User Management Rendszer - Feladat Leírás

Projekt Áttekintés

Egy **User Management (felhasználó kezelési)** rendszer elkészítése Node.js és Express.js használatával, **Clean Architecture** elveket követve, **CQRS (Command Query Responsibility Segregation)** pattern alkalmazásával.

Célkitűzés

Egy RESTful API létrehozása felhasználók kezeléséhez (CRUD műveletek), amely követi a modern szoftver architektúra elveket és jól strukturált, karbantartható kódot eredményez.

Technológiai Stack

Kötelező Technológiák

- **Node.js** (v18 vagy újabb)
- **Express.js** (v5.2.1 vagy újabb) - Web framework
- **ES6 Modules** - `import/export` szintaxis használata
- **JSON File Storage** - Adattárolás fájl alapú megoldással (későbbi adatbázis integrációhoz előkészítve)

Fejlesztői Eszközök

- **nodemon** - Automatikus újraindítás fejlesztés közben

Architektúra Áttekintés

A projekt **4 fő réteget** tartalmaz:

📋 **Projekt struktúra**

```
src/
├── API/                    # Prezentációs réteg (Web API)
├── Application/            # Üzleti logika réteg (CQRS)
├── Domain/                 # Domain modell és interfészek
└── Infrastructure/         # Adatelérési réteg és külső szolgáltatások
```

Rétegek Részletes Leírása

1. **Domain Réteg** (Domain Layer)

- **Felelősség**: Az alkalmazás magját képező interfészek és domain modellek
- **Tartalma**: Repository interfészek (IUserRepository)
- **Függőség**: Nincs más rétegre való függősége

2. **Infrastructure Réteg** (Infrastructure Layer)

- **Felelősség**: Technikai implementációk (adatbázis, fájlkezelés, külső API-k)
- **Tartalma**: Repository implementációk, adattárolás
- **Függőség**: Domain rétegtől függ (implementálja az interfészeket)

3. **Application Réteg** (Application Layer)

- **Felelősség**: Üzleti logika, use case-ek implementálása
- **Tartalma**: Commands, CommandHandlers, Queries, QueryHandlers
- **Függőség**: Domain rétegtől függ
- **CQRS Pattern**: Írás (Command) és olvasás (Query) műveletek szétválasztása

4. **API Réteg** (API Layer)

- **Felelősség**: HTTP kérések kezelése, routing, válaszok küldése
- **Tartalma**: Controllers, Routers, Server konfiguráció
- **Függőség**: Application és Infrastructure rétegtől függ

CQRS Pattern Magyarázat

Mi az a CQRS?

Command Query Responsibility Segregation - Az írási és olvasási műveletek szétválasztása.

Command (Parancs) - Írási műveletek

- Megváltoztatja a rendszer állapotát
- Nem ad vissza értéket (vagy csak a létrehozott entitást)
- Példák: CreateUser, UpdateUser, DeleteUser

Struktúra:

1. **Command osztály** - Tartalmazza a parancs adatait
2. **CommandHandler osztály** - Végrehajtja a parancsot, validálja az adatokat

Query (Lekérdezés) - Olvasási műveletek

- Nem változtatja meg a rendszer állapotát
- Adatokat ad vissza
- Példák: GetAllUsers, GetUserById

Struktúra:

1. **Query osztály** - Tartalmazza a lekérdezés paramétereit
2. **QueryHandler osztály** - Végrehajtja a lekérdezést

Részletes Implementációs Útmutató

1. Projekt Inicializálás

1.1 Projekt struktúra létrehozása

📋 **Teljes mappastruktúra**

```
Backend/
└── elso gyakorlat/
    ├── package.json
    └── src/
        ├── API/
        │   ├── server.js
        │   ├── controllers/
        │   │   └── userController.js
        │   └── routers/
        │       └── userRouter.js
        ├── Application/
        │   └── users/
        │       ├── command/
        │       │   ├── createUserCommand.js
        │       │   ├── createUserCommandHandler.js
        │       │   ├── updateUserCommand.js
        │       │   ├── updateUserCommandHandler.js
        │       │   ├── deleteUserCommand.js
        │       │   └── deleteUserCommandHandler.js
        │       └── query/
        │           ├── getAllUsersQuery.js
        │           ├── getAllUsersQueryHandler.js
        │           ├── getUserByIdQuery.js
        │           └── getUserByIdQueryHandler.js
        ├── Domain/
        │   └── IUserRepository.js
        └── Infrastructure/
            ├── user.json
            └── userRepository.js
```

1.2 package.json létrehozása

Cél: Node.js projekt konfigurációs fájl létrehozása ES6 modul támogatással.

> **Részletes leírás:**
>
> A `package.json` az **npm projekt szíve** - tartalmazza a metaadatokat, függőségeket és script-eket.
>
> **Kulcsfontosságú mezők:**
>
> **1. Alapinformációk:**

- `name` : Projekt neve (lowercase, kötőjellel)
- `version` : Verzió szám (semantic versioning: MAJOR.MINOR.PATCH)
- `description` : Rövid leírás
- `main` : Belépési pont (itt nem használt)

2. "type": "module" ⚠️ KRITIKUS!

- Engedélyezi az ES6 modulok használatát (`import/export`)
- Nélküle csak `require()` / `module.exports` működne (CommonJS)
- Ez kötelező, hogy `import` szintaxist használhassunk!

3. Scripts:

- `start` : A szerver indítása nodemon-nal
- `nodemon src/API/server.js` - automatikus újraindítás fájl módosításkor
- Így nem kell minden változtatás után manuálisan újraindítani
- `test` : Placeholder teszt script (később Jest/Mocha tesztek jöhetnek ide)

4. Dependencies (Futásidejű függőségek):

- `express` : ^5.2.1 - Web framework
- `^` (caret): Automatikus minor/patch update-ek engedélyezése
- 5.x.x verzió tartományon belül
- `nodemon` : ^3.1.11 - Fejlesztői szerver automatikus újraindítása
- Érdemes `devDependencies` -be tenni (csak fejlesztéshez kell)

📄 **Teljes package.json tartalom**

```
{
  "name": "elso-gyakorlat",
  "version": "1.0.0",
  "description": "User Management System",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon src/API/server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^5.2.1",
    "nodemon": "^3.1.11"
  }
}
```

⚠️ **FONTOS:** Ne felejtsed el hozzáadni a `"type": "module"` sort!

1.3 Függőségek telepítése

📋 **Parancs**

```
npm install
```

---

2. Domain Réteg Implementálás

2.1 IUserRepository.js - Repository Interface

Fájl: `src/Domain/IUserRepository.js`

Cél: Egy interfész-szerű osztály létrehozása, amely definiálja a user repository alapvető műveleteit.

> **Részletes leírás:**

A Domain réteg középpontjában az interfészek állnak. JavaScript-ben nincs natív interfész támogatás (mint pl. TypeScript-ben vagy Java-ban), de osztályokkal szimulálhatjuk ezt a viselkedést.

Mit kell létrehozni:

1. **IUserRepository osztály** - Ez lesz a "szerződés", amit minden repository implementációnak követnie kell
2. **5 metódus**, mind aszinkron (`async`):
   - `getAll()` - Összes felhasználó lekérése
   - `getById(id)` - Egy felhasználó lekérése ID alapján
   - `create(user)` - Új felhasználó létrehozása
   - `update(id, userData)` - Felhasználó módosítása
   - `delete(id)` - Felhasználó törlése
3. **Error dobás** - Minden metódus dobjon hibát alapértelmezetten

Fontos elvek:

- Ez az osztály NEM tartalmaz implementációt, csak az "interfészt" definiálja
- Minden metódus async, mert később fájlkezelés/adatbázis műveleteket fogunk végezni
- Ez a Dependency Inversion Principle része

Teljes kód

```
export class IUserRepository {
  async getAll() {
    throw new Error('Method not implemented');
  }

  async getById(id) {
    throw new Error('Method not implemented');
  }

  async create(user) {
    throw new Error('Method not implemented');
  }

  async update(id, userData) {
    throw new Error('Method not implemented');
  }

  async delete(id) {
    throw new Error('Method not implemented');
  }
}
```

---

3. Infrastructure Réteg Implementálás

3.1 user.json - Adat fájl

Fájl: `src/Infrastructure/user.json`

Cél: Egyszerű JSON fájl alapú adattárolás létrehozása.

> **Miért JSON fájl?**
>
> - Egyszerű és gyors kezdéshez
> - Nem kell adatbázist telepíteni
> - Könnyen olvasható és debuggolható
> - Demonstrálja a repository pattern használatát

Fájl tartalma

```
[]
```

3.2 userRepository.js - Repository Implementáció

Fájl: `src/Infrastructure/userRepository.js`

Cél: Az IUserRepository interfész konkrét implementációja JSON fájl alapú tárolással.

Teljes kód

```
import fs from 'fs/promises';
import path from 'path';
import { fileURLToPath } from 'url';
import { IUserRepository } from '../Domain/IUserRepository.js';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const USER_FILE = path.join(__dirname, 'user.json');

export class UserRepository extends IUserRepository {
  async getAll() {
    try {
      const data = await fs.readFile(USER_FILE, 'utf-8');
      return JSON.parse(data);
    } catch (error) {
      return [];
    }
  }

  async getById(id) {
    const users = await this.getAll();
    return users.find(user => user.id === id);
  }

  async create(user) {
    const users = await this.getAll();
    const newUser = {
      id: Date.now().toString(),
      ...user,
      createdAt: new Date().toISOString()
    };
    users.push(newUser);
    await fs.writeFile(USER_FILE, JSON.stringify(users, null, 2));
    return newUser;
  }

  async update(id, userData) {
    const users = await this.getAll();
    const index = users.findIndex(user => user.id === id);
    if (index === -1) return null;

    users[index] = {
      ...users[index],
      ...userData,
      id: users[index].id,
      updatedAt: new Date().toISOString()
    };
    await fs.writeFile(USER_FILE, JSON.stringify(users, null, 2));
```

```
    return users[index];
  }

  async delete(id) {
    const users = await this.getAll();
    const filteredUsers = users.filter(user => user.id !== id);
    if (users.length === filteredUsers.length) return false;

    await fs.writeFile(USER_FILE, JSON.stringify(filteredUsers, null, 2));
    return true;
  }
}
```

---

4. Application Réteg - Query Implementálás

4.1 getAllUsersQuery.js & getAllUsersQueryHandler.js

getAllUsersQuery.js

```
export class GetAllUsersQuery {
  constructor() {}
}
```

getAllUsersQueryHandler.js

```
export class GetAllUsersQueryHandler {
  constructor(userRepository) {
    this.userRepository = userRepository;
  }

  async handle(query) {
    return await this.userRepository.getAll();
  }
}
```

4.2 getUserByIdQuery.js & getUserByIdQueryHandler.js

getUserByIdQuery.js

```
export class GetUserByIdQuery {
  constructor(id) {
    this.id = id;
  }
}
```

getUserByIdQueryHandler.js

```
export class GetUserByIdQueryHandler {
  constructor(userRepository) {
    this.userRepository = userRepository;
  }

  async handle(query) {
    const user = await this.userRepository.getById(query.id);
    if (!user) {
      throw new Error('User not found');
    }
    return user;
  }
}
```

---

5. Application Réteg - Command Implementálás

5.1 createUserCommand.js & createUserCommandHandler.js

createUserCommand.js

```
export class CreateUserCommand {
  constructor(name, email, age) {
    this.name = name;
    this.email = email;
    this.age = age;
  }
}
```

createUserCommandHandler.js

```
export class CreateUserCommandHandler {
  constructor(userRepository) {
    this.userRepository = userRepository;
  }

  async handle(command) {
    if (!command.name || !command.email) {
      throw new Error('Name and email are required');
    }

    const userData = {
      name: command.name,
      email: command.email,
      age: command.age
    };

    return await this.userRepository.create(userData);
  }
}
```

5.2 updateUserCommand.js & updateUserCommandHandler.js

updateUserCommand.js

```
export class UpdateUserCommand {
  constructor(id, name, email, age) {
    this.id = id;
    this.name = name;
    this.email = email;
    this.age = age;
  }
}
```

updateUserCommandHandler.js

```
export class UpdateUserCommandHandler {
  constructor(userRepository) {
    this.userRepository = userRepository;
  }

  async handle(command) {
    const userData = {
      name: command.name,
      email: command.email,
      age: command.age
    };

    const updatedUser = await this.userRepository.update(command.id, userData);
    if (!updatedUser) {
      throw new Error('User not found');
    }

    return updatedUser;
  }
}
```

5.3 deleteUserCommand.js & deleteUserCommandHandler.js

deleteUserCommand.js

```
export class DeleteUserCommand {
  constructor(id) {
    this.id = id;
  }
}
```

deleteUserCommandHandler.js

```
export class DeleteUserCommandHandler {
  constructor(userRepository) {
    this.userRepository = userRepository;
  }

  async handle(command) {
    const deleted = await this.userRepository.delete(command.id);
    if (!deleted) {
      throw new Error('User not found');
    }
  }
}
```

---

6. API Réteg Implementálás

6.1 userController.js

Fájl: `src/API/controllers/userController.js`

Cél: HTTP kérések kezelése és koordináció az Application réteggel.

Teljes kód

```
import { UserRepository } from '../../Infrastructure/userRepository.js';
import { GetAllUsersQuery } from '../../Application/users/query/getAllUsersQuery.js
import { GetAllUsersQueryHandler } from '../../Application/users/query/getAllUsersQ
import { GetUserByIdQuery } from '../../Application/users/query/getUserByIdQuery.js
import { GetUserByIdQueryHandler } from '../../Application/users/query/getUserByIdQ
import { CreateUserCommand } from '../../Application/users/command/createUserCommar
import { CreateUserCommandHandler } from '../../Application/users/command/createUse
import { UpdateUserCommand } from '../../Application/users/command/updateUserCommar
import { UpdateUserCommandHandler } from '../../Application/users/command/updateUse
import { DeleteUserCommand } from '../../Application/users/command/deleteUserCommar
import { DeleteUserCommandHandler } from '../../Application/users/command/deleteUse

const userRepository = new UserRepository();
const getAllUsersQueryHandler = new GetAllUsersQueryHandler(userRepository);
const getUserByIdQueryHandler = new GetUserByIdQueryHandler(userRepository);
const createUserCommandHandler = new CreateUserCommandHandler(userRepository);
const updateUserCommandHandler = new UpdateUserCommandHandler(userRepository);
const deleteUserCommandHandler = new DeleteUserCommandHandler(userRepository);

export class UserController {
  async getAll(req, res) {
    try {
      const query = new GetAllUsersQuery();
      const users = await getAllUsersQueryHandler.handle(query);
      res.json(users);
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  }

  async getById(req, res) {
    try {
      const query = new GetUserByIdQuery(req.params.id);
      const user = await getUserByIdQueryHandler.handle(query);
      res.json(user);
    } catch (error) {
      res.status(404).json({ error: error.message });
    }
  }

  async create(req, res) {
    try {
      const command = new CreateUserCommand(
        req.body.name,
        req.body.email,
        req.body.age
      );
      const user = await createUserCommandHandler.handle(command);
```

```
      res.status(201).json(user);
    } catch (error) {
      res.status(400).json({ error: error.message });
    }
  }

  async update(req, res) {
    try {
      const command = new UpdateUserCommand(
        req.params.id,
        req.body.name,
        req.body.email,
        req.body.age
      );
      const user = await updateUserCommandHandler.handle(command);
      res.json(user);
    } catch (error) {
      res.status(404).json({ error: error.message });
    }
  }

  async delete(req, res) {
    try {
      const command = new DeleteUserCommand(req.params.id);
      await deleteUserCommandHandler.handle(command);
      res.status(204).send();
    } catch (error) {
      res.status(404).json({ error: error.message });
    }
  }
}
```

6.2 userRouter.js

Fájl: `src/API/routers/userRouter.js`

Teljes kód

```
import express from 'express';
import { UserController } from '../controllers/userController.js';

const router = express.Router();
const userController = new UserController();

router.get('/', (req, res) => userController.getAll(req, res));
router.get('/:id', (req, res) => userController.getById(req, res));
router.post('/', (req, res) => userController.create(req, res));
router.put('/:id', (req, res) => userController.update(req, res));
router.delete('/:id', (req, res) => userController.delete(req, res));

export default router;
```

6.3 server.js

Fájl: `src/API/server.js`

Teljes kód

```
import express from 'express';
import userRouter from './routers/userRouter.js';

const app = express();
const PORT = 3000;

// Middleware
app.use(express.json());

// Routes
app.use('/users', userRouter);

// Server
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Futtatás és Tesztelés

Szerver Indítása

📋 Parancs

```
npm start
```

API Endpoint-ok Tesztelése

1. Összes User Lekérése

📋 GET Request

```
GET http://localhost:3000/users

Válasz: []
```

2. Új User Létrehozása

📋 POST Request

```
POST http://localhost:3000/users
Content-Type: application/json

{
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30
}

Válasz:
{
  "id": "1707825600000",
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "createdAt": "2026-02-13T10:00:00.000Z"
}
```

3. User Lekérése ID alapján

📋 GET Request

```
GET http://localhost:3000/users/1707825600000

Válasz:
{
  "id": "1707825600000",
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "createdAt": "2026-02-13T10:00:00.000Z"
}
```

4. User Módosítása

📋 PUT Request

```
PUT http://localhost:3000/users/1707825600000
Content-Type: application/json

{
  "name": "John Updated",
  "email": "john.updated@example.com",
  "age": 31
}

Válasz:
{
  "id": "1707825600000",
  "name": "John Updated",
  "email": "john.updated@example.com",
  "age": 31,
  "createdAt": "2026-02-13T10:00:00.000Z",
  "updatedAt": "2026-02-13T10:05:00.000Z"
}
```

5. User Törlése

📋 DELETE Request

```
DELETE http://localhost:3000/users/1707825600000

Válasz: 204 No Content
```

User Adatmodell

📋 Adatstruktúra

```
{
  "id": "string",           // Automatikusan generált (timestamp)
  "name": "string",         // Kötelező
  "email": "string",        // Kötelező
  "age": number,            // Opcionális
  "createdAt": "ISO8601",   // Automatikusan generált létrehozáskor
  "updatedAt": "ISO8601"    // Automatikusan generált módosításkor
}
```

Hibakezelés

HTTP Státusz Kódok

Státusz Kód	Jelentés	Mikor használjuk
200 OK	Sikeres művelet	GET, PUT
201 Created	Sikeres létrehozás	POST
204 No Content	Sikeres törlés, nincs body	DELETE
400 Bad Request	Validációs hiba	Hiányzó/helytelen adatok
404 Not Found	Nem található	User nem létezik

500 Internal Server Error	Szerver hiba	Váratlan hiba

Best Practice-ek és Elvek

1. Separation of Concerns (SoC)

- Minden réteg csak a saját felelősségével foglalkozik
- Controller nem tartalmaz üzleti logikát
- Repository nem validál

2. Dependency Inversion Principle (DIP)

- A magasabb szintű modulok nem függenek az alacsonyabb szintűektől
- Mindketten az absztrakciótól (interface) függenek

3. Single Responsibility Principle (SRP)

- Minden osztálynak egy felelőssége van
- Command/Query külön handler-ben
- Repository csak adatelérésért felelős

4. CQRS Benefits

- Egyszerűbb kód, könnyebb megérteni
- Optimalizálható külön az olvasás és írás
- Könnyebb tesztelni

Ellenőrző Lista (Checklist)

- ☐ Projekt struktúra létrehozva
- ☐ package.json konfigurálva (`"type": "module"`)
- ☐ Függőségek telepítve
- ☐ Domain réteg: IUserRepository létrehozva
- ☐ Infrastructure réteg: UserRepository és user.json létrehozva
- ☐ Application réteg: Összes Query és Command létrehozva
- ☐ Application réteg: Összes Handler létrehozva
- ☐ API réteg: UserController létrehozva
- ☐ API réteg: userRouter létrehozva
- ☐ API réteg: server.js létrehozva és konfigurálva
- ☐ Szerver elindul hiba nélkül
- ☐ GET /users működik
- ☐ POST /users működik és ment a JSON fájlba
- ☐ GET /users/:id működik
- ☐ PUT /users/:id működik
- ☐ DELETE /users/:id működik
- ☐ Hibakezelés működik mindenhol
- ☐ Kód tiszta, kommentek megfelelőek

Összegzés

Ez a User Management rendszer egy jól strukturált, modern Node.js alkalmazás, amely követi a Clean Architecture és CQRS elveket. A kód könnyen karbantartható, skálázható, és előkészített későbbi bővítésekre (pl. adatbázis integráció).

Főbb tanulságok:

- Rétegelt architektúra előnyei
- CQRS pattern alkalmazása

- Dependency Injection használata
- REST API best practice-ek
- Async/Await és hibakezelés
- ES6 modulok használata Node.js-ben

Fejlesztési idő becslés: 3-5 óra (tapasztalattól függően)

Nehézségi szint: Közép-haladó

Dátum: 2026. február 13.
Verzió: 1.0