**Node.js #10 Пакетный менеджер (NPM & Packages)**

***const { lookupService } = require("dns")***

***NPM - Node Packet Manager*** - коллекция пакетов или модулей, которые используются при разработке Manager

Пакет - один или несколько ***JavaScript*** - файлов, представляющих собой какую - то библиотеку или инструмент (готовый блок функциональности, который можно использовать в своем проекте)

Для установки - команда ***npm init*** из папки, где планируется развернуть проект и ответить на несколько вопросов:

***package name: (js)*** - название проекта

***version: (1.0 .0)*** - версия(можно пустое)

***description***: -описание(можно пустое)

***entry point: (app.js)*** - точка входа(корневой файл проекта с которого предполагается запуск)

***test command***: - команда для запуска тестов(можно пустое)

***git repository***: -путь к гиту, если есть

***keywords***: -ключевые слова

***auyhor***: - авторство

***license***: ***(ISC)*** - лицензия

и еще раз подтвердить ***ОК***

Результатом - появится ключевой файл любого проекта в ***WEB*** - разработке **package.jso**, в котором можно добавить или изменить то, что ввели на этапе инициализации проекта

КАК ДОБАВИТЬ СТОРОННИЙ МОДУЛЬ В проект

Сначала нужно знать для решения какой проблемы нужно добавить модуль ***nodemon*** - сам останавливает и перезапускает сервер, чтобы не делать это вручную каждый раз при внесении изменений.

Все пакеты можно устанавливать глобально на компьютер(данный пакет будет доступен для всех проектов, которые у меня есть. С другой стороны на компьютере появится ОГРОМНАЯ папка в которой постоянно будут накапливаться модули. При этом модули периодически обновляются и иногда может получиться, что новая версия модуля не поддерживает что - то из предыдущей и у нас есть несколько проектов, которые используют разные версии. И вот тут начинаются проблемы.), а можно устанавливать локально в проект с добавлением зависимости в **packet.json**(для работы используют локальную установку с добавлением зависимости в **packet.json**, к тому же, без данной зависимости данный проект не возможно будет запустить на других компьютерах, а только на своем локальном)

2 - й вариант команда: ***npm install --save-dev nodemon***

После установки в **package.json** появится новое поле "***devDependencies***" (здесь появляются зависимости, которые требуются для разработки)

в папке ***dependencies*** будут появляться зависимости, без которых наш проект не будет работать

Просто установить модуль не достаточно. Его нужно начать использовать. Как? Читайте в документации.

В нашем случае, нужно создать отдельный скрипт и присвоить ему команду запуска сервера:

***"scripts": {***

***"dev": "nodemon app.js" // "dev" - имя команды, app.js будет запускаться с помощью пакета nodemon***

***},***

теперь запустить сервер можно командой:

***npm run dev***

и чтобы в терминале видеть что все работает в ***createServer*** добавляем:

***console.log('Just for Test');***

и в терминале после обновления браузера мы будем видеть 2 строки: )

после того как установили модуль он сам и все его зависимости будут храниться в папке **node\_modules**

Также появится файл

**package-lock.json**

он нужен для синхронизации всех версий пакетов на разных компьютерах, чтобы все версии пакетов были одинаковые у всех кто разрабатывает командой

И еще 2 модуля, которые понадобятся в будущих уроках - команда:

***npm i express ejs***

***express - node js framework***, для более элегантной работы с запросами и ответами

***ejs*** - шаблонизатор для добавления динамических данных на наш сайт и позволит рендерить их на UI

**Node.js #11 Node.js & Express (Node.js & Express)**

// устанавливаем путь запуска в **packed.json**:

***// "scripts": {***

***// "dev": "nodemon 10server.js"***

***// },***

// командой ***npm run dev*** запускаем прослушивание

// создаем сервер:

***const express = require('express');*** // подключаем модуль ***Express***

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

***const app = express();*** // создаем константу ***app***, в которую присваиваем вызов ***express***

***const PORT = 3000;*** // определяем константу для порта

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

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

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

***})*** // и запускаем прослушивание с помощью метода .***listen***. ***'localhost'*** можно было оставить, но уберем, т.к. он явно определится

// Теперь у нас есть сервер, в который интегрирован ***express*** и он прослушивает запросы на 3000-м порту

// приступаем к передаче данных

// чтобы отправить данные из сервера в браузер используем метод .***get***, который вызываем непосредственно в созданном ***app***

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

// ***res.send('<h1>Hello world!</h1>');*** // внутри используется один метод: ***res.send***. В него передаются данные, которые нужно отправить в браузер. В ***express*** не нужно передавать тип данных ***'Conten-type'***. Он сам определяет тип и передает их в браузер.

// ***});***

// методы в ***express*** принимают 2 основных параметра: ***rout*** - путь по которому идет обращение и ***callback***-функция, которая будет вызываться. У неё также есть два основных параметра - это объекты запроса и ответа

// Теперь можем перейти к созданию полноценного роутинга

// для начала подключаем вверху модуль ***path*** и создаем функцию ***createPath***, которые понадобятся для создания пути до файла, после чего обновляем логику метода .***get*** (метод .***send*** предназначенный для отправки данных заменяем на метод .***sendFile***, предназначенный для отправки файлов и внутрь передаем функцию ***createPath*** с именем домашней страницы и она принимает следующий вид:)

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

***res.sendFile(createPath('index'));***

***});***

// также делаем для контактов

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

***res.sendFile(createPath('contacts'));***

***});***

// реализуем redirect

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

***res.redirect('/contacts');*** // используя redirect переходим на /contacts

***})***

// теперь обработка ошибок

// для этого есть файл error.html. На этот раз вместо метода .***get*** мы воспользуемся методом .***use***. На этот раз путь передавать не нужно

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

***res***

***.status(404)*** // воспользовались встроенным методом .***status*** для

***.sendFile(createPath('error'));***

***})***

// на самом деле сейчас был создан не отдельный роут, а мидлвар, который будет перехватывать запросы по несуществующим путям и рендорить ошибку

**// Node.js #12 Подключение шаблонизатора (View Engine)**

// создадим пользовательский интерфейс и превратим статичные страницы в динамичные, которые будут получать данные от сервера и рендерить (воспроизводить) их

// в папке **views** появились новые страницы: **add-post, post, posts**. Чтобы увидеть как они выглядят добавим их в файле сервера

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

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

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

// подключаем ***ejs*** в ТРИ ШАГА. ПОСЛЕ этого у нас появляются новые возможности.

// 1 - ***partial template*** (частичный шаблон) - чтобы инкапсулировать повторяющиеся части, например навигацию в страницах:

// создаем папку ***partials*** в которой создаем файл ***nav.ejs***, открываем любой файл (например **index.ejs**) и вырезаем из него блок с навигацией (код, находящийся между тегами ***<nav>*** и ***</nav>***, включая эти тэги). Вырезанный блок с навигацией вставляем в файл **nav.ejs,** а в том месте, где мы его вырезали вставляем стандартный синтаксис ***ejs***, у которого в начале и в конце мы добавляем знак %. В остальном он похож на обычный тэг. Внутри мы указываем директиву ***include*** в которую передаем путь до файла с навигацией:

***// <%- include('./partials/nav.ejs') %>***

// копируем эту сроку и заменяем таким образом навигацию во всех остальных шаблонах папки **ejs-views**

// ПОДРОБНЕЕ О РАЗЛИЧНЫХ КОНСТРУКЦИЯХ EJS МОЖНО ПОЧИТАТЬ НА ОФИЦИАЛЬНОМ САЙТЕ С ДОКУМЕНТАЦИЕЙ: ***https://ejs.co/***

// Теперь рассмотрим ПЕРЕДАЧУ ДАННЫХ ШАБЛОННЫХ СТРАНИЦ. (Для начала базовый механизм на примере страницы контактов - 05:23)

// С сервера мы можем отдавать любое количество данных

// В тэге ***ejs*** мы можем описать любой ***JavaScript*** код. В примере мы создадим переменную, но это может быть и функция и любая другая логика

***// <% const name = 'Jack' %>***

***// <li><%= name %></li>*** // используя такой же синтаксис, но добавив = мы обращаемся к пространству имен шаблонизатора и вытягиваем значение ***name***

// основная задача заключается в том, чтобы на сервере формировать нужные данные и отправлять их в шаблон

***app.set('view engine', 'ejs');*** // ШАГ 1: устанавливаем для нашего приложения ***ejs*** в качестве ***view*** ***engine***

***const PORT = 3000;***

***const createPath = (page) => path.resolve(\_\_dirname, 'ejs-views', `${page}.ejs`);*** // ШАГ 2: обновим ***createPath***: все статичные страницы из папки ***views*** скопируем в папку ***ejs-views***, изменив расширение с .***html*** на .***ejs***. Это же расширение (.***ejs***) и папку (***ejs-views***) нужно использовать для построения пути до файла

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

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

***}) /***

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

***const title = 'Home';*** // в каждый роут добавляем константу ***title*** - Home - для главной страницы

***res.render(createPath('index'), { title });*** // ШАГ 3: заменяем метод ***sendFile*** на метод ***render*** (воспроизвести). В каждом из описанных роутов наше приложение начнет использовать ***ejs*** в качестве шаблонизатора для страниц

***});***

// ДОБАВИМ СВОЙ ИНДИВИДУАЛЬНЫЙ TITLE (значение, которое отображается на вкладке браузера) ДЛЯ КАЖДОЙ СТРАНИЦЫ - 07:37

// для этого в каждый роут добавляем константу ***title*** и передаем это значение вторым аргументом для каждого роута

// после этого переходим в каждый шаблон ***ejs*** и обновляем тэг ***title*** до вида:

***// <title>Node.js Course | <%= title %></title>***

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

***const title = 'Contacts'; // в каждый роут добавляем константу title***

***const contacts = [ // добавляем массив с контактами***

***{ name: 'YouTube', link: 'http://youtube.com/YauhenKavalchuk' },*** // формируем список социальных сетей,

***{ name: 'Twitter', link: 'http://twitter.com/YauhenKavalchuk' },*** // после этого в шаблоне **contacts.ejs**

***{ name: 'GitHub', link: 'http://github.com/YauhenKavalchuk' },*** // нужно этот список отрендерить:

// код ниже в файле **contacts.ejs**

// тут вступает в силу специфический синтаксис шаблонизатора: к каждой операции в фигурных скобках мы должны добавить обертку ***ejs*** ***<% %>***

// ***<ul>***

// ***<% if (contacts.length) { %>*** // проверяем логикой существуют ли отправляемые данные

***// <% contacts.forEach(({ link, name }) => { %>*** // с помощью .***forEach*** пробегаемся по получаемым данным

***// <li>***

***// <a href="<%= link %>"><%= name %></a>*** // и строим соответствующую разметку - тэг ***li***, внутри которого будет находиться ссылка, динамически получающая данные из полей ***link*** и ***name***

***// </li>***

***// <% }) %>***

***// <% } %>***

***// </ul>***

***]***

***res.render(createPath('contacts'), { contacts, title });*** // в метод ***render*** вторым аргументом передаем данные массива // и значение ***title***

***});***

***app.get('/posts/:id', (req, res) => {*** // использует передаваемый в ***url***-адресе ***id***, для отображения одного конкретно запрашиваемого поста

***const title = 'Post';*** // в каждый роут добавляем константу ***title***

***res.render(createPath('post'), { title });*** // ШАГ 3: заменяем метод ***sendFile*** на метод ***render*** (воспроизвести). // и передаем значение ***title*** вторым аргументом для каждого роута

***});***

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

***const title = 'Posts';*** // в каждый роут добавляем константу ***title***

***res.render(createPath('posts'), { title });*** // ШАГ 3: заменяем метод ***sendFile*** на метод ***render*** (воспроизвести). // и передаем значение ***title*** вторым аргументом для каждого роута

***});***

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

***const title = 'Add Post';*** // в каждый роут добавляем константу ***title***

***res.render(createPath('add-post'), { title });*** // ШАГ 3: заменяем метод ***sendFile*** на метод ***render*** (воспроизвести). // и передаем значение ***title*** вторым аргументом для каждого роута

***});***

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

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

***})***

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

***const title = 'Error Page';*** // в каждый роут добавляем константу ***title***

***res***

***.status(404)***

***.render(createPath('error'), { title });*** // ШАГ 3: заменяем метод ***sendFile*** на метод render (воспроизвести). // и передаем значение ***title*** вторым аргументом для каждого роута

***})***

**// Node.js #13 Промежуточное ПО (Middleware)**

// промежуточная функциональность, промежуточное ПО, некая логика, которая может быть добавлена в основной поток программы и которая может добавлять функциональность

// когда сервер получает ***request*** (запрос), то в Response (отклике) можно добавить одну или несколько функций, которые будут выполнять дополнительную обработку (установка заголовков, работа с куками, шифрование, логирование, сжатие и т.д.)

// самый популярный вариант промежуточного ПО - это регистрация

// ***Middleware*** - это функция, которая исполняется между (после) получением запроса серверу и (перед) возвратом ответа браузеру

//Рассмотрим пример ЛОГИРУЮЩЕГО ***Middleware*** - 01:17

// берем наше приложение и используем метод .***use*** в который передаем ***callback*** - см. ниже

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

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

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

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

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

***const PORT = 3000;***

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

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

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

***}) /***

// берем наше приложение и используем метод .***use*** в который передаем ***callback***

***// app.use((req, res, next) => {*** // особенность данной функции в том, что она будет принимать 3 аргумента: запроса, отклика и доп.функции ***next()***

***// console.log(`path: ${req.path}`);*** // выводит информацию о пути запроса

***// console.log(`method: ${req.method}`);*** // выводит информацию о методе запроса

***// next();*** // должна быть добавлена в самом конце для возврата контроля серверу, для дальнейшей работы с ответом

***// })***

// таким образом, мы можем добавить любое количество обработчиков и передавать им управление по цепочке

// Так как ***Middleware*** должны определяться между запросом и откликом, то они должны располагаться в самом начале, сразу после создания сервера и прослушивания порта и обязательно ДО создания роута с ответом, т.к. в каждом роуте у нас ечть метод ***res.render***, который возвращает ответ обратно в браузер и после этого промежуточная функциональность не успеет отработать

// для установки нормального логгера ***morgan*** (см.: ***http://expressjs.com/en/resources/middleware/morgan.html***) останавливаем сервер и вводим команду инсталляции: ***npm install morgan***. После установки перезапускаем наш сервер и импортируем морган в модуль (вверху - ***const morgan = require('morgan')).***

***app.use(morgan(':method :url :status :res[content-length] - :response-time ms'));*** // заменили самописный консольный логгер на нормальный. Строку, которую он будет передавать мы скопировали прямо из документации. Теперь при переходе по страницам мы будем видеть: метод, статус, длину контента, и время ответа в миллисекундах - действительно полезная информация, которая может помочь при отлове ошибок и проверке перфоманса сервера.

//-----// Когда мы добавляем статичные файлы в проект, по типу стилей или картинок, мы не можем получить к ним доступ. Если бы это работало, то пользователь мог бы получить доступ к любому файлу, хранящемуся на сервере. Поэтому по умолчанию ***Node.JS*** защищает все хранящиеся данные и для того, чтобы браузер мог получить статичные данные нам требуется это разрешить, т.е. нужно определить файлы и папки, к которым браузер может получить доступ и делается это созданием нового ***Middleware***

***app.use(express.static('styles'));*** // Мы использует стандартный ***app.use*** внутрь которого положим ***express.static***. Данный метод поможет добавить передаваемую папку в исключения, т.е. сделать ее общедоступной. Внутри мы определяем папку ***styles***, т.е. непосредственно папку с нашими стилями.

// Помимо самописных ***Middleware*** мы можем использовать и сторонние, список самых часто используемых здесь: ***http://expressjs.com/en/resources/middleware.html*** Среди них:

// ***body-parser*** - предназначен для чтения тела запроса

// ***compression*** - для того, чтобы в соответствии с передаваемыми аргументами осуществлять сжатие тела ответа

// ***cors*** - для работы с кросс-доменными запросами

// ***cookie-session*** - чтобы дополнительно в ответ устанавливать сессионные куки для получения дополнительной информации

// ***morgan*** - нормальный логгер

***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';***

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

***});***

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

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

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

***});***

***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 #14 Обработка POST запросов (Handling Post Requests)**

// // 01:45 PAUSE - SM LINE 46

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

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

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

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

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

***const PORT = 3000;***

***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 }));*** //внутрь принимает аргументы (в нашем случае - extended: false, т.к. расширенный парсинг нам не нужен)

***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 });***

***});***

//Добавляем "моковые" данные в файл сервера (***Mock***-объект (от англ. ***mock*** ***object***, букв. — «объект-пародия», «объект-имитация», а также «подставка») — в ООП — тип объектов, реализующих заданные аспекты моделируемого программного окружения. ***Mock***-объект представляет собой конкретную фиктивную реализацию интерфейса, предназначенную исключительно для тестирования взаимодействия и относительно которого высказывается утверждение. ***Mock***-объекты активно используются в разработке через тестирование.)

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

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

***const post = {*** //***post*** будет представлять из себя обычный объект

***id: '1',*** // поле ***id*** нужно для уникального идентификатора, который мы будем использовать для роутинга, его же мы будем использовать при удалении и редактировании данных

***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.',*** //***text*** - это полный текст статьи

***title: 'Post title',*** //***title*** - заголовок

***date: '12/02/2022',*** //доп информация - дата

***author: 'Nemo',*** // доп инф - автор

***};***

***res.render(createPath('post'), { title, post });*** // данные ***post*** мы должны передать в файл шаблона. Для этого в объект, который мы передаем в метод .***render*** вторым аргументом отправляем созданный объект ***post***. После этого переходим внутрь шаблона **post.ejs**

***});***

// Теперь все тоже самое для файла **posts.ejs**, но сначала вносим изменения здесь - в сервере:

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

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

***const posts = [{*** //поскольку ***post*** – это массив всех новостей, то все, что нам нужно сделать - это создать массив ***posts*** и скопировать в него объект с одним постом

***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 });*** // по аналогии передаем данные о наборе новостей в соответствующий шаблон - **posts.ejs** - переходим в него...

// после этого мы можем поработать с ***post***-запросом. Этот тип запроса помогает отправлять информацию на сервер.

// чтобы мы могли отправлять какие-то данные у нас есть файл **add-post.ejs**, который содержит форму (идем в этот файл)

***});***

// После редактирования **add-post.ejs** первое, что нужно сделать - это добавить мидлвар для парсинга входящего запроса. Для этого используем метод ***express.urlencoded*** (см. строку 23 + ***http://expressjs.com/en/4x/api.html#express.urlencoded*** (можно почитать об остальных аргументах))

// После того, как мидлвар подключен в файл сервера, создаем роут, который будет обрабатывать получаемые данные. В качестве метода будем использовать .***post***, все остальное идет по аналогии

***app.post('/add-post', (req, res) => {*** //указываем роут, на котором будет происходить обработка и передаем ***callback***, где будет описана логика

***const { title, author, text } = req.body;*** //из тела запроса при помощи деструктуризации вытягиваем ***title***, ***author***, ***text***

***const post = {*** // создаем новый объект ***post***, который будет содержать эти поля. Важно, чтобы здесь были все поля, которые использует шаблон ***post***.***ejs***, т.к. мы к нему обращаемся ниже

***id: new Date(),*** //добавили поле ***id***: куда присвоили ***new*** ***Date()***

***date: (new Date()).toLocaleDateString(),*** // добавили поле date: куда также присвоили (***new Date()***) и взяли метод ***.toLocaleDateString()***, чтобы дата отображалась в нормальном формате

***title,***

***author,***

***text,***

***}***

***res.render(createPath('post'), { post, title });*** //внутри указываем путь до файла ***post***, и в качестве аргумента созданный объект ***post***, а также значения ***title***, которое мы используем для вкладки браузера

***});***

***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 });***

***})***