Assignment 3 – Creative

Yi-Heng Liu (Ian Liu)
Student ID: 2276896
ianliutw@uw.edu

Background

Having reviewed the specifications of the Raspberry Pi 4 Model B, I am excited to create an optimal environment that meets my future development needs. No matter I am going to use the Raspberry Pi as a DNS server, forward proxy, or API server, I am planning to use Python 3 to build my projects. To achieve this, my top priority will be setting up a Python environment that is compatible with version 3.11, the latest stable release as of February 2023, along with essential packages and settings to streamline the development process. Since I am new to the embedded world and this is the first time I am using Raspberry PI, I will focus on determine if it is viable to use Raspberry Pi 4 as a portable developing machine and eventually as a deploying machine in my local network. This report will be an extension to the characterization results in assignment 2 and a guide to start Python development on the board.

Once the development environment is set up, I plan to showcase a project on the device that does not require any hardware extensions, but still explores the capabilities of the Raspberry Pi. This project will serve as a fun opportunity to set up the environment and further explore what the Raspberry Pi 4 is capable of. As a backend engineer experienced in developing API servers, I am curious to see whether deploying an API server on the Raspberry Pi 4 within my home network is a viable option. Through this experiment, I hope to test the performance of a Python project on the board and challenge myself to create new functions for an in-house API server. I will also try to brainstorm some ideas that might be useful for this setup.

Goals

- **Python Environment Setup**
  Create a Python dev environment supporting Python 3.11, isolated environment, and package management. Repeatable process for new machines.
- **Python API Server Project**
  Developing a Python project using the FastAPI package and deploying the code onto the board. Enhancing the API server by incorporating additional functionalities.

Note: The project will be based on OS Raspbian 11 (bullseye).
Note: Python codes will follow PEP 8 standards.
Note: All the reference links will be documented in the References section at the end of the report.

Python Environment Setup

I plan to use Python 3 for my upcoming projects and aim to create an optimal development environment. As per the characterization, the Raspberry Pi 4 Model B comes with Python 3.9.2. To upgrade it to the latest stable release of Python 3, i.e., version 3.11, I will use asdf.

To avoid cluttering the system with conflicting dependencies, I will use pipx to set up isolated environments for my projects. This way, each project will have its own dependencies, and I can avoid version conflicts. Moreover, you can run multiple versions of the same package without any conflicts.

As for package and dependency management, I prefer using Poetry. It simplifies managing and distributing Python packages while ensuring consistent and compatible dependencies across different projects.

asdf

asdf is a powerful tool that can manage multiple runtime versions of various programming languages, including Python. It enables us to install and switch between multiple versions of Python effortlessly.

Installation:

```
Clone the git repo:
git clone https://github.com/asdf-vm/asdf.git ~/.asdf --branch v0.11.2


Edit ~/.bashrc and add the followings codes:
sudo vim ~/.bashrc

Add the following codes:
. $HOME/.asdf/asdf.sh
. $HOME/.asdf/completions/asdf.bash


Source .bashrc:
source ~/.bashrc


Install Python 3:
sudo apt install build-essential libssl-dev zlib1g-dev libbz2-dev \
libreadline-dev libsqlite3-dev wget llvm libncurses5-dev libncursesw5-dev \
xz-utils tk-dev libffi-dev liblzma-dev libedit-dev

asdf plugin add python
asdf list-all python  # See all available versions
asdf install python 3.11.2
asdf global python 3.11.2

pip install --upgrade pip

python --version
python -m pip --version
```

pipx

Once you have installed the desired version of Python using asdf, you can use pipx to install and run Python applications in isolated environments.

Installation:

```
Installing using pip:
python -m pip install --user pipx
python -m pipx ensurepath
pipx -version


Add auto-completion:
pipx completions
sudo vim ~/.bashrc

Edit the ~/.bashrc and add:
# pipx
eval "$(register-python-argcomplete pipx)"

Source .bashrc:
source ~/.bashrc
```

Poetry

Poetry is my preferred tool for packaging and dependency management in Python. It offers a user-friendly command-line interface that simplifies the process of creating and managing virtual environments for your projects.

Installation:

```
Install Rust (Rust >=1.48.0 is required for cryptography package):
asdf plugin add rust
asdf install rust 1.67.1
asdf global rust 1.67.1

rustc --version


Install dependencies:
apt-get install build-essential libssl-dev libffi-dev python-dev
pip install cryptography

Installing using pipx:
pipx install poetry
poetry --version


Enable tab completion for Bash:
sudo install -m 777 /dev/null /etc/bash_completion.d/poetry.bash-completion
poetry completions bash > /etc/bash_completion.d/poetry.bash-completion


(Optional) Export PYTHON_KEYRING_BACKEND variable to avoid `Failed to create the
collection: Prompt dismissed..` issue:
echo 'export PYTHON_KEYRING_BACKEND=keyring.backends.null.Keyring' >> ~/.bashrc
source ~/.bashrc


(Optional) Set poetry to create the virtualenv inside the project's root directory:
poetry config virtualenvs.in-project true
```

Summary

After conducting a thorough characterization process, it is evident that Raspbian OS is a well-built and fully-functional Linux distribution that offers a wide range of features for developing various applications.

In this session, my objective is to assess whether setting up the development environment on Raspbian is comparable to that of my desktop computers. While I anticipate a seamless setup due to Raspbian's extensive array of features and software packages, I am prepared to troubleshoot any potential issues that may arise during the process.

Notably, the procedures used to establish the Python 3.11 environment are both developer-friendly and highly repeatable. Additionally, I also installed Rust 1.67 with minimal effort while installing the required packages for Poetry. Consequently, the machine now has Rust, opening up a range of new programming capabilities for future development work.

Discussion

- Characterization Self-Assessment for Assignment 2

The characterization results in Assignment 2 were consistent with my findings. The Raspberry Pi 4 Model B and Raspbian OS provide a solid foundation for Python projects.

- Applicability for Limited or Mass Production

For limited production, the installation steps are straightforward and easy to replicate. However, for mass production, I would create a custom image that includes the pre-installed and configured environment. This would allow for efficient and consistent deployment by simply burning the image to SD cards.

- Enhancements

In addition to the Python environment and supporting packages, I found asdf to be a useful tool for developing in various languages. For instance, I was able to install Rust quickly. This tool also facilitates the installation of other popular languages such as Java, Go, and many others. Moreover, for development or deployment purposes, it is worth considering using docker.

- Recommended enhancement to the characterization stage

Explore preinstalled packages in the distribution.

- Time taken

3 hours for the first time setting up and documenting. Approximately 15~20 minutes for repeating the documented process.

Python API Server Project

In this section, I am building an RESTful API server from scratch and keep adding some fun functionalities to it. Documentation of the example endpoints will be provided at the end of the section.

Opening the port

We will connect to the endpoints of API server within our private network. To manage the open ports, we can use ufw.

```
Install the package:
sudo apt-get update
sudo apt-get install ufw


Enable service and open desired port:
sudo ufw enable
sudo ufw allow 8000/tcp  # for the project
sudo ufw allow 22  # for the ssh
sudo ufw status
```

Now port 8000 and 22 are opened for project and ssh connections.

Installing API server packages

There are two packages that are essential for building the API server. In this project, I will use poetry to install and manage all packages. I am installing this with Poetry and I will not show the detailed commands for the installation. The relating commands can be found in Poetry documentation and I will submit my codes with this report, which will includes all my dependencies and their versions. Other dependencies may be installed during the development.

Core Packages:

1. FastAPI

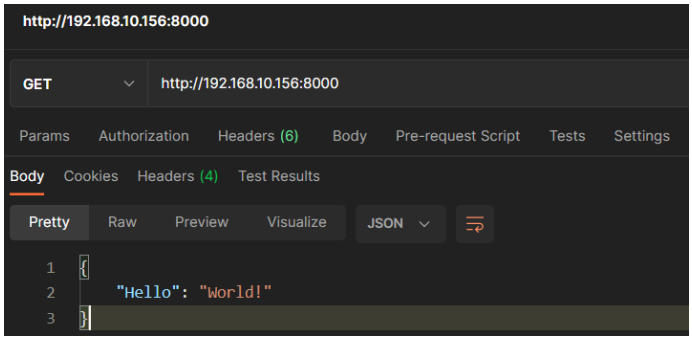
FastAPI is a Python web framework designed for building APIs with high performance and ease of use in mind. It features automatic data validation, API documentation generation, and built-in support for async/await syntax.

2. Uvicorn

Uvicorn is a lightning-fast ASGI server implementation, built on top of the asyncio event loop. It is a popular choice for running Python web applications with high performance and concurrency requirements.

Demonstration – Basic

I will demonstrate some basic function to show that the server is working well. In my local network, I can reach the Raspberry PI machine with 192.168.10.156. I used ssh to connect and develop on the board. Now since we have a server running and listening on port 8000, we can test if we can reach it as well:



Now the server is responding, we can add more functionality to it.

Demonstration – Creative

The purpose of this section is to explore creative extensions to the server that can add value for local network deployment. The goal is to enable automation of periodic tasks such as backup operations or health checks for other services, or to invoke one-time tasks by calling API endpoints. This will enhance the functionality of the server and make it a more useful addition to your local network.

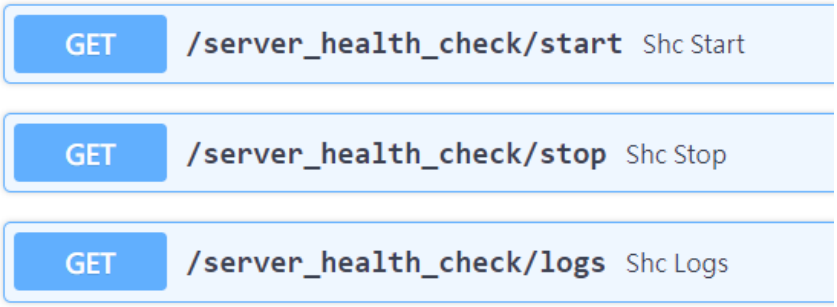
Note: To see the API documentation, please visit the following link: http://YOUR_SERVER_IP:8000/docs. The server needs to be running.

1. Server Health Check

One possible solution is to allow the machine to perform periodic checks on server health via the internet. In the provided example code, a simple script continuously makes API calls to OpenAI's Davinci engine, which hosts the GPT-3.5 model. Although this selection was made randomly, it's a well-known website that is frequently used. When an endpoint is called to start the service, the server will initiate a thread that monitors the server's availability and logs the results to a file. Additionally, the program could be extended to send email or text notifications in case of a failed health check. Ideally, the program should check multiple servers every hour and report failures to the appropriate channels.

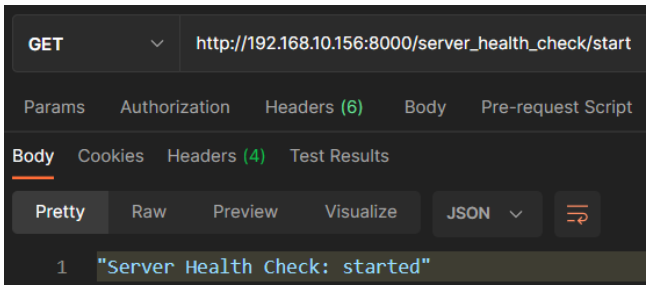
Note: To try this example, please find the codes in the attachment.
Note: You will need an API key from OpenAI, please refer to:
https://help.openai.com/en/articles/4936850-where-do-i-find-my-secret-api-key

Endpoints (see documentation in codes for details)



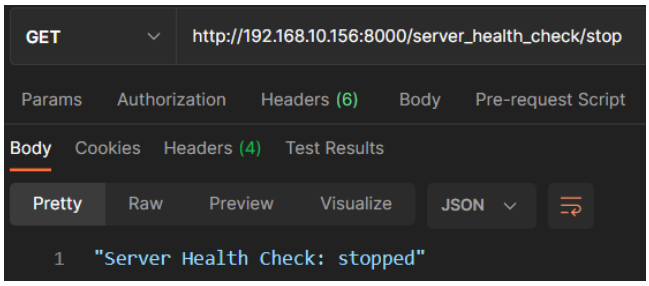
- /server_health_check/start

Start the server health check service.



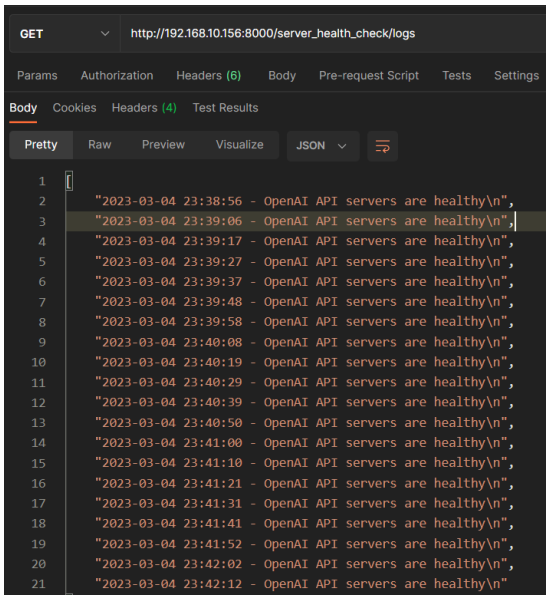
- /server_health_check/stop

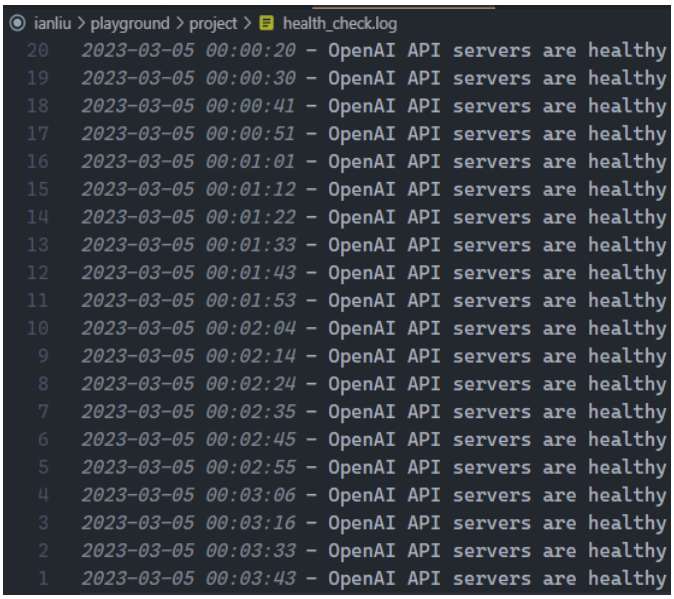
Stop the server health check service.



- /server_health_check/logs

Get the server health check logs. The number of records is specified in config.py.





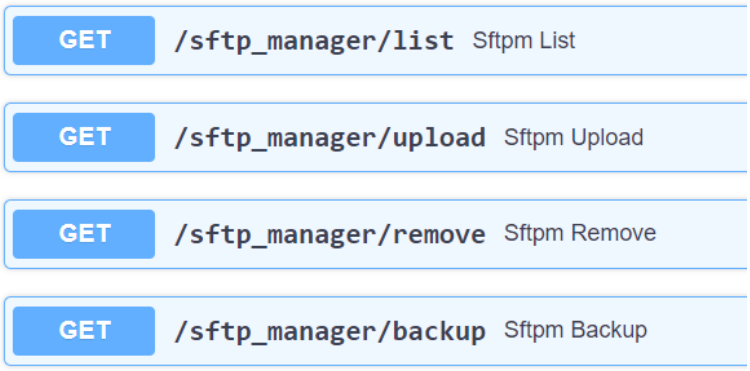
2. SFTP Operations

Working with FTP operations, like backing up or updating files, can often feel tedious, particularly in commercial or administrative settings where regular uploading of audit records to an operation server via FTP is required. In this example, I will show how to connect to an FTP server using an API server and perform several simple operations using SFTP protocol for added security. Although this is an exercise for a hypothetical requirement, it may prove valuable in real-world scenarios for addressing various problems.

In the provided code example, I used my account to log in to both the public SFTP demo server and the UW SFTP server for testing purposes. The basic operations I included are listing a SFTP server's directories, uploading and removing a file to and from a SFTP server, and backing up and removing files under a specific directory on a SFTP server to free space.

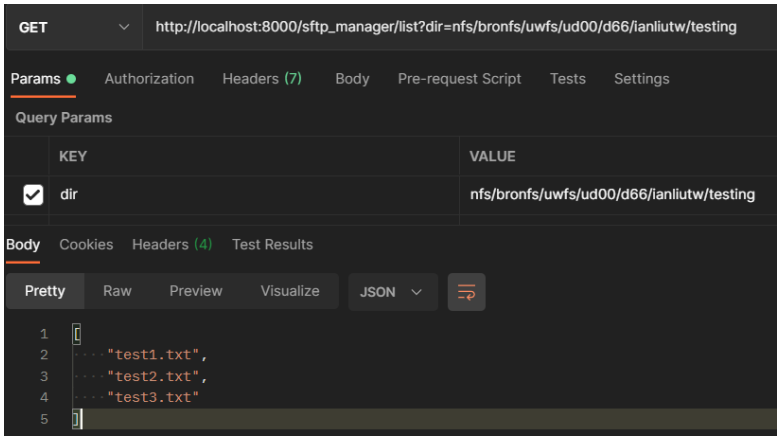
Note: To try this example, please find the codes in the attachment.

Endpoints (see documentation in codes for details)



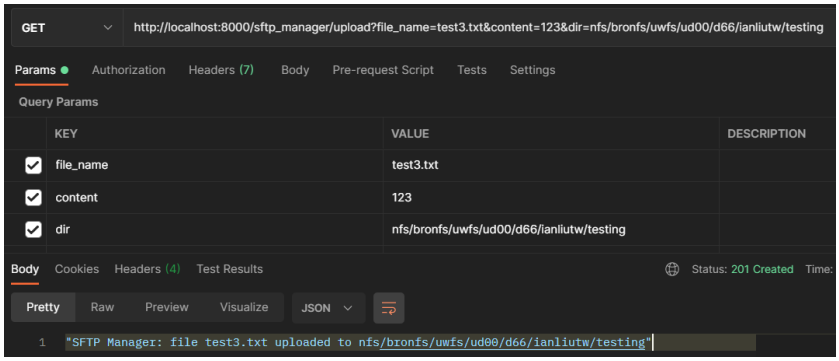
- /sftp_manager/list

See the files under a directory on the server.



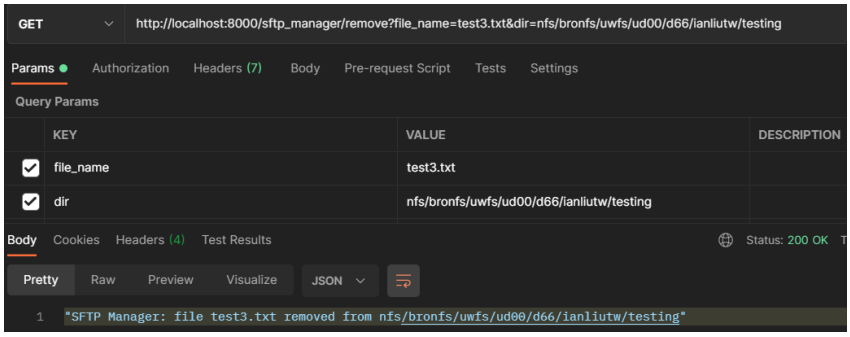
- /sftp_manager/upload

Create a file under a directory on the server.



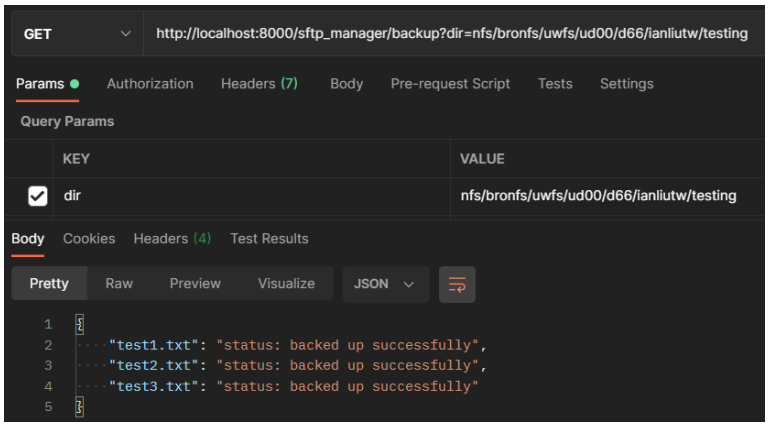
- /sftp_manager/remove

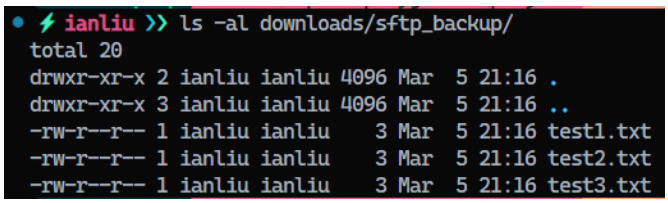
Remove a file under a directory on the server.



- /sftp_manager/backup

Back up and remove files under a directory on the server to a directory on the board.





Note: In the config file, you can specify the backup directory on the board and the file size limit for backing up files.

Note: Automating the backup process is a good practice if you need regular backups.

Summary

Although smartphones can accomplish some tasks, I find it enjoyable to have an API server that can be customized with added functionalities. The advantage of having a local server instead of a cloud-based one is that it consumes less energy, which translates to lower costs. Additionally, since the server remains connected to the Internet, it is possible to integrate other APIs into it to improve its capabilities.

In terms of security, API servers usually have various authentication methods to protect the endpoints. However, since the API server on the board is within a private network, I have not implemented any protection measures. If security is a concern, it would be prudent to upgrade the server's security.

Extending the functionality of the server is the most challenging yet enjoyable part of this project. While basic ideas can be implemented, the Raspberry Pi 4 hardware is capable of more complex and resource-intensive tasks. However, it is important to monitor the board's temperature since high temperatures are a known issue for this model.

As the project progresses, it is essential to consider the potential for future upgrades and improvements. By doing so, it is possible to enhance the server's capabilities and ensure that it remains useful over time.

Discussion

- Characterization Self-Assessment for Assignment 2

Based on my characterization in assignment 2, I discovered that the Raspberry Pi is a highly capable mini-computer with impressive performance potential. However, I had some doubts about how similar the Raspbian OS distribution is to popular distributions like Ubuntu. After researching this topic, I found that there are only minor differences between Raspbian and other Linux distributions, and most common packages come pre-installed. As a result, I feel that my characterization was thorough and accurate, and I did not encounter any significant challenges during setup or development.

- Applicability for Limited or Mass Production

When working on projects with limited production, it's simple to clone the code and set up individual servers on each board. However, for mass production, it's more efficient to containerize the code and use Docker to manage version control during deployment. Since this is a software project, it's also possible to create an image of the current storage state and use it to burn SD cards for distribution purposes.

- Enhancements

For this project, I'm utilizing Python 3.11. Compared to Python 3.10, it offers a 10-60% improvement in speed. I've also opted to use the FastAPI package, which is known for its high performance and ease of use. By combining these two technologies, I can significantly boost the server's execution speed, which compensates for the lower performance of previous Python versions. Additionally, FastAPI offers automatic interactive API documentation, which can be very user-friendly for those new to the server.

In the future, there are several additional enhancements that could be made to the server, such as turning it into a systemd service that automatically starts up when the system boots. This added convenience means that even if the machine is accidentally shut down, the server can quickly restart as soon as power is restored.

- Recommended enhancement to the characterization stage

Investigate the availability of ports and security functions within the OS distribution. Additionally, conduct further research into the software tools included in the distribution and determine whether it supports Docker.

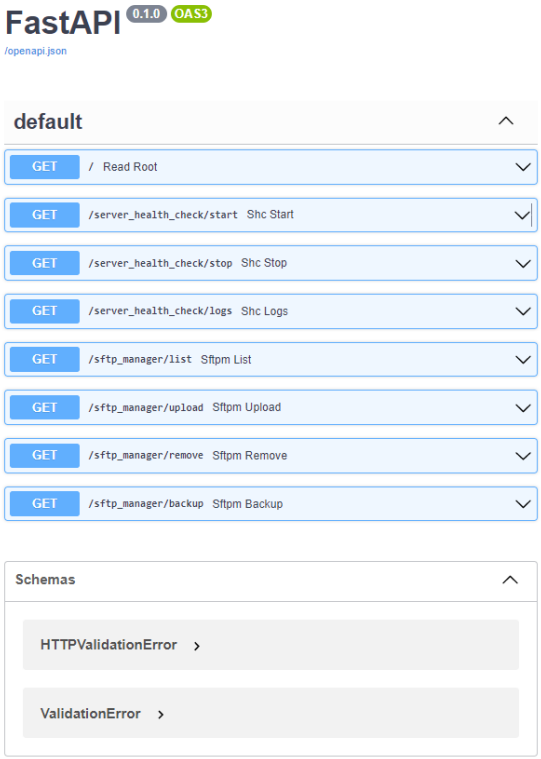
- Time taken

2 hours were spent setting up the network configuration and initiating the installation of the core packages for the project. It took 1 hour to build, run, and test the server for the first time. Brainstorming ideas and expanding/testing the functionality took 6 to 8 hours.

API Documentation

In the preceding sections, I presented a brief explanation and demonstration of the endpoints. The test results are displayed in Postman.

FastAPI offers built-in support for generating automatic interactive API documentation. To view the documentation, visit http://YOUR_SERVER_IP:8000/docs while the server is running. The documentation is presented in Swagger UI format.



Conclusion

The Raspberry Pi 4 Model B is a powerful and versatile single-board computer that is widely used for a variety of projects. One of the great things about the Raspberry Pi is that it can be used for both development and deployment, and it is relatively easy to set up a Python development environment on the device.

There are several benefits to setting up a Python development environment on a Raspberry Pi 4 Model B. For example, it provides a separate and portable machine that can be used for side projects. This is especially useful if you want to work on a project that is separate from your main computer, or if you want to work on a project that requires a different environment than your primary machine. Another benefit of using a Raspberry Pi 4 Model B for Python development is that it provides a solid foundation for other projects. Once you have your development environment set up, you can easily add tools and libraries to the device, and you can start exploring ideas.

Once you have your Python development environment set up on your Raspberry Pi 4 Model B, you can start experimenting with different projects. One idea is to deploy an API server in your home network. The Raspberry Pi 4 is well-suited for this kind of application because it can be an always-on machine in your local network that takes care of minor-load operations and periodic tasks. It is also possible to expose the server to the internet, which allows you to connect to it from anywhere. However, if you plan to expose your Raspberry Pi 4 to the internet, it is important to take security precautions. This includes setting up a reversed proxy server, such as Nginx or Apache, as well as using TLS transmission to encrypt your data.

In addition to deploying an API server, you can also extend your Raspberry Pi 4 to do other simple yet meaningful tasks. For example, you can use it to host a website, create a home automation system, or run a media server. The Raspberry Pi 4 provides a low-cost and energy-efficient solution for hosting these kinds of projects.

In conclusion, setting up a Python development environment on a Raspberry Pi 4 Model B is a great way to get started with experimenting with different projects. Although a server hosted in the cloud or on a PC can achieve the same goals, the Raspberry Pi 4 provides a fun and cost-effective solution that can be extended and deployed in a variety of settings. Whether you are working on a side project, setting up a home automation system, or experimenting with different concepts, it is a great tool to have in your toolkit.

References

- General

ChatGPT

https://chat.openai.com/chat

- Raspberry Pi

Raspberry Pi Documentation

https://www.raspberrypi.com/documentation/

Raspberry Pi 4 Model B - Schematics, Revision 4.0

https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-reduced-schematics.pdf

Raspberry Pi 4 Model B - Mechanical Drawings, PDF

https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-mechanical-drawing.pdf

- Coding standard

PEP 8 – Style Guide for Python Code

https://peps.python.org/pep-0008/

- Environment Setup

asdf

https://asdf-vm.com/

pipx

https://pypa.github.io/pipx/

Poetry

https://python-poetry.org/

- Python FastAPI Project

FastAPI

https://fastapi.tiangolo.com/

Uvicorn

https://www.uvicorn.org/

pysftp Documentation

https://pysftp.readthedocs.io/en/release_0.2.9/

Postman

https://www.postman.com/

OpenAI Help

https://help.openai.com/en/

Free Public SFTP Servers

https://www.sftp.net/public-online-sftp-servers

UW IT - Moving Files Using Secure FTP

https://itconnect.uw.edu/tools-services-support/storage-hosting/shared-web-hosting/getting-started/moving-files-using-secure-ftp/