Egy JavaScript alapú REST API keretrendszer megtervezése és megvalósítása

Készítette:
Nikli Erik

Konzulens:
Dr. Antal Gábor

A feladatról

- Motiváció
  - architektúra és backend fókusz
- Inspiráció
  - NestJS, Spring, ASP.NET
- Technológiák
  - REST, HTTP, Node, Typescript, Postgres, SolidJS
- Eredmények
  - Keretrendszer, bemutatóalkalmazás frontenddel
- Célközönség
  - Fejlesztők

Kiindulási pont

main.ts

```
async function main(): Promise<void> {
    const app = await startHttpApp(AppModule);
    app.logError(true);
}

main().catch(console.error);
```

app.module.ts

```
export const AppModule =
    createModule('App')
    .provideServerConfig({
        http: {
            port: 3000,
            ssl: false,
        },
        routePrefix: '/api',
    });
```

Legelső végpont

example.controller.ts

```
@Controller('/example')
export class ExampleController {

    @Get()
    public async test() {
        return { text: 'Example works!' };
    }
}
```

Parancssor

```
$ curl --location "http://localhost:3000/api/example"

{
  "text": "Example works!"
}
```

A vezérlő, és a hozzátartozó modul beregisztrálása

example.module.ts

```
export const ExampleModule = createModule('Example')
    .addController(ExampleController);
```

app.module.ts

```
export const AppModule =
    createModule('App')
    .provideServerConfig({
        http: {
            port: 3000,
            ssl: false,
        },
        routePrefix: '/api',
    })
    .import(ExampleModule);
```

Más metódusok

example.controller.ts

```
@Controller('/example')
export class ExampleController {

    @Post()
    public async testPost() {
        return {
            text: 'Example post works!'
        };
    }
}
```

Parancssor

```
$ curl --location \
    --request POST \
    "http://localhost:3000/api/example"

{
  "text": "Example post works!"
}
```

Vezérlőn belüli útvonalak

example.controller.ts

```
@Controller('/example')
export class ExampleController {

    @Get('/sub')
    public async test() {
        return { text: 'Example works!' };
    }
}
```

Parancssor

```
$ curl --location \
    "http://localhost:3000/api/example/sub"

{
  "text": "Example works!"
}
```

Bemenet keresőargumentumokkal (Query)

example.controller.ts

```
    @Get('echo')
    public async test(
        @Query('text')
        text: string,
    ) {
        return { text };
    }
```

Parancssor

```
$ curl --location \
    "http://localhost:3000/api/example/echo?text=hello"

{
  "text": "hello"
}
```

Query Bemenet transzformálása

example.controller.ts

```
    @Get('sum')
    public async test(
        @Query('a', { transform: Number, defaultValue: 0 })
        a: number,
        @Query('b', { transform: Number, defaultValue: 0 })
        b: number,
    ) {
        const sum = a + b;
        return { sum };
    }
```

Parancssor

```
$ curl --location \
    "http://localhost:3000/api/example/sum?a=2&b=3"

{
  "sum": 5
}
```

Paraméterezhető útvonal

example.controller.ts

```
    @Get('/:id')
    public async echoId(@Param('id') id: string) {
        return { id };
    }
```

Parancssor

```
$ curl --location \
    "http://localhost:3000/api/example/someId"

{
  "id": "someId"
}
```

Kérés törzsének (Body) olvasása és Data Transfer Object-ek

example.controller.ts

```
    @Post()
    public async create(
        @Body()
         dto: CreateDto,
    ) {
        return dto;
    }
```

create.dto.ts

```
@Dto()
export class CreateDto {
    @Field()
    name: string;

    @Field()
    age: number;

    @Field()
    email: string;

    @Field()
    location?: string;
}
```

Parancssor

```
$ curl -L -X POST
"http://localhost:3000/api/example" \
-H "Content-Type: application/json" \
--data-raw "{
    \"name\": \"Teszt Elek\",
    \"age\": 23,
    \"email\": \"tesztelek@gmail.com\",
    \"location\": \"Szeged\"
}"


{
    "name": "Teszt Elek",
    "age": 23,
    "email": "tesztelek@gmail.com",
    "location": "Szeged"
}
```

Kérés törzsének validálása

create.dto.ts

```ts
@Dto()
export class CreateDto {
    @Field()
    @IsString()
    @Min(5)
    @Max(30)
    name: string;

    @Field()
    @Convert(Number)
    @IsInteger()
    @IsPositive()
    @Max(100)
    age: number;

    @Field()
    @Convert(Date)
    @IsDate()
    birthdate: Date;

    @Field()
    @IsEmail()
    email: string;
}
```

Példa helyes bemenetre

```
$ curl -L "http://localhost:3000/api/example" \
-H "Content-Type: application/json" \
--data-raw "{
    \"name\": \"Teszt Elek\",
    \"age\": \"23\",
    \"email\": \"tesztelek@gmail.com\",
    \"birthdate\": \"2001-07-04\"
}"
```

```
{
    "name": "Teszt Elek",
    "age": 23,
    "birthdate": "2001-07-04T00:00:00.000Z",
    "email": "tesztelek@gmail.com"
}
```

Kérés törzsének validálása

create.dto.ts

```ts
@Dto()
export class CreateDto {
    @Field()
    @IsString()
    @Min(5)
    @Max(30)
    name: string;

    @Field()
    @Convert(Number)
    @IsInteger()
    @IsPositive()
    @Max(100)
    age: number;

    @Field()
    @Convert(Date)
    @IsDate()
    birthdate: Date;

    @Field()
    @IsEmail()
    email: string;
}
```

Példák helytelen bemenetekre

```
$ curl -L "http://localhost:3000/api/example" \
-H "Content-Type: application/json" \
--data-raw "{
    \"name\": \"1234\",
    \"age\": -1,
    \"email\": \"tesztelek@\",
    \"birthdate\": \"2001-07-04\"
}"

{
    "messages": [
        "Property 'name' has to be at least 5
characters long.",
        "Property 'age' was not set to a positive
number.",
        "Property 'email' was not set to a valid email
address."
    ],
    "status": 400
}
```

Kérés törzsének validálása

create.dto.ts

```ts
@Dto()
export class CreateDto {
    @Field()
    @IsString()
    @Min(5)
    @Max(30)
    name: string;

    @Field()
    @Convert(Number)
    @IsInteger()
    @IsPositive()
    @Max(100)
    age: number;

    @Field()
    @Convert(Date)
    @IsDate()
    birthdate: Date;

    @Field()
    @IsEmail()
    email: string;
}
```

Példák helytelen bemenetekre

```
$ curl -L "http://localhost:3000/api/example" \
-H "Content-Type: application/json" \
--data-raw "{
    \"name\": 7,
    \"age\": 0.3,
    \"email\": \"tesztelek@gmail.com\",
    \"birthdate\": \"2001-07-04\"
}"

{
    "messages": [
        "Property 'name' is not a valid string.",
        "Property 'age' must be an integer."
    ],
    "status": 400
}
```

Fájlok fel- és letöltése

Feltöltés

```
@Post('/:filename')
async upload(
    @Param('filename')
    filename: string,
    @Body({
        type: 'buffer',
    })
    data: Buffer,
) {
    const filepath = join('files', filename);
    await writeFile(filepath, data);
}
```

Letöltés

```
@Get(':/filename')
async download(
    @Param('filename')
    filename: string,
) {
    const filepath = join('files', filename);
    const file = await readFile(filepath);
    const contentType = // ...;
    return binary(file, contentType);
}
```

Státuszkódok és hibakezelés

Hiba mint kivétel

```
    @Get('/:id')
    public async find(@Param('id') id: string)   {
        const data = // adatbázis művelet;
        if (!data) {
            throw new NotFound(`Entry of '${id}' cannot be found.`);
        }
        return data;
    }
```

Parancssor

```
$ curl -L "http://localhost:3000/api/example/test"

{
    "messages": [
        "Entry of 'test' cannot be found."
    ],
    "status": 404
}

$ curl -L "http://localhost:3000/api/example/test"

...
< HTTP/1.1 404 Not Found
...
```

Státuszkódok és hibakezelés

Visszatérés státusszal

```
    @Get('/:id')
    public async find(@Param('id') id: string) {
        const data = // adatbázis művelet;
        if (!data) {
            return withStatus("NotFound", undefined);
        }
        return data;
    }
```

Parancssor

```
$ curl -L "http://localhost:3000/api/example/test"

...
< HTTP/1.1 404 Not Found
...
```

Státuszkódok és hibakezelés

Státusz setter használata

```
    @Get('/:id')
    public async test(
        @Param('id') id: string,
        @Status() status: StatusSetter
    ) {
        const data = // adatbázis művelet;
        if (!data) {
            status(HttpStatus.NotFound);
            return undefined;
        }
        return data;
    }
```

Parancssor

```
$ curl -L "http://localhost:3000/api/example/test"

...
< HTTP/1.1 404 Not Found
...
```

Dependency Injection

user.controller.ts

```
@Controller('/users')
export class UserController {



}
```

Dependency Injection

user.controller.ts

```ts
@Controller('/users')
export class UserController {



}
```

user.service.ts

```ts
export class UserService {
    public async add(
        userInfo: DeepPartial<User>,
    ): Promise<User>

    public async list(): Promise<User[]>

    public async getById(id: string)
         : Promise<User>

    public async update(
        id: string,
        toUpdate: DeepPartial<User>,
    ): Promise<User>

    public async remove(id: string)
         : Promise<void>
}
```

Dependency Injection

user.controller.ts

```ts
@Controller('/users')
export class UserController {
    public constructor(
        private readonly userService: UserSevice
    ) {}
}
```

user.service.ts

```ts
export class UserService {
    public async add(
        userInfo: DeepPartial<User>,
    ): Promise<User>

    public async list(): Promise<User[]>

    public async getById(id: string)
        : Promise<User>

    public async update(
        id: string,
        toUpdate: DeepPartial<User>,
    ): Promise<User>

    public async remove(id: string)
        : Promise<void>
}
```

Dependency Injection

user.controller.ts

```
@Controller('/users')
export class UserController {
    public constructor(
        private readonly userService: UserSevice
    ) {}
}
```

user.module.ts

```
export const UserModule =
    createModule('Users')
    .addController(UserController)
    .provideType(UserService);
```

user.service.ts

```
@Injectable()
export class UserService {
    public async add(
        userInfo: DeepPartial<User>,
    ): Promise<User>

    public async list(): Promise<User[]>

    public async getById(id: string)
        : Promise<User>

    public async update(
        id: string,
        toUpdate: DeepPartial<User>,
    ): Promise<User>

    public async remove(id: string)
        : Promise<void>
}
```

Adatbázis elérése

app.module.ts

```
export const AppModule = createModule('App')
    .provideServerConfig(
        //  ...
    )
    .import(
        //  ...
        TypeOrmModule.root({
            type: 'postgres',
            host: 'localhost',
            port: 5432,
            username: 'postgres',
            password: 'postgres',
            database: 'test',
            synchronize: true,
            entities: [User, Ad, Image],
            namingStrategy: new SnakeNamingStrategy(),
        }),
    );
```

user.module.ts

```
export const UserModule =
    createModule('Users')
    .addController(UserController)
    .provideType(UserService)
    .import(TypeOrmModule.get());
```

Adatbázis elérése

user.repository.ts

```
@Injectable()
export class UserRepository {
    private readonly repository: Repository<User>;

    public constructor(repositoryProvider: RepositoryProvider) {
        this.repository = repositoryProvider.get(User);
    }

    public async findByEmail(email: string): Promise<User | undefined> {
        const user = await this.repository.findOne({
            where: {
                email,
            },
        });

        return user ?? undefined;
    }
}
```

Adatbázis elérése

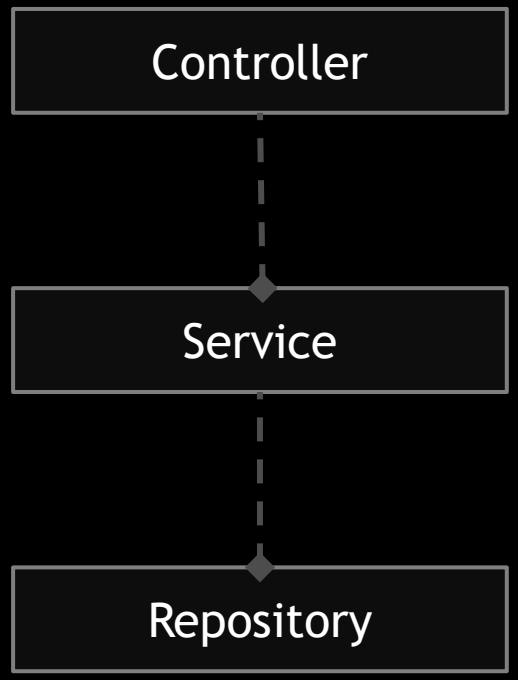
user.repository.ts

```
@Injectable()
export class UserRepository {
    private readonly repository: Repository<User>;

    public constructor(repositoryProvider: RepositoryProvider) {
        this.repository = repositoryProvider.get(User);
    }
}
```

user.service.ts

```
@Injectable()
export class UserService {
    public constructor(
        private readonly userRepository: UserRepository
    ) {}
}
```

Általános architektúra



Middleware-k

auth.middleware.ts

```
@Injectable()
export class AuthMiddleware implements Middleware {


    public async execute(
        req: RequestData,
        res: ResponseData,
        next: () => Promise<void>,
    ): Promise<void> {









        await next();
    }
}
```

Middleware-k

auth.middleware.ts

```
@Injectable()
export class AuthMiddleware implements Middleware {
    public constructor(private readonly userService: UserService) {}

    public async execute(
        req: RequestData,
        res: ResponseData,
        next: () => Promise<void>,
    ): Promise<void> {
        let authorization = req.headers.authorization;
        const [username, password] = authorization.split(':');
        const user = await this.userService.find(username, password);
        if (!user) {
            throw new Unauthorized();
        }
        req['user'] = user;

        await next();
    }
}
```

Middleware-k beregisztrálása

main.ts

```
async function main(): Promise<void> {
    const app = await startHttpApp(AppModule);
    app.logError(true);
    app.addMiddlewares({
        pattern: (url) => !url.startsWith('/api/auth'),
        middlewares: [AuthMiddleware],
    });
}
```

Saját parameter decorator végponthoz

current-user.decorator.ts

```
export function CurrentUser() {
    return paramDecor(({ request }) => {
        const user: User = request['user'];

        if (!user) {
            throw new Unauthorized();
        }

        return user;
    });
}
```

Saját parameter decorator végponthoz

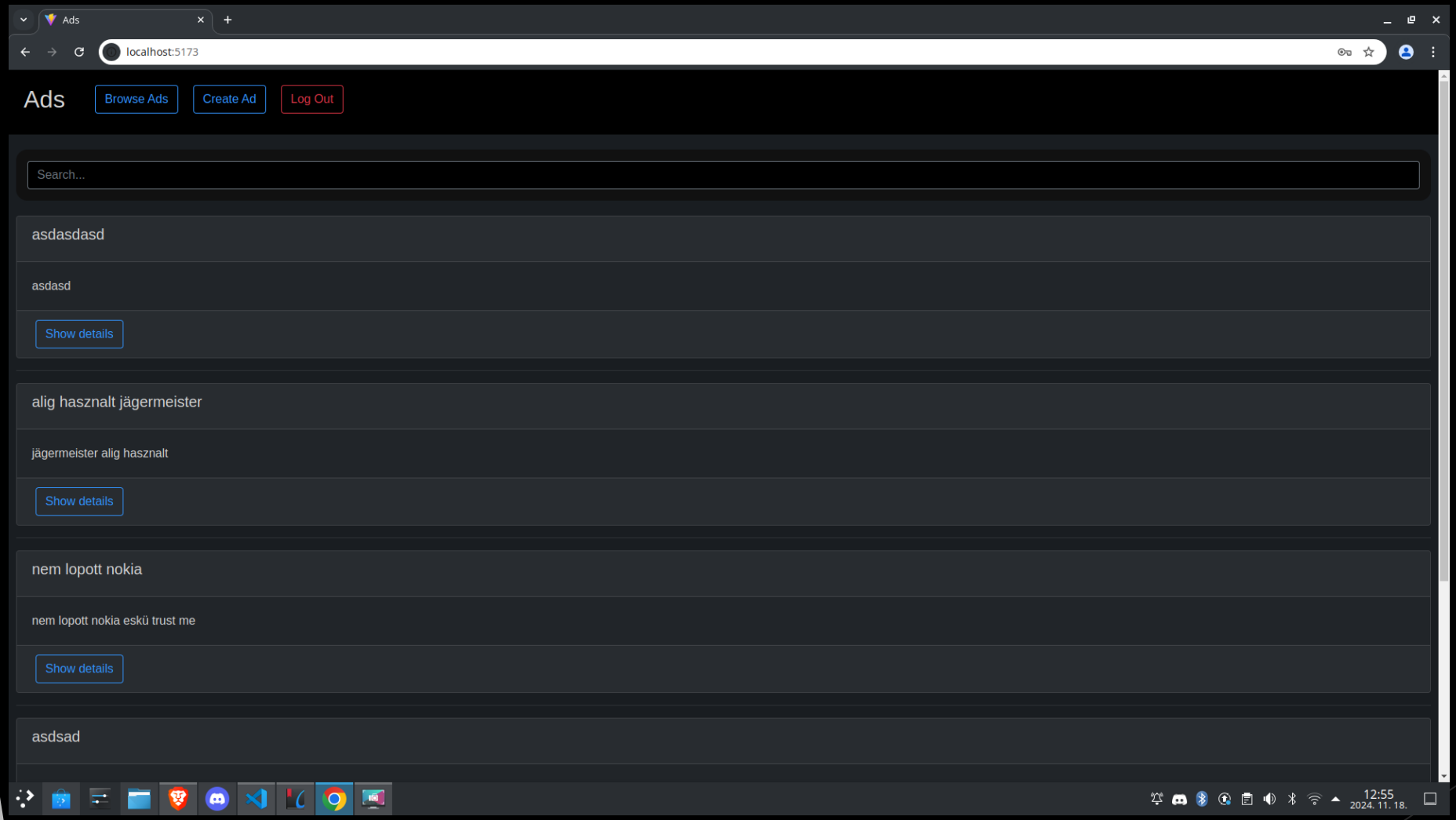
current-user.decorator.ts

```
export function CurrentUser() {
    return paramDecor(({ request }) => {
        const user: User = request['user'];

        if (!user) {
            throw new Unauthorized();
        }

        return user;
    });
}
```

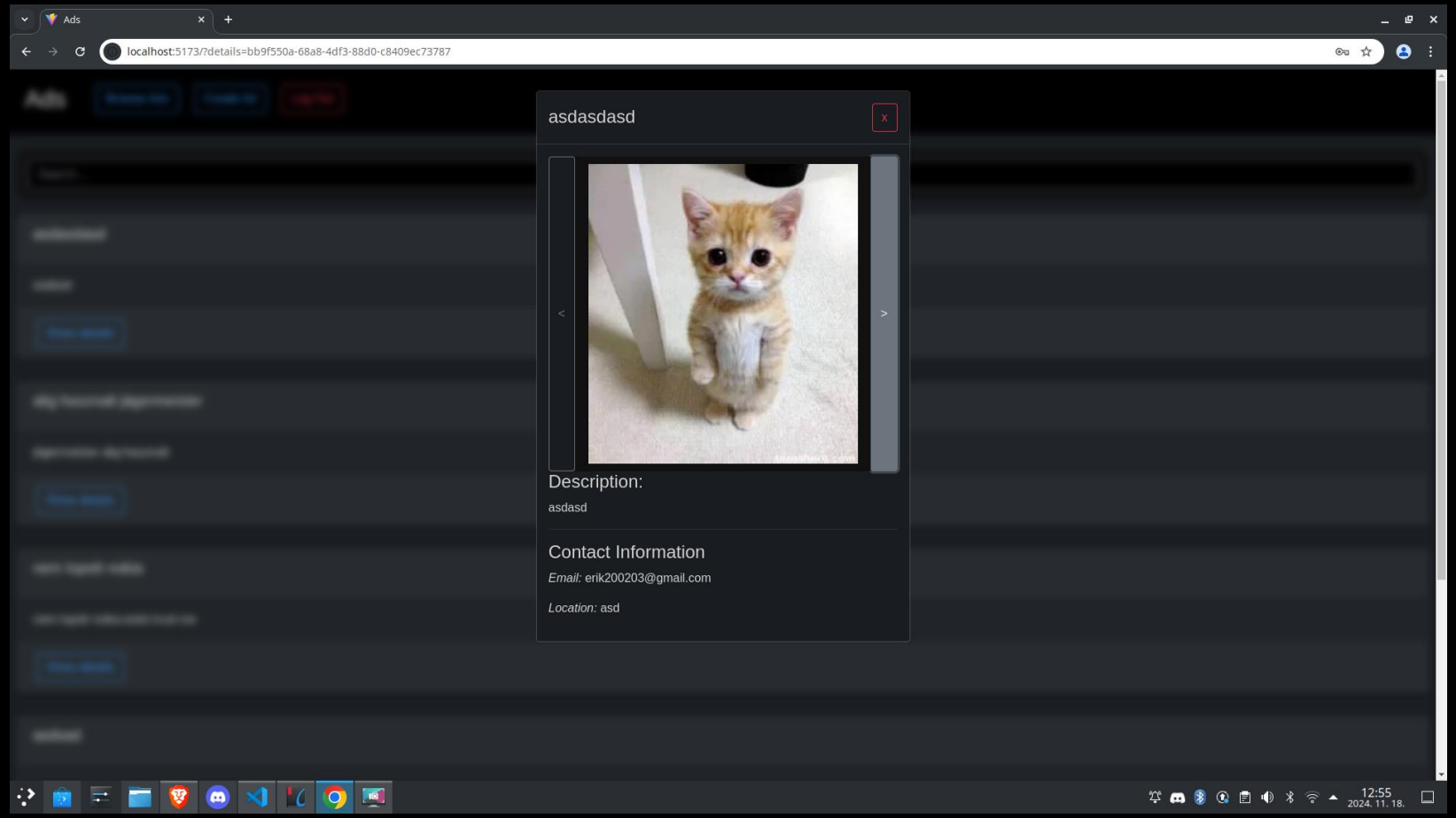
Dekorátor használata

```
    @Post()
    public async create(
        @CurrentUser() user: User,
    ) {}
```

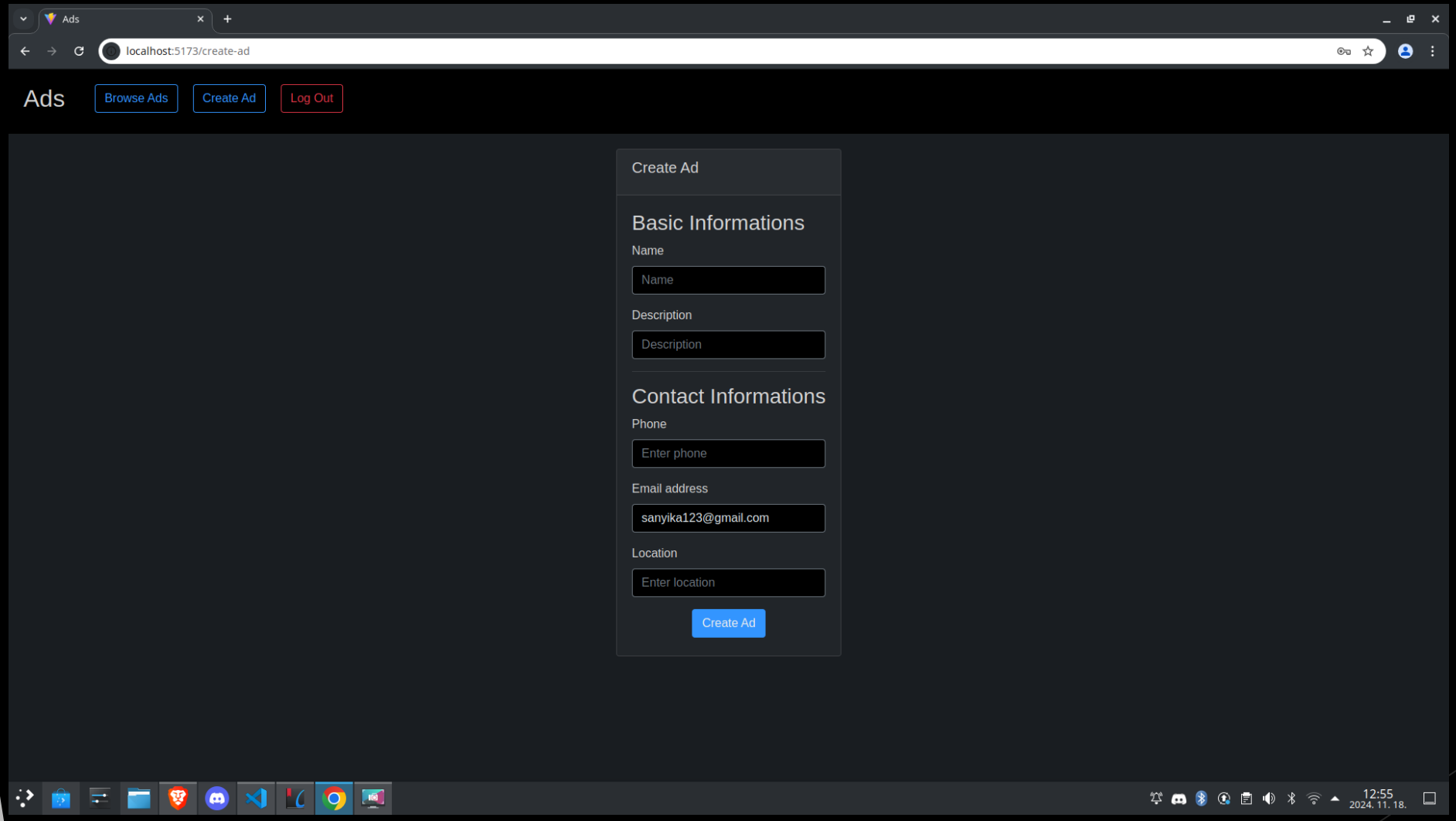
Bemutató alkalmazás frontendje



Bemutató alkalmazás frontendje



Bemutató alkalmazás frontendje



Összefoglaló

1. Feladatról
2. Vezérlő és végpontok
   a. Végpont készítése
   b. Query bemenet
   c. Paraméterezhető URL
   d. Törzs olvasása és DTO-k
   e. Státuszkódok és hibakezelés
3. Dependency Injection és szervízek
4. Adatbázis és TypeORM
5. Middleware-k
6. Saját decorator
7. Bemutató alkalmazás frontendje