

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 “Багатосторінкий веб-сайт”

|  |  |  |
| --- | --- | --- |
| 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**

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.

**Вказівки до виконання**

#### **Частина 1. HTML та CSS**

1. Всі HTML файли розміщувати у директорії views.
2. Реалізувати основні розділи сайту:
   1. **Home**. Додати на головну сторінку (index.html) назву сайту і додаткову інформацію про сайт і його призначення (параграф опису).
   2. **Users**. Сторінки користувачів:
      1. Створити файл users.html зі списком користувачів.
      2. Створити файли для одного користувача (наприклад, user.html). Додати на сторінку інформацію про будь-якого користувача:
         * Логін.
         * Повне ім'я.
         * Дату реєстрації на сайті.
         * Параграф із біографією.
      3. Зв'язати сторінки так, щоби з index можна було перейти до users, а з users до сторінки з користувача (user.html).
   3. **{Entities}** (**назву замінити**). Сторінки сутностей за варіантом:
      1. Створити файл зі списом сутностей (наприклад, {entities}.html) кожен елемент списку - посилання на сутність.
      2. Створити файли для однієї сутності (наприклад, {entity}.html). У файлі розмістити детальну інформацію про будь-яку сутність.
      3. Зв'язати сторінки так, щоби з index можна було перейти до списку сутностей, а зі списку сутностей до конкретної сутності.
   4. **About**. Створити сторінку (about.html) із додатковим описом сайту і інформацією про автора сайту.
3. Додаткові зміни:
   1. Меню сайту. Створити список посилань на основні розділи сайту (Home, Users, {Entities} - **замінити на назву ваших сутностей**, About) і додати його до всіх створених сторінок.
   2. Зображення:
      1. Створити у public/ директорію images/ і додати у неї мінімум 3 зображення довільних форматів (наприклад, images/logo.png, images/user.png, images/{entity}.png). Також можна додати декілька зображень для сутностей.
      2. Додати зображення за допомогою тегів img на відповідні сторінки (index, user, {entity}).
   3. Стилізація сайту:
      1. Створити у public/ директорію stylesheets/, додати у неї файл з CSS-стилем (stylesheets/style.css) та підключити цей стиль до всіх створених веб-сторінок.
      2. Оформити стиль веб-сторінок [за вимогами](https://docs.google.com/presentation/d/1v486JQZ2ZaDPHeHXAljYgqXPXQBL5Hezc98mKUOEBYQ/edit?usp=sharing).
4. Відкрити файл views/index.html у будь-якому веб-браузері та перевірити показ веб-сторінок та навігації по статичному сайту.

#### **Частина 2. Шаблонізація веб-сторінок**

1. Використати [**ExpressJS**](https://expressjs.com/) веб-сервер із попереднього завдання.
   1. Налаштувати у сервері шлях для отримання статичних файлів із директорії public (див. Додатки). В результаті, якщо сервер запущений на порті 3000, у браузері має стати доступний файл стилю по шляху http://localhost:3000/stylesheets/style.css та всі інші файли з директорії public.
   2. Оновити у HTML-сторінках URL-шляхи до CSS файлів та статичних зображень.
   3. Реалізувати на сервері обробники запитів (роутери і контроллери) для отримання відповідних веб-документів із директорії views. Наприклад:
   4. GET / -> views/index.html
   5. GET /users -> views/users.html
   6. GET /users/:id -> views/user.html
   7. GET /{entities} -> views/{entities}.html
   8. GET /{entities}/:id -> views/{entity}.html
   9. GET /about -> views/about.html

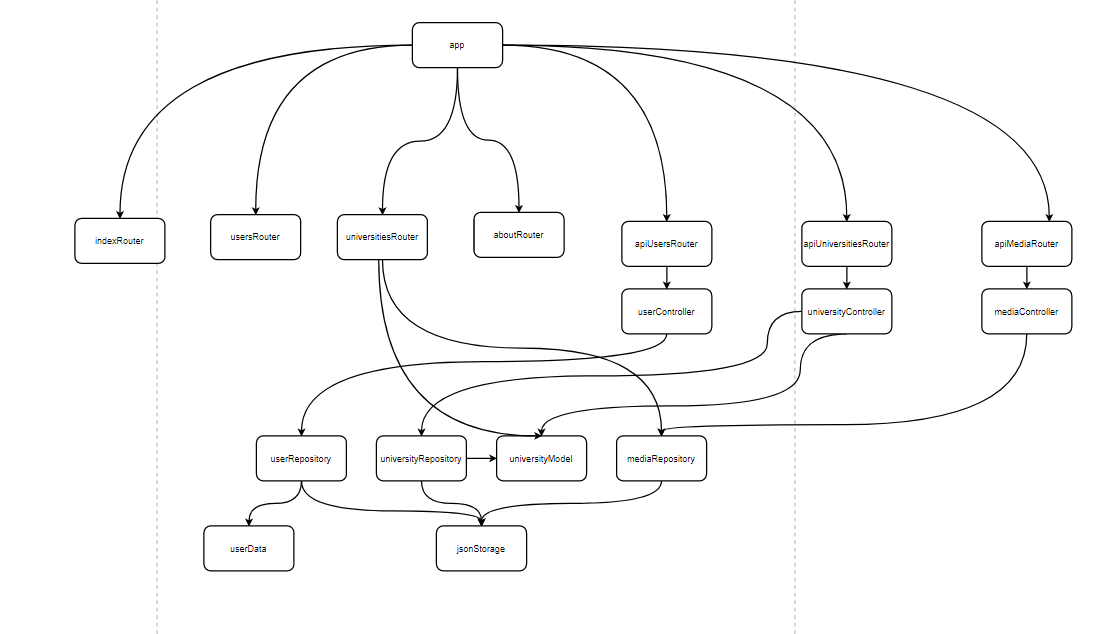
Для включення цієї можливості використайте consolidate i swig та код налаштування із Додатків (Використання HTML view).

1. Шаблонізація
   1. Виділити із HTML-сторінок шаблони для вставки у них об’єктів моделей за допомогою обраного шаблонізатора (наприклад, [**Mustache**](https://github.com/janl/mustache.js)). Повинні з'явитися наступні шаблони:
   2. views/
   3. index.mst
   4. users.mst
   5. user.mst
   6. {entities}.mst
   7. {entity}.mst
   8. about.mst
   9. Видалити HTML файли.
   10. Виділити 3 часткові шаблони і помістити їх у views/partials/:
       * head.mst - винести сюди все спільне з тегу head шаблонів
       * header.mst - винести спільну розмітку "шапки" із тегу body шаблонів
       * footer.mst - винести спільну розмітку "підвалу" із тегу body шаблонів
   11. Включити часткові шаблони у всі інші шаблони.

#### **Частина 3. HTML-форми і CRUD операції для сутностей**

1. Пошук сутностей по назві
   1. Модифікувати веб-сторінку /{entities} з сутностями так, щоби можна було шукати сутності по частковому співпадінню рядка назви за допомогою HTML форми із GET запитом.
   2. Сторінка має містити спеціальні компоненти для інформування користувача про те, для якого пошукового запиту відображаються результати.
   3. Додати пагінацію результатів. Показувати користувачу номер поточної сторінки та загальну кількість сторінок із посиланнями для переходу на попередню/наступну/довільну сторінку.
   4. Якщо в результаті пошуку не було знайдено жодних об'єктів, додавати на сторінку спеціальне повідомлення про це.
2. Створення нових сутностей
   1. Створити сторінку /{entities}/new і додати на неї HTML форму для створення нової сутності (за варіантом) за допомогою POST запиту.
   2. Дана форма обов’язково повинна мати одне поле для завантаження на сервер файлу (зображення чи ін.), що пов'язаний із типом об'єкта.
   3. Завантажені файли розміщувати у директорії data/media/. Додати у веб-сервер обробник GET запитів на отримання файлів із data/media/. У поля сутностей зберігати URL для отримання цих файлів через GET запити до веб-сервера.
   4. Після створення нової сутності і присвоєння їй ідентифікатора, перенаправити на сторінку нової сутності (/{entities}/:id).
3. Видалення сутностей
   1. На сторінці перегляду інформації про окрему сутність (/{entities}/:id) додати форму (яка виглядатиме як кнопка, посилання чи зображення) для видалення поточної сутності за допомогою POST запиту.

**Діаграма залежностей модулів:**



**Код розроблених модулів:**

|  |
| --- |
| **app** |
| const express = require('express');  const fileUpload = require('express-fileupload');  const path = require('path');  const bodyParser = require('body-parser');  const exphbs = require('express-handlebars');  const routes = [];  routes['index'] = require('./routes/index');  routes['users'] = require('./routes/users');  routes['universities'] = require('./routes/universities');  routes['about'] = require('./routes/about');  routes['apiUsers'] = require('./routes/apiUsers');  routes['apiUniversities'] = require('./routes/apiUniversities');  routes['apiMedia'] = require('./routes/apiMedia');  const PORT = process.env.PORT || 3000;  const app = express();  const hbs = exphbs.create({      defaultLayout: 'main',      extname: 'hbs',      helpers: {          if\_eq: function (a, b, opts) {              console.log(a);              console.log(b);              if (a === b) {                  return true;              } else {                  return opts.inverse(this);              }          },      },  });  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:${PORT}`,          produces: ['application/json'],      },      basedir: \_\_dirname,      files: ['./routes/\*\*/\*.js', './models/\*\*/\*.js'],  };  expressSwagger(options);  app.engine('hbs', hbs.engine);  app.set('view engine', 'hbs');  app.use(      fileUpload({          createParentPath: true,      })  );  app.use(bodyParser.json());  app.use(bodyParser.urlencoded({ extended: true }));  app.use(express.static(path.join(\_\_dirname, 'public')));  app.use('/', routes['index']);  app.use('/users', routes['users']);  app.use('/universities', routes['universities']);  app.use('/about', routes['about']);  app.use('/api/users', routes['apiUsers']);  app.use('/api/universities', routes['apiUniversities']);  app.use('/api/media', routes['apiMedia']);  app.listen(PORT, () => {      console.log(`Server has been started on http://localhost:${PORT}`);  }); |

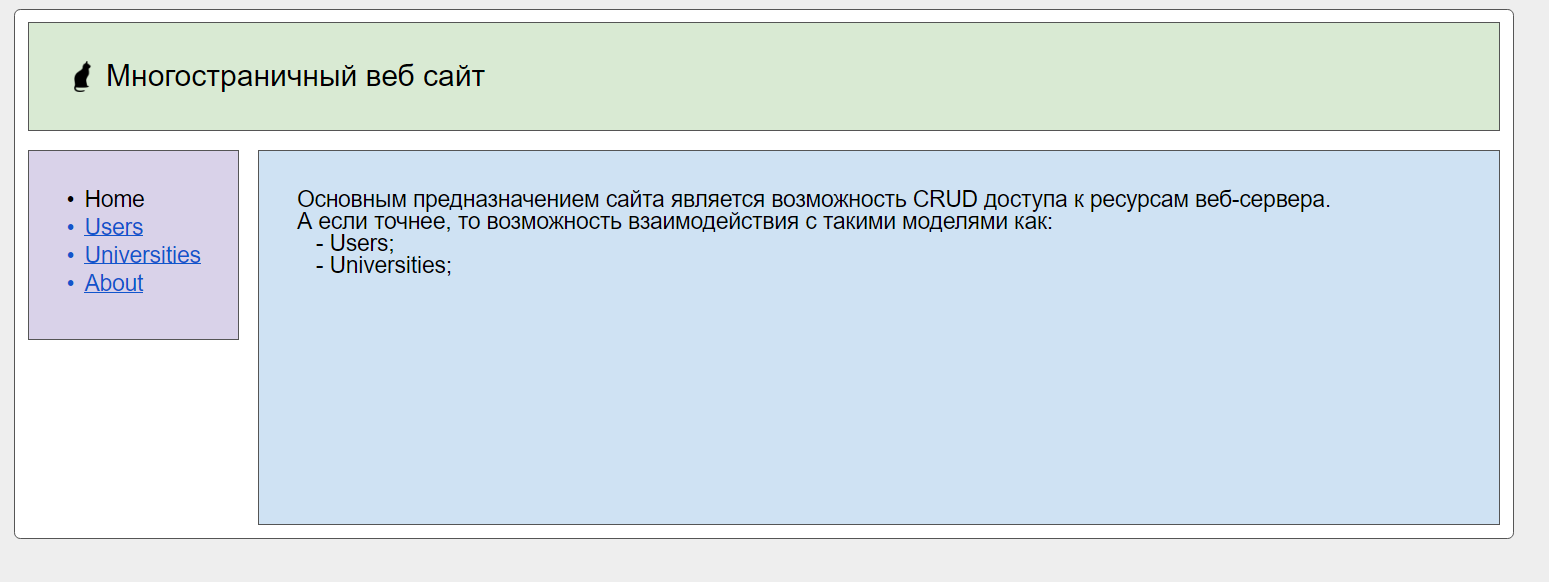
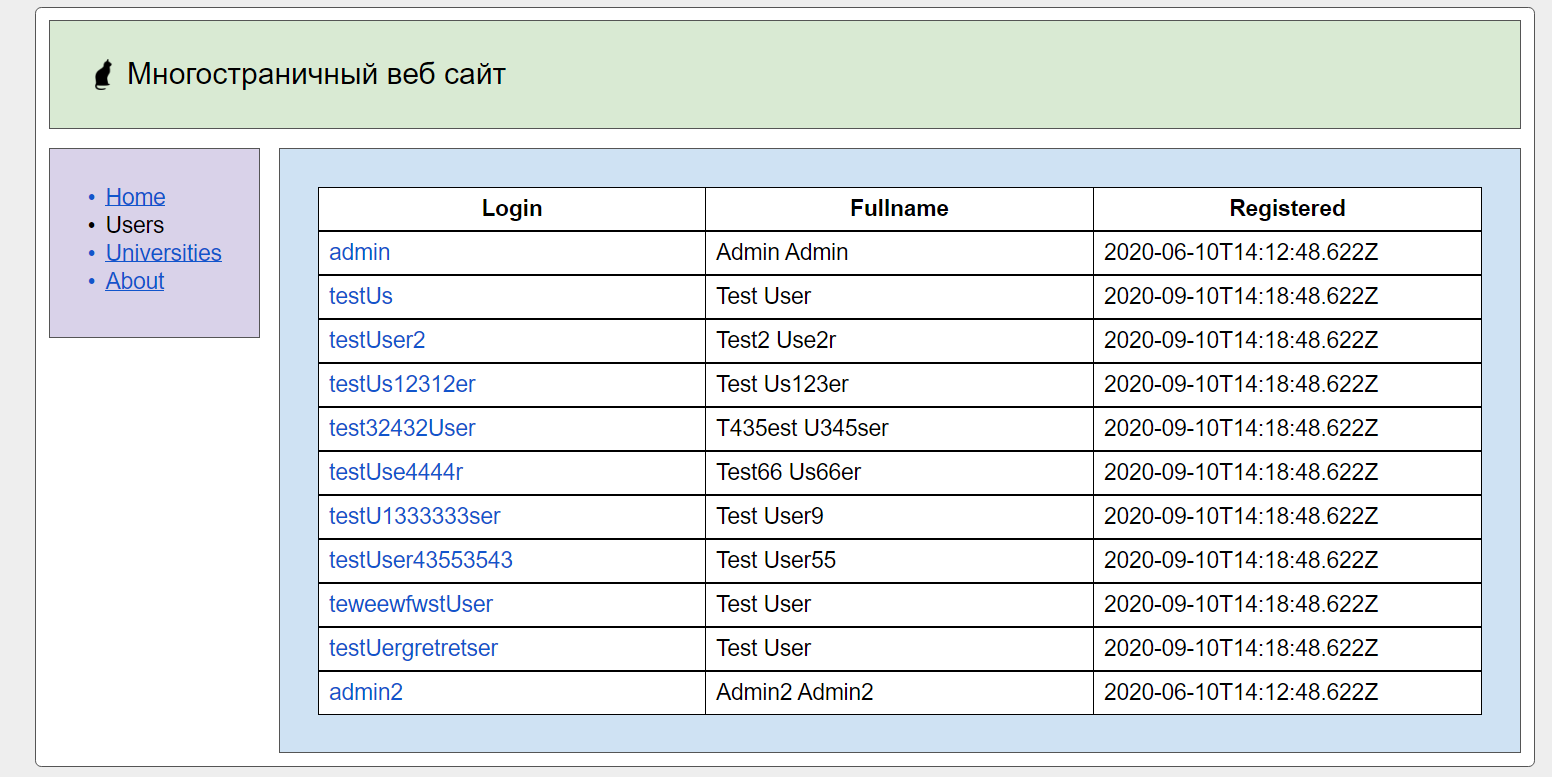
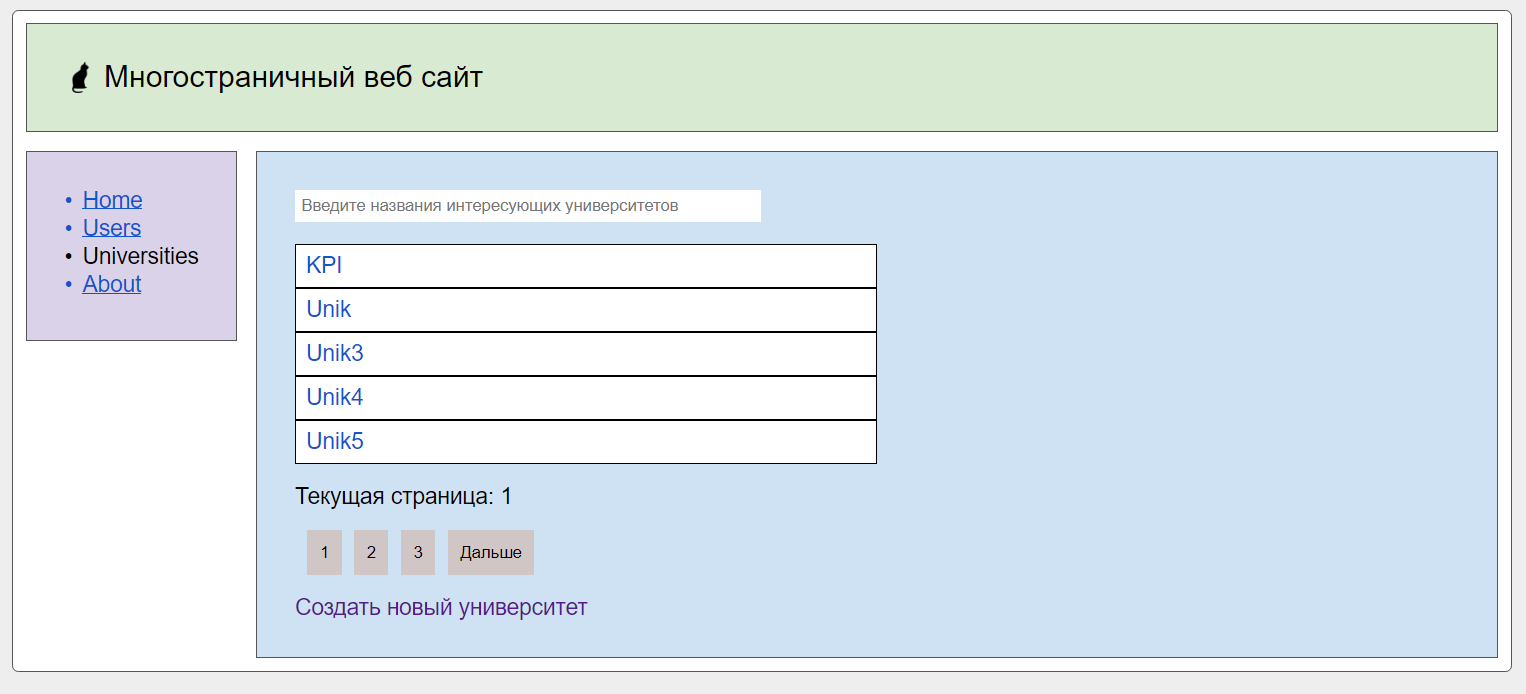
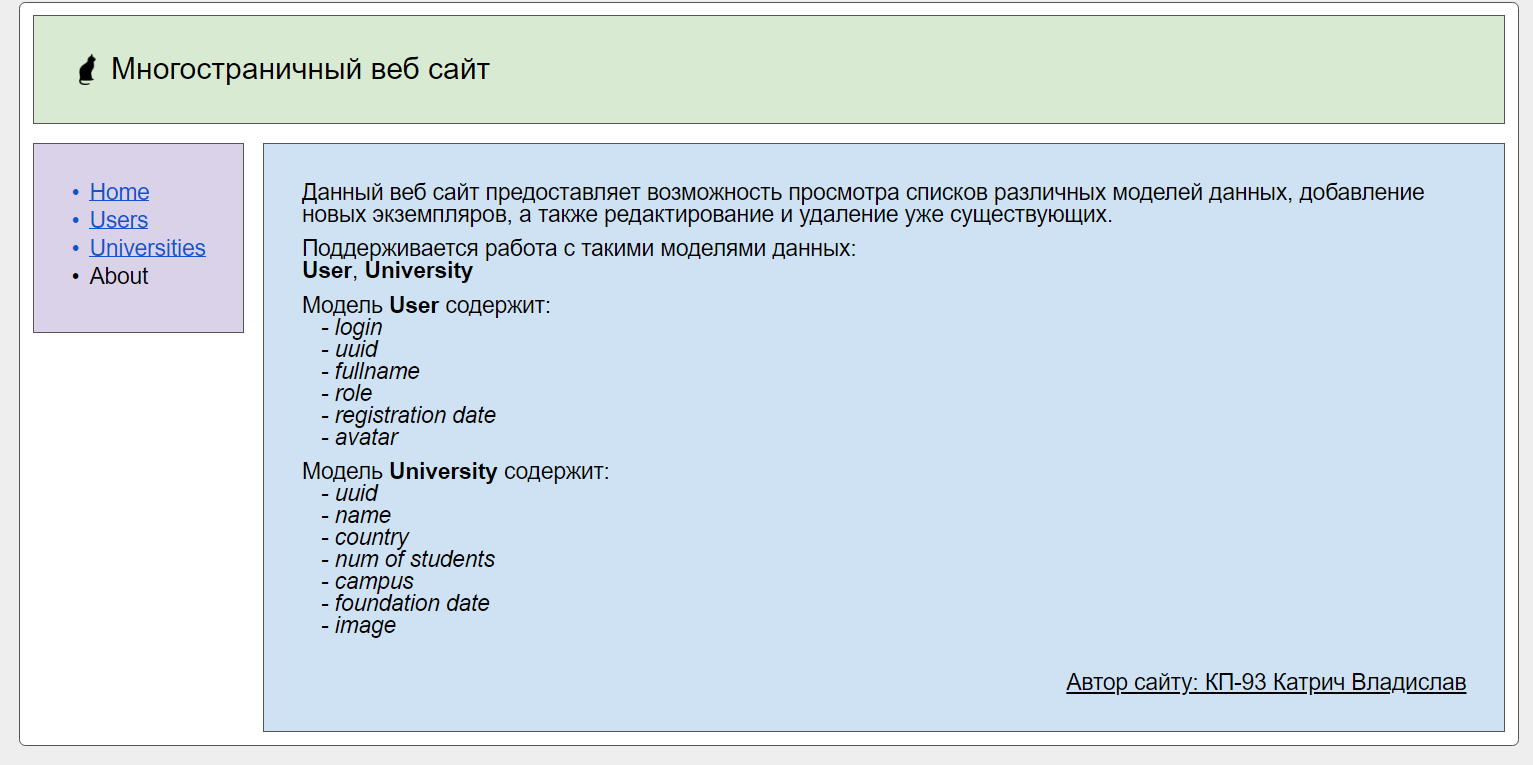
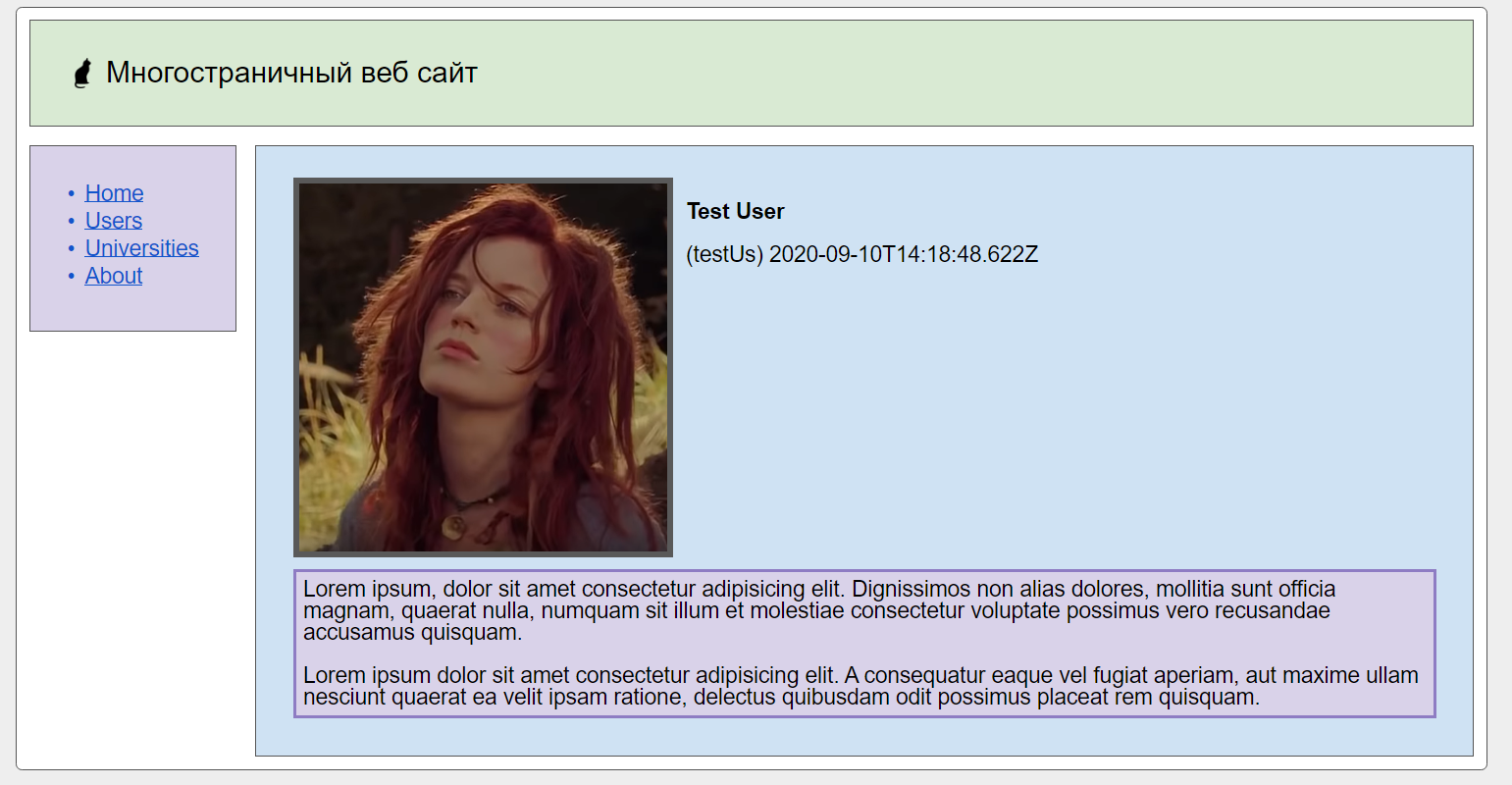
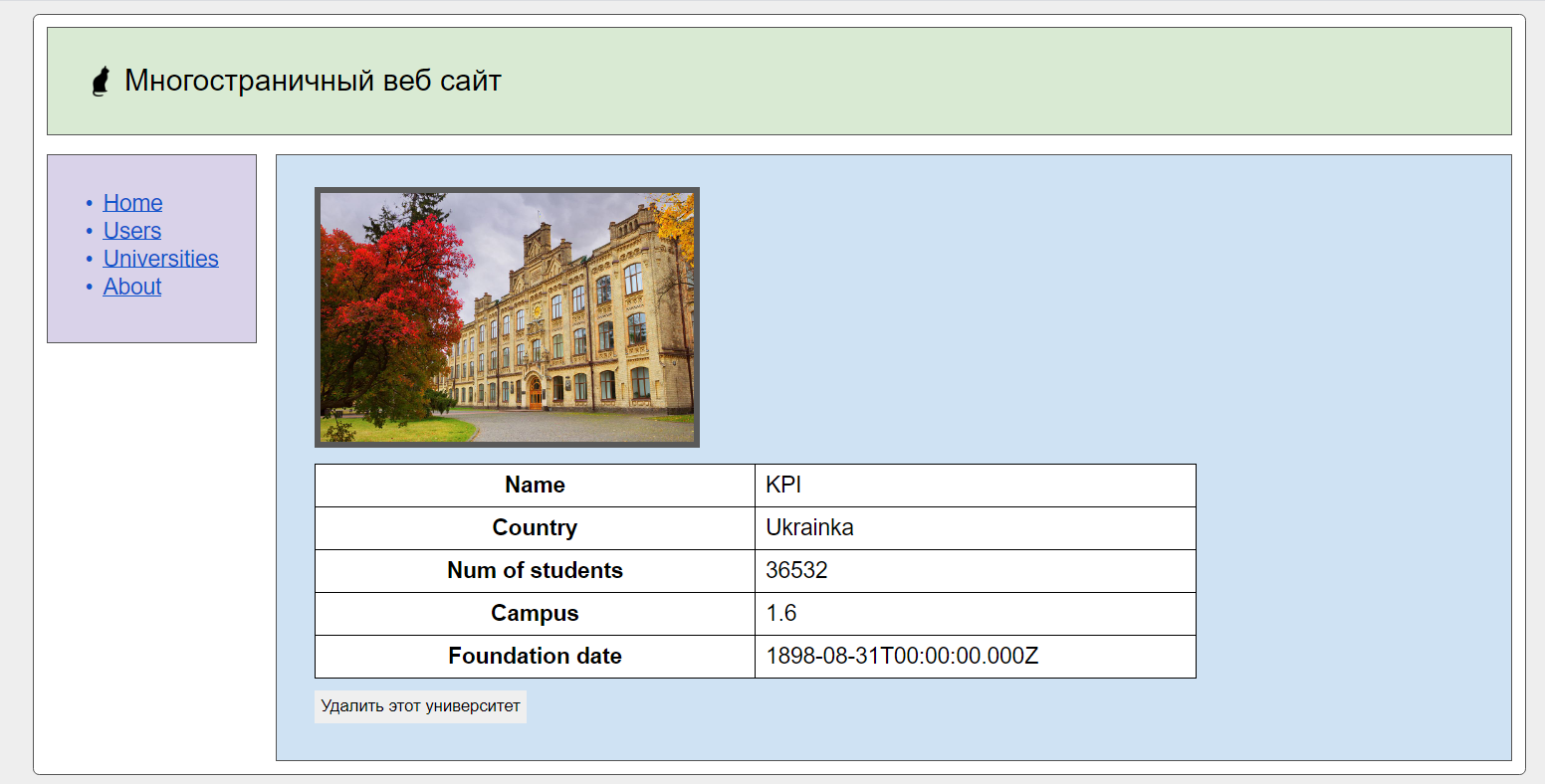
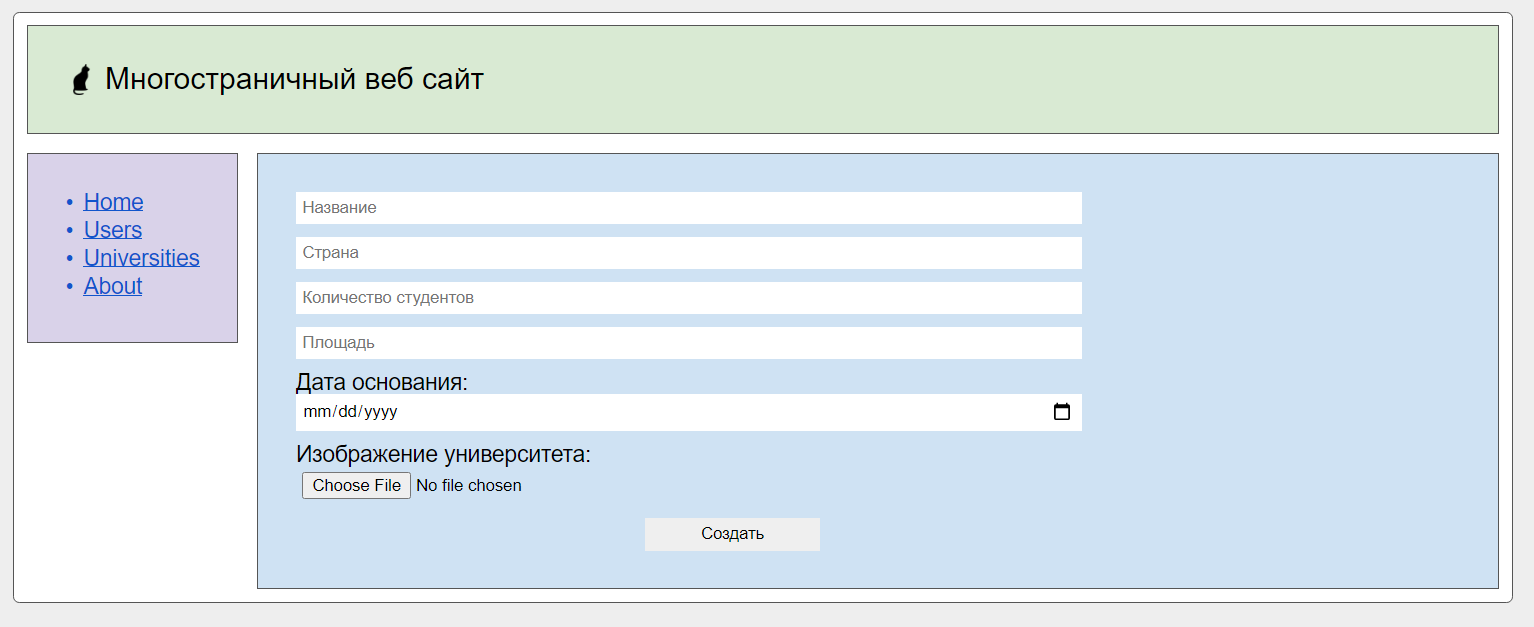
|  |
| --- |
| **universitiesRouter** |
| const { Router } = require('express');  const router = Router();  const PORT = process.env.PORT || 3000;  const http = require('http');  const uuidv4 = require('uuid-v4');  const University = require('../models/university');  const MediaRepository = require('../repositories/mediaRepository');  const mediaRepository = new MediaRepository('data/media.json');  router.get('/', (req, res) => {      const name = req.query.name;      const currentPage = req.query.page || 1;      http.get(          `http://localhost:${PORT}/api/universities?page=${currentPage}`,          async (apiRes) => {              apiRes.setEncoding('utf8');              let rawData = '';              await apiRes.on('data', (chunk) => {                  rawData += chunk;              });              const universities = JSON.parse(rawData);              if (name)                  universities.results = universities.results.filter(                      (university) =>                          university.name                              .toUpperCase()                              .includes(name.toUpperCase())                  );              universities.page = [                  ...Array(universities.numOfPages + 1).keys(),              ].slice(1);              // console.log(universities);              res.render('universities', {                  title: 'Universities',                  isUniversities: true,                  universities,                  name,                  currentPage,              });          }      );  });  router.get('/new', (req, res) => {      res.render('newUniversity', {          title: 'Create new university',      });  });  router.post('/new', (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']      );      if (req.files)          university.unImage = `/api/media/${req.files['unImage'].md5}`;      // for (const key in university) {      //  console.log('Ключ: ' + key + ' значение: ' + university[key]);      // }      const apiReqToUniversity = http.request(          {              hostname: 'localhost',              port: PORT,              path: '/api/universities',              method: 'POST',              headers: {                  'Content-Type': 'application/json',              },          },          (apiRes) => {              console.log(`statusCode: ${apiRes.statusCode}`);              apiRes.on('data', (d) => {                  process.stdout.write(d);              });          }      );      apiReqToUniversity.on('error', (error) => {          console.error(error);      });      apiReqToUniversity.write(JSON.stringify(university));      apiReqToUniversity.end();      if (req.files) {          const image = {};          image['id'] = req.files['unImage'].md5;          image['path'] = `./uploads/${req.files['unImage'].name}`;          const images = mediaRepository.getMedia();          const isImage = images.find((img) => img.id === image.id);          if (!isImage) {              req.files['unImage'].mv('./uploads/' + req.files['unImage'].name);              images.push(image);              mediaRepository.addMedia(images);          }      }      res.status(201).redirect(`/universities/${university.uuid}`);  });  router.get('/:id', (req, res) => {      http.get(          `http://localhost:${PORT}/api/universities/${req.params.id}`,          async (apiRes) => {              apiRes.setEncoding('utf8');              let rawData = '';              await apiRes.on('data', (chunk) => {                  rawData += chunk;              });              const university = JSON.parse(rawData);              // console.log(university);              res.render('university', {                  title: university.name,                  university,              });          }      );  });  router.post('/:id', (req, res) => {      const apiReqToUniversity = http.request(          {              hostname: 'localhost',              port: PORT,              path: `/api/universities/${req.body.uuid}`,              method: 'DELETE',              headers: {                  'Content-Type': 'application/json',              },          },          (apiRes) => {              console.log(`statusCode: ${apiRes.statusCode}`);              apiRes.on('data', (d) => {                  process.stdout.write(d);              });          }      );      apiReqToUniversity.on('error', (error) => {          console.error(error);      });      apiReqToUniversity.write(req.body.uuid);      apiReqToUniversity.end();      res.status(200).redirect('/universities');  });  module.exports = router; |

|  |
| --- |
| **userRouter** |
| const { Router } = require('express');  const router = Router();  const PORT = process.env.PORT || 3000;  const http = require('http');  router.get('/', (req, res) => {      http.get(`http://localhost:${PORT}/api/users?limit=999`, async (apiRes) => {          apiRes.setEncoding('utf8');          let rawData = '';          await apiRes.on('data', (chunk) => {              rawData += chunk;          });          const users = JSON.parse(rawData);          // console.log(users);          res.render('users', {              title: 'Users',              isUsers: true,              users,          });      });  });  router.get('/:id', (req, res) => {      http.get(          `http://localhost:${PORT}/api/users/${req.params.id}`,          async (apiRes) => {              apiRes.setEncoding('utf8');              let rawData = '';              await apiRes.on('data', (chunk) => {                  rawData += chunk;              });              const user = JSON.parse(rawData);              // console.log(user);              res.render('user', {                  title: user.login,                  user,              });          }      );  });  module.exports = router; |

|  |
| --- |
| **indexRouter** |
| const { Router } = require('express');  const router = Router();  router.get('/', (req, res) => {      res.render('index', {          title: 'Home',          isIndex: true,      });  });  module.exports = router; |

|  |
| --- |
| **aboutRouter** |
| const { Router } = require('express');  const router = Router();  router.get('/', (req, res) => {      res.render('about', {          title: 'About',          isAbout: true,      });  });  module.exports = router; |

**Зображення всіх розроблених сторінок**

**Висновок**

Виконавши цю лабораторну роботу, ми навчились створювати веб-сайт, що надає CRUD доступ до ресурсів веб-сервера, ознайомились із шаблонізаторами веб-сторінок, а також навчились створювати HTML-форми для взаємодії користувача із веб-сайтом.