

# MVC és REST alapok gyakorlat

Két technológiai út: Spring Boot (Java) és FastAPI (Python)

A gyakorlat célja, hogy az *MVC + REST alapelvek* és a *REST API fejlesztés előadás* anyagát egy összefüggő, lépésről-lépésre felépített feladatsoron keresztül gyakoroljátok. A feladatok során REST-ben fogtok gondolkodni: erőforrásokat azonosítotok, URI-kat terveztek, HTTP metódusokat és státszkódokat választotok, majd egy egyszerű REST API-t implementáltok és Swagger UI-val kipróbáltok.

## Áttekintés és követelmények

### Tanulási célok

- Megérteni a REST erőforrás-alapú szemléletet (resource, URI).
- Tudatosan választani HTTP metódusokat és státszkódokat.
- Egyszerű REST API implementálása (GET, POST, DELETE).
- Swagger/OpenAPI dokumentáció megtekintése és használata.
- Kapcsolat megértése: REST kívülről szerződés, MVC belül felépítés.

### Használt példa domain

Egy minimális „User” (felhasználó) erőforrást kezelünk. Nem használunk adatbázist, hanem in-memory (memóriában tárolt) adatszerkezetet, hogy a REST alapokra koncentráljunk.

- User mezők: id (azonosító), name (név), email (opcionális).
- API alap útvonal: /api/v1/users
- Végpontok: listázás, lekérdezés id alapján, létrehozás, törlés.

### Két megvalósítási út

- A) Spring Boot (Java, Maven):
  - IDE-be importálható project
  - springdoc-openapi Swagger UI-val
- B) FastAPI (Python):
  - venv alapú környezet
  - uvicorn futtatással
  - beépített Swagger UI

## 1. rész – REST tervezés

### Feladat 1 – Erőforrások azonosítása

Írd le, milyen erőforrások vannak a rendszerben, és milyen gyökér URI-k illenek hozzájuk. A cél: főnevekben gondolkodni, nem műveletekben.

- Milyen erőforrás(oka)t kezel? (pl. users, orders)
- Mi legyen az erőforrás gyökér URI-ja? (pl. /users)
- Mi az erőforrás azonosítója? (pl. /users/{id})

Várt eredmény (minimum):

/users  
/users/{id}

### Feladat 2 – URI design + HTTP metódusok

A következő műveletekhez tervezd meg a REST-es végpontokat (URI + HTTP metódus). Fontos: az URI legyen főnév, a műveletet a HTTP metódus fejezte ki.

- Felhasználók listázása
- Egy felhasználó lekérdezése
- Új felhasználó létrehozása
- Felhasználó törlése
- Egy user rendeléseinek lekérdezése (csak tervezés szinten)

Minta megoldás:

```
GET    /users
GET    /users/{id}
POST   /users
DELETE /users/{id}
GET    /users/{id}/orders
```

### Feladat 3 – Státszkód párosítás

Párosítsd a helyzeteket megfelelő HTTP státszkódokkal.

Helyzet	Ajánlott státszkód
Sikeres létrehozás	201 Created
Nem létező user kérése	404 Not Found
Hibás input (validáció)	400 Bad Request
Sikeres törlés	204 No Content

Megjegyzés: a státszkód az API kommunikáció része. A kliens gyakran státszkód alapján dönt a következő lépésről.

## 2. rész – Implementáció + Swagger

### Válassz technológiai utat

A következő feladatokat választhatóan Spring Boot vagy FastAPI megoldással készítsd el. A két megoldás ugyanazt az API-t valósítja meg, csak más nyelven és keretrendszerben.

### A) Spring Boot (Java) – teljes projekt

#### Projekt letöltése és importálása

- A projekt egy Maven alapú Spring Boot 3 alkalmazás.
- Importáld IDE-be mint Maven projekt (pom.xml).
- Előfeltétel: JDK 17 és Maven 3.9+.

Projekt mappa neve: spring-boot-rest-lab

Futtatás a projekt gyökeréből:

```
mvn spring-boot:run
```

#### Swagger UI

A springdoc-openapi automatikusan generál OpenAPI leírást és Swagger UI-t.

- Swagger UI: <http://localhost:8080/swagger-ui.html>
- OpenAPI JSON: <http://localhost:8080/v3/api-docs>

#### Implementált végpontok

```
GET    /api/v1/users
GET    /api/v1/users/{id}
POST   /api/v1/users
PUT    /api/v1/users/{id}
DELETE /api/v1/users/{id}
```

#### Tesztelés curl-lel (opcionális)

```
curl -s http://localhost:8080/api/v1/users
```

```
curl -i -X POST http://localhost:8080/api/v1/users \
-H "Content-Type: application/json" \
-d '{"name": "Charlie", "email": "charlie@example.com"}'
```

```
curl -i -X DELETE http://localhost:8080/api/v1/users/1
```

#### Mini feladatok (Spring Boot)

- Próbáld ki a POST végpontot Swagger UI-ból és figyeld meg a 201 Created státuszt.
- Küldj hibás kérést (pl. üres name) és figyeld meg a 400 Bad Request választ és a hibamezőket.
- Törölj egy nem létező felhasználót és ellenőrizd a 404 Not Found választ.

## B) FastAPI (Python) – teljes projekt

### Projekt előkészítése (venv)

A projekt Python venv-et használ. A parancsok operációs rendszerenként kicsit eltérnek.

Projekt mappa neve: fastapi-rest-lab

#### Windows (PowerShell)

```
python -m venv .venv  
.\\venv\\Scripts\\Activate.ps1  
pip install -r requirements.txt
```

#### Linux/macOS (bash)

```
python3 -m venv .venv  
source .venv/bin/activate  
pip install -r requirements.txt
```

### Futtatás

```
uvicorn app.main:app --reload --port 8000
```

### Swagger UI / OpenAPI

- Swagger UI: <http://localhost:8000/docs>
- OpenAPI JSON: <http://localhost:8000/openapi.json>

### Implementált végpontok

```
GET    /api/v1/users  
GET    /api/v1/users/{id}  
POST   /api/v1/users  
PUT    /api/v1/users/{id}  
DELETE /api/v1/users/{id}
```

### Mini feladatok (FastAPI)

- Swagger UI-ból próbáld ki a GET és POST végpontot, figyeld a válasz struktúrát (response\_model).
- Kérdezz le nem létező id-t és figyeld meg a 404-es hibát (HTTPException).
- Törlés után kérdezd le újra a listát és ellenőrizd, hogy a user eltűnt.

## Összefoglalás és önenellenőrzés

### Ellenőrző kérdések

- Miért mondjuk azt, hogy REST-ben az URI főnév, a HTTP metódus pedig az ige?
- Mi a különbség a 200 OK és a 201 Created között?
- Mitől 'stateless' egy REST API? Mit NEM tárol a szerver?
- Miért hasznos a Swagger UI a fejlesztés és tesztelés során?
- MVC-ben mi tekinthető 'View'-nak egy JSON-t visszaadó REST API esetén?