MVC és REST architektúra alapjai és elvei

MVC és REST architektúra, URI design, versioning, documentation

Fontos megérteni:

A web nem csak HTTP kérésekből áll

Kell egy **strukturált gondolkodásmód**, hogyan épül fel egy alkalmazás

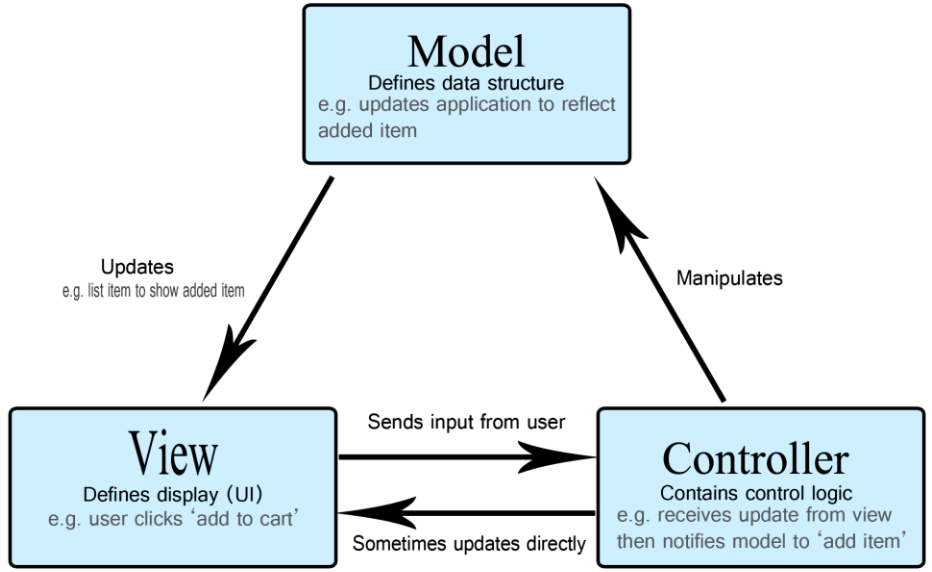
Az MVC és a REST **nem technológiák**, hanem **elvek és minták**

Miért van szükség architektúrára?

- Növekvő komplexitás
- Karbantarthatóság
- Skálázhatóság
- Csapatmunka

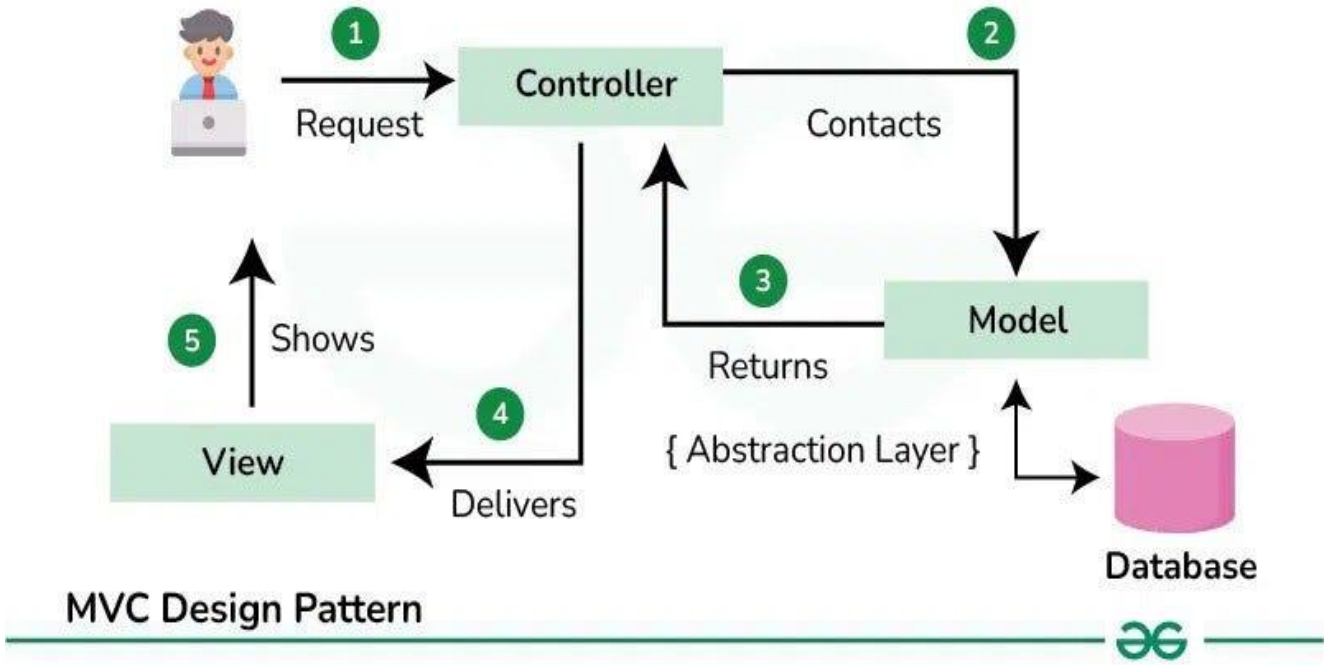
MVC architektúra áttekintése

- **MVC = Model – View – Controller**
- Cél: szétválasztani a felelősségeket
- **Model**: üzleti logika és adatok
- **View**: megjelenítés
- **Controller**: vezérlés



MVC működési folyamata

- HTTP kérés érkezik
- **Controller** fogadja és feldolgozza
- **Model** végrehajtja az üzleti logikát
- **View** előállítja a választ (megjelenítés)



MVC előnyei és korlátai

Előnyök

- Átlátható
- Tesztelhető
- Jól strukturált

Korlátok

- Nem ideális tisztán API-hoz
- Modern weben a frontend (view) már külön van
- Nagy rendszernél túl sok controller

Mi az a REST?

- **REST** = **Re**presentational **S**tate **T**ransfer
- Egy architektúrális stílus
- HTTP protokollra épül
- Erőforrás (Resource) központú szemlélet

REST alapelvek

- Client–Server felépítés
- Állapotnélküli (stateless)
- Egységes felület (Uniform Interface)
- Gyorsítótárazás (Cacheable)

REST és HTTP kapcsolata

- REST **kihasználja** a HTTP protokollt
- HTTP metódusok: műveletek
- Státuszkódok: állapotjelzés
- Fejlécek: kiegészítő metaadatok

URI design alapelvek

- Legyen erőforrás-alapú
- Használjunk többes számot
- Nincs ige, hanem főnév használata

❌ `/getUser?id=5`
✅ `/users/5`

REST verziózás

- API mint **szerződés** a kliens és a szerver között
- Verziózás szükségessége
  - Stabilitás, karbantarthatóság, kontrollált változtatások
- Visszafelé kompatibilitás
- Verziózási stratégiák:
  - URL alapú: /api/v1/users
  - HTTP fejléc alapú: X-API-Version: 1
  - Query paraméter alapú: ?version=1

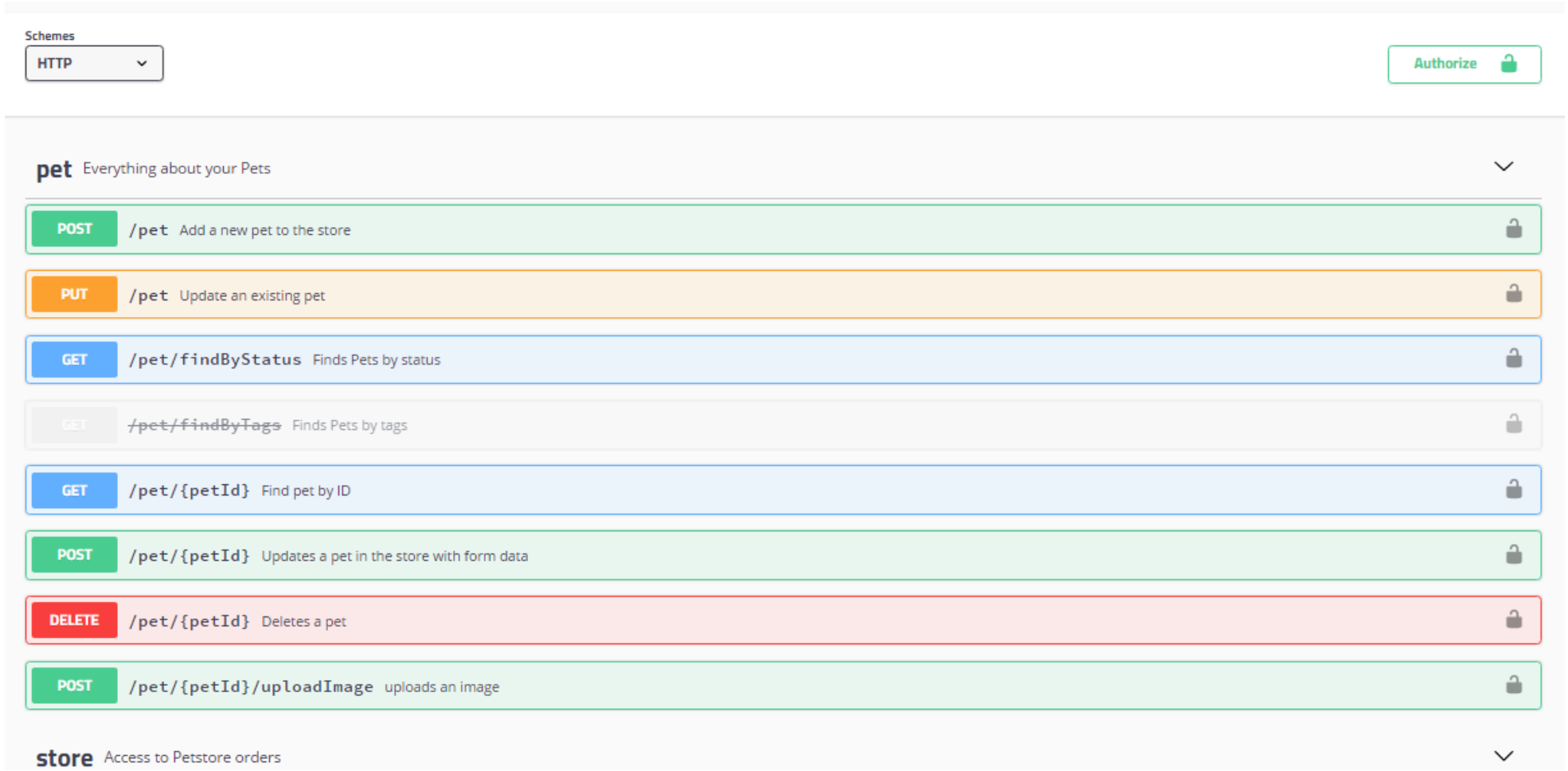
REST dokumentáció szükségessége

- Fejlesztők közti kommunikáció
- Egyértelmű API szerződés
- Automatizált tesztelés
- Kliens generálás
- Verziók követhetősége

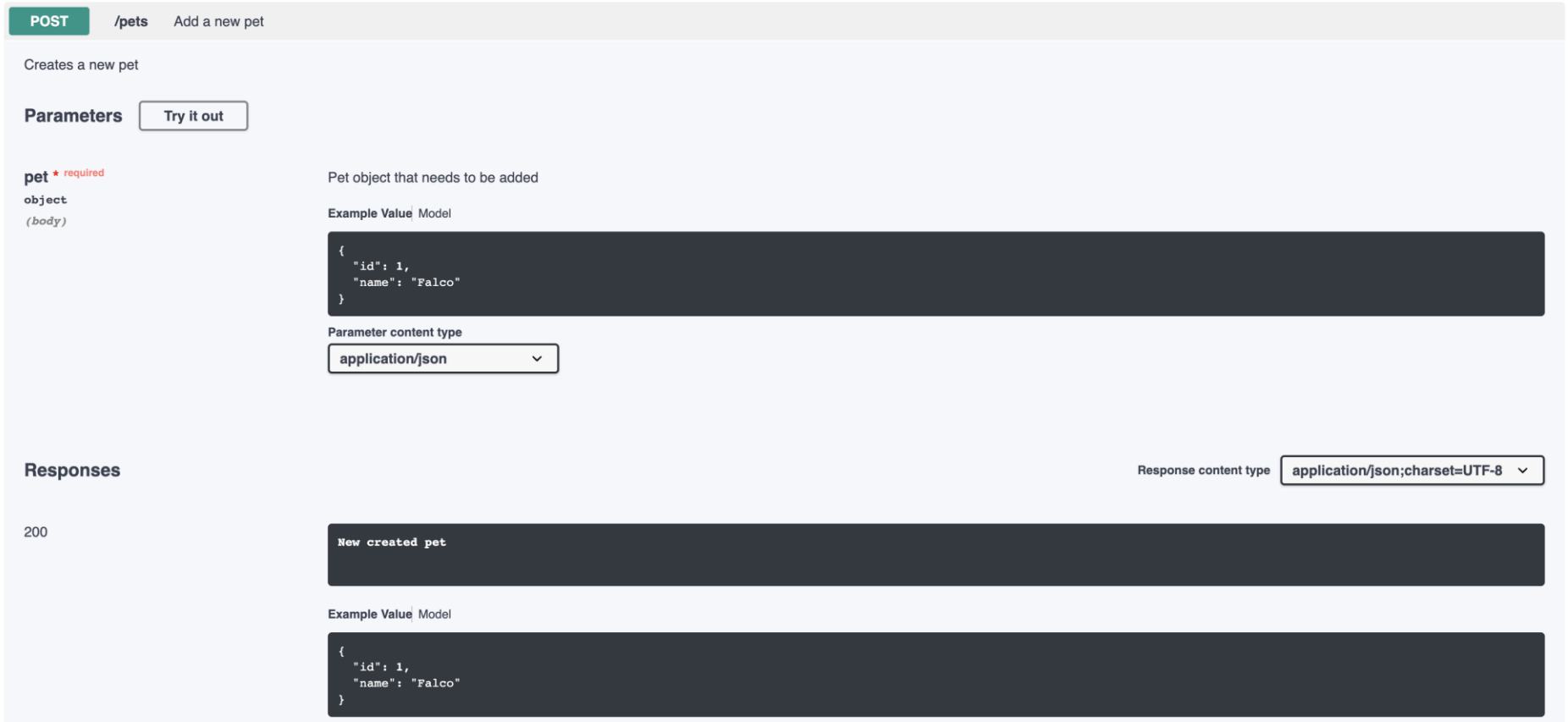
Swagger és OpenAPI

- OpenAPI specifikáció
- Swagger UI
- Interaktív dokumentáció
- Böngészőből tesztelhető API
- Fejlesztési eszköz

Swagger UI


Schemes
HTTP
Authorize
pet Everything about your Pets
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status
GET /pet/findByTags Finds Pets by tags
GET /pet/{petId} Find pet by ID
POST /pet/{petId} Updates a pet in the store with form data
DELETE /pet/{petId} Deletes a pet
POST /pet/{petId}/uploadImage uploads an image
store Access to Petstore orders

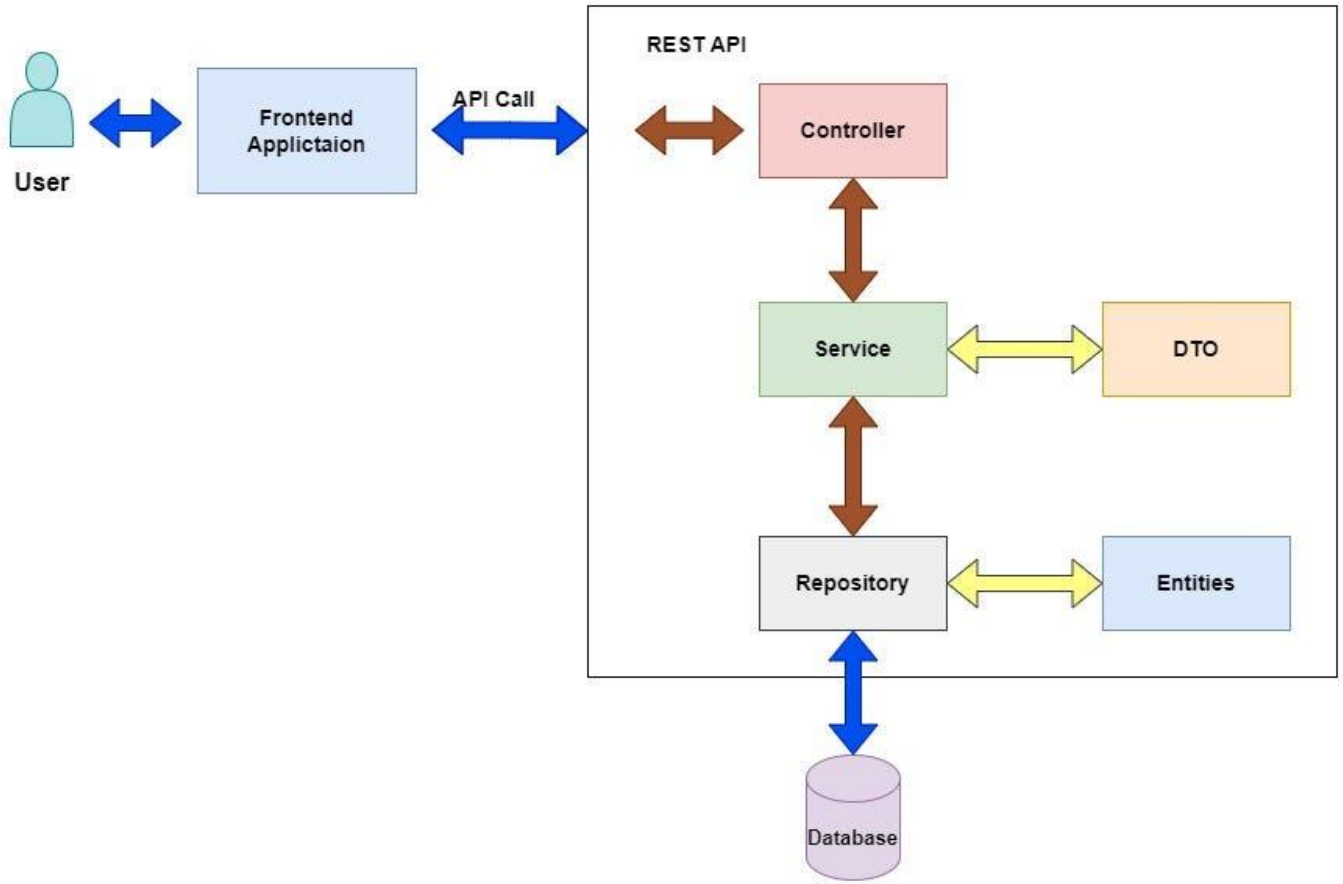
Swagger UI


POST
/pets
Add a new pet
Creates a new pet
Parameters
Try it out
pet * required
object
(body)
Pet object that needs to be added
Example Value Model
{
"id": 1,
"name": "Falco"
}
Parameter content type
application/json
Responses
Response content type
application/json;charset=UTF-8
200
New created pet
Example Value Model
{
"id": 1,
"name": "Falco"
}

MVC és REST kapcsolata

- MVC a szerveren
- REST API mint backend interfész
- Frontend külön alkalmazás
- **Controller** → REST endpoint
- **Model** → üzleti logika
- **View** → JSON válasz

MVC és REST kapcsolata



Összefoglalás

- MVC gondolkodás a szerveren
- REST alapelvek
- Erőforrás-alapú URI-k
- Verziózás = szerződésvédelem
- Dokumentáció (OpenAPI / Swagger)