**// Node.js #15 Интеграция БД в приложение (MongoDB Integration)**

// Настройка БД mongodb - видео 00:00 - 05:55

// сайт - ***mongodb.com***, вход через Google akaunt - oleh.kolesnyk.dnepr@gmail.com

// name DataBase - node-blog, password - pass321

***const express = require('express');***

***const path = require('path');***

***const morgan = require('morgan');***

// в терминале вводим команду: ***npm i mongoose***

***const mongoose = require('mongoose');*** // после установки импортируем mongoose в наш сервер

***const Post = require('./models/post');*** //импортируем только что созданную модель

***const app = express();***

***app.set('view engine', 'ejs');***

***const PORT = 3000;***

***const db = 'mongodb+srv://Oleh:pass321@cluster0.nnedg.mongodb.net/node-blog?retryWrites=true&w=majority';*** // создали константу ***db***, которой присвоили строку подключения, не забыв изменить в ней пароль (***pass321***) и имя БД (***node-blog***)

***mongoose*** //устанавливаем подключение к БД

***.connect(db, { useNewUrlParser: true, useUnifiedTopology: true })*** // в метод передаем константу ***db***, и еще два дополнительных аргумента

***.then((res) => console.log('Conected to DB'))*** // поскольку это асинхронная операция можем воспользоваться методами .***then*** и .***catch***. Здесь мы выведем в консоль строку о том, что успешно подключились к БД

***.catch((error) => console.log(error));*** // выведем в консоль если появится ошибка. После этого останавливаем и перезапускаем сервер.

***const createPath = (page) => path.resolve(\_\_dirname, 'ejs-views', `${page}.ejs`);***

***app.listen(PORT, (error) => {***

***error ? console.log(error) : console.log(`listening port ${PORT}`);***

***})***

***app.use(morgan(':method :url :status :res[content-length] - :response-time ms'));***

***app.use(express.urlencoded({ extended: false }));***

***app.use(express.static('styles'));***

***app.get('/', (req, res) => {***

***const title = 'Home';***

***res.render(createPath('index'), { title });***

***});***

***app.get('/contacts', (req, res) => {***

***const title = 'Contacts';***

***const contacts = [***

***{ name: 'YouTube', link: 'http://youtube.com/YauhenKavalchuk' },***

***{ name: 'Twitter', link: 'http://twitter.com/YauhenKavalchuk' },***

***{ name: 'GitHub', link: 'http://github.com/YauhenKavalchuk' },***

***]***

***res.render(createPath('contacts'), { contacts, title });***

***});***

***app.get('/posts/:id', (req, res) => {***

***const title = 'Post';***

***const post = {***

***id: '1',***

***text: 'You think water moves fast? You should see ice. It moves like it has a mind. Like it knows it killed the world once and got a taste for murder. After the avalanche, it took us a week to climb out.',***

***title: 'Post title',***

***date: '12/02/2022',***

***author: 'Nemo',***

***};***

***res.render(createPath('post'), { title, post });***

***});***

***app.get('/posts', (req, res) => {***

***const title = 'Posts';***

***const posts = [{***

***id: '1',***

***text: 'You think water moves fast? You should see ice. It moves like it has a mind. Like it knows it killed the world once and got a taste for murder. After the avalanche, it took us a week to climb out.',***

***title: 'Post title',***

***date: '12/02/2022',***

***author: 'Nemo', //***

***}]***

***res.render(createPath('posts'), { title, posts });***

***});***

//изменяем логику добавления поста

***app.post('/add-post', (req, res) => {***

***const { title, author, text } = req.body;*** //оставляем только деструктуризацию данных из запроса

***const post = new Post({ title, author, text });*** //используя модель ***post*** собираем новый объект ***Post***. Внутрь конструктора передаем получаемые из запроса данные

***post***

***.save()*** // после чего используем метод .***save***. Он также является асинхронным, поэтому мы можем обработать приходящий результат

***.then((result) => res.send(result))*** //в этом методе, используя ***res.send***, отправляем данные на ***UI***, т.е. если все удачно сохранится, то сохраненные данные мы увидим в браузере.

***.catch((error) => {*** // здесь перехватим ошибку

***console.log(error);***

***res.render(createPath('error'), { title: 'Error' });*** //и дополнительно сделаем редирект на страницу ошибки

***})***

***});***

***app.get('/add-post', (req, res) => {***

***const title = 'Add Post';***

***res.render(createPath('add-post'), { title });***

***});***

***app.get('/about-us', (req, res) => {***

***res.redirect('/contacts');***

***})***

***app.use((req, res) => {***

***const title = 'Error Page';***

***res***

***.status(404)***

***.render(createPath('error'), { title });***

***})***

**// Node.js #16 Добавление и получение данных**

**(Get & Post Requests)**

//через сайт БД в нашей БД создаем новую коллекцию ***contacts***. Внутри коллекции кликаем кнопку ***INSERT*** ***DOCUMENT***. В появившемся окне мы можем добавить пару КЛЮЧ-ЗНАЧЕНИЕ. 1-я пара: ***name-YouTube***, 2-я пара: ***link-http://youtube.com/YauhenKavalchuk***. Справа есть оп-ции для выбора формата данных. В нашем случае - ***String*** (строка). Снова кликаем кнопку ***INSERT*** ***DOCUMENT*** и создаем еще 2 пары: ***name-Twitter*** и ***link-http://twitter.com/YauhenKavalchuk***. После того, как мы добавили нужные данные в ***mongodb***, для этих контактов нам нужно создать модель. Для этого копируем модель файла постов, переименовываем его, удаляем лишние аргументы, удаляем ***timestamps*** (т.к. значения контактов не будем создавать в приложении, а будем брать готовые из БД) и вместо ***post*** прописываем ***contact***. Поля переименовываем на ***name*** и ***link***. (см.файл '**./models/contact**')

***const express = require('express');***

***const path = require('path');***

***const morgan = require('morgan');***

***const mongoose = require('mongoose');***

***const Post = require('./models/post');***

***const Contact = require('./models/contact');*** //импортируем БД в файл сервера

***const app = express();***

***app.set('view engine', 'ejs');***

***const PORT = 3000;***

***const db = 'mongodb+srv://Oleh:pass321@cluster0.nnedg.mongodb.net/node-blog?retryWrites=true&w=majority';***

***mongoose***

***.connect(db, { useNewUrlParser: true, useUnifiedTopology: true })***

***.then((res) => console.log('Conected to DB'))***

***.catch((error) => console.log(error));***

***const createPath = (page) => path.resolve(\_\_dirname, 'ejs-views', `${page}.ejs`);***

***app.listen(PORT, (error) => {***

***error ? console.log(error) : console.log(`listening port ${PORT}`);***

***})***

***app.use(morgan(':method :url :status :res[content-length] - :response-time ms'));***

***app.use(express.urlencoded({ extended: false }));***

***app.use(express.static('styles'));***

***app.get('/', (req, res) => {***

***const title = 'Home';***

***res.render(createPath('index'), { title });***

***});***

//обновим логику, чтобы все данные получать непосредственно из БД

***app.get('/contacts', (req, res) => {***

***const title = 'Contacts';***

***Contact*** //обращаемся к модели

***.find()*** // используем метод .***find*** - он помогает найти и вернуть все значения, которые находятся в коллекции

***.then((contacts) => res.render(createPath('contacts'), { contacts, title }))*** //метод асинхронный, поэтому в .***then*** можем отрендерить наш ***UI*** с полученными данными

***.catch((error) => {*** // в случае ошибки выведем ее в консоль и отрисуем страницу ошибки

***console.log(error);***

***res.render(createPath('error'), { title: 'Error' });***

***});***

***});***

// заключительный роут, который использует данные, хранящиеся на сервере это страница отдельного поста. Здесь из БД нужно возвращать отдельное значение. Копируем конструкцию из ***app.get('/posts'...***

***app.get('/posts/:id', (req, res) => {***

***const title = 'Post';***

***Post***

***.findById(req.params.id)*** //здесь используем метод ***.findById***, который по ID находит конкретное значение и возвращает его. ***ID*** будем брать из адресной строки браузера, получить его можно из запроса (из поля ***.params***, которое хранит различные значения из адресной строки)

***.then((post) => res.render(createPath('post'), { post, title }))*** // после этого в метод ***.then*** возвращается один ***post***, его мы отправляем в ответ, а в качестве страницы на ***UI*** используем файл-шаблон ***post***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath('error'), { title: 'Error' });***

***});***

***});***

// с роутом ***posts*** (который предназначен для вывода всех новостей) сделаем те же изменения, как и с ***contacts*** и список новостей также будем получать из БД

***app.get('/posts', (req, res) => {***

***const title = 'Posts';***

***Post***

***.find()***

***.sort({ createdAt: -1 })*** //для отображения последних добавленных постов в самом верху добавим сортировку по убыванию по дате: добавили метод ***.sort***, внутрь которого передали объект, в качестве ключа указываем поле, по которому будет производиться сортировка (***createdAt***), а в качестве значения -1, т.е. сортировать по убыванию

***.then((posts) => res.render(createPath('posts'), { posts, title }))***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath('error'), { title: 'Error' });***

***});***

***});***

//после изменений в двух предыдущих роутах у нас в постах перестает отображаться дата поста. На сервере в ***mongodb*** для хранения дат есть 2 поля: ***createdAt*** (дата и время создания поста) и ***updateAt*** (дата и время редактирования поста). Для того чтобы отображалась дата создания поста необходимо внести изменения в файлы шаблонов: **./ejs-views/post.ejs** и **./ejs-views/posts.ejs** (см. в них)

***app.post('/add-post', (req, res) => {***

***const { title, author, text } = req.body;***

***const post = new Post({ title, author, text });***

***post***

***.save()***

***.then((result) => res.redirect('/posts'))*** //чтобы создать редирект на страницу со всеми новостями при создании новости, устанавливаем метод ***.redirect*** (логику редиректа), внутрь передаем страницу на которую должен произойти переход

***.catch((error) => {***

***console.log(error);***

***res.render(createPath('error'), { title: 'Error' });***

***})***

***});***

***app.get('/add-post', (req, res) => {***

***const title = 'Add Post';***

***res.render(createPath('add-post'), { title });***

***});***

***app.get('/about-us', (req, res) => {***

***res.redirect('/contacts');***

***})***

***app.use((req, res) => {***

***const title = 'Error Page';***

***res***

***.status(404)***

***.render(createPath('error'), { title });***

***})***

**// Node.js #17 Удаление и редактирование данных   
(Delete & Update Requests)**

//добавлены кнопки удаления и редактирования. Изменения внесены в файлы **post.ejs, posts.ejs, head.ejs**

// Полностью организовать удаление через сервер не возможно, т.к. с ***UI*** нужно передавать данные. Удаление, которое мы реализуем, основывается на ***ID*** элемента, поэтому нам нужно пробросить это поле из браузера (см. шаблон **posts.ejs**)

//После изменений в файле **posts.ejs** создаем здесь роут, который будет обрабатывать удаление

//также внесем изменения в файл **post.ejs**

//чтобы научить изменять данные, копируем файл **add-post.ejs** и переименовываем его в **edit-post.ejs**

// Подключаем модуль ***method-override***:

// ***https://www.npmjs.com/package/method-override - Install***, - устанавливаем модуль в проект, после чего перезапускаем сервер

// ***Examples - override using a query value*** (Примеры - переопределить, используя значение запроса)

***const express = require("express");***

***const path = require("path");***

***const morgan = require("morgan");***

***const mongoose = require("mongoose");***

***const methodOverride = require("method-override");*** //импортируем модуль ***method-override*** в файл сервера, т.к. это пакет является ***midlewar*** то его нужно подключить

***const Post = require("./models/post");***

***const Contact = require("./models/contact");***

***const app = express();***

***app.set("view engine", "ejs");***

***const PORT = 3000;***

***const db =***

***"mongodb+srv://Oleh:pass321@cluster0.nnedg.mongodb.net/node-blog?retryWrites=true&w=majority";***

***mongoose***

***.connect(db, { useNewUrlParser: true, useUnifiedTopology: true })***

***.then((res) => console.log("Conected to DB"))***

***.catch((error) => console.log(error));***

***const createPath = (page) =>***

***path.resolve(\_\_dirname, "ejs-views", `${page}.ejs`);***

***app.listen(PORT, (error) => {***

***error ? console.log(error) : console.log(`listening port ${PORT}`);***

***});***

***app.use(express.urlencoded({ extended: false }));***

***app.use(***

***morgan(":method :url :status :res[content-length] - :response-time ms")***

***);***

***app.use(express.static("styles"));***

***app.use(methodOverride("\_method"));*** // т.к. это ***method-override*** является ***midlewar*** то его нужно подключить, используя конструкцию ***app.use***, внутрь которой передаем данный модуль со строкой, на которую он будет реагировать (тип такого подключения описан на странице документации: ***https://www.npmjs.com/package/method-override***)

***app.get("/", (req, res) => {***

***const title = "Home";***

***res.render(createPath("index"), { title });***

***});***

***app.get("/contacts", (req, res) => {***

***const title = "Contacts";***

***Contact.find()***

***.then((contacts) => res.render(createPath("contacts"), { contacts, title }))***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

***app.get("/posts/:id", (req, res) => {***

***const title = "Post";***

***Post.findById(req.params.id)***

***.then((post) => res.render(createPath("post"), { post, title }))***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

//создаем здесь роут, который будет обрабатывать удаление, используя новый тип метода - ***delete***

***app.delete("/posts/:id", (req, res) => {***

  //в ***callback*** функции мы снова обращаемся к модели ***Post***

***const title = "Post";***

***Post.findByIdAndDelete(req.params.id)*** //используем метод ***.findByIdAndDelete*** В качестве аргумента передаем ***ID***, который мы отправляем из ***UI***

***.then((result) => {***

***res.sendStatus(200);*** // при успешном выполнении вернем 200-й статус

***})***

***.catch((error) => {***

      //а конструкцию ***catch*** можем скопировать из любого другого роута и тогда при наличии ошибки у нас будет отрисовываться страница с ошибкой.

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

// добавим роут, который будет возвращать созданную страницу с формой, внутри которой заполнены все данные

***app.get("/edit/:id", (req, res) => {***

  // используем метод ***.get***, роут определяем как ***/edit/:id***,

***const title = "Edit Post";*** //внутри задаем ***title***

***Post.findById(req.params.id)***

***.then((post) => res.render(createPath("edit-post"), { post, title }))*** // логику возврата шаблона с данными остается точно такая же как и в роуте страницы одного поста, только вместо шаблона ***post*** мы используем шаблон ***edit-post***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});*** // чтобы попадать на эту страницу нужно внести изменения в файле ***post.ejs*** (см. стр. 13-14)

// добавляем роут (логику), чтобы вносимые изменения обновляли существующий в БД пост

***app.put("/edit/:id", (req, res) => {***

  // создаем новый роут с методом ***.put***

***const { title, author, text } = req.body;*** // данные о заголовке,авторе и текст поста мы будем получать из тела запроса

***const { id } = req.params;***

***Post.findByIdAndUpdate(id, { title, author, text })*** // у модели Post используя метод ***.findByIdAndUpdate*** мы не сохраняем новый, а обновляем существующий пост. Данный метод ищет в БД элемент по номеру ***id***, после чего обновляет его поля новыми полученными данными, поэтому в качестве аргумента он принимает ***id***, который берем из параметров запроса, а в качестве второго аргумента идет объект с данными

***.then((result) => res.redirect(`/posts/${id}`))*** // в случае успешного обновления нужно выполнить переход обратно на страницу постов

***.catch((error) => {***

***console.log(error);*** //в случае ошибки отрисовываем соответствующую страницу

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

***app.get("/posts", (req, res) => {***

***const title = "Posts";***

***Post.find()***

***.sort({ createdAt: -1 })***

***.then((posts) => res.render(createPath("posts"), { posts, title }))***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

***app.post("/add-post", (req, res) => {***

***const { title, author, text } = req.body;***

***const post = new Post({ title, author, text });***

***post***

***.save()***

***.then((result) => res.redirect("/posts"))***

***.catch((error) => {***

***console.log(error);***

***res.render(createPath("error"), { title: "Error" });***

***});***

***});***

***app.get("/add-post", (req, res) => {***

***const title = "Add Post";***

***res.render(createPath("add-post"), { title });***

***});***

***app.get("/about-us", (req, res) => {***

***res.redirect("/contacts");***

***});***

***app.use((req, res) => {***

***const title = "Error Page";***

***res.status(404).render(createPath("error"), { title });***

***});***