

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 №**3

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

# тема “**Багатосторінковий веб-сайт**”

|  |  |  |
| --- | --- | --- |
| 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 веб-сайт, що надає CRUD доступ до ресурсів веб-сервера.  
Ознайомитись із шаблонізаторами веб-сторінок для генерації контенту.  
Навчитись створювати HTML-форми для взаємодії користувача із веб-сайтом та обробляти на сервері внесені користувачем дані.

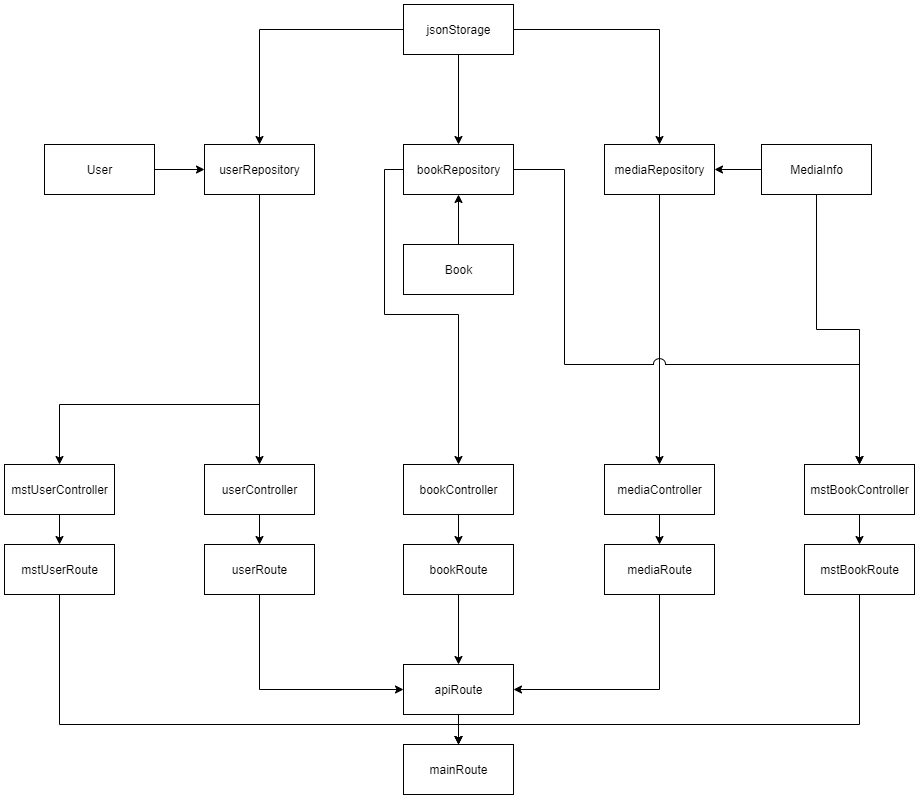
**Постановка завдання**

Реалізувати багатосторінковий веб-сайт із заданою структурою розділів:



Забезпечити можливість пошуку, пагінації, додавання та видалення сутностей за варіантом.

**Діаграма залежності модулів**



**Тексти коду програм**

|  |
| --- |
| mstUserController.js |
| const UserRepository = require('./../repositories/userRepository');  const User = require('./../models/user');  const userRepository = new UserRepository('./data/users');  const HttpError = require('./../httpError');  const { off } = require('../routes/mainRoute');  const page\_size = 4;  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 max\_page = Math.ceil(size/page\_size);              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,                      registered\_at: user.registered\_at                  });              }              let prev\_page = '<span>&lt;</span>';              let next\_page = '<span>&gt;</span>';              if (page != 1) prev\_page = '<a href=\"/users?page=' + (page-1) + '\">&lt;</a>'              if (offset + page\_size < size) next\_page = '<a href=\"/users?page=' + (page+1) + '\">&gt;</a>'              let pages = []              if (page > 5)              {                  pages.push('<a href=\"/users?page=1\">1</a>');                  if (page != 6) pages.push('<span>...</span>');              }              for (i = Math.max(page-4, 1); i < page; i++)              {                  pages.push('<a href=\"/users?page=' + i + '\">' + i + '</a>');              }              pages.push('<span>' + page + '</span>');              for (i = page+1; i <= Math.min(page+4, max\_page); i++)              {                  pages.push('<a href=\"/users?page=' + i + '\">' + i + '</a>');              }              if (page < max\_page - 4)              {                  if (page != max\_page - 5) pages.push('<span>...</span>');                  pages.push('<a href=\"/users?page=' + max\_page + '\">' + max\_page + '</a>');              }              params = {head\_title: 'Users', users\_page: arr, users\_current: 'current', next\_page: next\_page, prev\_page: prev\_page, pages: pages}              output.status(200).render('users', params);          }          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              {                  let role = 'user';                  if (user.role === 1) role = 'admin';                  let ava\_img = '<img src=\"' + user.ava\_url + '" alt=\"ava\" class=\"user\_img\">'                  if (user.ava\_url === null) ava\_img = '<div class=\"user\_img\"><span>No image</span></div>'                  let bio = 'No biography'                  if (user.biography != null) bio = user.biography                  const obj = {                      id: user.id,                      login: user.login,                      fullname: user.fullname,                      role: role,                      registered\_at: user.registered\_at,                      ava\_url: ava\_img,                      is\_enabled: user.is\_enabled,                      biography: bio                  };                  output.status(200).render('user', {head\_title: 'User', users\_current: 'current', user: obj});              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      }  } |

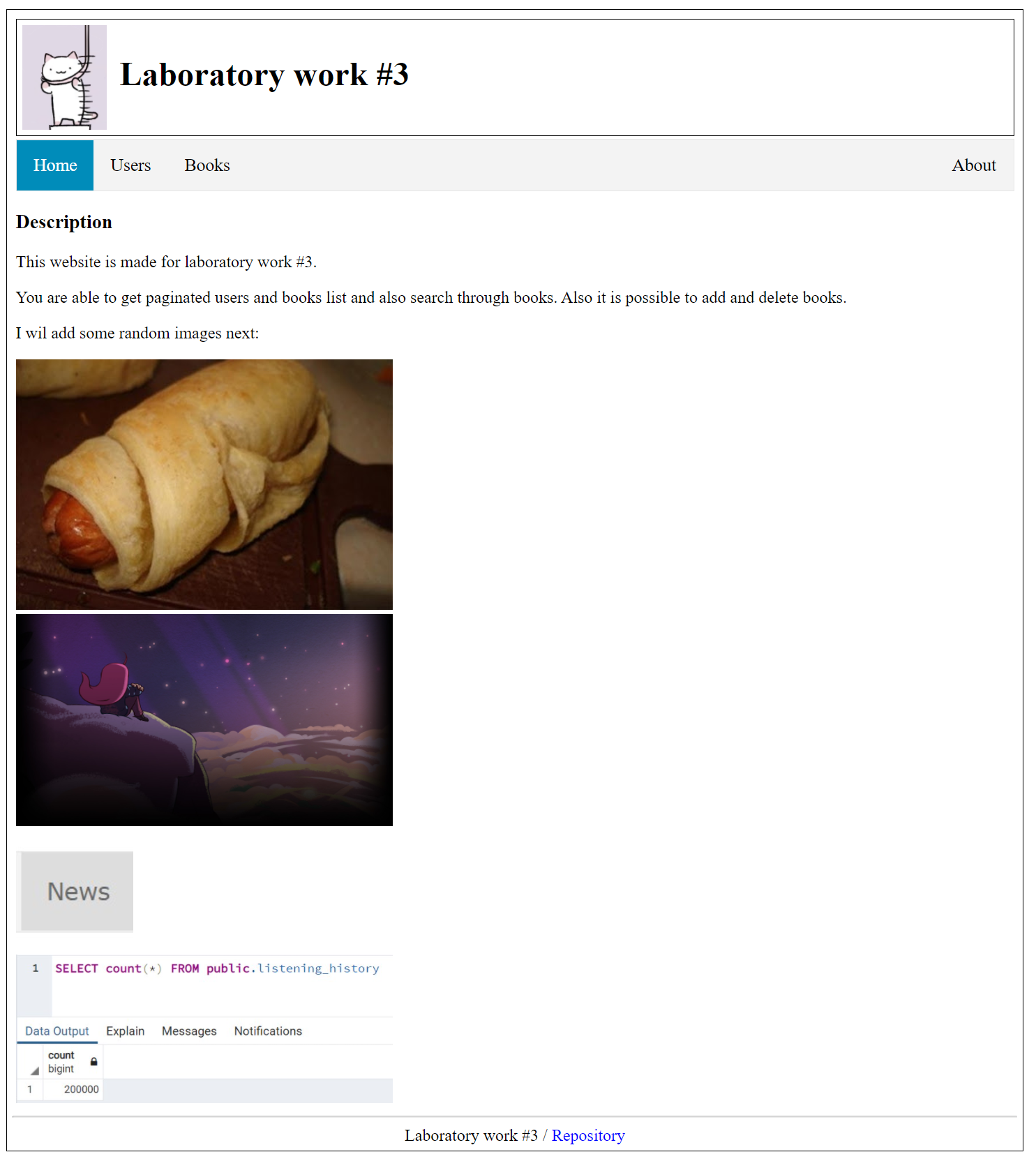
|  |
| --- |
| mstBookController.js |
| const BookRepository = require('./../repositories/bookRepository');  const MediaRepository = require('./../repositories/mediaRepository');  const Book = require('./../models/book');  const MediaInfo = require('./../models/media');  const bookRepository = new BookRepository('./data/books');  const mediaRepository = new MediaRepository('./data/media');  const HttpError = require('./../httpError');  const moment = require('moment');  const page\_size = 4;  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,          file\_url: book.file\_url      };  }  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 (!moment(obj['added'], moment.ISO\_8601, true).isValid()) throw new HttpError(400, 'invalid date format');          else          {              year = parseInt(obj.publishment\_year)              pages = parseInt(obj.pages)              if (isNaN(year) || isNaN(pages)) throw new HttpError(400, 'Invalid data format');              return new Book(                  id,                  obj['title'],                  obj['original\_language'],                  year,                  pages,                  obj['added'],                  obj['file\_url']              );          }      }      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;              const title\_search = input.query.title;              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)');              }              let books = bookRepository.getBooks();              if (!(title\_search === undefined))              {                  books = books.filter(item => item.title.includes(title\_search));              }                const size =  books.length;              const max\_page = Math.ceil(size/page\_size);              const offset = page\_size \* (page - 1);              if (offset === 0 && size === 0)              {                  output.status(200).render('books', {head\_title: 'Books', books\_page: null, books\_current: 'current',                      next\_page: '<span>&gt;</span>', prev\_page: '<span>&lt;</span>', page: null, title\_value: title\_search,                      null\_result: '<tr><td colspan=3 style=\"text-align: center;\">Nobody here but us chickens!</td></tr>'});                  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));              }              let prev\_page = '<span>&lt;</span>';              let next\_page = '<span>&gt;</span>';              let title\_query = '';              if (!(title\_search === undefined)) title\_query = '&title=' + title\_search;              if (page != 1) prev\_page = '<a href=\"/books?page=' + (page-1) + title\_query + '\">&lt;</a>'              if (offset + page\_size < size) next\_page = '<a href=\"/books?page=' + (page+1) + title\_query + '\">&gt;</a>'              let pages = []              if (page > 5)              {                  pages.push('<a href=\"/books?page=1' + title\_query + '\">1</a>');                  if (page != 6) pages.push('<span>...</span>');              }              for (i = Math.max(page-4, 1); i < page; i++)              {                  pages.push('<a href=\"/books?page=' + i + title\_query + '\">' + i + '</a>');              }              pages.push('<span>' + page + '</span>');              for (i = page+1; i <= Math.min(page+4, max\_page); i++)              {                  pages.push('<a href=\"/books?page=' + i + title\_query + '\">' + i + '</a>');              }              if (page < max\_page - 4)              {                  if (page != max\_page - 5) pages.push('<span>...</span>');                  pages.push('<a href=\"/books?page=' + max\_page + title\_query + '\">' + max\_page + '</a>');              }              params = {head\_title: 'Books', books\_page: arr, books\_current: 'current', next\_page: next\_page, prev\_page: prev\_page, pages: pages, title\_value: title\_search}              output.status(200).render('books', params);          }          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).render('book', {head\_title: 'Books', books\_current: 'current', book: obj});              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      },      addBook(input, output)      {          try {              const media = mediaRepository.addMedia(new MediaInfo(0, input.file.originalname, input.file.path));              let body = input.body;              body.file\_url = 'http://localhost:55555/api/media/' + media.id;              const book\_model = bookParser(body);              const book = bookRepository.addBook(book\_model);              const obj = bookToObject(book);              output.status(303).redirect('http://localhost:55555/books/' + obj.id)          }          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              {                  output.status(303).redirect('http://localhost:55555/books')              }          }          catch (err)          {              if (err instanceof HttpError) throw err;              else throw new HttpError(500, err.message);          }      }  } |

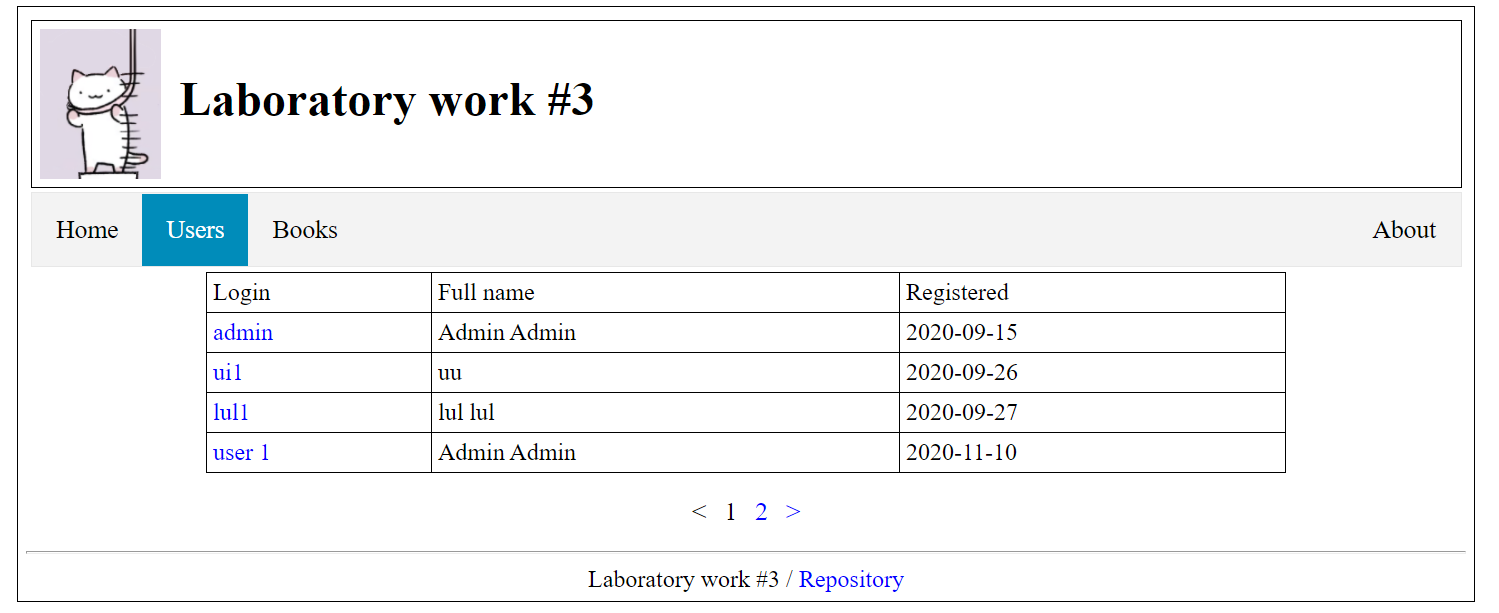
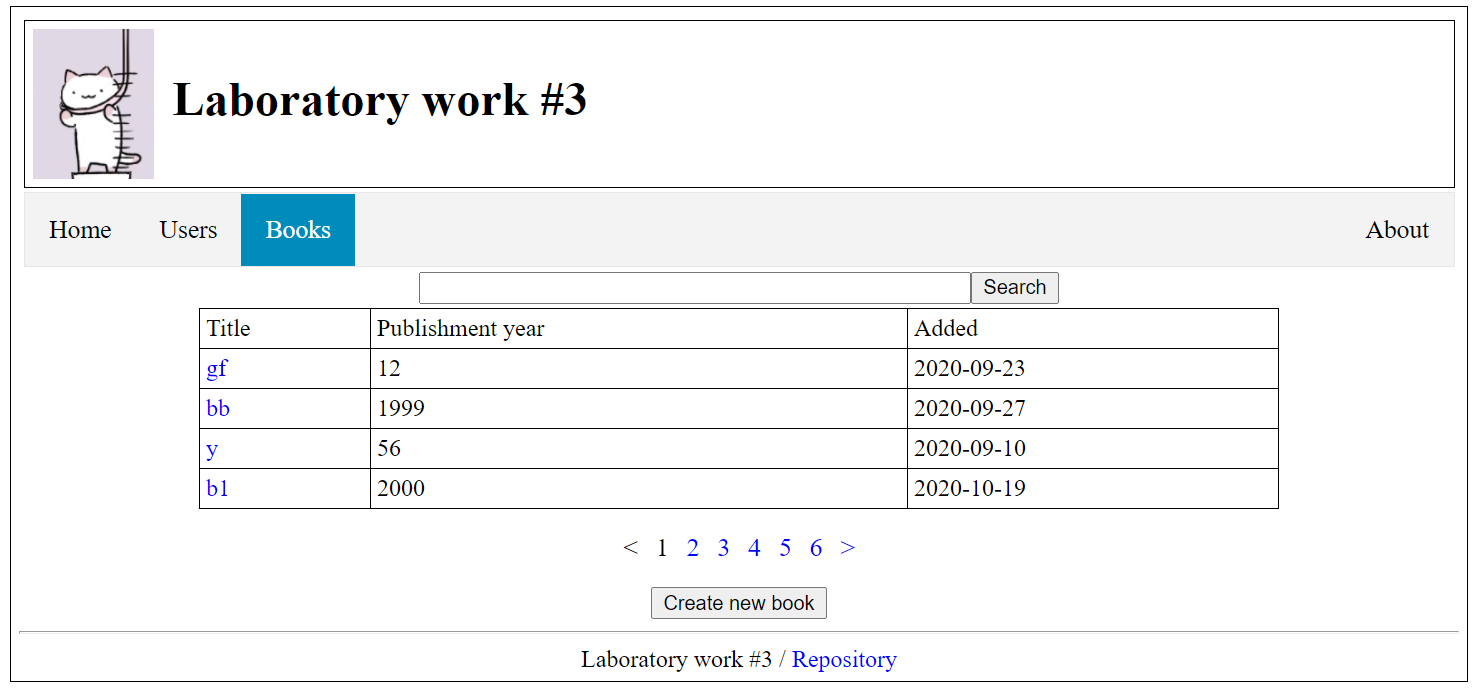
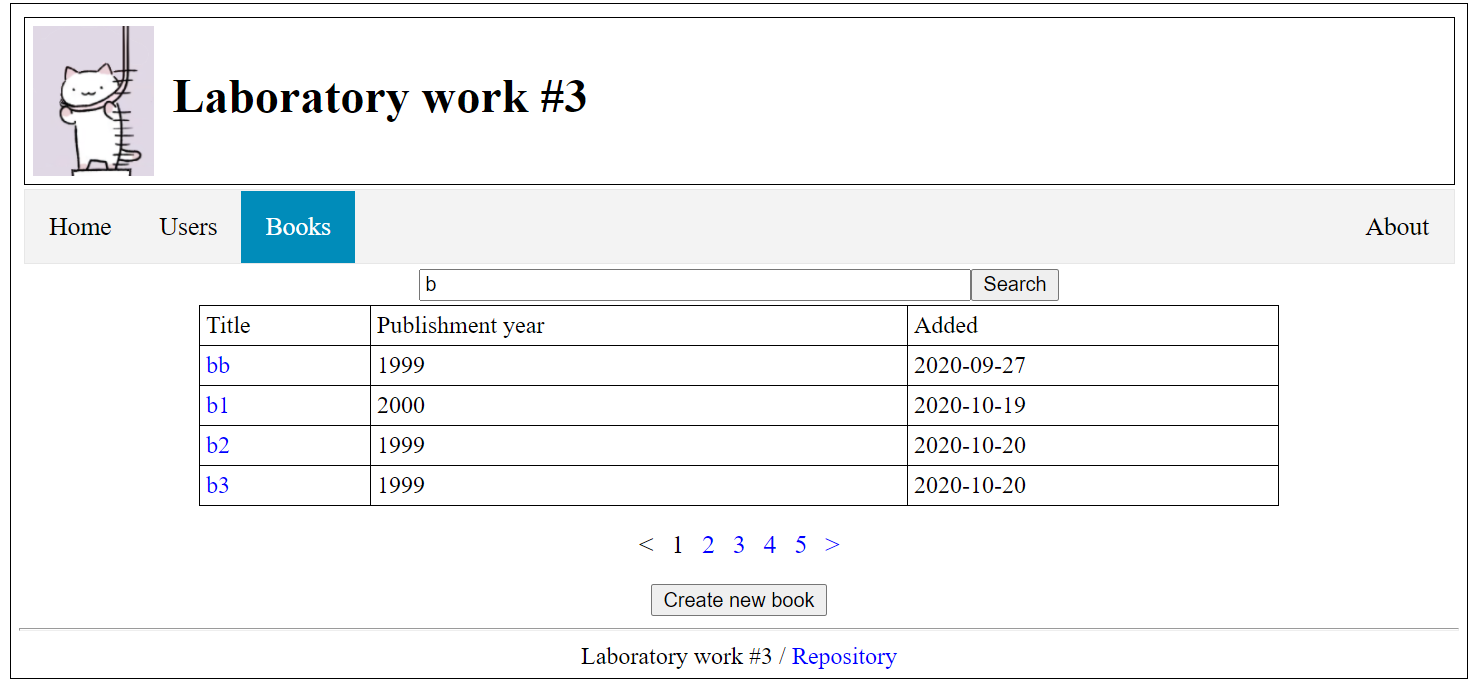
|  |
| --- |
| mstUserRoute.js |
| const mstUserController = require('./../controllers/mstUserController');  const mstUserRoute = require('express').Router();  const HttpError = require('./../httpError');  mstUserRoute      .get('/', mstUserController.getUsers)      .get('/:id', mstUserController.getUser);  mstUserRoute.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = mstUserRoute; |

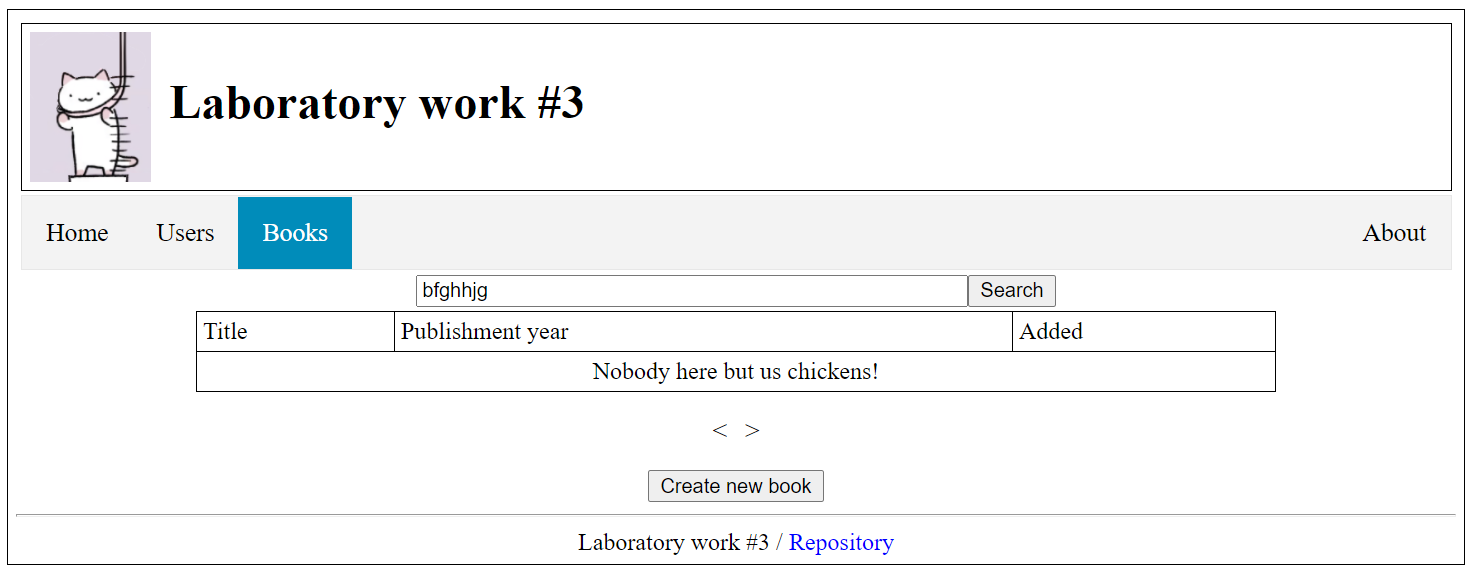
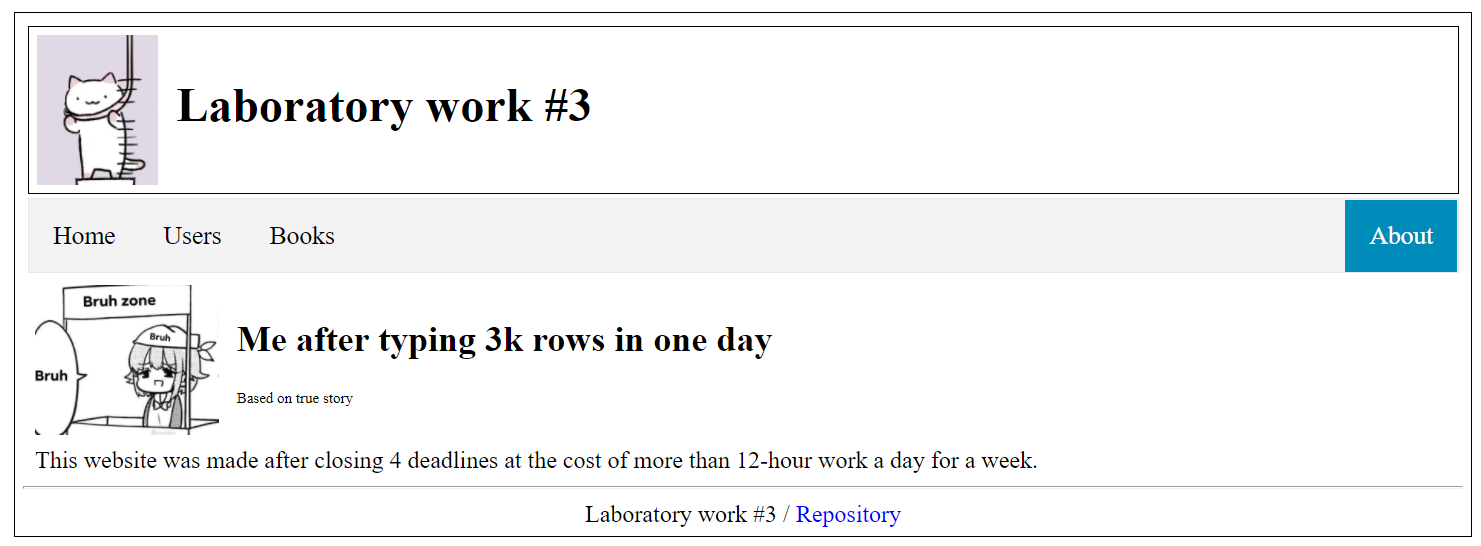
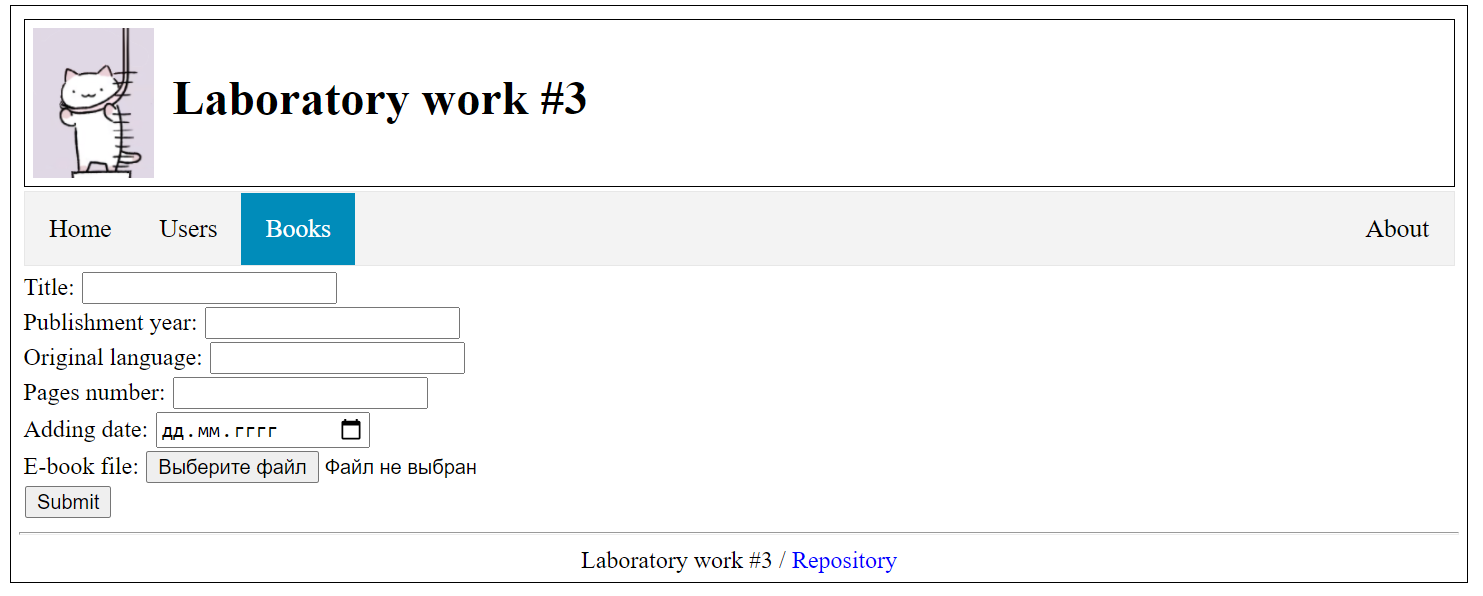
|  |
| --- |
| mstBookRoute.js |
| const mstBookController = require('./../controllers/mstBookController')  const mstBookRouter = require('express').Router();  const HttpError = require('./../httpError');  mstBookRouter      .get('/', mstBookController.getBooks)      .get('/new', (req, res) => { res.status(200).render('new\_book', {head\_title: 'New book', books\_current: 'current'}) })      .get('/:id', mstBookController.getBook)      .post('/', mstBookController.addBook)      .delete('/:id',mstBookController.deleteBook);  mstBookRouter.use((req, res) => {      throw new HttpError(400, 'command not found');  });  module.exports = mstBookRouter; |

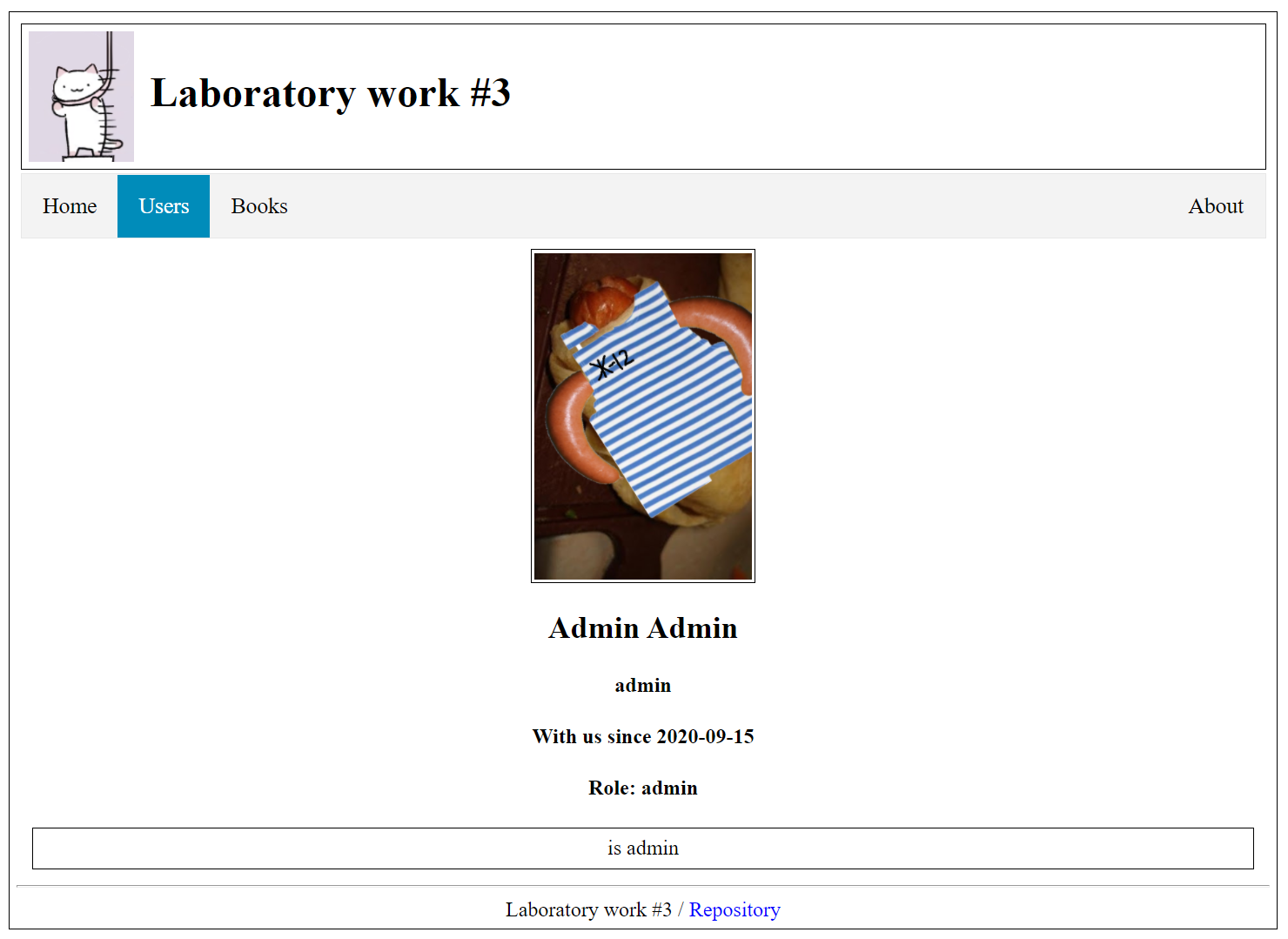
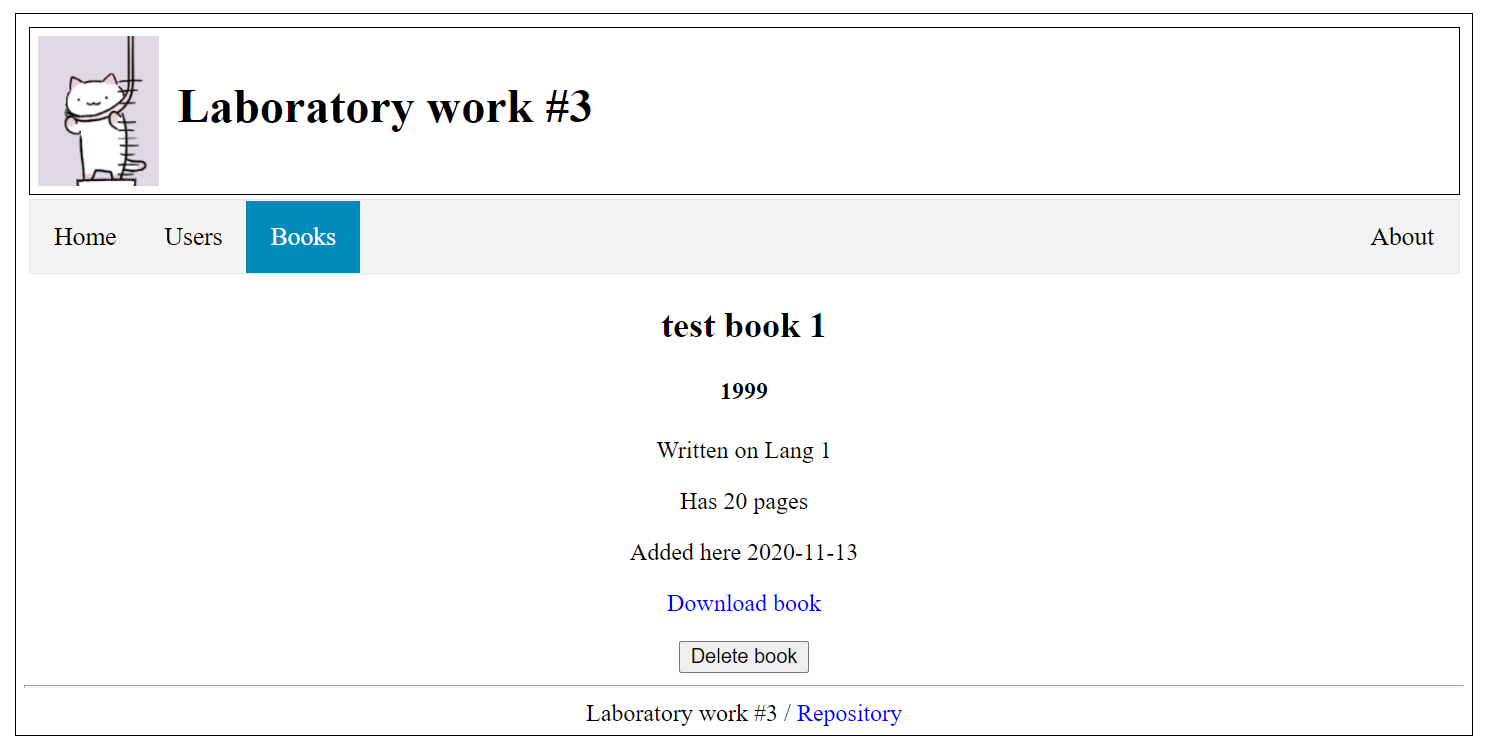
|  |
| --- |
| mainRoute.js |
| const express = require('express');  const path = require('path');  const mustache = require('mustache-express');  const methodOverride = require('method-override');  const HttpError = require('./../httpError');  const body\_parser = require('body-parser')  const multer  = require('multer');  const upload = multer({ dest: 'data/images/' });  const mainRouter = express();  const apiRouter = require('./apiRoute');  const mstUserRouter = require('./mstUserRoute');  const mstBookRouter = require('./mstBookRoute');  //#region swagger documentation  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);  //#endregion  // const consolidate = require('consolidate');  // mainRouter.engine('html', consolidate.swig);  // mainRouter.set('views', path.join(\_\_dirname, '../views'));  // mainRouter.set('view engine', 'html');  const viewsDir = path.join(\_\_dirname, '../views');  mainRouter.engine("mst", mustache(path.join(viewsDir, "partials")));  mainRouter.set('views', viewsDir);  mainRouter.set('view engine', 'mst');  //  mainRouter.use(methodOverride('\_method'));  //#region api  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('media\_file'), (req, res, next) => { next(); });  mainRouter.use('/api', apiRouter);  //#endregion  //#region web documents  mainRouter.use(express.static('public'));  mainRouter.post('/books', upload.single('book\_file'));  mainRouter.post('/books', body\_parser.urlencoded({ extended: true }));  mainRouter      .use('/users', mstUserRouter)      .use('/books', mstBookRouter)  mainRouter.use('/about/', (req, res) => {      res.render('about', {head\_title: 'About', about\_current: 'current'});  });  mainRouter.use('/', (req, res) => {      res.render('index', {head\_title: 'Home', home\_current: 'current'});  });  //#endregion  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).render('error', {head\_title: 'Error', error: err.status\_code + ' | ' + 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.