Table of Contents

To IITD-AIA-FSM 2022,

Introduction

FastAPI is currently the go-to framework for building robust and high-performance APIs that scale in production environments.[1] FastAPI has gained a lot of popularity lately and saw a huge increase in user adoption among web developers, data scientists, and ML engineers.

FastAPI feature: Automatic Docs

Interactive API documentation and exploration of web user interfaces. One of the primary ones is, **Swagger UI**, with interactive exploration, call and test your API directly from the browser. FastAPI's syntax is simple and this makes it fast to use.[2]
It actually resembles Flask's syntax. Instantiating Flask and FastAPI apps are merely the same. However, unlike Flask, FastAPI doesn't come with an integrated webserver. FastAPI is specifically designed to build APIs. It has no responsibility to serve them.

Flask code:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
   app.run()
```

FastAPI code:

```
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
    uvicorn.run("hello_world_fastapi: app")
```

In Flask and FastAPI, path parameters are parsed from the path we pass to the route. With FastAPI, we add these parameters as arguments to the function.

Deployment using FastAPI

Prerequisites

Install the FastAPI module and requirements

Run the following command to install all the requirements and application-specific dependencies to build a Restful API using FastAPI.

```
pip install pip-tools
pip-compile requirements.in
```

This will generate a requirements.txt file in the same directory.

```
pip install requirements.txt
```

Folder structure

```
├───main.py
├───requirements.in
├───requirements.txt
├───pcb-fault-detection
│   ├───model
│   └───yolov5
│       ├───models
│       │   └───hub
│       └───utils
│           ├───aws
│           ├───flask_rest_api
│           ├───google_app_engine
│           └───loggers
│               └───wandb
├───training data
└───yolov5
    ├───.github
    │   ├───ISSUE_TEMPLATE
    │   └───workflows
    ├───data
    │   ├───hyps
    │   ├───images
    │   └───scripts
    ├───models
    │   ├───hub
    │   └───__pycache__
    ├───runs
    │   └───train
    │
    │       ├───exp
    │           └───weights
    ├───utils
    │   ├───aws
    │   ├───docker
    │   ├───flask_rest_api
    │   ├───google_app_engine
    │   ├───loggers
    │   │   ├───wandb
    │   │   │   └───__pycache__
    │   │   └───__pycache__
    │   └───__pycache__
    └───__pycache__
```

Modifications based on our requirements

This folder structure can be duplicated using the following commands:

```
git clone https://github.com/keivalya/FSM-INT-2022.git
```

Main change that needs to be carried out is in the `Codes/pcb-fault-detection/model` folder.
Update the PyTorch model (`best.pt`) present inside the directory with your created model.

Deployment

We use `gunicorn` in order to deploy our FastAPI, as it is easy to use, has tons of support and is highly compatible cross-platform.

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker main:app
```

If all the requirements are met properly, our application will be successfully deployed locally.

References

1. How to Deploy a Machine Learning Model with FastAPI, Docker and Github Actions | by Ahmed Besbes | Towards Data Science (https://towardsdatascience.com/how-to-deploy-a-machine-learning-model-with-fastapi-docker-and-github-actions-13374cbd638a)
2. Features - FastAPI official documentation (https://fastapi.tiangolo.com/features/)