3 FastAPI

CS 425
Web Applications Development

Alex Petrovici

Content

Introduction /

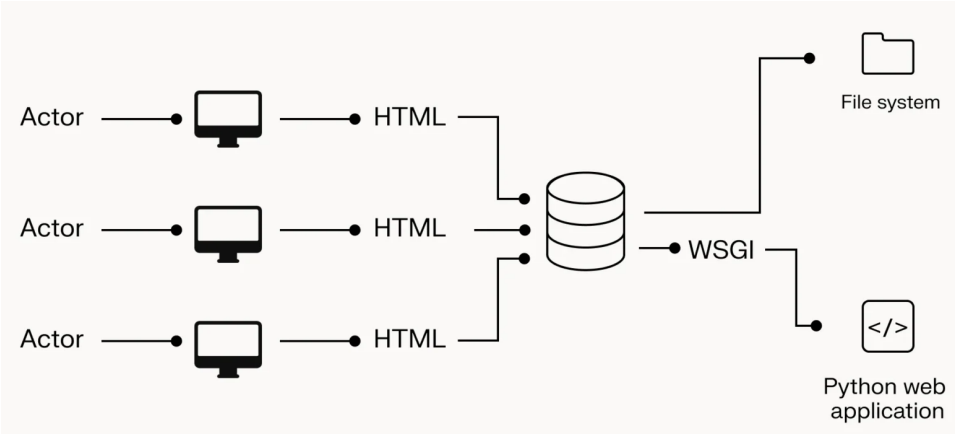
What is FastAPI ?

FastAPI is a modern, fast, web framework for building APIs with Python 3.8+.
Began in 2018 by Sebastián Ramírez, as the sole developer.

What are the core principles of FastAPI ?

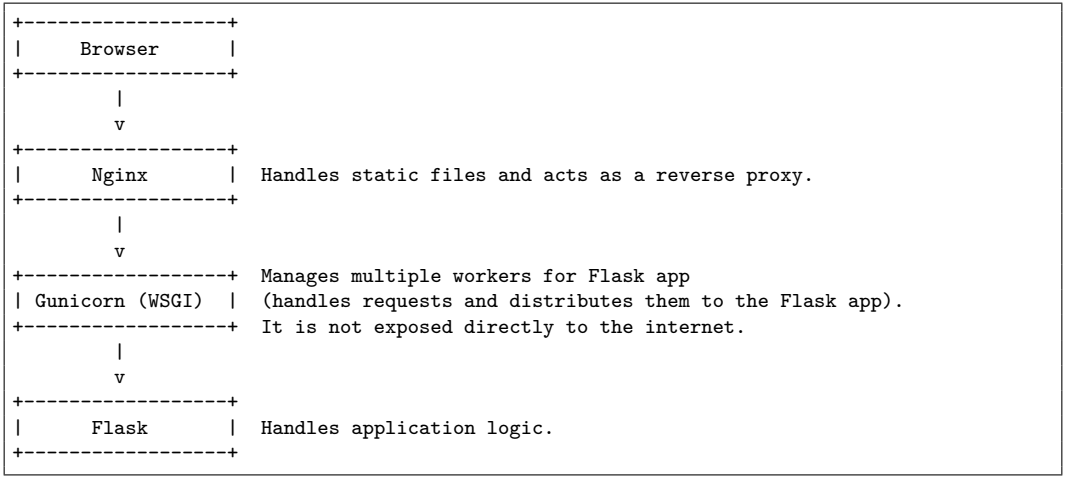
- Extensible. Uses Pydantic for data validation.
- Documentation. Automatically generated documentation for the API following the OpenAPI specification.
- Simple. Doesn't focus on rendering templates.
- Fast. Based on Starlette, a lightweight ASGI framework.

Introduction / SGI (Server Gateway Interface)

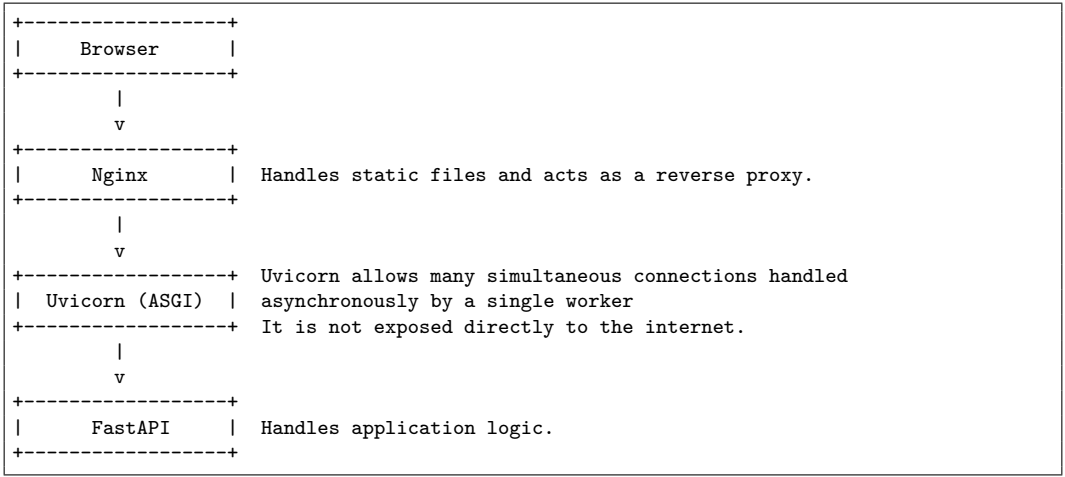


Source: https://www.liquidweb.com/blog/what-is-wsgi/

Introduction / SGI (Server Gateway Interface)



Introduction / SGI (Server Gateway Interface)



Introduction / SGI (Server Gateway Interface)

SGI (Server Gateway Interface)

- Werkzeug is a WSGI (Web Server Gateway Interface) utility library that provides the core HTTP handling for Flask request parsing, response generation, routing, and error handling. It's the foundation that lets Flask act as a lightweight framework rather than a full stack. Werkzeug operates synchronously.
- Starlette supports asynchronous I/O (async/await).

Endpoints /

main.py

```python
import uvicorn
from fastapi_app import create_app

if __name__ == "__main__":
    app = create_app()
    uvicorn.run(app, host="127.0.0.1", port=7000, log_level="info")
```

Endpoints /

fastapi_app.py

```python
from fastapi import FastAPI

def create_app() -> FastAPI:
    app = FastAPI(
        title="FastAPI App", version="0.1.0",
        description="A simple FastAPI application for testing purposes",
        docs_url="/docs",  # Swagger UI at /docs
        redoc_url="/redoc",  # ReDoc at /redoc
    )

    @app.get(
        "/",
        summary="Home endpoint",
        description="Returns a welcome message",
        response_description="A simple greeting message",
    )
    def home():
        """Get the home page with a welcome message."""
        return "Hello, FastAPI!"

    return app
```

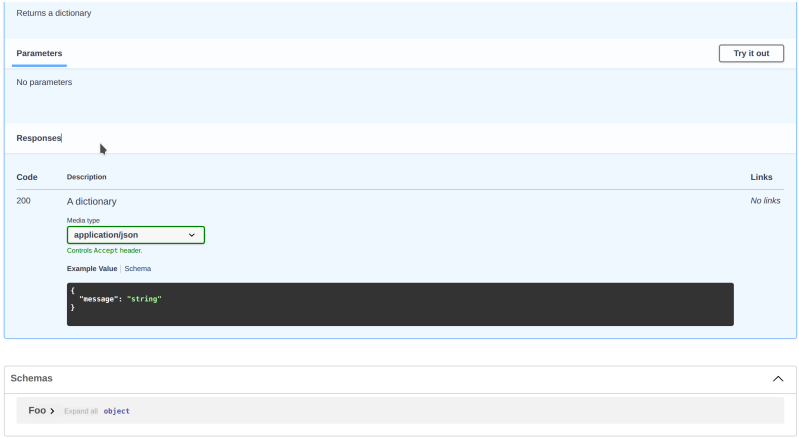
Endpoints / Enforcing Schemas

fastapi_app.py

```python
from pydantic import BaseModel


class Foo(BaseModel):
    message: str

...

@app.get(
    "/object",
    summary="Object endpoint",
    description="Returns a dictionary",
    response_description="A dictionary",
)
def get_object() -> Foo:
    """Object test endpoint."""

    return Foo(message="This is a simple FastAPI application.")
```

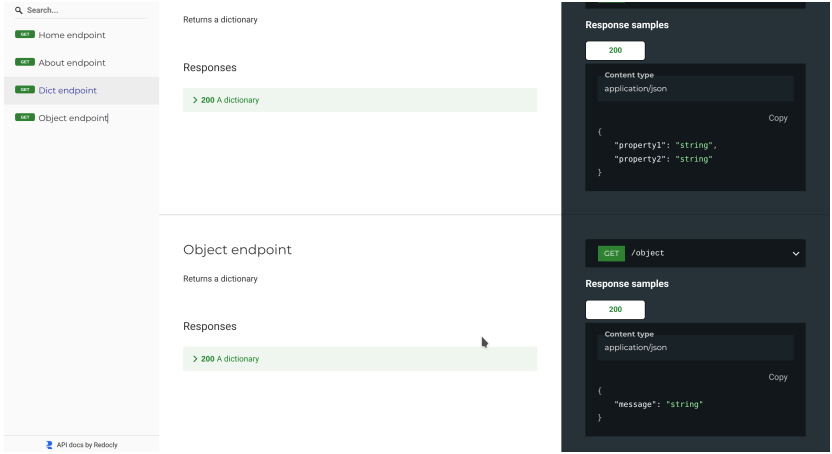
Documentation / Swagger

Available at `http://127.0.0.1:7000/docs`



Documentation / Redocly

Available at `http://127.0.0.1:7000/redoc`



Modularization using routes (blueprints) / Flask

```python
from flask import Flask, Blueprint

users = Blueprint("users", __name__)

@users.route("/profile")
def profile():
    return "User profile"

app = Flask(__name__)
app.register_blueprint(users)
```

Modularization using routes (blueprints) / FastAPI

```python
from fastapi import FastAPI, APIRouter

users = APIRouter()

@users.get("/profile")
def profile():
    return {"message": "User profile"}

app = FastAPI()
app.include_router(users)
```

Serving static and templates / Flask

```python
from app.blueprints.api import bp as bp_api
from app.blueprints.main import bp as bp_main
from app.config import Config
from flask import Flask

def create_app(config_class=Config) -> Flask:
    template_dir = Path(__file__).parent / "app" / "templates"
    static_dir = Path(__file__).parent / "app" / "static"
    app = Flask(__name__, template_folder=template_dir, static_folder=static_dir)
    app.config.from_object(config_class)

    app.register_blueprint(bp_main)
    app.register_blueprint(bp_api)

    return app
```

Serving static and templates / FastAPI

```python
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

def create_app() -> FastAPI:
    template_dir = Path(__file__).parent / "app" / "templates"
    static_dir = Path(__file__).parent / "app" / "static"

    app = FastAPI(
        title="code_fixer_ui", version="0.1.0",
        description="The front end of the Code Fixer framework.",
        docs_url="/docs", redoc_url="/redoc",
    )

    # Configure Jinja2 templates
    templates = Jinja2Templates(directory=str(template_dir))
    app.state.templates = templates

    # Mount static files
    app.mount("/static", StaticFiles(directory=str(static_dir)), name="static")
    app.include_router(bp_main)
    app.include_router(bp_api)
    return app
```

Serving static and templates / Flask

```python
@bp.route("/view/<string:id>")
def view_error_log(id):
    _id = UUID(id)
    item = db.get_error_log_by_id(id=_id, flavour=ErrorLog)

    if item is None:
        # Handle case where item is not found
        return render_template("tpl_view_error_log.html", item=None)

    enhanced_item: dict
    ...

    return render_template("tpl_view_error_log.html", item=enhanced_item)
```

Serving static and templates / FastAPI

```python
from fastapi.responses import HTMLResponse

@bp.get("/view/{id}", response_class=HTMLResponse)
def view_error_log(request: Request, id: str):
    _id = UUID(id)
    item = db.get_error_log_by_id(id=_id, flavour=ErrorLog)

    if item is None:
        # Handle case where item is not found
        return request.app.state.templates.TemplateResponse(
        "main/tpl_view_error_log.html",
        {"request": request, "item": None},
    )

    enhanced_item: dict
    ...

    return request.app.state.templates.TemplateResponse(
        "main/tpl_view_error_log.html",
        {"request": request, "item": enhanced_item},
    )
```