

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 II курсу  групи КП-93  Катрич Владислав Сергійович  (*прізвище, ім’я, по батькові*)  варіант №25 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2019

**Мета роботи**

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

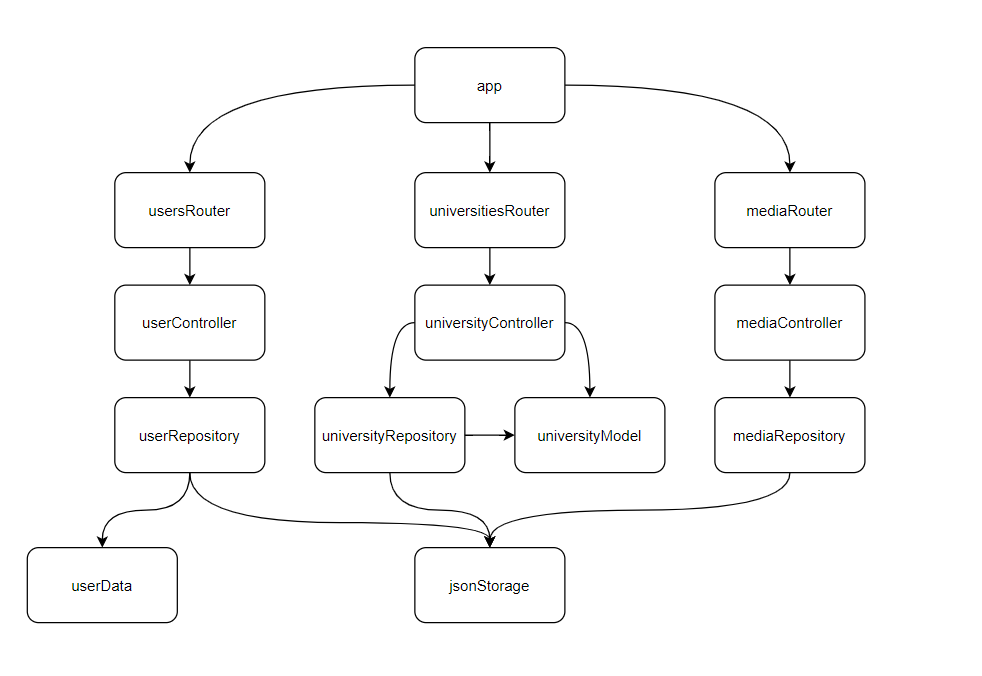
**Постановка завдання**

Реалізувати JSON HTTP API веб-сервер зі сторінкою документації по цьому API.  
API має дозволяти отримувати дані про користувачів а також забезпечувати CRUD операції над сутностями за варіантом.  
Додатково реалізувати URI для завантаження та перегляду зображень.

**Вказівки до виконання**

1. Видалити код, що отримує ввід від користувача через консоль.
2. Створити у app.js простий веб-сервер за допомогою фреймворку [**ExpressJS**](https://expressjs.com/).
3. JSON HTTP API
   1. Користувачі:
      1. Описати роутер користувачів (у директорії routes) із обробниками GET запитів, що повертають список користувачів або одного користувача за ідентифікатором у форматі JSON (з відповідним Content-type):
      2. GET /api/users -> [{User}, {User}]
      3. GET /api/users/:id -> {User}
      4. Додати HTTP контроллер користувачів (у директорії controllers), що на запити від HTTP клієнта повертає JSON відповіді.
   2. Сутності за варіантом:
      1. Описати роутер сутностей за варіантом (у директорії routes) із обробниками GET, POST, PUT, DELETE запитів для CRUD операцій над цими моделями.
      2. GET /api/{entities} -> [{Entity}, {Entity}]
      3. GET /api/{entities}/:id -> {Entity}
      4. POST /api/{entities} -> {Entity} // return added entity
      5. PUT /api/{entities} -> {Entity} // return changed entity
      6. DELETE /api/{entities}/:id -> {Entity} // return removed entity
      7. Налаштувати body-parser для розбору JSON тіла HTTP запиту.
      8. Додати HTTP контроллер сутностей за варіантом (у директорії controllers), що на запити від HTTP клієнта повертає JSON відповіді.
4. Пагінація результатів (див. [пояснення і приклад](https://docs.google.com/document/d/1LOpOhvi3AunxM0mmfkeOFB0Jeo9plkBI2VYFkMY4uIw/edit)):
   1. Додати пагінацію результатів для GET /api/users
   2. Додати пагінацію результатів для GET /api/{entities}
5. Документація API за допомогою Swagger (див. [приклад](https://docs.google.com/document/d/131QVkR2um-Q-oJj87xkcscA-um1WlnSx6xHZTI9IIWw/edit?usp=sharing)):
   1. Налаштувати express-swagger-generator у app.js і підключити шляхи до routes i models файлів.
   2. Задокументувати всі моделі за допомогою коментарів.
   3. Задокументувати всі роутери за допомогою коментарів.
   4. Навчитись перевіряти API через Swagger сторінку /api-docs.
6. Зображення:
   1. Додати роутер media із двома шляхами:
   2. POST /api/media -> {media\_id}
   3. GET /api/media/{media\_id} -> Image
   4. Створити відповідні контроллер, репозиторій та спеціальне сховище медіа.
   5. Завантажені файли розміщувати у директорії data/media/
7. Статуси помилок:
   1. У випадку помилки (неіснування ресурсу з таким id) повертати HTTP код статуса 404 Not found.
   2. У випадку некоректно сформованого запиту (наприклад, сутність, яка додається або оновлюється не містить обов'язкових полів) повертати HTTP код статуса 400 Bad request.
   3. Якщо запит на створення нової сутності пройшов успішно - потрібно повертати код 201 Created, а в тілі весь новостворений об'єкт із новим ідентифікатором, за яким клієнтська програма зможе, наприклад, перенаправити користувача на сторінку створеної сутності
8. Будь-яка API відповідь від сервера (успішна чи ні) має повертати у тілі JSON об'єкт або масив з даними.

**Діаграма залежностей модулів:**



**Код розроблених модулів:**

|  |
| --- |
| **app** |
| const express = require('express');  const fileUpload = require('express-fileupload');  const bodyParser = require('body-parser');  const exphbs = require('express-handlebars');  const routes = [];  routes['users'] = require('./routes/users');  routes['universities'] = require('./routes/universities');  routes['media'] = require('./routes/media');  const PORT = process.env.PORT || 3000;  const app = express();  const hbs = exphbs.create({      defaultLayout: 'main',      extname: 'hbs',  });  const expressSwagger = require('express-swagger-generator')(app);  const options = {      swaggerDefinition: {          info: {              description:                  'JSON HTTP API web server that provides access to resources from the repository',              title: 'lab2 - JSON HTTP API Web Server',              version: '1.0.0',          },          host: `localhost:${PORT}`,          produces: ['application/json'],      },      basedir: \_\_dirname,      files: ['./routes/\*\*/\*.js', './models/\*\*/\*.js'],  };  expressSwagger(options);  app.engine('hbs', hbs.engine);  app.set('view engine', 'hbs');  app.use(      fileUpload({          createParentPath: true,      })  );  app.use(bodyParser.json());  app.use('/api/users', routes['users']);  app.use('/api/universities', routes['universities']);  app.use('/api/media', routes['media']);  app.listen(PORT, () => {      console.log(`Server has been started on http://localhost:${PORT}`);  }); |

|  |
| --- |
| **mediaRouter** |
| const { Router } = require('express');  const mediaController = require('../controllers/mediaController');  const router = Router();  /\*\*   \* Post an image   \* @route POST /api/media   \* @group Media - upload and get images   \* @consumes multipart/form-data   \* @param {file} image.formData.required - uploaded image   \* @returns {integer} 200 - added image id   \*/  router.post('/', mediaController.apiPostMedia);  /\*\*   \* Get an image   \* @route GET /api/media/{id}   \* @group Media - upload and get images   \* @param {string} id.path.required - id of the Image   \* @consumes multipart/form-data   \* @returns {integer} 200 - get Image   \* @returns {Error} 404 - Image not found   \*/  router.get('/:id', mediaController.apiGetMedia);  module.exports = router; |

|  |
| --- |
| **universitiesRouter** |
| const { Router } = require('express');  const router = Router();  const universityController = require('../controllers/universityController');  /\*\*   \* Get all universities   \* @route GET /api/universities   \* @group Universities - university operations   \* @returns {Array.<University>} University - all universities   \*/  router.get('/', universityController.getUniversities);  /\*\*   \* Get university by id   \* @route GET /api/universities/{id}   \* @group Universities - university operations   \* @param {string} id.path.required - id of the University   \* @returns {University.model} 200 - University object   \* @returns {Error} 404 - University not found   \*/  router.get('/:id', universityController.getUniversity);  /\*\*   \* Post new university   \* @route POST /api/universities   \* @group Universities - university operations   \* @param {University.model} university.body.required - new University object   \* @returns {University.model} 201 - added University object   \*/  router.post('/', universityController.postUniversity);  /\*\*   \* Put university   \* @route PUT /api/universities/{id}   \* @group Universities - university operations   \*   \* @param {string} id.path.required - id of the University   \* @param {University.model} university.body.required - new University object   \* @returns {University.model} 200 - changed University object   \* @returns {Error} 404 - University not found   \*/  router.put('/:id', universityController.putUniversity);  /\*\*   \* Delete university   \* @route DELETE /api/universities/{id}   \* @group Universities - university operations   \* @param {string} id.path.required - id of the University   \* @returns {University.model} 200 - deleted University object   \* @returns {Error} 404 - University not found   \*/  router.delete('/:id', universityController.deleteUniversity);  module.exports = router; |

|  |
| --- |
| **userRouter** |
| const { Router } = require('express');  const router = Router();  const userController = require('../controllers/userController');  /\*\*   \* Get all users   \* @route GET /api/users   \* @group Users - user operations   \* @returns {Array.<User>} User - all users   \*/  router.get('/', userController.getUsers);  /\*\*   \* Get user by id   \* @route GET /api/users/{id}   \* @group Users - user operations   \* @param {string} id.path.required - id of the User   \* @returns {User.model} 200 - User object   \* @returns {Error} 404 - User not found   \*/  router.get(`/:id`, userController.getUser);  module.exports = router; |

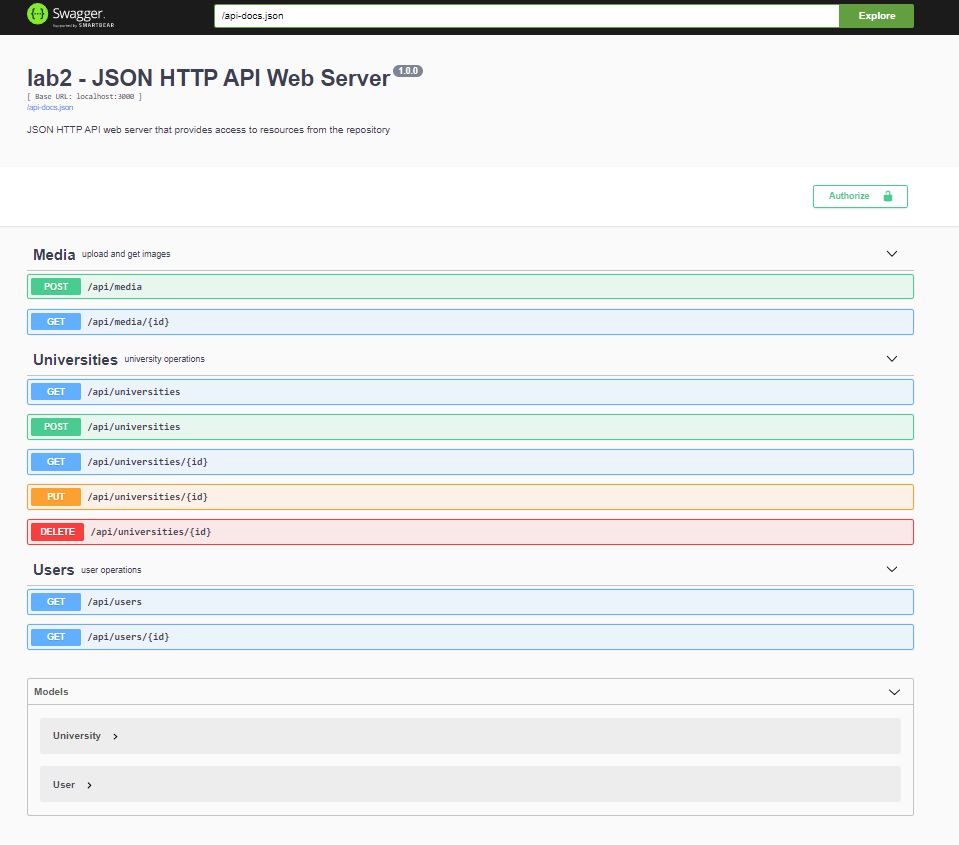
|  |
| --- |
| **mediaRepository** |
| const JsonStorage = require('../jsonStorage');  const fs = require('fs');  class MediaRepository {      constructor(filePath) {          this.storage = new JsonStorage(filePath);      }      getMedia() {          return this.storage.readItems();      }      getMediaById(id) {          for (const item of this.storage.readItems()) {              if (item.id === id) {                  return fs.readFileSync(item.path);              }          }          return null;      }      addMedia(mediaModel) {          this.storage.writeItems(mediaModel);      }  }  module.exports = MediaRepository; |

|  |
| --- |
| **mediaController** |
| const MediaRepository = require('../repositories/mediaRepository');  const mediaRepository = new MediaRepository('data/media.json');  module.exports = {      apiPostMedia(req, res) {          try {              if (!req.files) {                  res.send({                      status: false,                      message: 'No file uploaded',                  });              } else {                  const image = {};                  image['id'] = req.files['image'].md5;                  image['path'] = `./uploads/${req.files['image'].name}`;                  const images = mediaRepository.getMedia();                  const isImage = images.find((img) => img.id === image.id);                  if (!isImage) {                      req.files['image'].mv(                          './uploads/' + req.files['image'].name                      );                      images.push(image);                      mediaRepository.addMedia(images);                      res.send(req.files);                  } else                      res.send({                          status: false,                          message: 'This file is already uploaded',                      });              }          } catch (err) {              res.status(500).send(err);          }      },      apiGetMedia(req, res) {          res.set({ 'Content-Type': 'image/png' });          const image = mediaRepository.getMediaById(req.params.id);          if (image) res.send(image);          else {              res.set({ 'Content-Type': 'text/html; charset=utf-8' });              res.status(404).send('This id is missing');          }      },  }; |

|  |
| --- |
| **universityController** |
| const UniversityRepository = require('./../repositories/universityRepository');  const uuidv4 = require('uuid-v4');  const universityRepository = new UniversityRepository('data/universities.json');  const University = require('../models/university');  module.exports = {      getUniversities(req, res) {          const page = parseInt(req.query.page) || 1;          const limit = parseInt(req.query.limit) || 3;          const startIndex = (page - 1) \* limit;          const endIndex = page \* limit;          const universities = universityRepository.getUniversities();          const results = {};          if (startIndex > 0) {              results.previous = {                  page: page - 1,                  limit: limit,              };          }          if (endIndex < universities.length) {              results.next = {                  page: page + 1,                  limit: limit,              };          }          results.results = universities.slice(startIndex, endIndex);          res.send(results);      },      getUniversity(req, res) {          const university = universityRepository              .getUniversities()              .find((u) => u.uuid === req.params.id);          if (!university)              res.status(404).send('No university found for this ID');          res.send(university);      },      postUniversity(req, res) {          const university = new University(              uuidv4(),              req.body['name'] || res.status(400).send(`name not entered`),              req.body['country'] || res.status(400).send(`country not entered`),              req.body['numOfStudents'],              req.body['campus'],              req.body['foundationDate']          );          const universities = universityRepository.getUniversities();          universities.push(university);          universityRepository.addUniversity(universities);          res.status(201).send(university);      },      putUniversity(req, res) {          const universities = universityRepository.getUniversities();          const requiredUuid = universities.findIndex(              (university) => university.uuid === req.params.id          );          if (requiredUuid === -1)              res.status(404).send('No university found for this ID');          else {              universities[requiredUuid].name =                  req.body['name'] || universities[requiredUuid].name;              universities[requiredUuid].country =                  req.body['country'] || universities[requiredUuid].country;              universities[requiredUuid].numOfStudents =                  req.body['numOfStudents'] ||                  universities[requiredUuid].numOfStudents;              universities[requiredUuid].campus =                  req.body['campus'] || universities[requiredUuid].campus;              universities[requiredUuid].foundationDate =                  req.body['foundationDate'] ||                  universities[requiredUuid].foundationDate;              universityRepository.updateUniversity(universities);              res.send(universities[requiredUuid]);          }      },      deleteUniversity(req, res) {          const universities = universityRepository.getUniversities();          const requiredUuid = universities.findIndex(              (university) => university.uuid === req.params.id          );          if (requiredUuid === -1)              res.status(404).send('No university found for this ID');          else {              const university = universities[requiredUuid];              universityRepository.deleteUniversity(requiredUuid);              res.send(university);          }      },  }; |

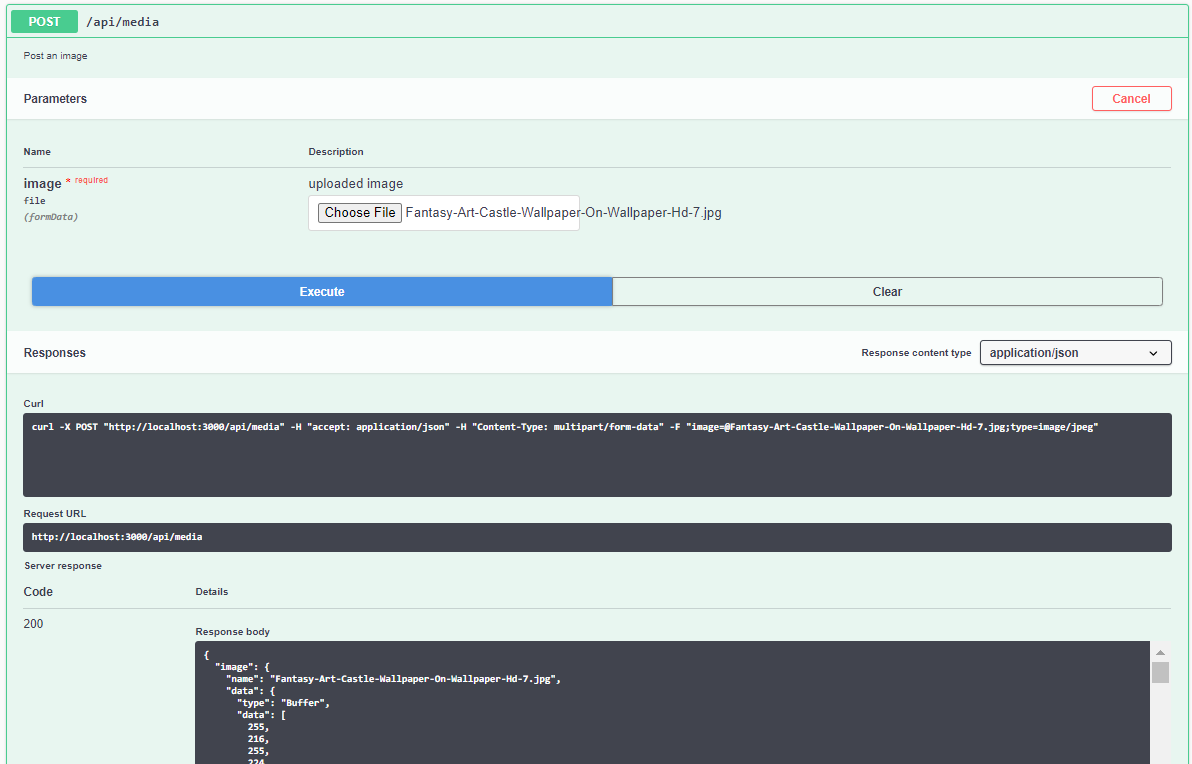
|  |
| --- |
| **userController** |
| const UserRepository = require('./../repositories/userRepository');  const userRepository = new UserRepository('data/users.json');  module.exports = {      getUsers(req, res) {          const page = parseInt(req.query.page) || 1;          const limit = parseInt(req.query.limit) || 3;          const startIndex = (page - 1) \* limit;          const endIndex = page \* limit;          const users = userRepository.getUsers();          const results = {};          if (startIndex > 0) {              results.previous = {                  page: page - 1,                  limit: limit,              };          }          if (endIndex < users.length) {              results.next = {                  page: page + 1,                  limit: limit,              };          }          results.results = users.slice(startIndex, endIndex);          res.send(results);      },      getUser(req, res) {          const user = userRepository              .getUsers()              .find((u) => u.uuid === req.params.id);          if (!user) res.status(404).send('No user found for this ID');          res.send(user);      },  }; |

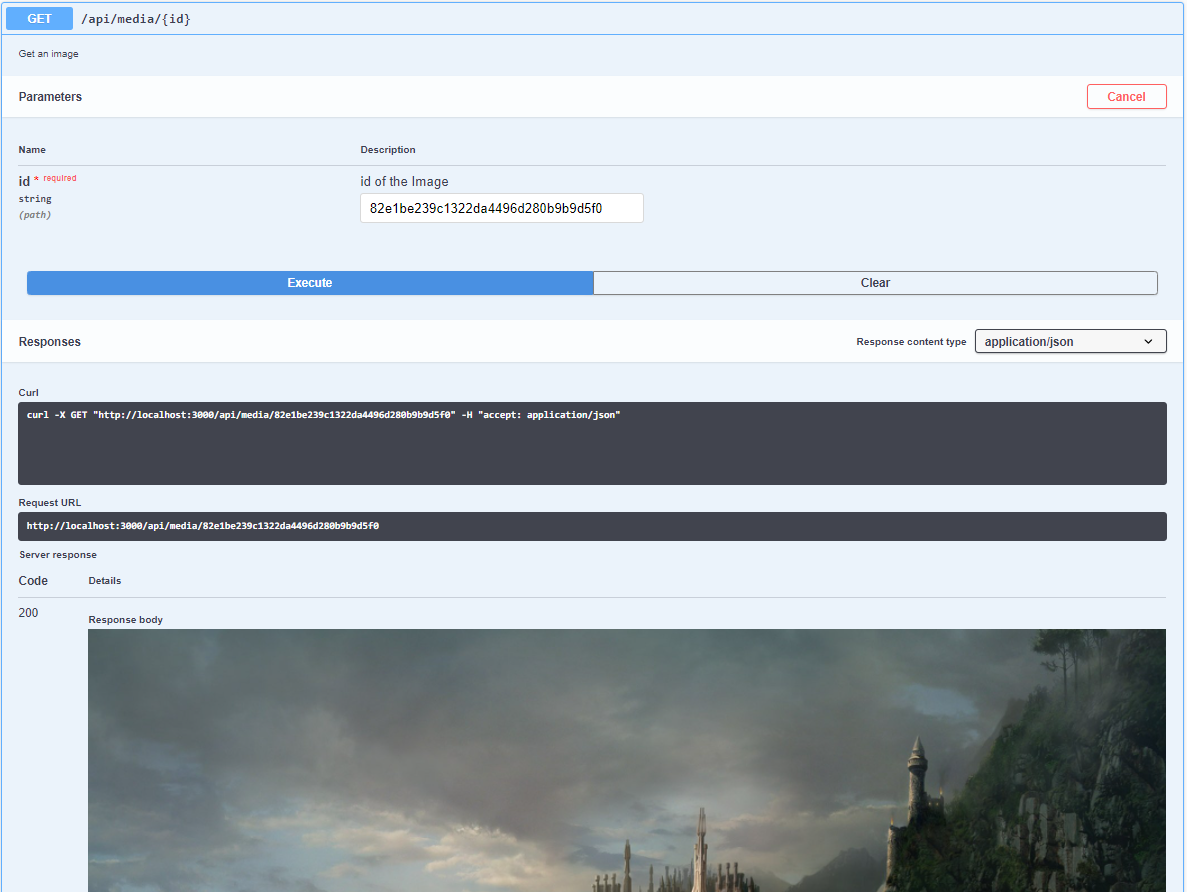
**Swagger сторінка**

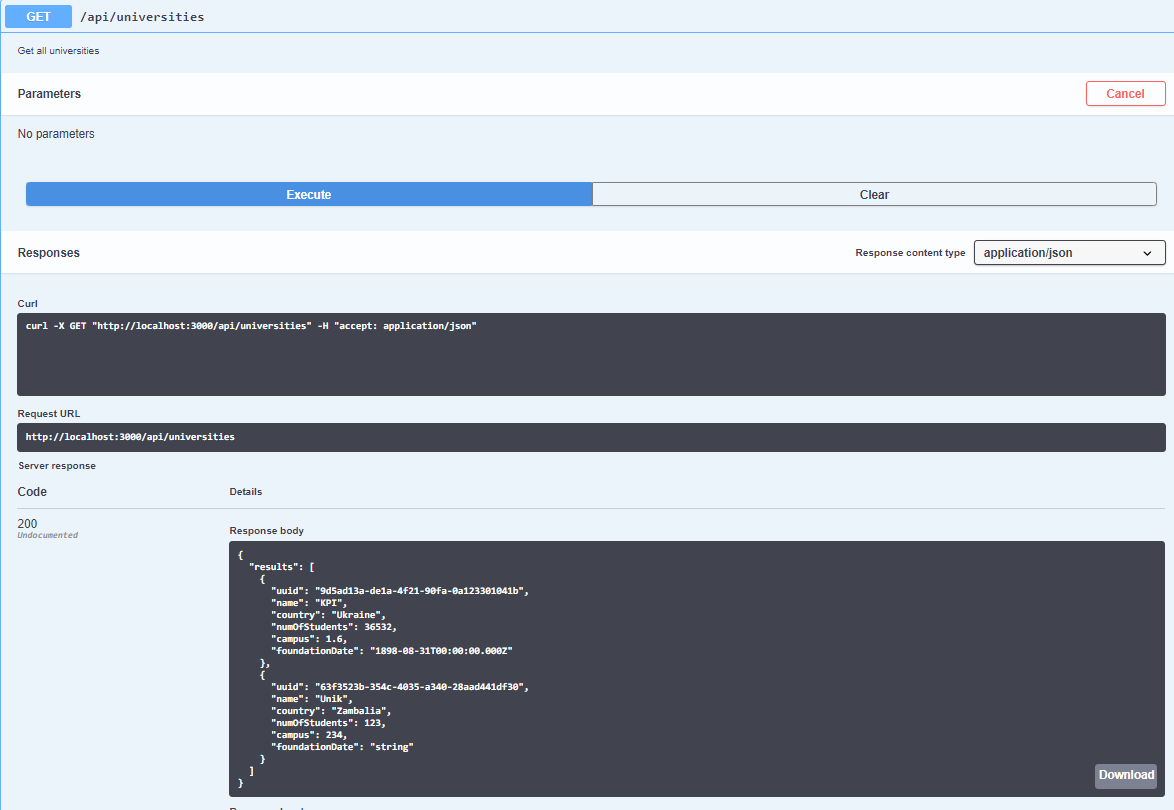


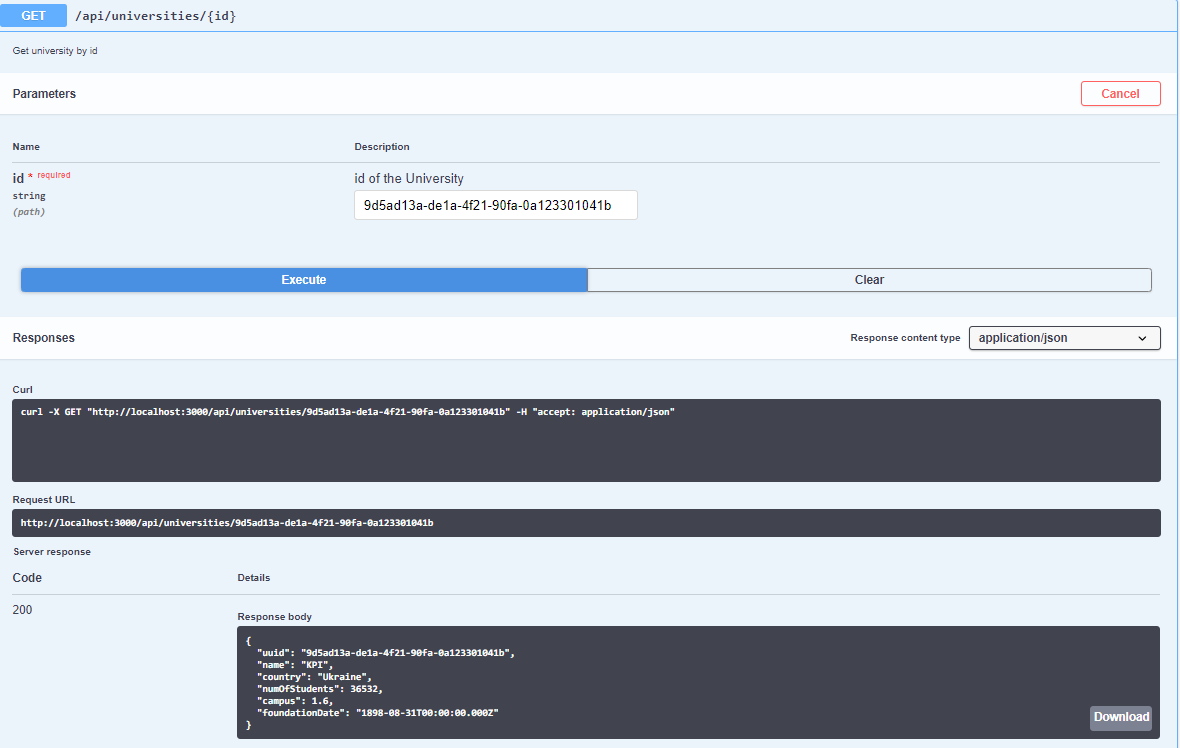


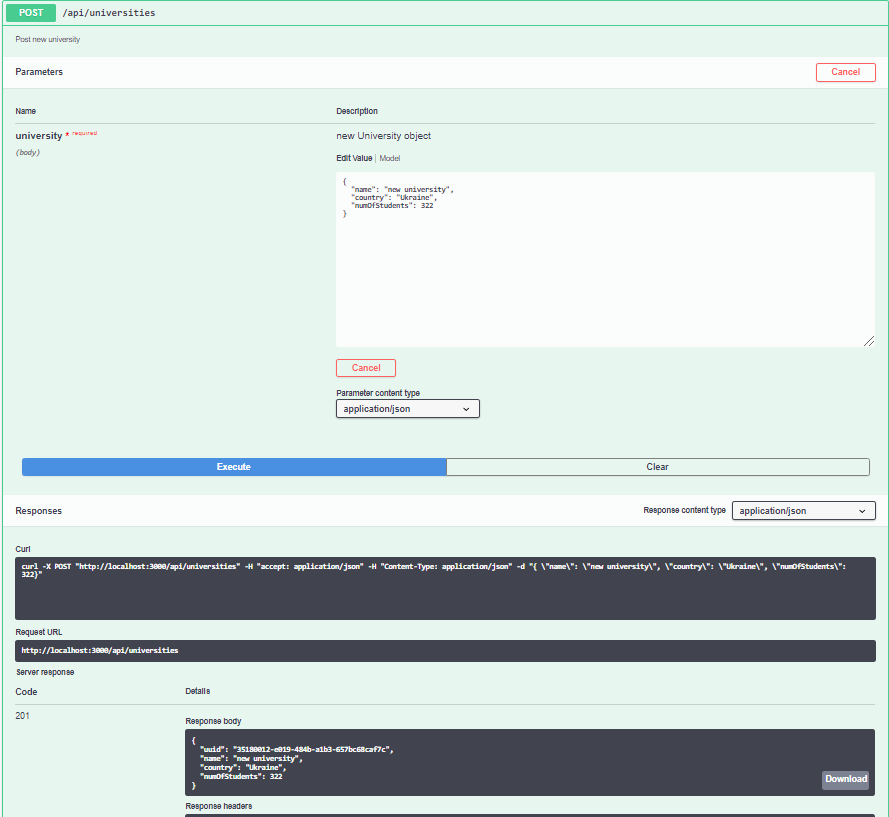
**Приклади взаємодії з JSON REST API**

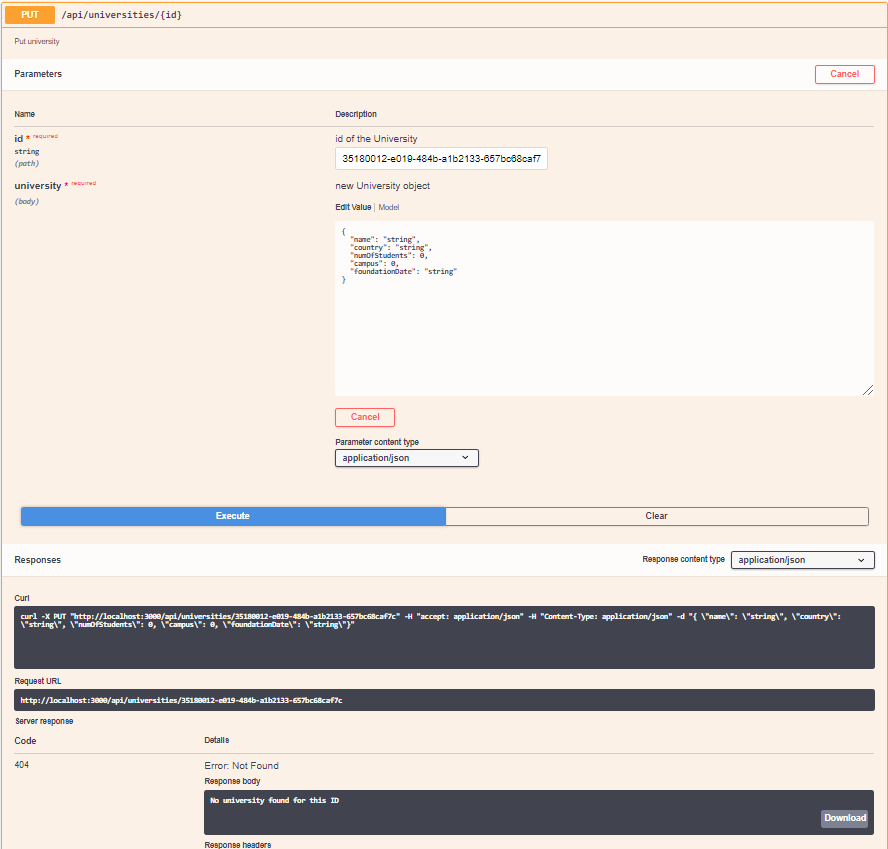


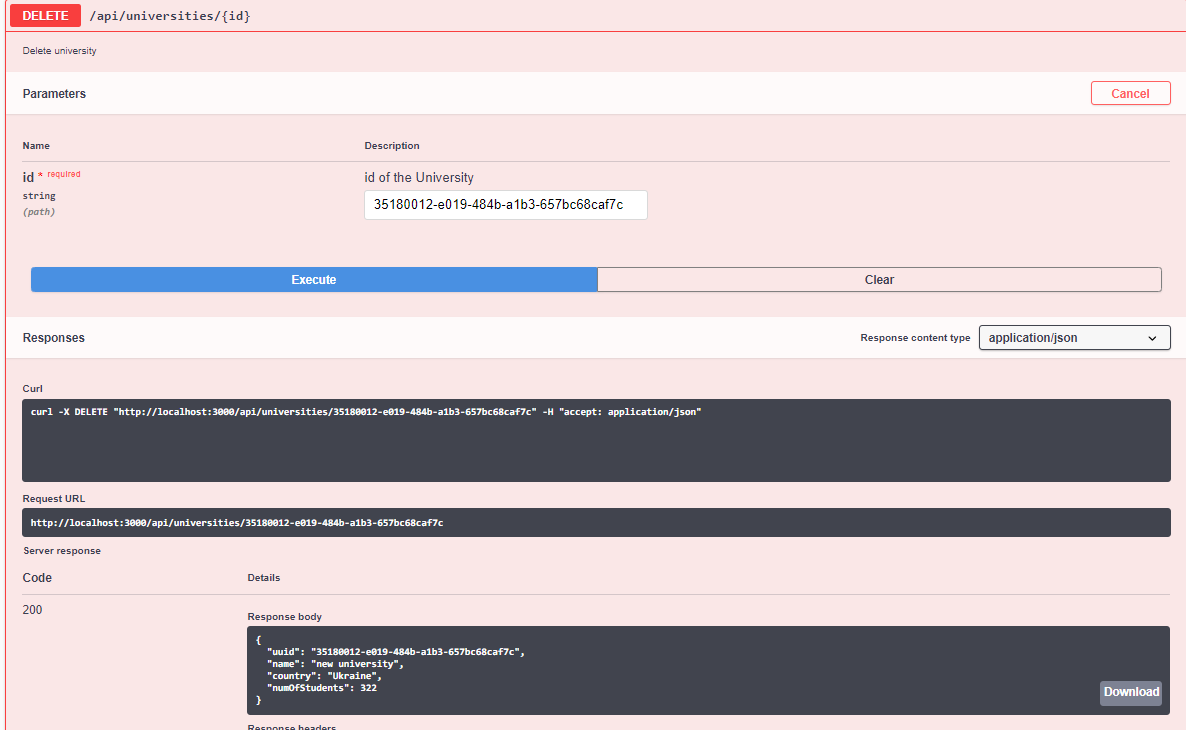


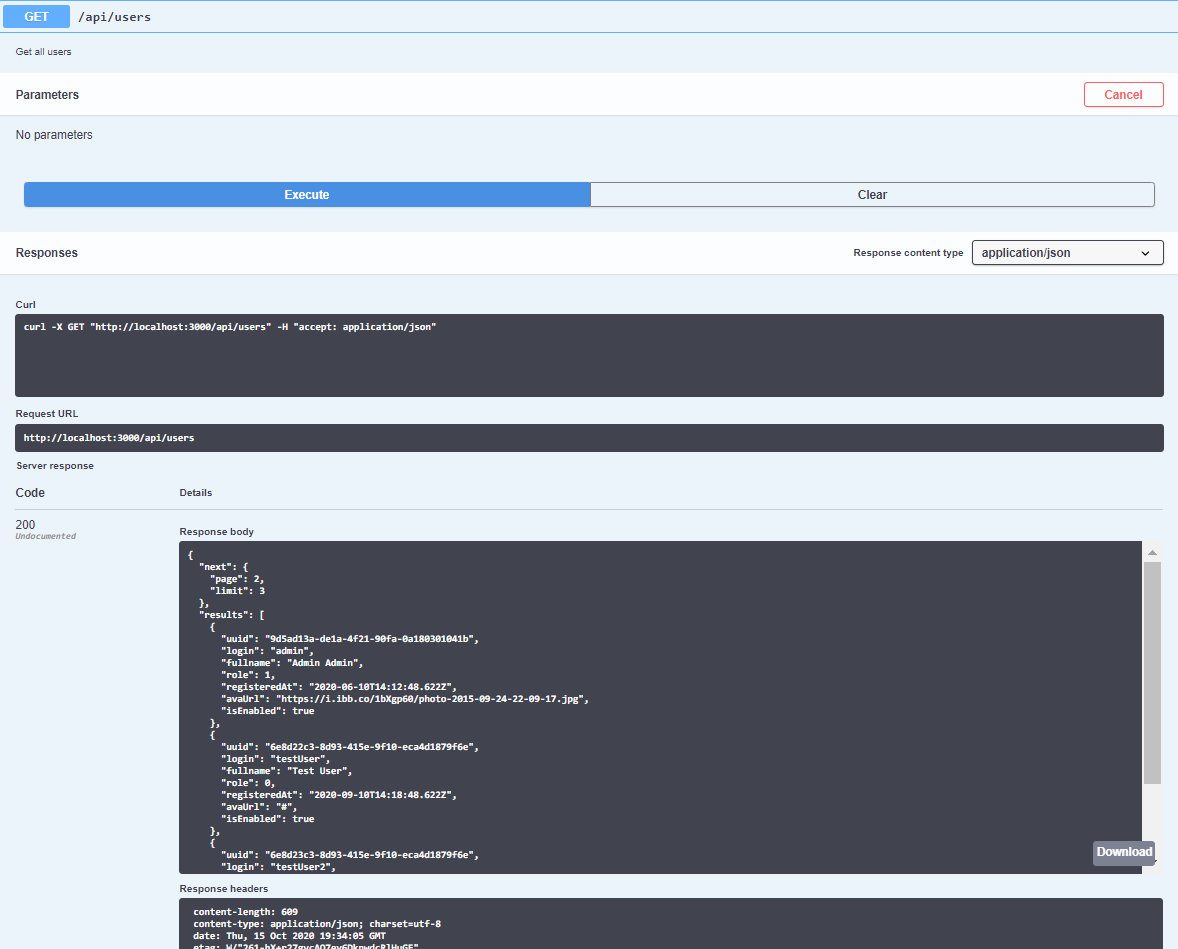


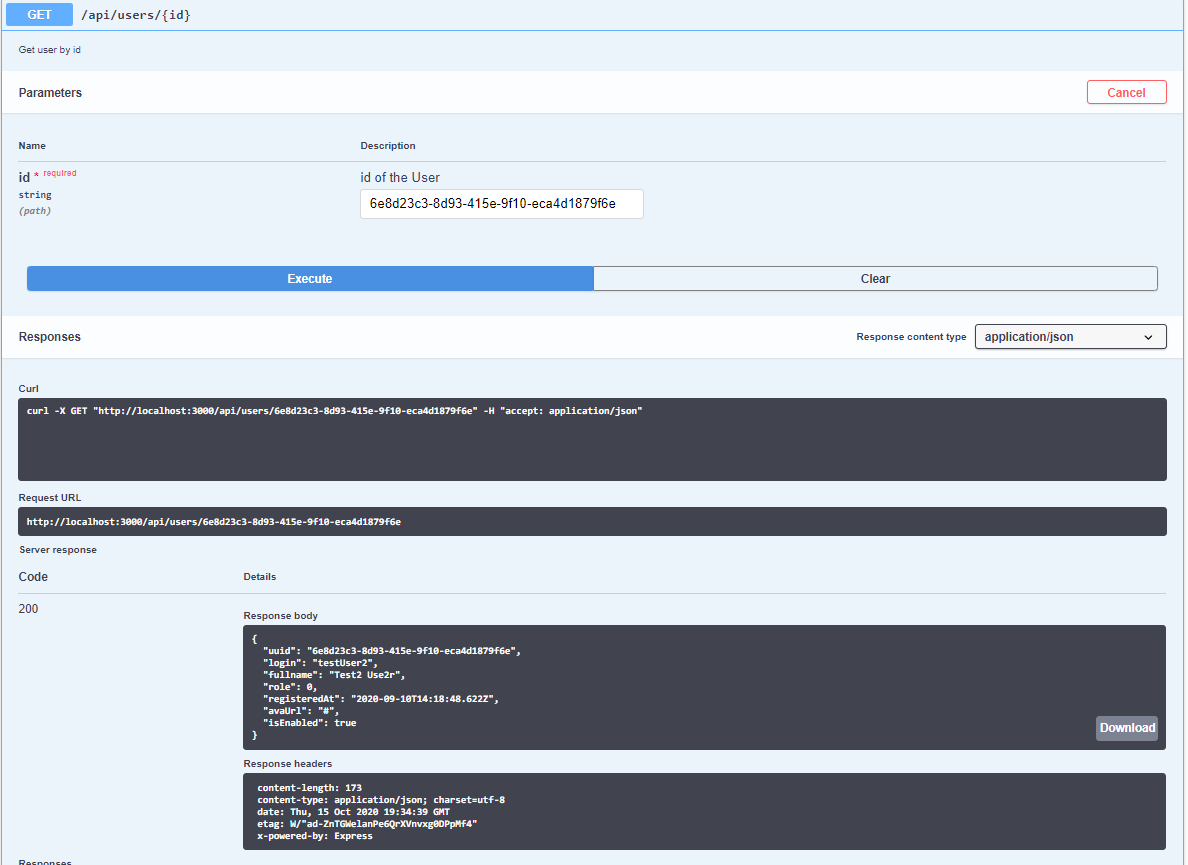












**Висновок**

Виконавши цю лабораторну роботу, ми навчились створювати простий JSON HTTP API веб-сервер, який надає доступ до ресурсів зі сховища.