Programming and Communications III: FastAPI

Jordi Ricard Onrubia Palacios

Departament d'Informàtica i Enginyeria Industrial Universitat de Lleida

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Overview

- ➢ Overview
- ❑ Installation
- ❑ Examples

Overview

FastAPI is a modern, high-performance web framework for building APIs with Python. It's designed for quick development, supports asynchronous programming, and is based on Python type hints for validation and documentation.

- Speed: FastAPI is built on Starlette and Pydantic, making it one of the fastest Python frameworks.
- Automatic Documentation: It generates interactive API documentation with Swagger UI and ReDoc.
- Validation: Automatic request and response validation based on Python type annotations.
- Asynchronous Support: Native support for async and await, ideal for high-concurrency applications.
- Ease of Use: Developer-friendly with minimal boilerplate.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Installation

- ❑ Overview
- ➢ Installation
- ❑ Examples

Installation

- Create a virtual environment (Not mandatory, but recommended)
  - python3 venv -m <path/to/install/env>
  - source env/bin/activate (linux)
  - env/scripts/activate (windows)

- Install
  - Fastapi -> pip install fastapi
  - Uvicorn -> pip install "uvicorn[standard]"

- Import
  - from fastapi import FastAPI

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Installation

- ❑ Overview
- ❑ Installation
- ➢ Examples

Examples: APP Definition

```python
from fastapi import FastAPI

app = FastAPI()
```

- The first step consists on instance the FastAPI class that serves as the main entry point for your application. It's where you define your API routes, middleware, dependencies, and configuration.
- The FastAPI() class creates an application object that handles all incoming requests and maps them to the appropriate handlers (routes).
- This instance is passed to the ASGI server (like Uvicorn) to run the application.

Examples: GET all

```
@app.get("/users")
def get_all():
    return [User(user_id=1, id=1, title='user1_id_1', body='descr_1_1'),
            User(user_id=1, id=2, title='user1_id_2', body='descr_1_2'),
            User(user_id=1, id=3, title='user1_id_3', body='descr_1_3'),
            User(user_id=2, id=1, title='user2_id_1', body='descr_2_1'),
            User(user_id=2, id=2, title='user2_id_2', body='descr_2_2'),
            User(user_id=3, id=3, title='user3_id_3', body='descr_3_3'),
            User(user_id=3, id=1, title='user2_id_1', body='descr_2_1'),
            User(user_id=3, id=2, title='user2_id_2', body='descr_2_2'),
            User(user_id=3, id=3, title='user3_id_3', body='descr_3_3')]
```

- @app.get("/users"): Defines a GET endpoint at the URL (/users).
- The function get_all() is called when the endpoint is accessed, returning a JSON response with all the users

Examples: GET some

```python
@app.get("/users/partial")
def get_some(limit:int, skip:int):
    return [User(user_id=1, id=1, title='user1_id_1', body='descr_1_1'),
            User(user_id=1, id=2, title='user1_id_2', body='descr_1_2'),
            User(user_id=1, id=3, title='user1_id_3', body='descr_1_3'),
            User(user_id=2, id=1, title='user2_id_1', body='descr_2_1'),
            User(user_id=2, id=2, title='user2_id_2', body='descr_2_2'),
            User(user_id=3, id=3, title='user3_id_3', body='descr_3_3'),
            User(user_id=3, id=1, title='user2_id_1', body='descr_2_1'),
            User(user_id=3, id=2, title='user2_id_2', body='descr_2_2'),
            User(user_id=3, id=3, title='user3_id_3', body='descr_3_3')][skip:][:limit]
```

- @app.get("/users/partial"): Defines a GET endpoint at the URL (/users/partial).
- This function will ask for 2 Query Parameters, this parameters do not belong to the path but are appended at the end of the url /users/partial?limit=1&skip1
- The function get_some() is called when the endpoint is accessed, returning a JSON response with limit number of users starting from skip

Examples: GET one

```python
@app.get("/users/{user_id}")
async def get_filter(user_id:int):
    users = [User(user_id=1, id=1, title='user1_id_1', body='descr_1_1'),
             User(user_id=1, id=2, title='user1_id_2', body='descr_1_2'),
             User(user_id=1, id=3, title='user1_id_3', body='descr_1_3'),
             User(user_id=2, id=1, title='user2_id_1', body='descr_2_1'),
             User(user_id=2, id=2, title='user2_id_2', body='descr_2_2'),
             User(user_id=3, id=3, title='user3_id_3', body='descr_3_3'),
             User(user_id=3, id=1, title='user2_id_1', body='descr_2_1'),
             User(user_id=3, id=2, title='user2_id_2', body='descr_2_2'),
             User(user_id=3, id=3, title='user3_id_3', body='descr_3_3')]
    return [user for user in users if user.user_id == user_id]
```

- @app.get("/users/{user_id}"): Defines a GET endpoint at the URL (/users/{user_id}). User_id must be de id of the user we want to retrieve
- This is known as Path parameter, it defines a parameter inside the url path, FastAPI will get that path parameter and make it accessible through the variable defined at the header function user_id
- The function get_filter(user_id:int) is called when the endpoint is accessed, returning a JSON response with all the users that satisfy the condition, in this case it will return only the user with the desired id

Examples: Helping Classes

```python
class PatchUser(BaseModel):
    user_id: int
    id: Optional[int] = None
    title: Optional[str] = None
    body: Optional[str] = None
```

```python
class PostUsers(BaseModel):
    users: list = []

post_users = PostUsers()
```

```python
class User(BaseModel):
    user_id: int
    id: int
    title: str
    body: str

    def update_user(self, patch_user: PatchUser):
        if patch_user.id:
            self.id = patch_user.id

        if patch_user.title:
            self.title = patch_user.title

        if patch_user.body:
            self.body = patch_user.body
```

Examples: POST

```python
@app.post("/users/")
async def post_user(user: User):
    post_users.users.append(user)
    return post_users.users
```

- @app.post("/users"): Defines a POST endpoint at the URL (/users).
- The endpoint expects a User like json in the body, FastAPI will map automatically the the payload inside the body with the User class defined previously
- The function post_user() is called when the endpoint is accessed, adding the user to a list like database.

Examples: PUT

```python
@app.put("/users/")
async def update_user(user: User):
    post_users.users = [old_user for old_user in post_users.users
                        if user.user_id ≠ old_user.user_id]
    post_users.users.append(user)
    return post_users.users
```

- @app.put("/users"): Defines a PUT endpoint at the URL (/users).
- The endpoint expects a User like json in the body, FastAPI will map automatically the the payload inside the body with the User class defined previously
- The function update_user() is called when the endpoint is accessed, updating the user with the same user_id

Examples: PATCH

```python
@app.patch("/users/")
async def patch_user(user: PatchUser):
    to_update_user: User = [old_user for old_user in post_users.users
                            if user.user_id == old_user.user_id][0]
    to_update_user.update_user(user)
    return post_users
```

- @app.patch("/users"): Defines a PATCH endpoint at the URL (/users).
- The endpoint expects a PatchUser like json in the body, FastAPI will map automatically the the payload inside the body with the PatchUser class defined previously
- The function patch_user() is called when the endpoint is accessed, updating the fields only present in the PatchUser object

Examples: DELETE

```
@app.delete("/users/{user_id}")
async def delete_suer(user_id: int):
    post_users.users = [user for user in post_users.users
                        if user_id ≠ user.user_id]
    return {'user_id': user_id}
```

- @app.delete("/users/{user_id}"): Defines a DELETE endpoint at the URL (/users/{user_id})
- The endpoint expects a Path Parameter representing the id of the user to be deleted
- The function delete_user(user_id) is called when the endpoint is accessed, removing the user from the list like database

Examples: Link

FastAPI Code: Link