Egyéni Feladat – User Management Rendszer

> ***Ez a leírás teljesen kezdőknek is szól!*** *Ha még soha nem fejlesztettél, akkor is végig tudsz menni rajta. Minden lépésnél részletes magyarázatot, példákat és forrásokat találsz.*

---

0. Alapok, amiket tudni érdemes

Mi az a User Management rendszer?

Egy olyan program, amiben felhasználókat tudsz létrehozni, listázni, módosítani, törölni. Ilyen van minden weboldalon, ahol regisztrálni lehet.

Mik azok a REST API-k?

Olyan szabályok szerint működő webes szolgáltatások, amikhez más programok (vagy akár te is) tudsz kéréseket küldeni (pl. "add vissza az összes felhasználót").

Mik azok a rétegek (layer-ek)?

A programot több részre bontjuk, hogy átláthatóbb legyen. Pl. külön rész kezeli az adatokat, külön rész a webes kéréseket.

Mik azok a parancsok (Command) és lekérdezések (Query)?

- **Command:** Valamit módosít (pl. új user létrehozása)
- **Query:** Csak lekérdez (pl. összes user listázása)

---

1. Szükséges eszközök, telepítés

1. **Node.js letöltése:**

- Menj a https://nodejs.org/ oldalra, töltsd le a LTS verziót, telepítsd.

2. **Kód szerkesztő:**
   - Ajánlott: Visual Studio Code

3. **Postman vagy Thunder Client:**
   - Ezekkel tudod tesztelni az API-t. Postman letöltése

---

2. Projekt létrehozása, első lépések

1. Hozz létre egy új mappát pl. `user-management` néven.
2. Nyisd meg a mappát VS Code-ban.
3. Nyiss egy terminált (Terminal > New Terminal).
4. Írd be: `npm init -y` (létrehozza a package.json-t)
5. Telepítsd az Express-t: `npm install express`
6. (Ha JSON fájlt használsz adattárolásra, nem kell adatbázis.)

---

3. Mappastruktúra kialakítása

Javasolt szerkezet:

📋 **Projekt struktúra**

```
user-management/
  src/
    API/
    Application/
    Domain/
    Infrastructure/
  package.json
```

Minden mappába majd külön fájlokat teszünk (lásd lentebb).

---

4. Kódolás lépésről lépésre (példákkal)

4.1. API réteg (Express szerver)

- Hozz létre egy `src/API/server.js` fájlt.
- Írd bele az alábbi mintát:

Express szerver példa

```
import express from 'express';
const app = express();
app.use(express.json());

app.get('/', (req, res) => {
  res.send('Hello, User Management!');
});

app.listen(3000, () => {
  console.log('Szerver fut a http://localhost:3000 címen');
});
```

> **Futtasd:** `node src/API/server.js`
>
> (Ha hibát ír, nézd meg, hogy mindenhol helyes-e az elérési út és a kód)

4.2. Domain réteg (interfész, entitás)

- Hozz létre egy `src/Domain/IUserRepository.js` fájlt.
- Írd bele, hogy milyen műveleteket vársz el (pl. createUser, getUserById, stb.)

IUserRepository.js példa

```
export default class IUserRepository {
  createUser(user) {}
  getUserById(id) {}
  getAllUsers() {}
  updateUser(id, user) {}
  deleteUser(id) {}
}
```

4.3. Infrastructure réteg (adattárolás)

- Hozz létre egy `src/Infrastructure/userRepository.js` fájlt.
- Ebben valósítsd meg az IUserRepository metódusait, pl. JSON fájlba írással/olvasással.

userRepository.js példa (részlet)

```
import fs from 'fs/promises';
import path from 'path';
import IUserRepository from '../Domain/IUserRepository.js';

const DATA_PATH = path.resolve('src/Infrastructure/user.json');

export default class UserRepository extends IUserRepository {
  async createUser(user) {
    // ...
  }
  // ... többi metódus ...
}
```

- Hozz létre egy `user.json` fájlt is, pl. így:

user.json példa

```
[]
```

4.4. Application réteg (Command/Query handlerek)

- Hozz létre minden CRUD művelethez külön Command/Query és Handler fájlt.
- Pl. `createUserCommand.js`, `createUserCommandHandler.js`, stb.

createUserCommand.js példa

```
export default class CreateUserCommand {
  constructor(user) {
    this.user = user;
  }
}
```

createUserCommandHandler.js példa

```
import UserRepository from '../../Infrastructure/userRepository.js';
export default class CreateUserCommandHandler {
  constructor() {
    this.userRepository = new UserRepository();
  }
  async handle(command) {
    return await this.userRepository.createUser(command.user);
  }
}
```

4.5. API végpontok (Controller, Router)

- Hozz létre egy `userController.js`-t és egy `userRouter.js`-t az API mappában.
- A controller hívja a handlereket, a router összeköti az útvonalakat a controllerrel.

userController.js példa

```
import CreateUserCommand from '../../Application/users/command/createUserCommand.js
import CreateUserCommandHandler from '../../Application/users/command/createUserCo

export async function createUser(req, res) {
  const command = new CreateUserCommand(req.body);
  const handler = new CreateUserCommandHandler();
  const result = await handler.handle(command);
  res.status(201).json(result);
}
```

---

5. Tesztelés, hibakeresés

- Indítsd el a szervert: `node src/API/server.js`
- Küldj kéréseket Postmanből vagy Thunder Clientből (GET, POST, PUT, DELETE)
- Ha hibát kapsz, olvasd el figyelmesen a hibaüzenetet, keresd meg a fájlban a hibás sort.
- Próbáld ki az összes végpontot, nézd meg, hogy minden működik-e.

> 💡 **Tipp:** Használd a `console.log()` parancsot, hogy lásd, mi történik a kódodban!

---

6. Dokumentáció, beadás

Készíts egy rövid leírást (README.md), amiben leírod:

- Mit csinál a programod?
- Hogyan kell elindítani?
- Milyen extra funkciót raktál bele?

Csatold a forráskódot és a tesztadatokat.

---

7. Hasznos források, videók

> 📚 **Tanulási anyagok:**
>
> - Node.js alapok magyarul (videó)
> - Express.js alapok magyarul (videó)
> - REST API magyarázat (videó)
> - JSON fájlok kezelése Node.js-ben (angol)
> - VS Code rövid bemutató (magyar)

> ✅ **Sikeres munkát kívánunk!**
>
> Ha elakadsz, ne félj segítséget kérni. A programozás tanulás kérdezéssel és gyakorlással megy a legjobban!