

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 №**1

# з дисципліни “**Основи web-програмування**”

# тема “**Основи JavaScript**”

|  |  |  |
| --- | --- | --- |
| Виконав  студент IІ курсу  групи КП-91  Ігор Цой Леонідович  (*прізвище, ім’я, по батькові*)  варіант №23 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2020

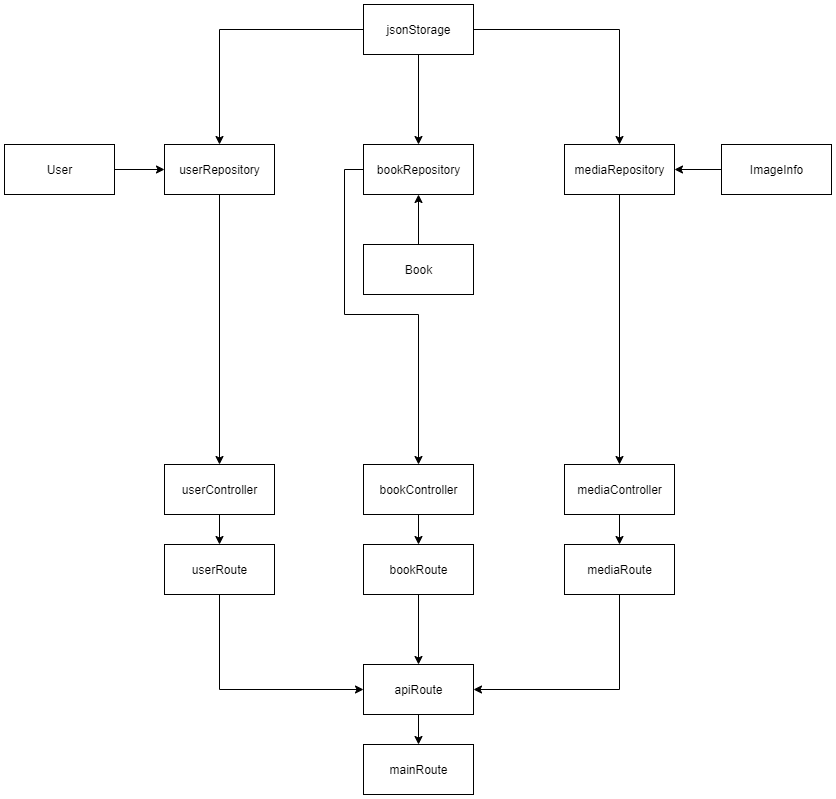
**Мета роботи**

Навчитись створювати простий JSON HTTP API веб-сервер, що надає доступ до ресурсів зі сховища.

**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.

**Діаграма залежності модулів**



**Тексти коду програм**

|  |
| --- |
| image.js |
| /\*\*   \* @typedef ImageInfo   \* @property {integer} id   \* @property {string} file\_name - file name   \*/  class ImageInfo  {      constructor (id, file\_name, file\_path = null)      {          this.id = file\_name;          this.file\_name = file\_name;          this.file\_path = file\_path;      }  }  module.exports = ImageInfo; |

|  |
| --- |
| mediaRepository.js |
| const ImageInfo = require('./../models/image')  const JsonStorage = require('../jsonStorage');  const fs = require('fs');  class MediaRepository  {      constructor(file\_path)      {          this.storage = new JsonStorage(file\_path);      }      getImageById(image\_id)      {          try {              const items = this.storage.readItems();              for (const item of items['items']) {                  if (item['id'] === image\_id) {                      return new ImageInfo(                          image\_id,                          item['file\_name'],                          item['file\_path']                      )                  }              }              return null;          } catch (err) {              throw err;          }      }      addImage(image\_model)      {          try {              let items = this.storage.readItems();              image\_model.id = this.storage.nextId;              this.storage.incrementNextId();              items['items'].push({                  id: image\_model.id,                  file\_name: image\_model.file\_name,                  file\_path: image\_model.file\_path              });              this.storage.writeItems(items);              return image\_model;          } catch (err) {              throw err;          }      }  }  module.exports = MediaRepository; |

|  |
| --- |
| userController.js |
| const UserRepository = require('./../repositories/userRepository');  const User = require('./../models/user');  const userRepository = new UserRepository('./data/users');  const HttpError = require('./../httpError');  const page\_size = 8;  module.exports =  {      getUsers(input, output)      {          try {              const page\_str = input.query.page;              let page;              if (page\_str === undefined) page = 1;              else              {                  page = Number(page\_str);                  if (isNaN(page)) throw new HttpError(400, 'page is not a number');                  if (!Number.isInteger(page)) throw new HttpError(400, 'page is not an integer');                  if (page < 1) throw new HttpError(400, 'invalid page value (page < 1)');              }              const users = userRepository.getUsers();              const size =  users.length;              const offset = page\_size \* (page - 1);              if (offset === 0 && size === 0)              {                  output.status(200).json([]);                  return;              }              if (offset >= size) throw new HttpError(400, 'offset is bigger than users number (page size is 8)');              const users\_page = users.slice(offset, offset + page\_size);              const arr = [];              for (const user of users\_page)              {                  arr.push({                      id: user.id,                      login: user.login,                      fullname: user.fullname,                      role: user.role,                      registered\_at: user.registered\_at,                      ava\_url: user.ava\_url,                      is\_enabled: user.is\_enabled                  });              }              output.status(200).json(arr);          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      getUser(input, output)      {          try {              const id\_str = input.params.id;              const id = Number(id\_str);              if (isNaN(id)) throw new HttpError(400, 'id is not a number');              if (!Number.isInteger(id)) throw new HttpError(400, 'id is not an integer');              if (id < 1) throw new HttpError(400, 'invalid id value (id < 1)');                const user = userRepository.getUserById(id);              if (user === null) throw new HttpError(404, 'user not found');              else              {                  const obj = {                      id: user.id,                      login: user.login,                      fullname: user.fullname,                      role: user.role,                      registered\_at: user.registered\_at,                      ava\_url: user.ava\_url,                      is\_enabled: user.is\_enabled                  };                  output.status(200).json(obj);              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      }  } |

|  |
| --- |
| bookController.js |
| const BookRepository = require('./../repositories/bookRepository');  const Book = require('./../models/book');  const bookRepository = new BookRepository('./data/books');  const HttpError = require('./../httpError');  const moment = require('moment');  const page\_size = 8;  function bookToObject(book)  {      return {          id: book.id,          title: book.title,          original\_language: book.original\_language,          publishment\_year: book.publishment\_year,          pages: book.pages,          added: book.added      };  }  function bookParser(obj, id\_check = false)  {      try      {          let id;          if (id\_check)          {              id = obj['id'];              if (!Number.isInteger(id)) { throw new HttpError(400, 'id is not an integer'); }              else if (id < 1) { throw new HttpError(400, 'invalid id value (id < 1)'); }          }          else { id = 0; }          if (typeof obj['title'] != 'string' || typeof obj['original\_language'] != 'string' ||              typeof obj['added'] != 'string' || !Number.isInteger(obj['publishment\_year']) ||              !Number.isInteger(obj['pages'])) throw new HttpError(400, 'invalid field types');          else if (!moment(obj['added'], moment.ISO\_8601, true).isValid()) throw new HttpError(400, 'invalid date format');          else          {              return new Book(                  id,                  obj['title'],                  obj['original\_language'],                  obj['publishment\_year'],                  obj['pages'],                  obj['added']              );          }      }      catch (err)      {          if (err instanceof HttpError) throw err;          else throw new HttpError(400, err.message);      }  }  module.exports =  {      getBooks(input, output)      {          try {              const page\_str = input.query.page;              let page;              if (page\_str === undefined) page = 1;              else              {                  page = Number(page\_str);                  if (isNaN(page)) throw new HttpError(400, 'page is not a number');                  if (!Number.isInteger(page)) throw new HttpError(400, 'page is not an integer');                  if (page < 1) throw new HttpError(400, 'invalid page value (page < 1)');              }              const books = bookRepository.getBooks();              const size =  books.length;              const offset = page\_size \* (page - 1);              if (offset === 0 && size === 0)              {                  output.status(200).json([]);                  return;              }              if (offset >= size) throw new HttpError(400, 'offset is bigger than books number (page size is 8)');              const books\_page = books.slice(offset, offset + page\_size);              const arr = [];              for (const book of books\_page)              {                  arr.push(bookToObject(book));              }              output.status(200).json(arr);          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      getBook(input, output)      {          try {              const id\_str = input.params.id;              const id = Number(id\_str);              if (isNaN(id)) throw new HttpError(400, 'id is not a number');              if (!Number.isInteger(id)) throw new HttpError(400, 'id is not an integer');              if (id < 1) throw new HttpError(400, 'invalid id value (id < 1)');                const book = bookRepository.getBookById(id);              if (book === null) throw new HttpError(404, 'book not found');              else              {                  const obj = bookToObject(book);                  output.status(200).json(obj);              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      addBook(input, output)      {          try {              const book\_model = bookParser(input.body);              const book = bookRepository.addBook(book\_model);              const obj = bookToObject(book);              output.status(201).json(obj);          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      updateBook(input, output)      {          try {              const book\_model = bookParser(input.body, true);              const book = bookRepository.updateBook(book\_model);              if (book === null) throw new HttpError(404, 'book not found' );              else              {                  const obj = bookToObject(book);                  output.status(200).json(obj);              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      deleteBook(input, output)      {          try          {              const id\_str = input.params.id;              const id = Number(id\_str);              if (isNaN(id)) throw new HttpError(400, 'id is not a number');              if (!Number.isInteger(id)) throw new HttpError(400, 'id is not an integer' );              if (id < 1) throw new HttpError(400, 'invalid id value (id < 1)');              const book = bookRepository.deleteBook(id);              if (book === null) throw new HttpError(404, 'book not found');              else              {                  const obj = bookToObject(book);                  output.status(200).json(obj);              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      }  } |

|  |
| --- |
| mediaController.js |
| const MediaRepository = require('./../repositories/mediaRepository');  const ImageInfo = require('./../models/image');  const mediaRepository = new MediaRepository('./data/images');  const HttpError = require('./../httpError');  module.exports =  {      addImage(input, output)      {          try {              const img = mediaRepository.addImage(new ImageInfo(0, input.file.originalname, input.file.path));              output.status(201).json({id: img.id, file\_name: img.file\_name});          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      getImage(input, output)      {          try {              const id\_str = input.params.id;              const id = Number(id\_str);              if (isNaN(id)) throw new HttpError(400, 'id is not a number');              if (!Number.isInteger(id)) throw new HttpError(400, 'id is not an integer');              if (id < 1) throw new HttpError(400, 'invalid id value (id < 1)');                const image = mediaRepository.getImageById(id);              if (image === null) throw new HttpError(404, 'image not found');              else output.status(200).download(image.file\_path, image.file\_name);          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      }  } |

|  |
| --- |
| userRoute.js |
| const userController = require('./../controllers/userController');  const userRouter = require('express').Router();  const HttpError = require('./../httpError');  userRouter      /\*\*       \* @route GET /api/users/{id}       \* @group Users - user operations       \* @param {integer} id.path.required - id of the User       \* @returns {User.model} 200 - User object       \* @returns {Error} 400 - Bad request       \* @returns {Error} 404 - User not found      \*/      .get('/', userController.getUsers)      /\*\*       \* @route GET /api/users       \* @group Users - user operations       \* @param {integer} page.query - page of the users list (page size is 8, default = 1)       \* @returns {Array.<User>} 200 - Users page       \* @returns {Error} 400 - Bad request       \*/      .get('/:id', userController.getUser);  userRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = userRouter; |

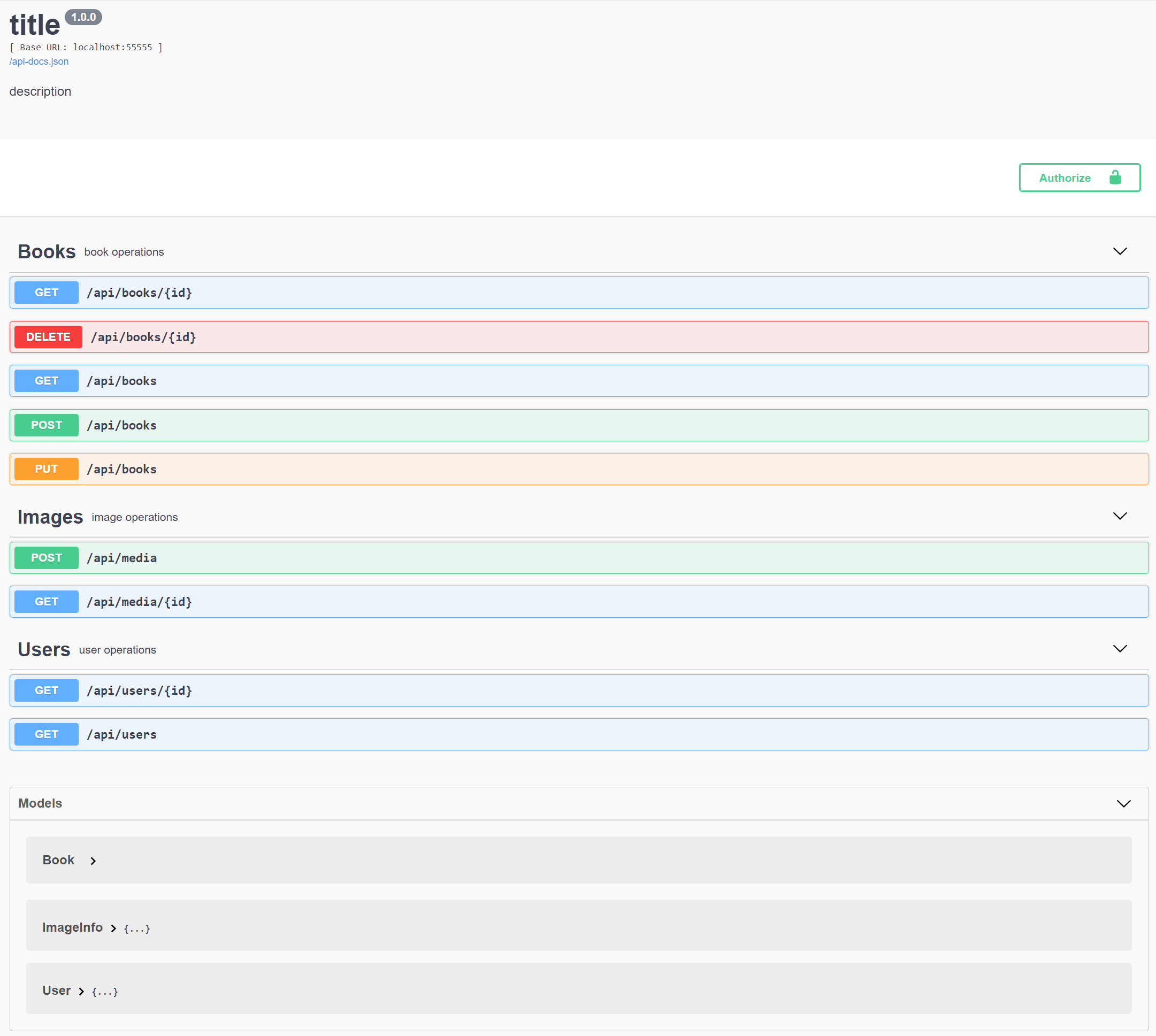
|  |
| --- |
| bookRoute.js |
| const bookController = require('./../controllers/bookController')  const bookRouter = require('express').Router();  const HttpError = require('./../httpError');  bookRouter      /\*\*       \* @route GET /api/books/{id}       \* @group Books - book operations       \* @param {integer} id.path.required - id of the Book       \* @returns {Book.model} 200 - Book object       \* @returns {Error} 400 - Bad request       \* @returns {Error} 404 - Book not found      \*/      .get('/', bookController.getBooks)      /\*\*       \* @route GET /api/books       \* @group Books - book operations       \* @param {integer} page.query - page of the books list (page size is 8, default = 1)       \* @returns {Array.<Book>} 200 - Books page       \* @returns {Error} 400 - Bad request      \*/      .get('/:id', bookController.getBook)      /\*\*       \* @route POST /api/books       \* @group Books - book operations       \* @param {Book.model} book\_model.body.required - book model to add       \* @returns {Book.model} 201 - Book created       \* @returns {Error} 400 - Bad request      \*/      .post('/', bookController.addBook)      /\*\*       \* @route PUT /api/books       \* @group Books - book operations       \* @param {Book.model} book\_model.body.required - book model to update       \* @returns {Book.model} 200 - Book updated       \* @returns {Error} 400 - Bad request       \* @returns {Error} 404 - Book not found      \*/      .put('/', bookController.updateBook)      /\*\*       \* @route DELETE /api/books/{id}       \* @group Books - book operations       \* @param {integer} id.path.required - id of the Book       \* @returns {Book.model} 200 - Book object       \* @returns {Error} 400 - Bad request       \* @returns {Error} 404 - Book not found       \*/      .delete('/:id',bookController.deleteBook);  bookRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = bookRouter; |

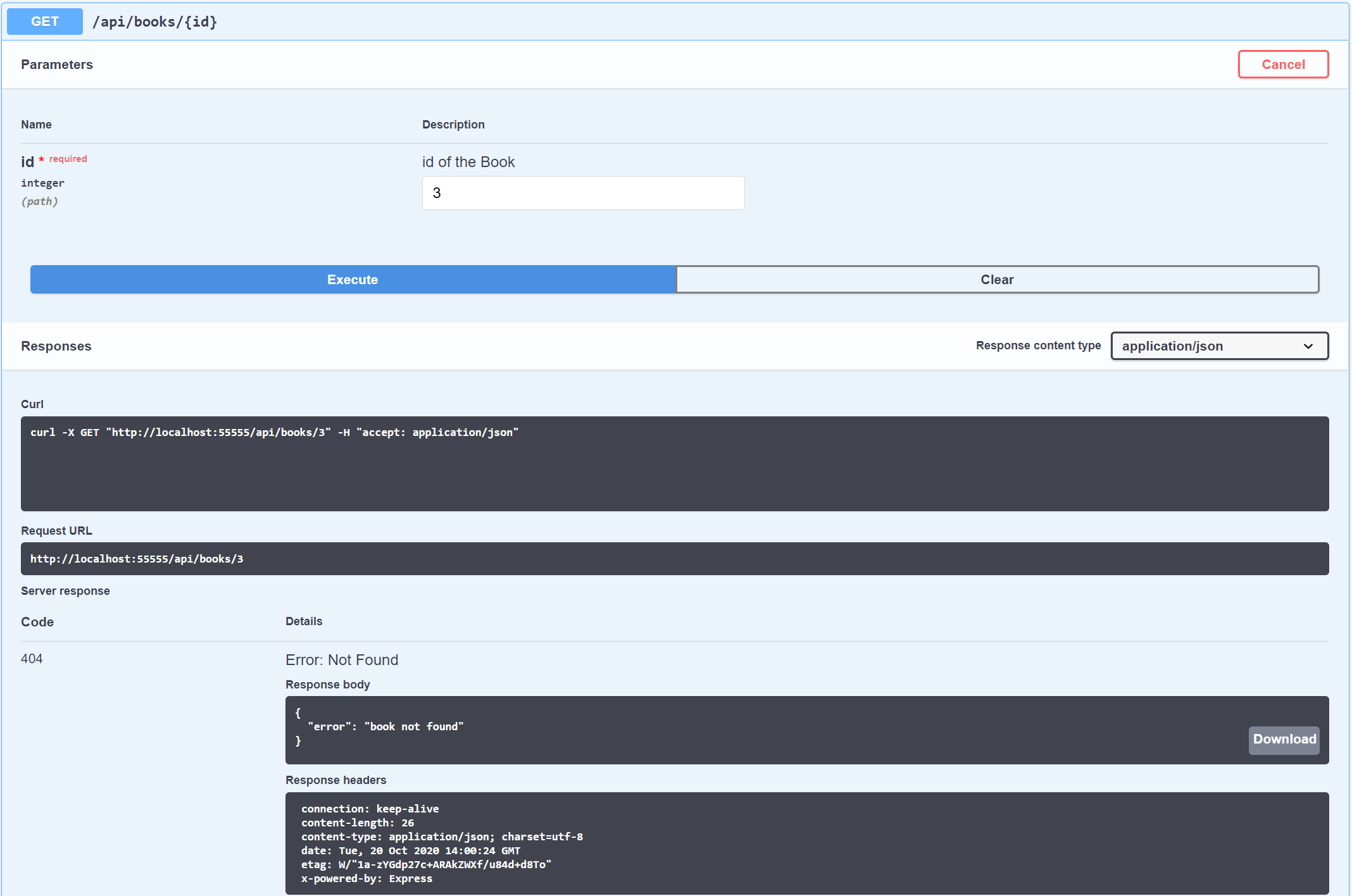
|  |
| --- |
| mediaRoute.js |
| const mediaController = require('../controllers/mediaController');  const mediaRouter = require('express').Router();  const HttpError = require('./../httpError');  mediaRouter      /\*\*       \* @route POST /api/media       \* @group Images - image operations       \* @param {file} image\_file.formData.required - image model to add       \* @returns {ImageInfo.model} 201 - Image created       \* @returns {Error} 400 - Bad request       \*/      .post('/', mediaController.addImage)      /\*\*       \* @route GET /api/media/{id}       \* @group Images - image operations       \* @param {integer} id.path.required - id of the Image       \* @returns {file} 200 - Image file bytes       \* @returns {Error} 400 - Bad request       \* @returns {Error} 404 - Image not found      \*/      .get('/:id', mediaController.getImage);  mediaRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = mediaRouter; |

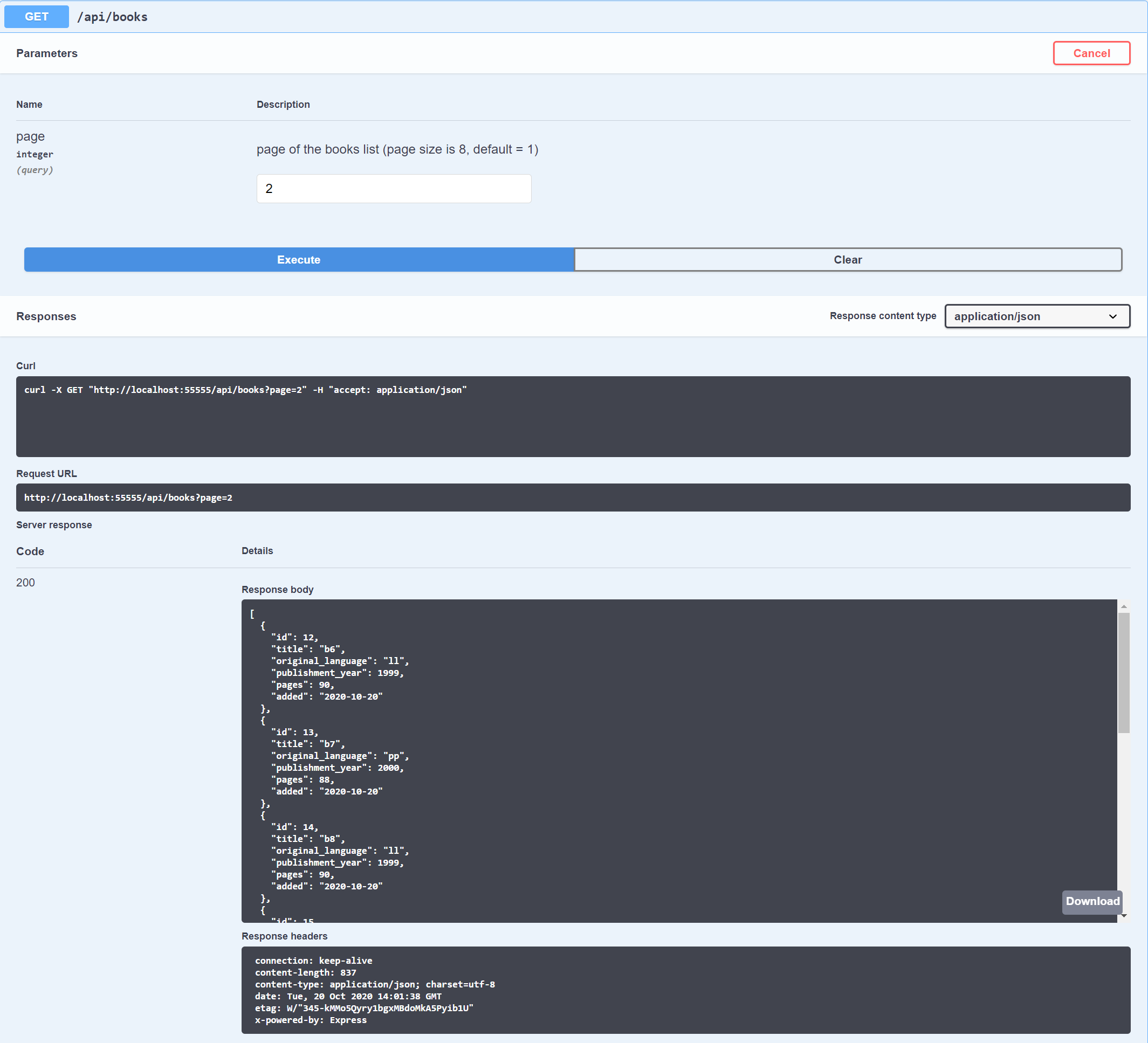
|  |
| --- |
| apiRoute.js |
| const userRouter = require('./userRoute');  const bookRouter = require('./bookRoute');  const mediaRouter = require('./mediaRoute');  const apiRouter = require('express').Router();  const HttpError = require('./../httpError');  apiRouter      .use('/users', userRouter)      .use('/books', bookRouter)      .use('/media', mediaRouter);  apiRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = apiRouter; |

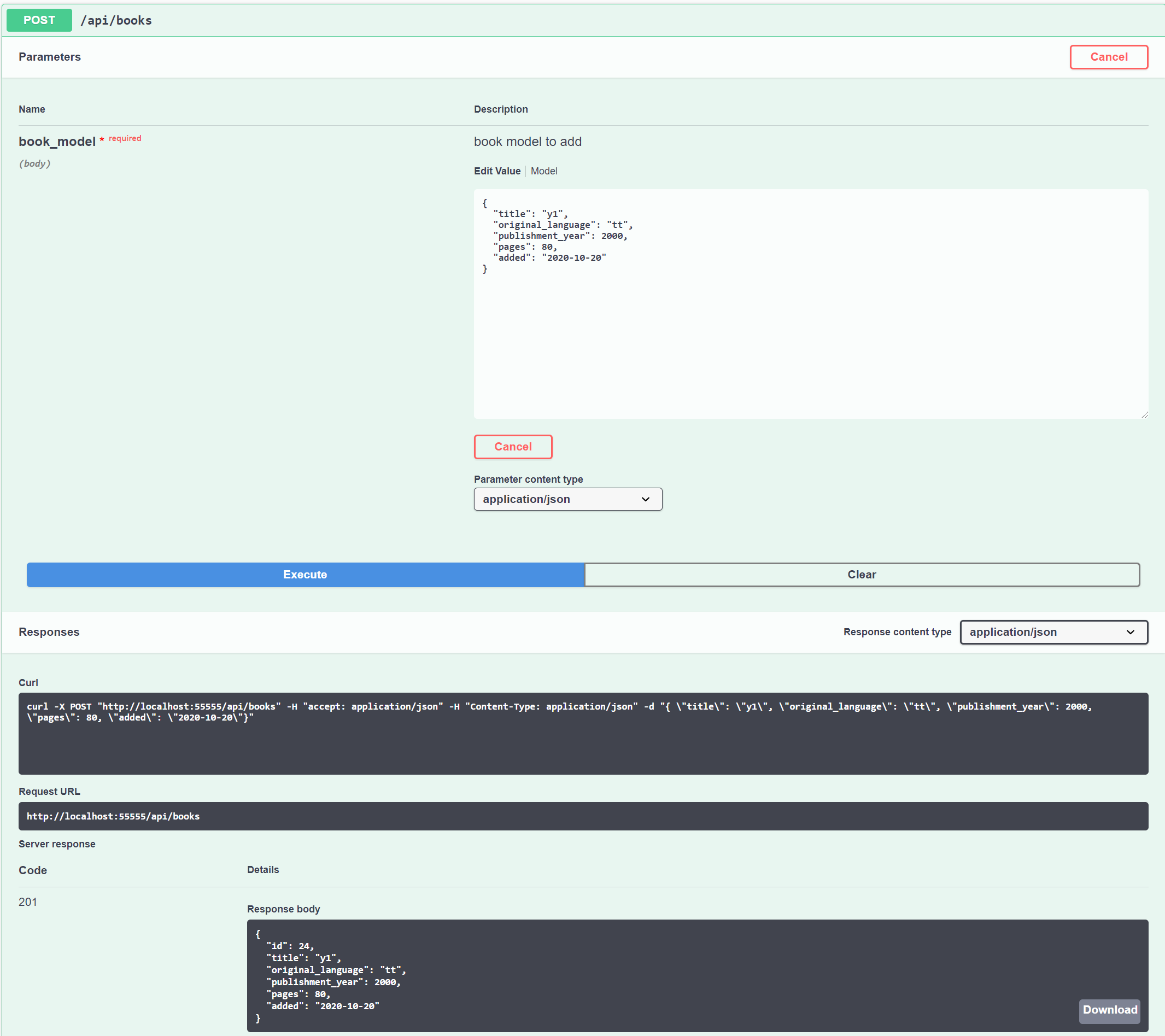
|  |
| --- |
| mainRoute.js |
| const express = require('express');  const HttpError = require('./../httpError');  const body\_parser = require('body-parser')  const multer  = require('multer');  const upload = multer({ dest: 'data/images/' });  const mainRouter = express();  const apiRouter = require('./apiRoute');  const expressSwaggerGenerator = require('express-swagger-generator');  const expressSwagger = expressSwaggerGenerator(mainRouter);    const config = {      swaggerDefinition: {          info: {              description: 'description',              title: 'title',              version: '1.0.0',          },          host: 'localhost:55555',          produces: [ "application/json", "image/png", "image/jpeg" ],      },      basedir: \_\_dirname,      files: ['./../routes/\*\*/\*.js', './../models/\*\*/\*.js'],  };  expressSwagger(config);  mainRouter.post('/api/books/', body\_parser.json(), (err, req, res, next) => {      if (err) throw new HttpError(400, err.message);      next();  });  mainRouter.put('/api/books/', body\_parser.json(), (err, req, res, next) => {      if (err) throw new HttpError(400, err.message);      next();  });  mainRouter.post('/api/media/', upload.single('image\_file'), (res, req, next) => { next(); });  mainRouter.use('/api', apiRouter);  mainRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  mainRouter.use((err, req, res, next) => {      console.log('error caught\n{')      console.log('    status code: ' + err.status\_code + ', message: ' + err.message + '\n}');      res.status(err.status\_code).json({ error: err.message });  });  module.exports = mainRouter; |

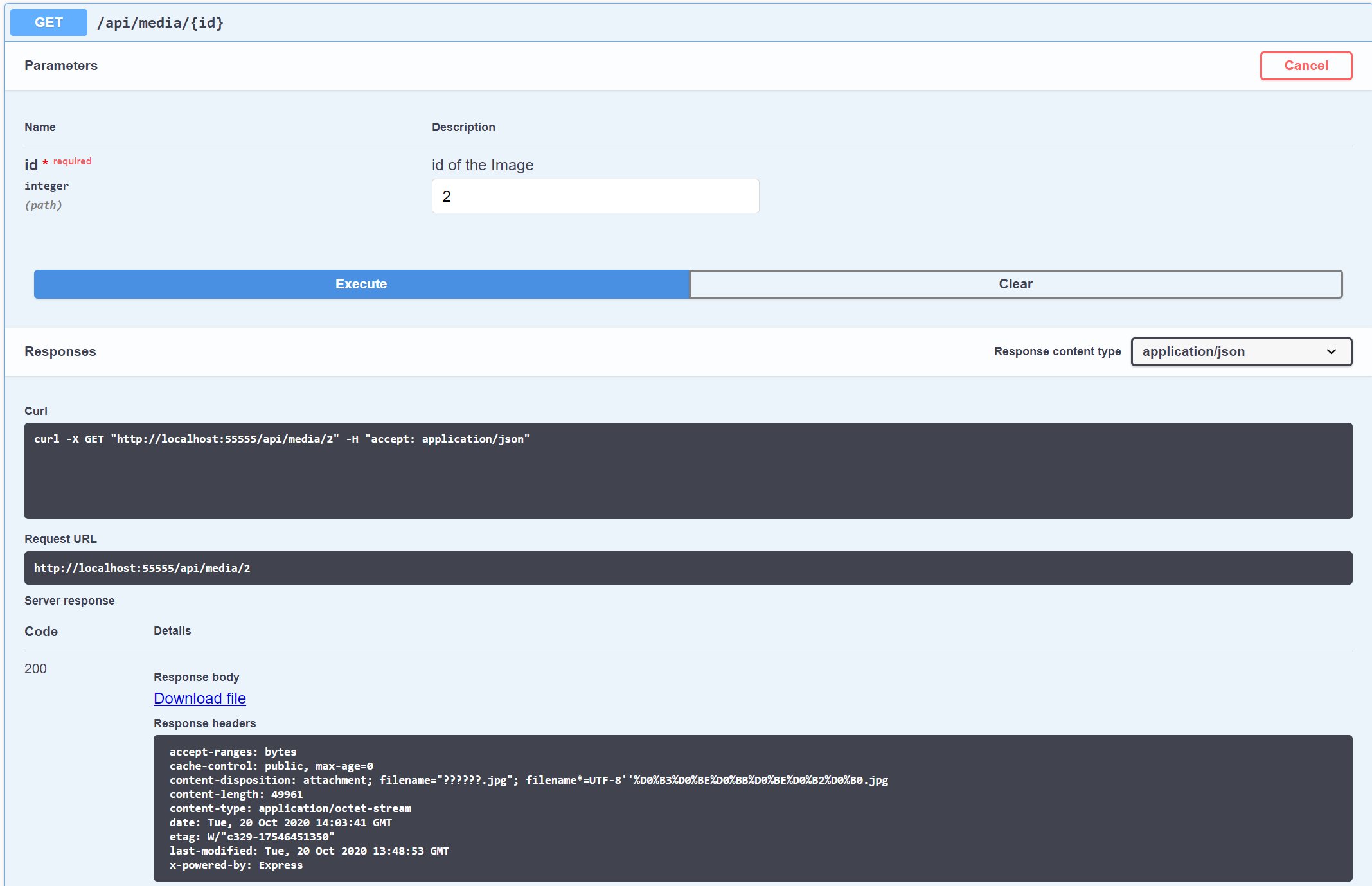
**Приклад роботи програми**











**Висновки**

У цій лабораторній роботі було здобуто навичок створення JSON HTTP API веб сервера, що надає ресурсів до сховища.

Було ознайомлено з бібліотекою для створення веб серверів express. Також були використані бібліотеки multer та body-parser.

Був створений власний клас помилок, наслідуваний від Error, що має у собі поле зі статус кодом відповіді.