fastapi_tutorial

March 22, 2025

1 Entwicklung einer REST-API mit FastAPI

Ziel: Einführung in FastAPI für Python-Entwickler **Dauer:** 50 Minuten

1.1 Agenda

- Einführung in APIs
- REST-Konzept und HTTP-Methoden
- FastAPI-Grundlagen
- Validierung mit Pydantic
- Endpoint-Beispiele
- Testen mit Postman

1.2 Was ist eine API?

- **Application Programming Interface**
- Schnittstelle zur Kommunikation zwischen Softwarekomponenten
- Beispiel: Wetterdaten abfragen, Bezahlvorgänge, Social-Media-Integrationen

1.2.1 REST-API: Das Konzept

- **REpresentational State Transfer (REST)**
- Architekturstil für verteilte Systeme
- **Prinzipien:**
  - Zustandslosigkeit
  - Ressourcenorientierung (URLs als eindeutige IDs)
  - Standard-HTTP-Methoden (GET, POST, PUT, DELETE)

1.3 HTTP-Methoden im Detail

- **GET:** Daten abrufen (z.B. `/items`)
- **POST:** Neue Ressource erstellen (z.B. `/items`)
- **PUT:** Ressource vollständig aktualisieren (z.B. `/items/{id}`)
- **DELETE:** Ressource löschen (z.B. `/items/{id}`)

Beispiel-URLs: - `GET /items` → Liste aller Artikel - `GET /items/42` → Artikel mit ID 42 - `POST /items` → Neuen Artikel anlegen - `PUT /items/42` → Artikel 42 aktualisieren

1.4 FastAPI: Warum?

- **Schnell:** ASGI-basiert (asynchrone Verarbeitung)
- **Einfach:** Automatische Dokumentation (Swagger/Redoc)
- **Modern:** Integration mit Pydantic für Validierung
- **Python-Typhinting:** Code-Vorhersage und Fehlervermeidung

1.4.1 Installation

```
pip install fastapi uvicorn
```

UVicorn: ASGI-Server zum Ausführen der API

1.5 Grundgerüst einer FastAPI-Anwendung

```
from fastapi import FastAPI
app = FastAPI()

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

Starten des Servers:

```
uvicorn main:app --reload
```

1.5.1 Beispiel: GET-Endpoint mit Validierung

```
from fastapi import Path

@app.get("/items/{item_id}")
def read_item(item_id: int = Path(..., ge=1)):
    return {"item_id": item_id}
```

- `Path(..., ge=1)`: Validierung für `item_id  1`

1.6 Pydantic: Datenvalidierung

- **Datenmodelle** definieren die Struktur der Ein-/Ausgaben

```
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    category: str
```

1.6.1 Verwendung im Endpoint

```
@app.post("/items/")
def create_item(item: Item):
    items[item.id] = item
    return item
```

- FastAPI validiert automatisch den Request-Body gegen das `Item`-Modell.

1.7 Beispiel: Vollständiger CRUD-Endpoint

```
items = {}

@app.post("/items/")
def create_item(item: Item):
    items[item.id] = item
    return item

@app.get("/items/{item_id}")
def read_item(item_id: int):
    return items[item_id]
```

1.7.1 Fehlerbehandlung

```
from fastapi import HTTPException

@app.get("/items/{item_id}")
def read_item(item_id: int):
    if item_id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return items[item_id]
```

1.8 Testen mit Postman

1. **GET-Request:** `http://localhost:8000/items/42`
2. **POST-Request:** Body (JSON) an `http://localhost:8000/items/` senden
3. **Response validieren:** Statuscode, JSON-Daten

Beispiel-POST-Body:

```
{
    "name": "Laptop",
    "price": 999.99,
    "category": "Electronics"
}
```

1.8.1 Automatische Dokumentation

- **Swagger UI:** `http://localhost:8000/docs`
- **Redoc:** `http://localhost:8000/redoc`
- Testen Sie Endpoints direkt im Browser!

1.9 Zusammenfassung

- FastAPI vereinfacht API-Entwicklung durch Typhinting und Pydantic
- REST-Prinzipien ermöglichen klare Struktur
- Testen mit Postman oder integrierter Dokumentation

Nächste Schritte: - Datenbankanbindung (z.B. SQLAlchemy) - Authentifizierung (OAuth2) - Deployment mit Docker

Sprechernotizen: - Betonen, dass FastAPI für Anfänger geeignet ist, aber auch skalierbar bleibt. - Auf Fragen zu komplexeren Validierungen (z.B. Regex) eingehen. - Praktische Übung: Eigenen GET/POST-Endpoint erstellen.