear and simple graphical user interface (GUI) to generate random numbers. Once you have set everything up correctly, you will be able to use the Angular frontend to generate numbers. You can access the GUI by typing in the static IP address of the Raspberry Pi in your Browser's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



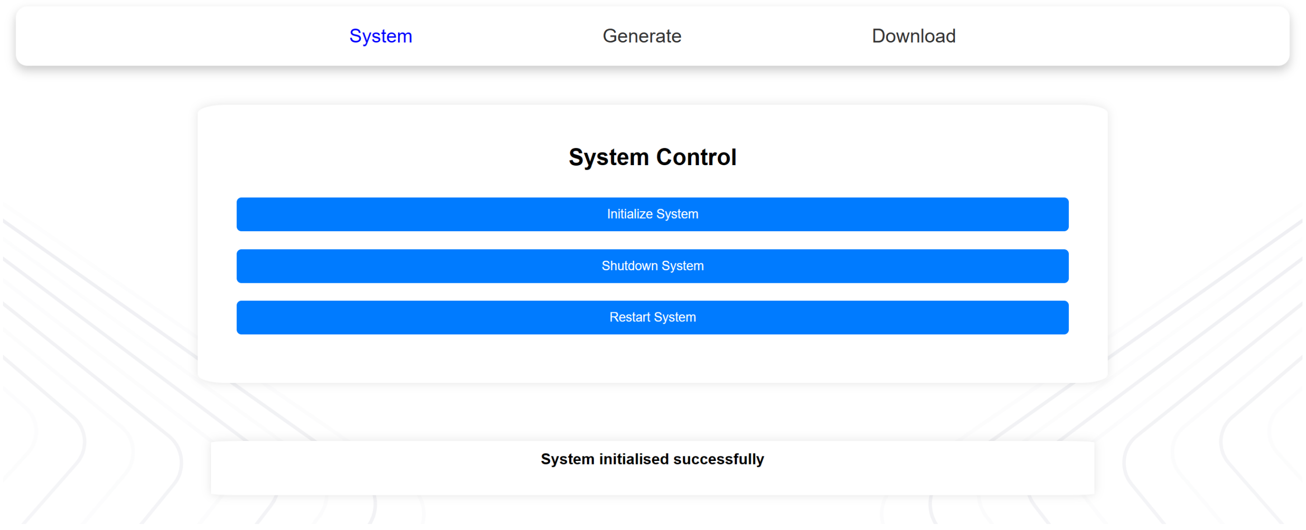
Before generating any numbers, initialize the system (TRNG) by navigating to the **System** tab.



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.



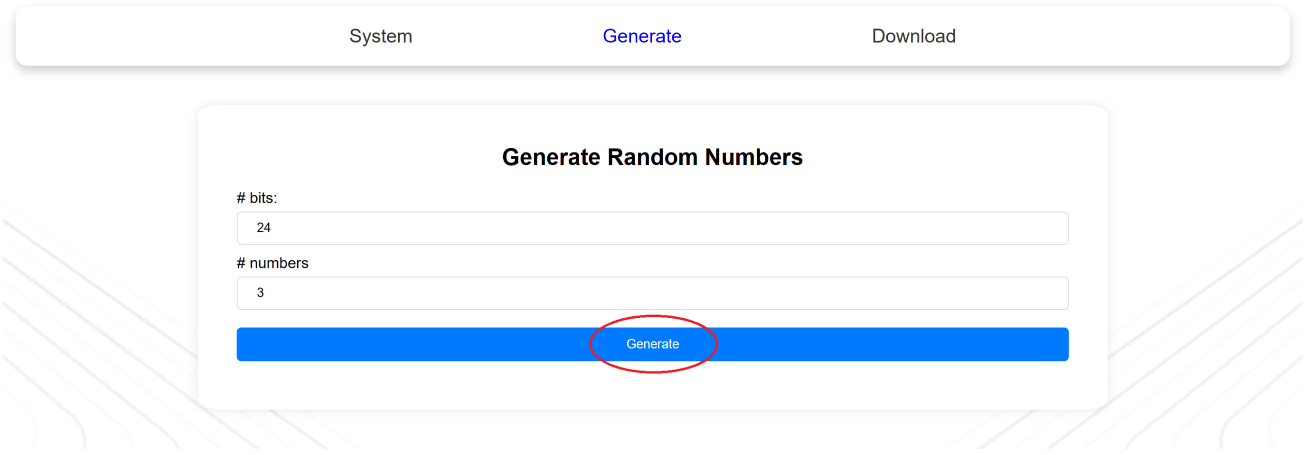
After clicking **Initialize**, a status bar will appear below the buttons, indicating the system is being initialized. Wait for a few seconds until a confirmation message appears, indicating a successful initialization.



System initialised successfully

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

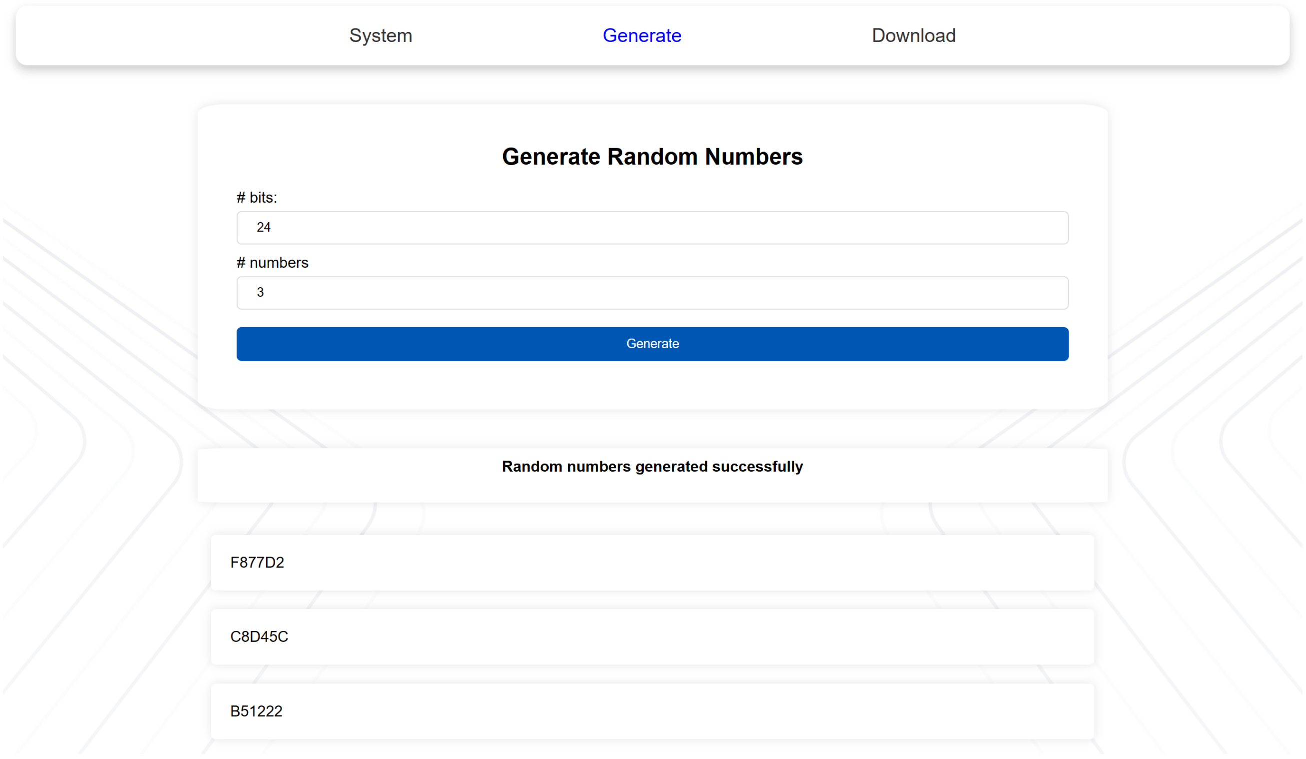
# numbers. The default value for the # bits field is set to '128', representing the length of each random number in bits. For example, if you want to generate **24**-bit numbers, simply enter '24' in this field. The # numbers field specifies the quantity of numbers to generate, with a default value of '10'. In this example, we will generate '**3**' numbers.



# bits: 24

# numbers 3

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.



# bits: 24

# numbers 3

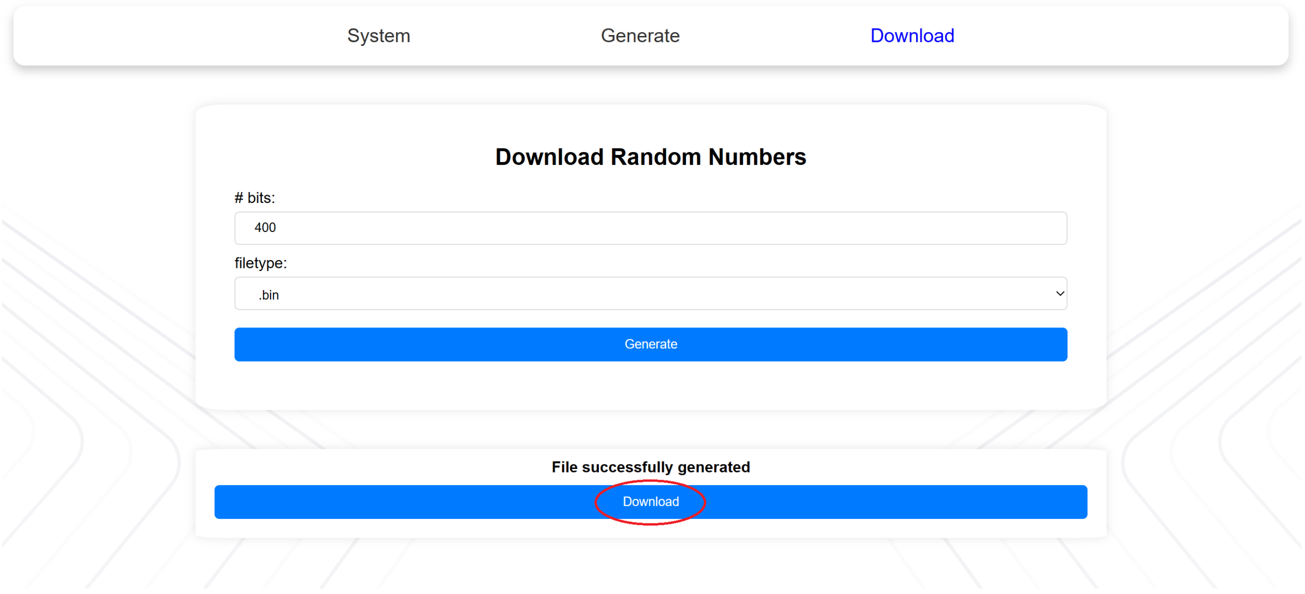
Random numbers generated successfully

```
F877D2
C8D45C
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.



Download Random Numbers

bits: 400

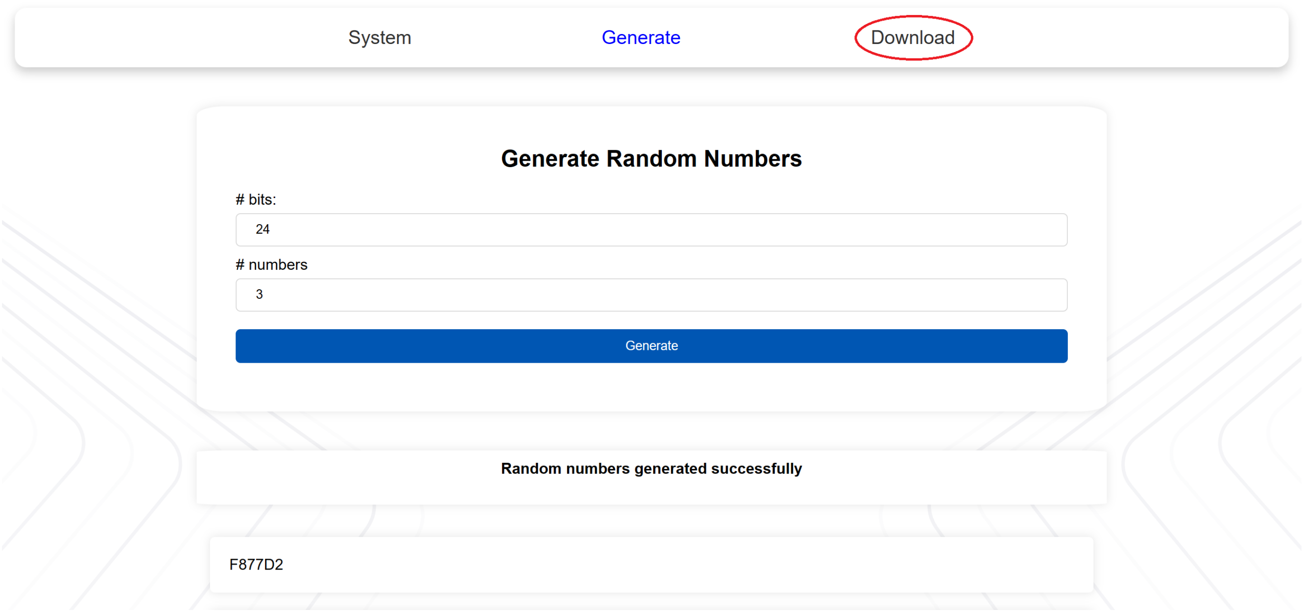
filetype: .bin

File successfully generated

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.



Generate Random Numbers

bits:
24

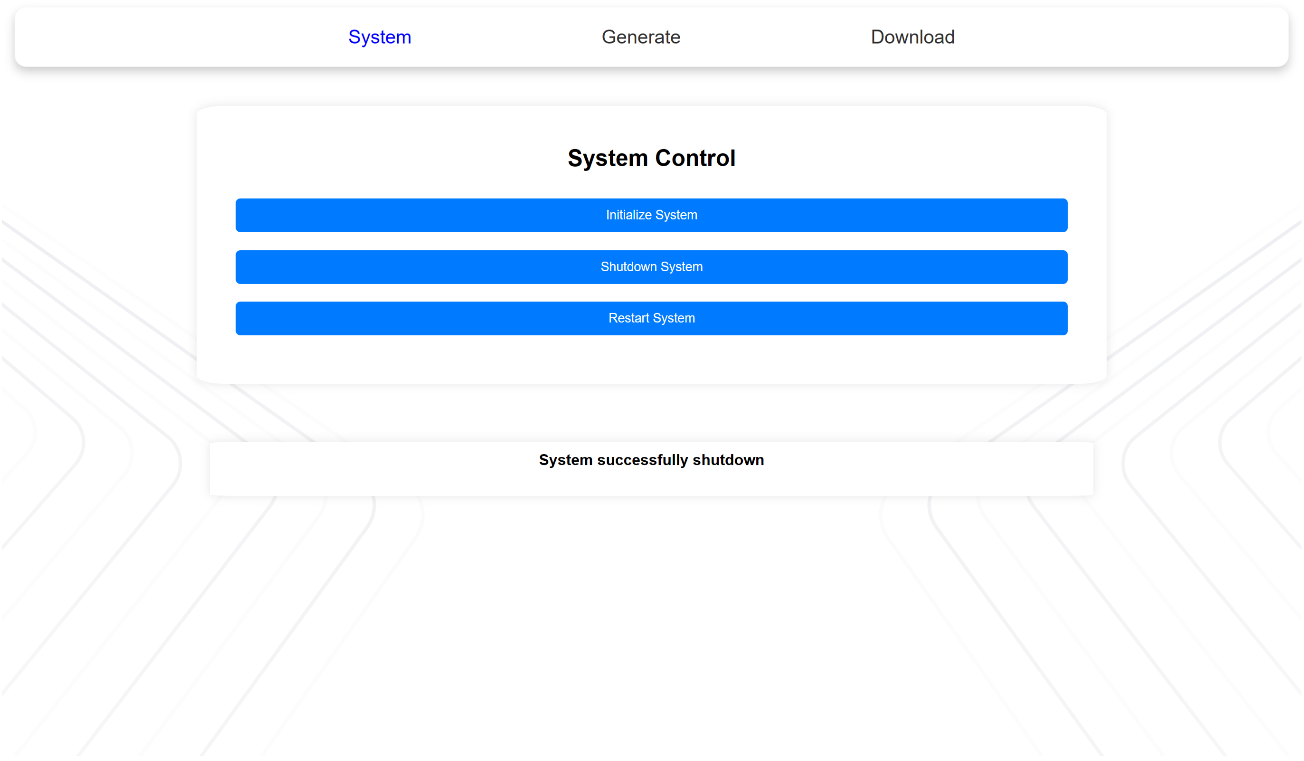
numbers
3

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 Control

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.

APPENDIX A - ENDPOINTS

APPENDIX B - ENVIRONMENT SETUP