Why FastAPI?

INTRODUCTION TO FASTAPI

Matt Eckerle
Software and Data Engineering Leader

What is FastAPI?

Let's start with some terminology

1. **API:** Application Programming Interface - refers to web applications using the HTTP protocol to transmit structured data
2. **Web Application:** application that serves traffic over the web
3. **Web Framework:** software framework that helps build web applications

FastAPI is a fast way to build high-performance APIs using Python

FastAPI key features

- **Fast:** Very high performance
- **"Low code" and easy to learn:** Python annotations and type hints
- **Robust:** Production-ready code with autodoc
- **Standards-based:** Based on OpenAPI and JSON Schema

JSON Schema

[1] https://fastapi.tiangolo.com/

FastAPI vs. other Python web frameworks

Flask

- Build web-based (GUI) apps
- ORM optional

FastAPI

- Build APIs
- ORM optional

Django

- Build web-based (GUI) apps
- ORM built in

Key differences

- For APIs without database operations
- Data and machine learning transactions

Building our first web application with FastAPI

1. Install FastAPI

```
pip install fastapi
```

2. Create your app in `main.py`

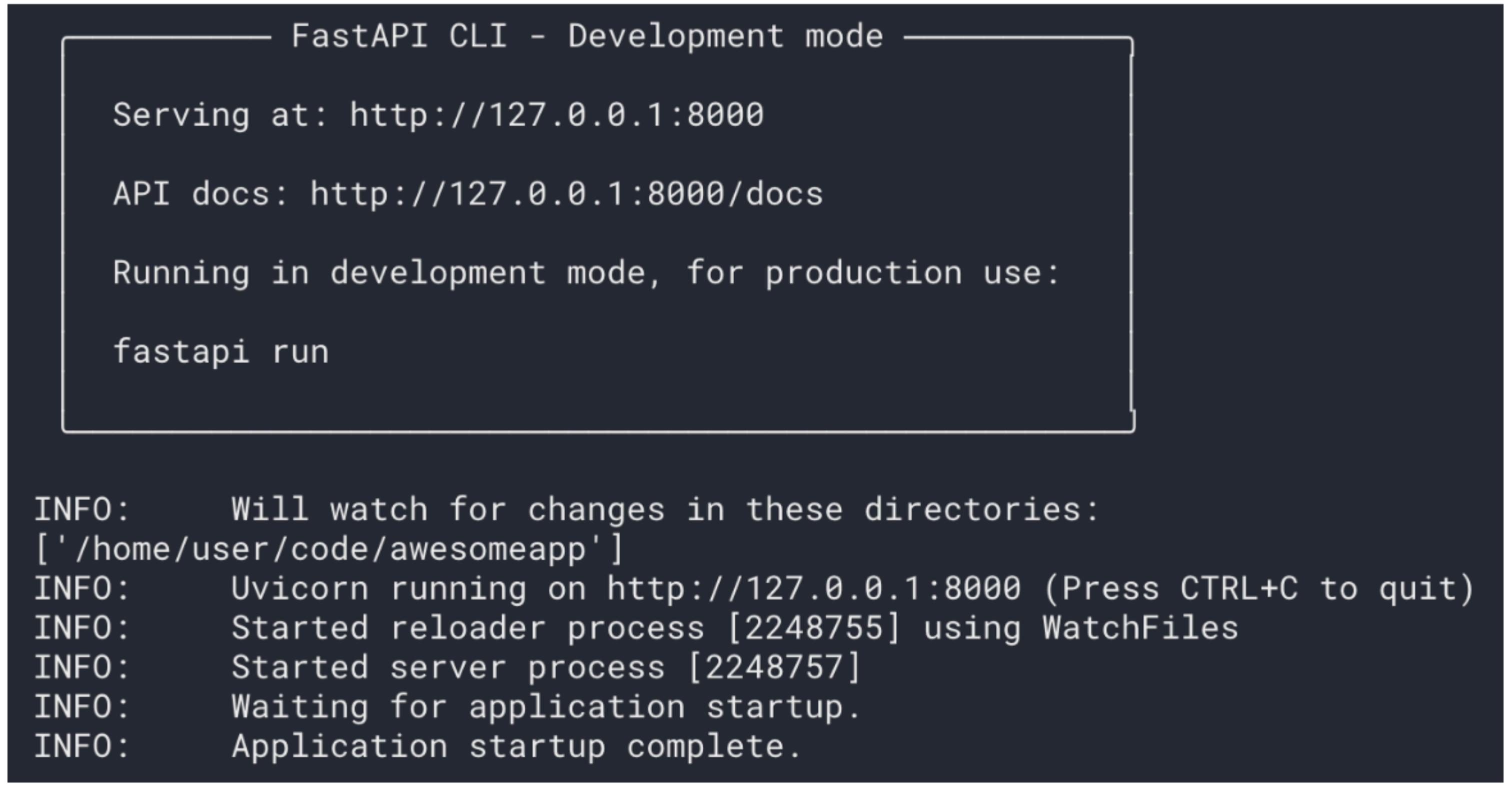
```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello World"}
```

3. Run the server

```
fastapi dev main.py
```



```
 ┌────────── FastAPI CLI - Development mode ──────────┐
 │                                                     │
 │  Serving at: http://127.0.0.1:8000                  │
 │                                                     │
 │  API docs: http://127.0.0.1:8000/docs               │
 │                                                     │
 │  Running in development mode, for production use:   │
 │                                                     │
 │  fastapi run                                        │
 │                                                     │
 └─────────────────────────────────────────────────────┘

INFO:     Will watch for changes in these directories:
['/home/user/code/awesomeapp']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [2248755] using WatchFiles
INFO:     Started server process [2248757]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Before we practice with FastAPI

Some notes

1. Can't run the FastAPI server with the "Run this code" button
2. Define server code in the Python editor as `main.py` instead
3. Run it from the terminal using the command `fastapi dev main.py`
4. Verify that the logs in the terminal show `Application startup complete.`
5. Stop the live server by pressing `Control + C` in the same terminal
6. You should install FastAPI in your own Python environment to get used to practicing there as well

Let's practice!

INTRODUCTION TO FASTAPI

GET operations

INTRODUCTION TO FASTAPI

Matt Eckerle
Software and Data Engineering Leader

GET operation review

HTTP protocol - several types of operations

- GET is the most common

Example: `https://www.google.com:80/search?q=fastapi`

The key parts of a GET request are:

- Host, e.g. `www.google.com`
- Port, e.g. `80` (default)
- Path, e.g. `/search`
- Query String, e.g. `?q=fastapi`

FastAPI GET operation

The simplest FastAPI application:

```python
from fastapi import FastAPI

# Instantiate app
app = FastAPI()

# Handle get requests to root
@app.get("/")
def root():
    return {"message": "Hello World"}
```

[1] https://fastapi.tiangolo.com/tutorial/first-steps/

Using the cURL web client

Key cURL options:

```
$ curl -h
Usage: curl [options...] <url>
 -v, --verbose                Make the operation more talkative
 -H, --header <header/@file> Pass custom header(s) to server
 -d, --data <data>            HTTP POST data
```

Example usage:

```
$ curl http://localhost:8000
{"message":"Hello World"}
```

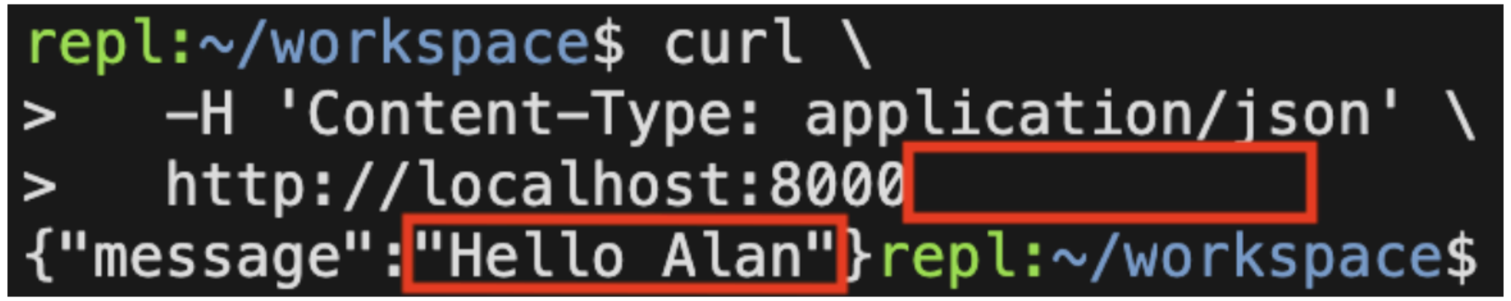
Query Parameters

New endpoint:

- Path: "/hello"
- Query parameter: "name"
    - Default value: "Alan"

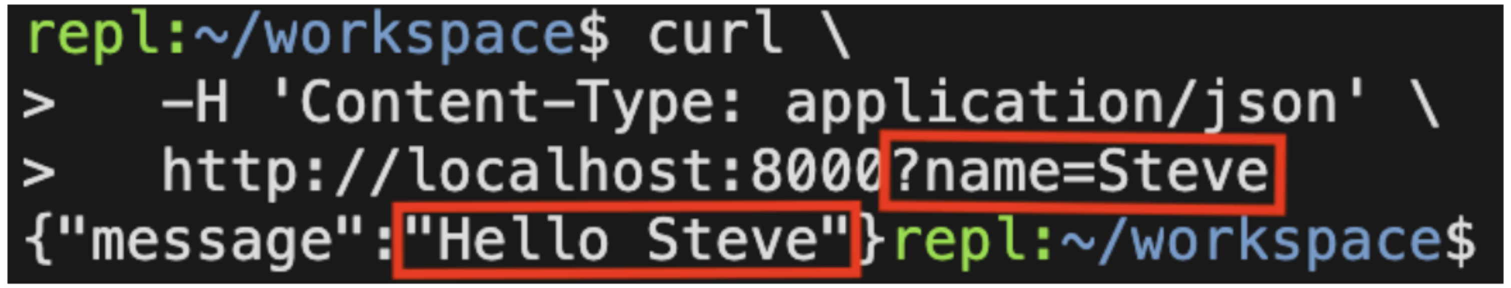
```python
@app.get("/hello")
def hello(name: str = "Alan"):
    return {"message": f"Hello {name}"}
```

Name not in request:



```
repl:~/workspace$ curl \
>    -H 'Content-Type: application/json' \
>    http://localhost:8000
{"message":"Hello Alan"}repl:~/workspace$
```

Name in request:



```
repl:~/workspace$ curl \
>    -H 'Content-Type: application/json' \
>    http://localhost:8000?name=Steve
{"message":"Hello Steve"}repl:~/workspace$
```

Let's practice!

INTRODUCTION TO FASTAPI

POST operations

INTRODUCTION TO FASTAPI

Matt Eckerle
Software and Data Engineering Leader

GET vs. POST Operations

GET Operations

- Traditional use: request info about an object
- Parameters sent via query string
- Can be sent from a web browser

```
api = "http://moviereviews.co/reviews/1"
response = requests.get(api)
```

POST Operations

- Traditional use: create a new object
- Parameters sent via query string as well as request body
- Requires an application or framework
  - e.g. `cURL` , `requests`

```
api = "http://moviereviews.co/reviews/"
body = {"text": "A great movie!"}
response = requests.post(api, json=body)
```

HTTP Request Body

- Data sent after the HTTP request header
- Header specifies body encoding
- Supports nested data structures
- JSON and XML are the most common encodings for APIs
- JSON is FastAPI default encoding

JSON Example

```
# Create a record for a movie review
{"movie": "The Neverending Story",
 "review": {"num_stars": 4,
            "text": "Great movie!",
            "public": true}}
```

Using pydantic's BaseModel

`pydantic` : interface to define request and response body schemas

> **Note**
>
> We are nesting `Review` inside `MovieReview`

```python
from pydantic import BaseModel

class Review(BaseModel):
    num_stars: int
    text: str
    public: bool = False

class MovieReview(BaseModel):
    movie: str
    # Nest Review in MovieReview
    review: Review
```

Handling a POST Operation

POST endpoint to create a new movie review:

- Endpoint: `/reviews`
- Input: `MovieReview` (from previous slide)
- Output: `db_review` (defined elsewhere)

```python
@app.post("/reviews", response_model=DbReview)
def create_review(review: MovieReview):
    # Persist the movie review to the database
    db_review = crud.create_review(review)
    # Return the review including database ID
    return db_review
```

[1] https://fastapi.tiangolo.com/tutorial/sql-databases/#crud-utils

Let's practice!

INTRODUCTION TO FASTAPI