

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №2**

з дисципліни “Основи web-програмування”

тема “**JSON HTTP API веб-сервер**”

|  |  |  |
| --- | --- | --- |
| Виконав  студент 2 курсу  групи КП-91  Маховой Олександр Вікторович  Варіант №17 |  | Перевірив.  викладач  Гадиняк Руслан Анатолійович |

Київ 2020

**Мета роботи**

Познайомитись з мовою програмування JavaScript та середовищем Node.js.  
Навчитись створювати модулі, синхронно працювати із файловими потоками та обробляти JSON-текст.

**Завдання**

Реалізувати на мові JavaScript (або TypeScript) консольну програму для керування двома типами сутностей із файлового сховища у форматі JSON: сутність "Користувач системи" та сутність за варіантом.

В якості сутностей за варіантом використати основну або додаткову сутність за темою вашої курсової роботи з "Основ програмування".

### Вказівки до виконання

Середовище розроблення:

* Встановити найновішу доступну LTS версію [Node.js (node)](https://nodejs.org/en/).
* Встановити глобально пакет ESLint за допомогою команди:  
  npm install -g eslint із sudo на Linux).

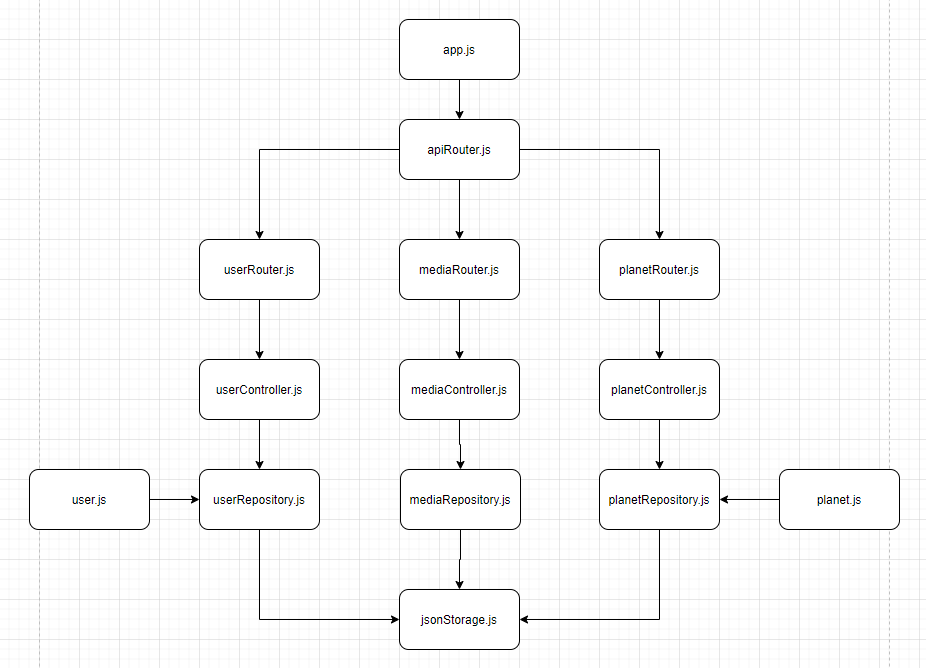
Проект завдання:

* Створити у навчальному репозиторії webprogbase директорію проекту: labs/lab1 і перейти до неї у редакторі (IDE).
* Створити у корені проекту файл app.js та .eslintrc.json (див. Додаток).
* Додати у корінь проекту файл .gitignore ([вміст файлу](https://github.com/github/gitignore/blob/master/Node.gitignore))
* Відкрити термінал у кореневій директорії завдання і виконати команду: npm init --yes, яка автоматично створить файл package.json.

Кроки виконання завдання:

1. Створити у корені проекту директорії data, models, repositories.
2. Реалізувати модуль jsonStorage.js JSON Сховища об'єктів із [цього завдання](http://progbase.herokuapp.com/modules/webprogbase/tasks/json_storage) для зчитування даних з data/users.json.
3. Модель користувача і репозиторій користувачів:
   1. Створити модуль models/user.js (див. [приклад](https://docs.google.com/document/d/1-Uz-Y-H6AtDygYpv_OhcOel6atb6clcGaTd04FD58WE/edit?usp=sharing)):  
      Кожен користувач повинен містити такі поля:
      * id - унікальний числовий (або uuid) ідентифікатор
      * login - унікальний рядок символів
      * fullname - рядок повного імені користувача
      * role - ціле число (0 - простий користувач, 1 - адміністратор).
      * registeredAt - рядок із датою у форматі [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601).
      * avaUrl - рядок з URL зображення.
      * isEnabled - відмітка чи користувача було активовано/деактивовано.
   2. Створити модуль repositories/userRepository.js (див. [приклад](https://docs.google.com/document/d/1-Uz-Y-H6AtDygYpv_OhcOel6atb6clcGaTd04FD58WE/edit?usp=sharing)) для отримання моделей всіх користувачів зі сховища та отримання моделі користувача за ідентифікатором.
4. Реалізувати у app.js командний інтерфейс користувача для виконання операцій з сутностями типу Користувач (формат команд можна змінити і придумати самостійно):
   1. На команду get/users завантажити через репозиторій і показати користувачу список (або таблицю) з короткою інформацією про всі сутності.
   2. На команду get/users/{id} (замість {id} вводиться ідентифікатор сутності) вивести у консоль детальну інформацію про обрану сутність.
5. Сутності за варіантом:
   1. Створити модуль models/{entity}.js (замість {entity} використати назву сутності англійською мовою) і підключити його у app.js. Використати у модулі затверджений тип сутностей за варіантом для створення модуля-репозиторія колекції об'єктів цього типу.  
      Кожен об'єкт сутності за варіантом повинен містити як мінімум:
      * унікальний ідентифікатор
      * 2 рядкових значеня
      * 2 числових значення
      * 1 рядок із датою у форматі [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601).
   2. Модуль репозиторія repository/{entity}Repository.js повинен містити такі синхронні функції:
      * add{Entity}(entityModel) - додати у сховище модель та повернути її новий ідентифікатор
      * get{Entities}() - отримати списком всі моделі зі сховища
      * get{Entity}ById(entityId) - отримати модель зі сховища за ідентифікатором
      * update{Entity}(entityModel) - оновити дані моделі у сховищі
      * delete{Entity}(entityId) - видалити модель зі сховища за ідентифікатором
   3. Реалізувати репозиторій сутностей таким чином, щоби всі описані функції синхронно працювали зі сховищем з JSON файлом (data/{entities}.json, замість {entities} використати назву сутності англійською мовою у множині), що буде містити масив із об'єктами.  
      Кожна зміна вмісту сховища повинна перезаписувати файл. Текст JSON зберігати із відступами (indented).
6. Реалізувати оброблення консольних команд, за допомогою яких користувач буде мати можливість використовувати всі функцій із модуля-репозиторія сутностей за варіантом, наприклад (формат команд можна змінювати):
   1. get/{entities} - отримати список всіх сутностей за варіантом
   2. get/{entities}/{id} - отримати детальну інформацію про обрану сутність
   3. delete/{entities}/{id} - видалити з репозиторія обрану сутність
   4. update/{entities}/{id} - ввести нові дані для обраної сутності і оновити її у репозиторії
   5. post/{entities} - розпочати заповнення даних нової сутності, яку додати у репозиторій
7. При вводі ідентифікатора чи даних сутності обов'язково перевіряти коректність введених даних і сповіщати користувачу результати перевірки.
8. Для всіх випадків неможливості виконання операції виводити про це повідомлення.

**Діаграма залежностей модулів:**

****

**Модулі:**

1. users.json

{

"nextId": 3,

"items": [

{

"id": 1,

"login": "admin",

"fullname": "Admin Admin",

"role": 1,

"registeredAt": "2020-01-10",

"avaUrl": "https://cutt.ly/QgrUZtK",

"isEnabled": true

},

{

"id": 2,

"login": "testUser",

"fullname": "Test User",

"role": 1,

"registeredAt": "2020-29-09",

"avaUrl": "-",

"isEnabled": false

}

]

}

2. planets.json

{

"nextId": 37,

"items": [

{

"id": 1,

"name": "Earth",

"number": 3,

"galaxy": "Milky Way",

"temperature": 45,

"book\_release": "1986-10-13"

},

{

"id": 2,

"name": "MARS",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 7,

"name": "planet",

"number": "12",

"galaxy": "asd",

"temperature": "21",

"book\_release": "2012-02-11T00:00:00.000Z"

},

{

"id": 8,

"name": "MAAAAAARS",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 9,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 17,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 18,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 19,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 20,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 21,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 22,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 23,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 24,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 25,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 26,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 27,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 28,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 29,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 30,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 31,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 32,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 33,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 34,

"name": "ABC",

"number": 4,

"galaxy": "Milky Way",

"temperature": -40,

"book\_release": "2014-02-11T11:30:30"

},

{

"id": 35,

"name": "striiiiiiiiiiiiiiiiing",

"number": 12,

"galaxy": "striiiiiiiiiiiiiiiiing",

"temperature": 21,

"book\_release": "2012-05-05"

},

{

"id": 36,

"name": "shiiiiit",

"number": 0,

"galaxy": "string",

"temperature": 0,

"book\_release": "2005-05-05"

}

]

}

5. planetRepository.js

const Planet = require('./../models/planet');

const JsonStorage = require('./jsonStorage');

class PlanetRepository {

constructor(filePath) {

this.storage = new JsonStorage(filePath);

}

getPlanets() {

const items = this.storage.readItems();

const planets = [];

for (const item of items) {

planets.push(new Planet(item.id, item.name, item.number, item.galaxy, item.temperature, item.book\_release));

}

return planets;

}

getPlanetById(planetId) {

const items = this.storage.readItems();

for (const item of items) {

if (parseInt(item.id) === parseInt(planetId)) {

return new Planet(item.id, item.name, item.number, item.galaxy, item.temperature, item.book\_release);

}

}

return null;

}

addPlanet(planet) {

const planets = this.getPlanets();

planet.id = this.storage.nextId();

planets.push(planet);

this.storage.incrementNextId();

this.storage.writeItems(planets);

return planet;

}

updatePlanet(updPlanet) {

const planets = this.getPlanets();

const planet = this.getPlanetById(updPlanet.id);

if (planet !== null) {

const index = planets.findIndex((plnt) => {

return plnt.id === planet.id;

});

planets.splice(index, 1, updPlanet);

this.storage.writeItems(planets);

return updPlanet;

}

else return null;

}

deletePlanet(id) {

const planets = this.getPlanets();

const planet = this.getPlanetById(id);

if (planet !== null) {

const index = planets.findIndex((plnt) => {

return plnt.id === planet.id;

});

planets.splice(index, 1);

this.storage.writeItems(planets);

return planet;

}

else return null;

}

};

module.exports = PlanetRepository;

6. mediaRepository.js

const JsonStorage = require('./jsonStorage');

const fs = require("fs");

class MediaRepository {

constructor(filePath) {

this.path = filePath;

this.storage = new JsonStorage(filePath + ".json"); //+

}

getNextId() {

this.storage.readItems();

return this.storage.nextId();

}

incrementId() {

const items = this.storage.readItems();

this.storage.incrementNextId();

this.storage.writeItems(items);

}

supportedFileFormats() {

return this.storage.readItems();

}

getMediaPath(id) {

for (const item of this.supportedFileFormats()) {

const fullPath = this.path + '/' + String(id) + '.' + item;

if (fs.existsSync(fullPath)) {

return fullPath;

}

}

return null;

}

};

module.exports = MediaRepository;

7. app.js

'use strict';

const express = require('express');

const app = express();

const bodyParser = require('body-parser');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

const apiRouter = require('./routes/apiRouter');

app.use('/api', apiRouter);

const expressSwaggerGenerator = require('express-swagger-generator');

const expressSwagger = expressSwaggerGenerator(app);

const options = {

swaggerDefinition: {

info: {

description: 'description',

title: 'title',

version: '1.0.0',

},

host: 'localhost:3000',

produces: ["application/json"],

},

basedir: \_\_dirname,

files: ['./routes/\*\*/\*.js', './models/\*\*/\*.js'],

};

expressSwagger(options);

app.all('/', function (req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

app.listen(3000, function () { console.log('Server is ready'); });

8. jsonStorage.js

const fs = require('fs');

class JsonStorage{

constructor(filePath) {

this.filePath = filePath;

}

nextId() {

return (JSON.parse(fs.readFileSync(this.filePath))).nextId;

}

incrementNextId() {

const file = JSON.parse(fs.readFileSync(this.filePath));

file.nextId++;

fs.writeFileSync(this.filePath, JSON.stringify(file, null, 4, (err) => {

if (err) throw err;

}));

}

readItems() {

return (JSON.parse(fs.readFileSync(this.filePath))).items;

}

writeItems(items) {

const file = (JSON.parse(fs.readFileSync(this.filePath)));

file.items = items;

fs.writeFileSync(this.filePath, JSON.stringify(file, null, 4), (err) => {

if(err) throw err;

});

}

};

module.exports = JsonStorage;

9. mediaRepository.js

const JsonStorage = require('./jsonStorage');

const fs = require("fs");

class MediaRepository {

constructor(filePath) {

this.path = filePath;

this.storage = new JsonStorage(filePath + ".json");

}

getNextId() {

this.storage.readItems();

return this.storage.nextId();

}

incrementId() {

const items = this.storage.readItems();

this.storage.incrementNextId();

this.storage.writeItems(items);

}

supportedFileFormats() {

return this.storage.readItems();

}

getMediaPath(id) {

for (const item of this.supportedFileFormats()) {

const fullPath = this.path + '/' + String(id) + '.' + item;

if (fs.existsSync(fullPath)) {

return fullPath;

}

}

return null;

}

};

module.exports = MediaRepository;

10. apiRouter.js

const express = require('express');

const router = express.Router();

const userRouter = require('./userRouter');

const planetRouter = require('./planetRouter');

const mediaRouter = require('./mediaRouter');

router.use('/users', userRouter);

router.use('/planets', planetRouter);

router.use('/media', mediaRouter);

module.exports = router;

11. mediaRouter.js

const router = require('express').Router();

const mediaController = require('./../controllers/mediaController');

/\*\*

\* @route GET /api/media/{id}

\* @group Media

\* @param {integer} id.path.required ID of the media

\* @returns {file} 200 - media file by ID

\* @returns {Error} 404 - not found

\*/

router.get("/:id", mediaController.getMediaById);

/\*\*

\* @route POST /api/media

\* @group Media

\* @comsumes multipart/form-data

\* @param {file} image.formData.required - new file

\* @returns {file} 201 - Media sdded

\* @returns {Error} 400 - bad reguest

\*/

router.post("/", mediaController.addMedia);

module.exports = router;

12. planetRouter.js

const express = require('express');

const planetController = require('../controllers/planetController');

const router = express.Router();

/\*\*

\* returns all planets

\* @route GET /api/planets/

\* @group Planets - planet operations

\* @param {integer} page.query - page number

\* @param {integer} per\_page.query - items per page

\* @returns {Array.<Planet>} Planets - all planets

\*/

router.get('/', planetController.getPlanets);

/\*\*

\* return planet by id

\* @route GET /api/planets/{id}

\* @group Planets - planet operations

\* @param {integer} id.path.required - id of the Planet - eg: 1

\* @returns {Planet.model} 200 - Planet object

\* @returns {Error} 404 - Planet not found

\*/

router.get('/:id', planetController.getPlanetById);

/\*\*

\* add planet

\* @route POST /api/planets/

\* @group Planets - planet operations

\* @param {Planet.model} id.body.required - new Planet object

\* @returns {Planet.model} 201 - added Planet object

\*/

router.post('/', planetController.addPlanet);

/\*\*

\* update planet

\* @route PUT /api/planets/

\* @group Planets - planet operations

\* @param {Planet.model} id.body.required - new Planet object

\* @returns {Planet.model} 200 - changed Planet object

\*/

router.put('/', planetController.updatePlanet);

/\*\*

\* delete planet

\* @route DELETE /api/planets/{id}

\* @group Planets - planet operations

\* @param {integer} id.path.required - id of the Planet - eg: 1

\* @returns {Planet.model} 200 - deleted Planet object

\* @returns {Error} 404 - Planet not found

\*/

router.delete('/:id', planetController.deletePlanet);

module.exports = router;

13. userRouter.js

const express = require('express');

const userController = require('./../controllers/userController');

const router = express.Router();

/\*\*

\* returns user by id

\* @route GET /api/users/{id}

\* @group Users - user operations

\* @param {integer} id.path.required - id of the User - eg: 1

\* @returns {User.model} 200 - User object

\* @returns {Error} 404 - User not found

\*/

router.get('/:id', userController.getUserById);

/\*\*

\* returns all users

\* @route GET /api/users

\* @group Users - user operations

\* @param {integer} page.query - page number

\* @param {integer} per\_page.query - items per page

\* @returns {Array.<User>} 200 - User array

\*/

router.get('/', userController.getUsers);

module.exports = router;

14. mediaController.js

const path = require('path');

const multer = require('multer');

const MediaRepository = require('./../repositories/mediaRepository');

const mediaRepository = new MediaRepository(path.resolve(\_\_dirname, '../data/media'));

const upload = multer

({

storage: multer.diskStorage

({

destination: function (req, file, cb) {

cb(null, mediaRepository.path);

},

filename: function (req, file, cb) {

const fileFormat = file.mimetype.split('/')[1];

cb(null, `${String(mediaRepository.getNextId())}.${fileFormat}`);

}

}),

}).any();

module.exports = {

async getMediaById(req, res) {

try {

const mediaId = Number(req.params.id);

const path = mediaRepository.getMediaPath(mediaId);

if (path) {

res.sendFile(path);

}

else {

res.status(404).send({ media: null, message: 'Not found' });

}

} catch (err) {

console.log(err.message);

res.status(500).send({ media: null, message: 'Server error' });

}

},

async addMedia(req, res) {

try {

upload(req, res, (err) => {

if (err) {

console.log('err ', err.message);

res.status(500).send({ media: null, message: 'Server error' });

} else if (req.files) {

res.status(201).send({ mediaId: mediaRepository.getNextId(), message: 'Success. Media added' });

mediaRepository.incrementId();

} else {

res.status(400).send({ message: 'Bad request' });

}

});

}

catch (err) {

console.log(err.message);

res.status(500).send({ media: null, message: 'Server error' });

}

}

};

15. planetController.js

const planetRepository = require('../repositories/planetRepository');

const repo = new planetRepository("./data/planets.json");

module.exports = {

getPlanets(req, res) {

const planets = repo.getPlanets();

const page = req.query.page;

const per\_page = req.query.per\_page;

const pages\_total = Math.ceil(planets.length / per\_page);

if (page && per\_page)

if (page <= pages\_total && page > 0 && per\_page > 0)

res.status(200).send(planets.slice(per\_page \* (page - 1), page \* per\_page));

else

res.sendStatus(400);

else

res.status(200).send(planets);

res.end;

},

getPlanetById(req, res) {

const planet = repo.getPlanetById(req.params.id);

if (planet)

res.status(200).send(planet);

else

res.sendStatus(404);

res.end();

},

addPlanet(req, res) {

const planet = repo.addPlanet(req.body);

res.status(200).send(planet);

res.end();

},

updatePlanet(req, res) {

const planet = repo.updatePlanet(req.body);

if (planet)

res.status(200).send(planet);

else

res.sendStatus(404);

},

deletePlanet(req, res) {

const planet = repo.deletePlanet(req.params.id);

if (planet)

res.status(200).send(planet);

else

res.sendStatus(404);

res.end();

},

};

16. userController.js

const userRepository = require('../repositories/userRepository');

const repo = new userRepository("./data/users.json");

module.exports = {

getUsers(req, res) {

const users = repo.getUsers();

const page = req.query.page;

const per\_page = req.query.per\_page;

const pages\_total = Math.ceil(users.length / per\_page);

if (page && per\_page)

if (page <= pages\_total && page > 0 && per\_page > 0)

res.status(200).send(users.slice(per\_page \* (page - 1), page \* per\_page));

else

res.sendStatus(400);

else

res.status(200).send(users);

res.end;

},

getUserById(req, res) {

const user = repo.getUserById(req.params.id);

if (user)

res.status(200).send(user);

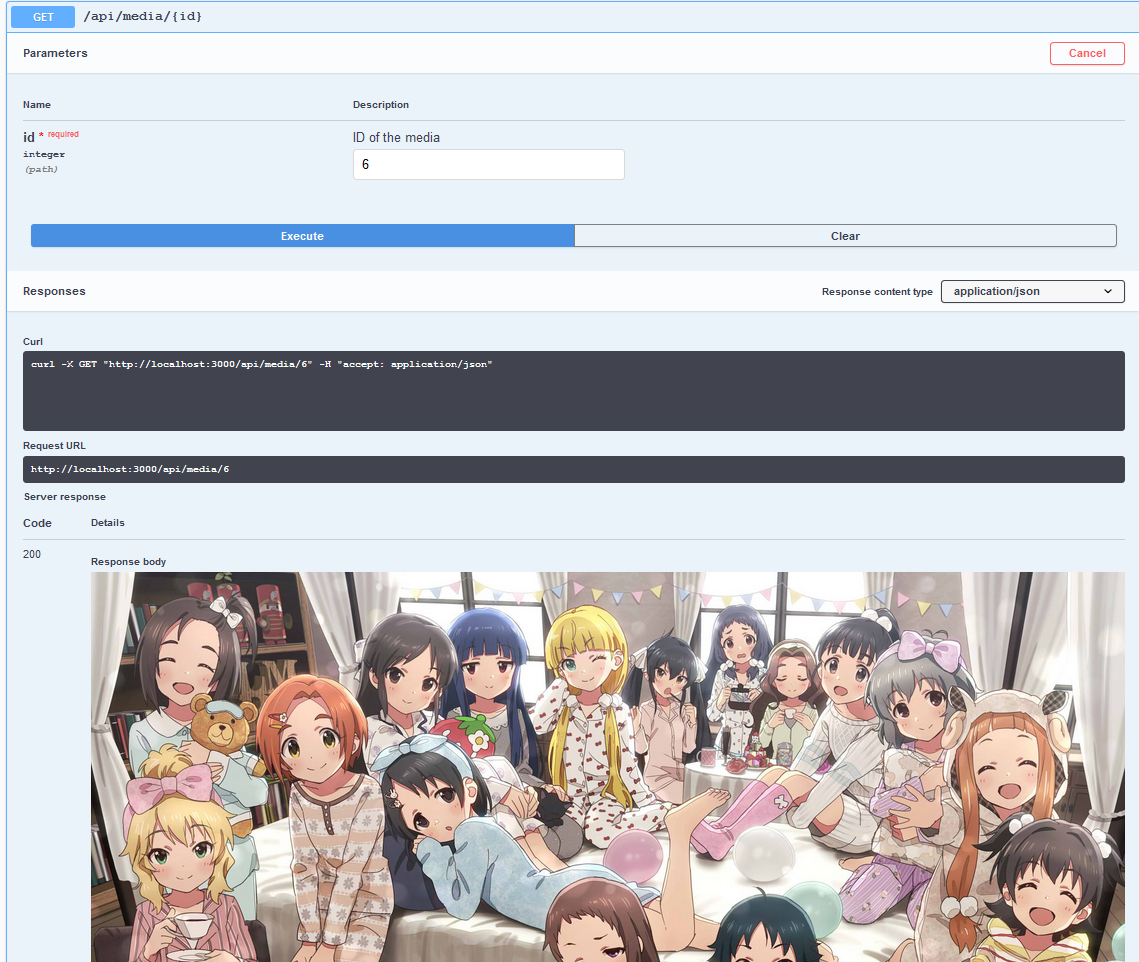
else {

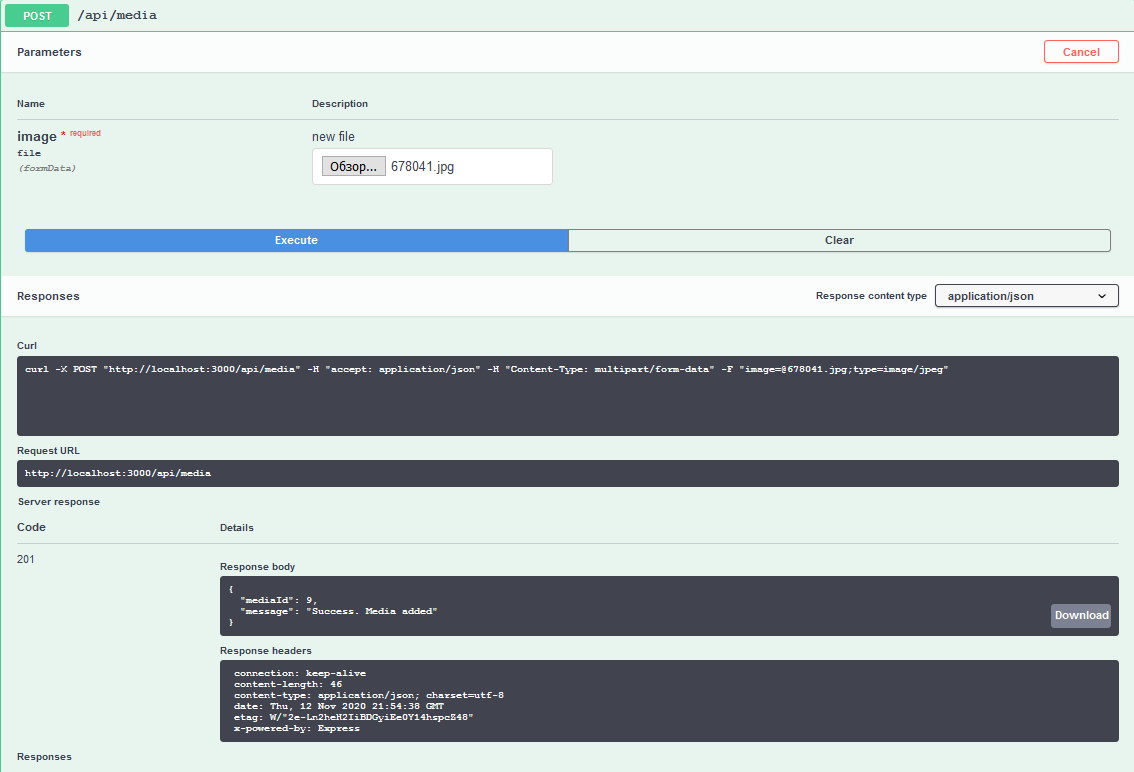
res.sendStatus(404);

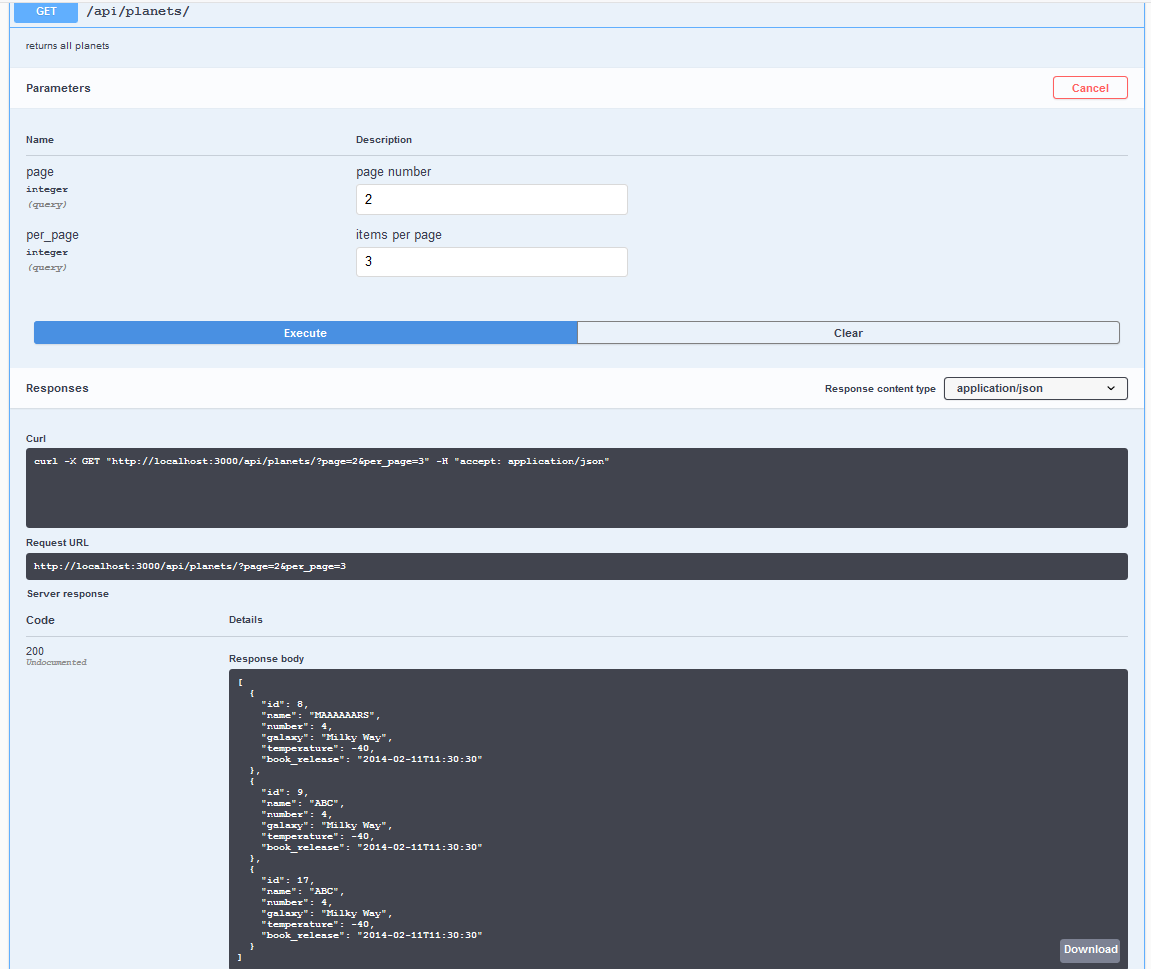
}

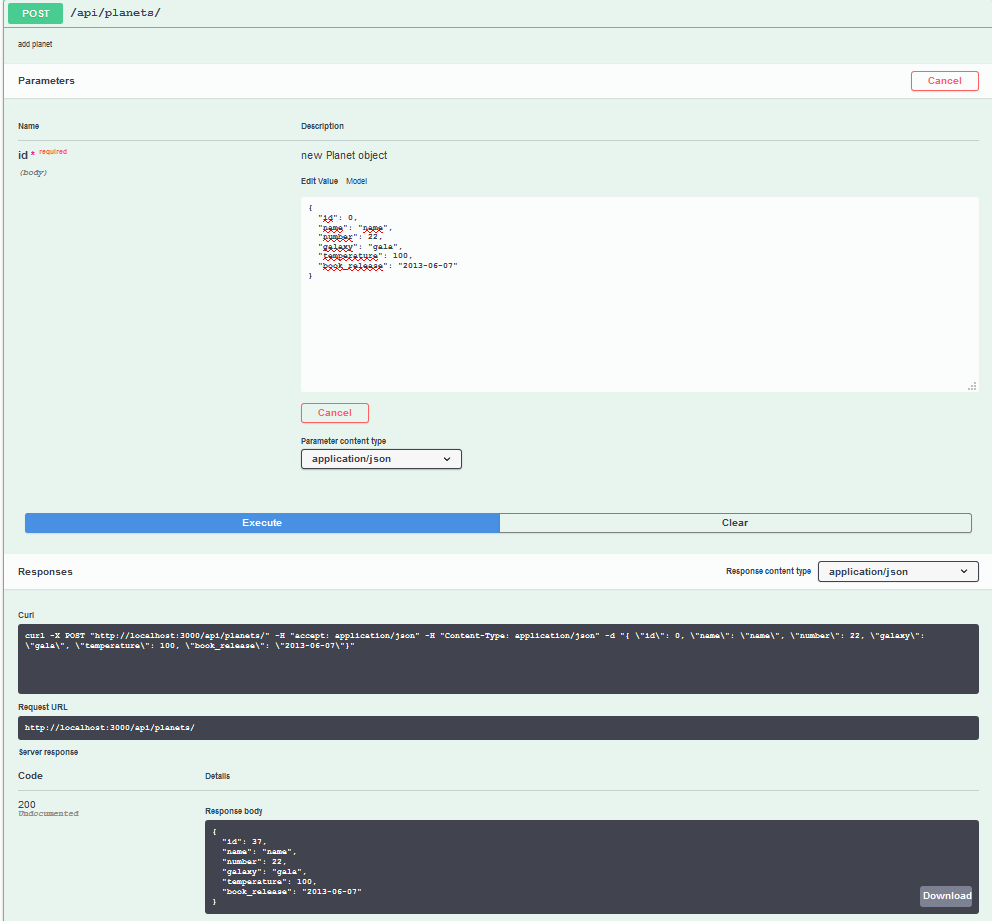
},

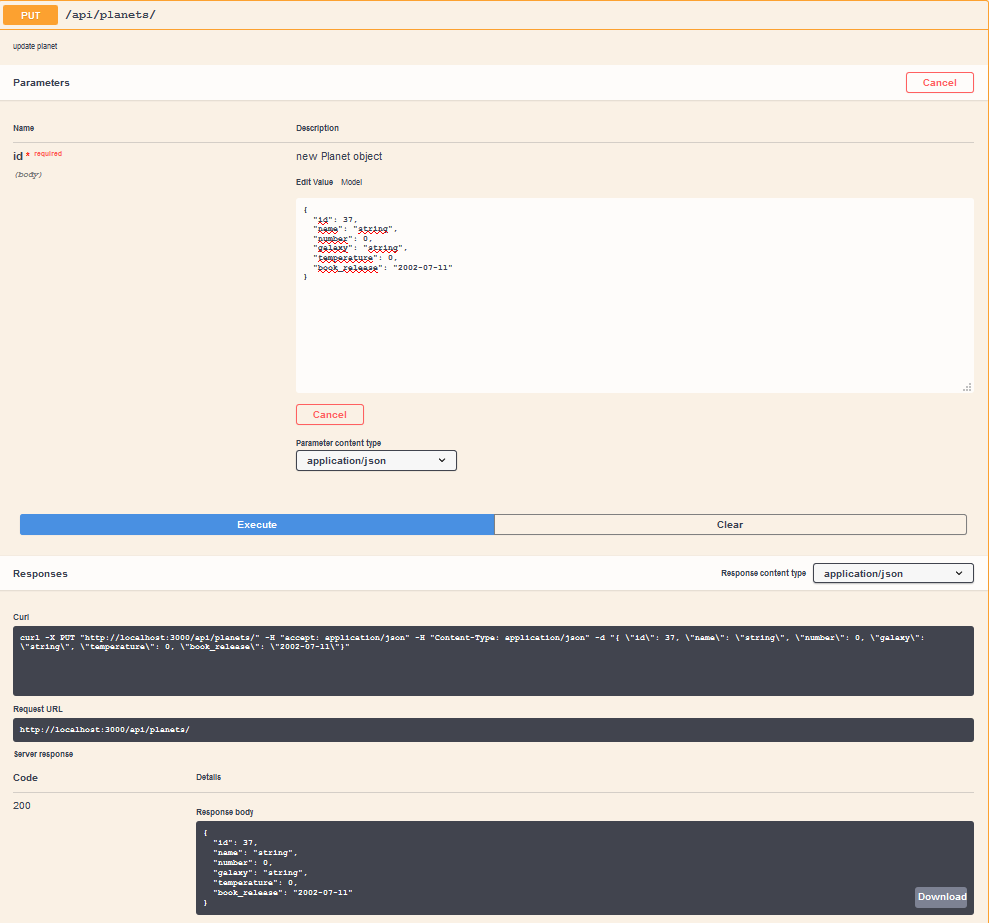
};

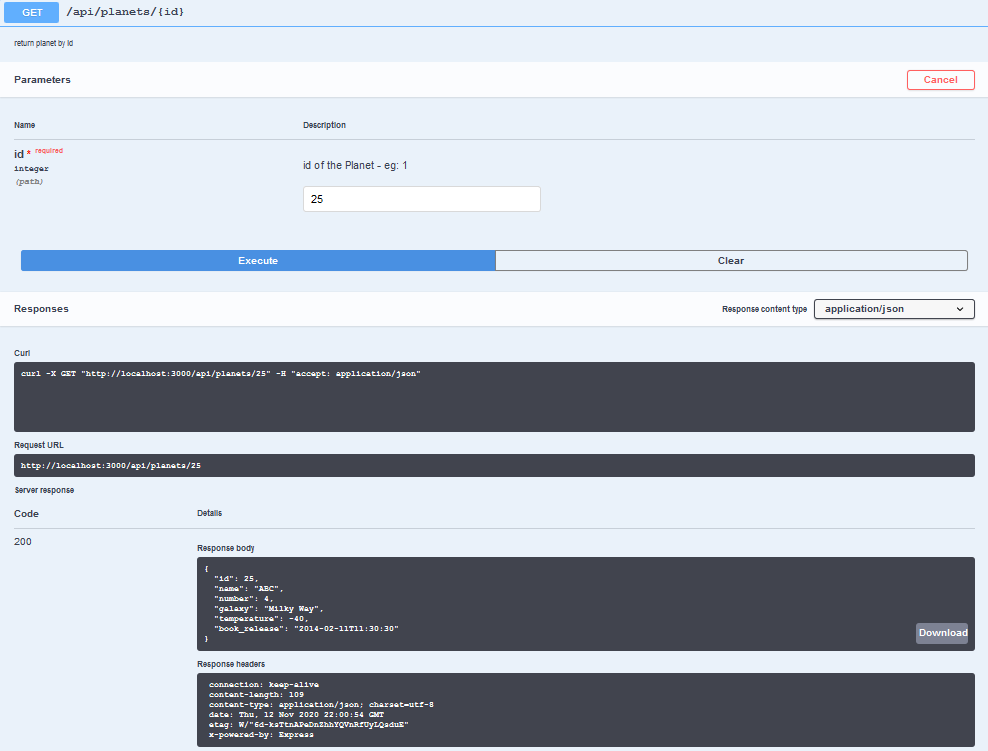
Результати:

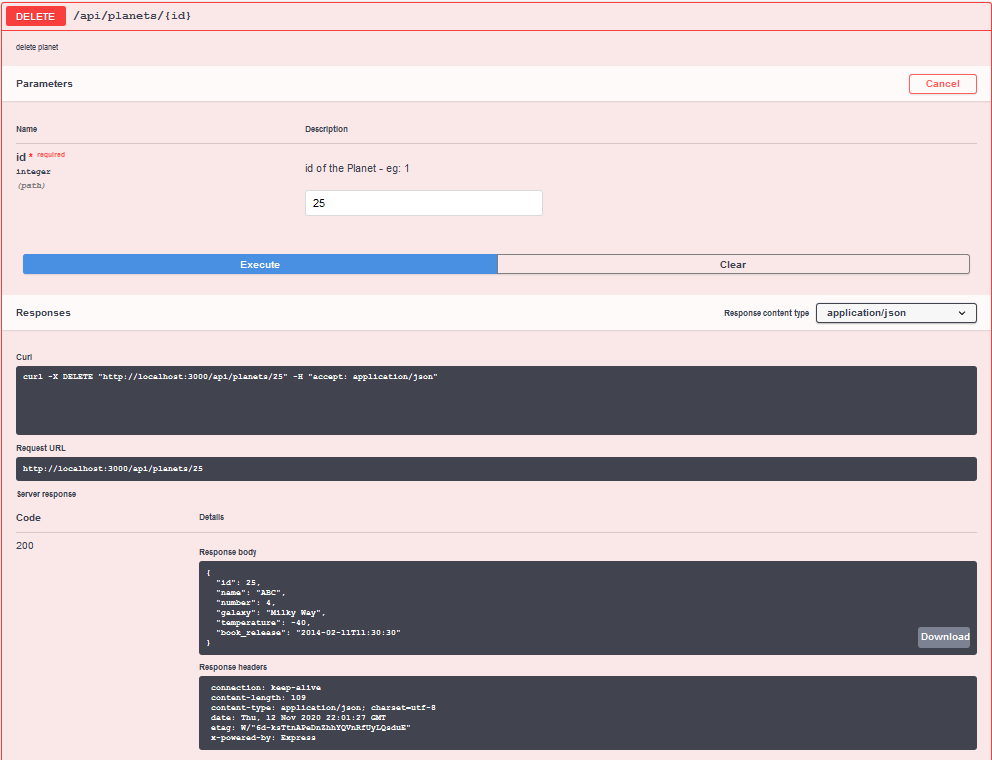


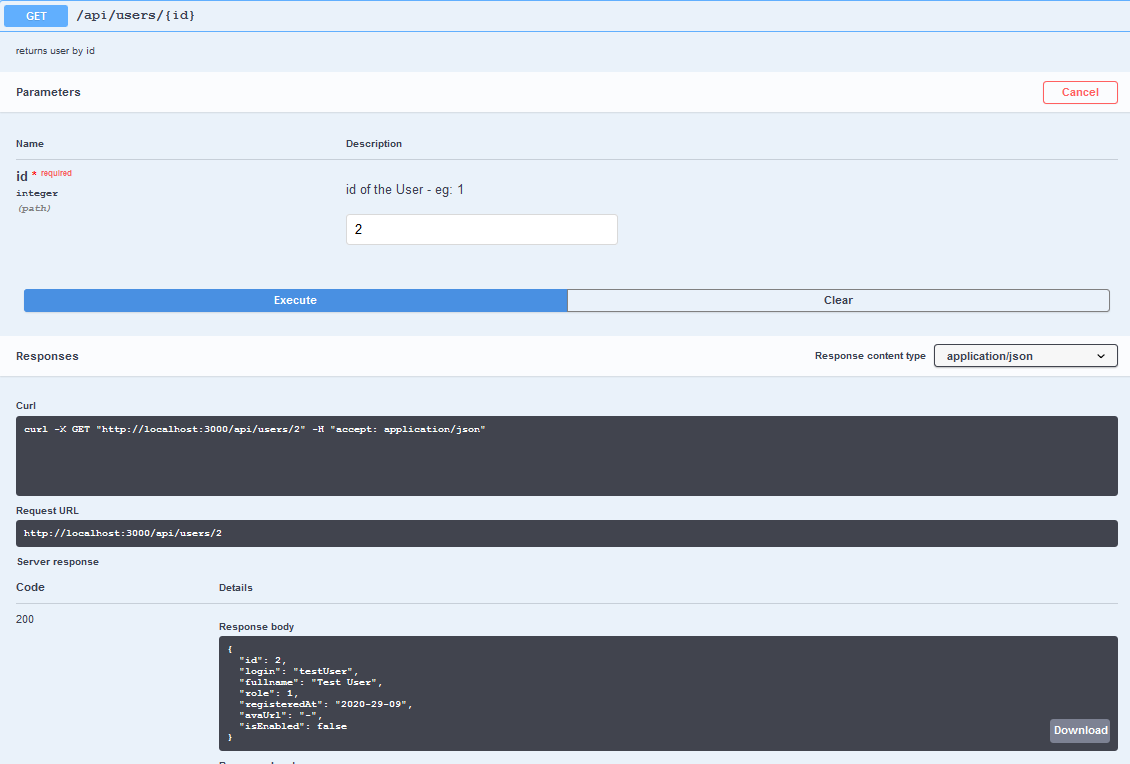


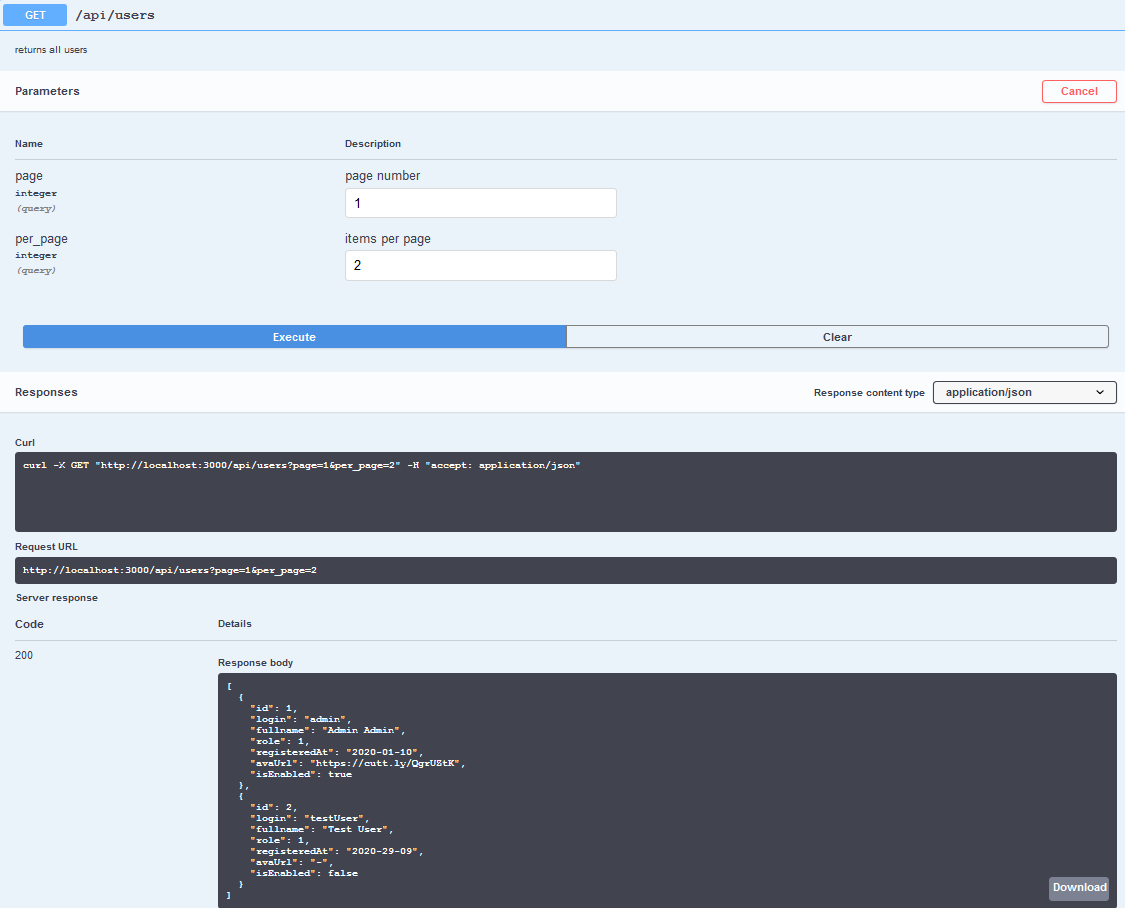












Висновки: В даній лабораторній роботі я навчився створювати простий JSON HTTP API веб-сервер, що надає доступ до ресурсів зі сховища. Документувати API за допомогою Swagger та вивчив коди статутів помилок серверів