**Node.js #7 Клиент и сервер (Client & Server)**

***UI*** - интерфейс взаимодействия, в нашем случае - браузер

***Request*** - запрос, бывают:

***Get*** - обычный запрос данных

***Post*** - используется для отправки данных

***Put*** - заменяет хранящиеся на сервере данные новыми

***Delete*** - удаляет данные

***Response*** - отклик (ответ на запрос)

***HOST*** - сервер, который содержит сайт

***\*\*\*.com*** - доменное имя

***HTTP/HTTPS*** - ***Hyper Text Transfer Protocol Secure***

***Protocol*** - набор коммуникационных правил, которые принимают стороны, чтобы взаимодействовать

***Socket*** - канал связи для передачи данных. Отправка данных происходит по протоколу ***TCP*** - ***Transmittion Control Protocol*** c помощью пакетов ***Packets***

***Packets*** - определенным образом оформленный блок данных, передаваемый по сети в пакетном режиме, т.е. информация отправляется небольшими порциями

***127.0.0.1 - loop Back - специальный IP-адрес, который возвращает отправляемые*** запросы на наш компьютер

***localhost:3000*** - локальный хост с портом 3000

**Node.js #7 Клиент и сервер (Client & Server)**

**07app.js**

***const http = require('http');*** // импортируем модуль ***http*** в наш ***js*** файл

***const { request } = require('https');***

***const PORT = 3000;*** // сделали минимальный рефакторинг - вынесли ***PORT*** в отдельную константу и обновили эти данные в методе .***listen*** (находится ниже)

// создаем сервер, используя встроенный метод .***createServer***. В качестве аргумента метод принимает ***callback***-функцию, которая будет вызываться каждый раз, когда к серверу идет какое-то обращение

***const server = http.createServer((req, res) => {*** // сама функция принимает 2 аргумента - это объекты запроса и ответа (отклика). ***request*** - хранит информацию, которую мы можем использовать в нашем сервере, а ***response*** - объект который мы будем формировать и отправлять в браузер

***console.log('Server request');*** // каждый раз когда к серверу идет обращение - выводим сообщение в консоль

***console.log(req.url, req.method);*** // выведет информацию о запрашиваемом пути и методе

***console.log(req);*** // увидим в консоли что из себя представляет весь объект ***request***

// Для передачи вспомогательной информации, с которой мы можем взаимодействовать в браузере, служит ***ХЭДЭР*** ответа. Чтобы в него записать какие-то данные используем метод .***setHeader***

***res.setHeader('Content-Type', 'text/plain');*** //передаем информацию о возвращаемом типе контента, т.е. обычный текст

***res.setHeader('Content-Type', 'text/html');*** //передаем информацию о возвращаемом типе контента, т.е. текст ***html***

***res.setHeader('Content-Type', 'application/json');*** ///передаем информацию о возвращаемом типе контента, т.е. формат ***application/json***

// Для возврата серверу что-то в ответ используем объект ***response***. Здесь есть ДВА основных метода. ***write*** - с его помощью записываем какой-то ответ. Использовать метод .***write*** можно сколько угодно раз, главное, чтобы это было ДО метода .***end***

//***end*** - сообщает о том, что все нужные данные были добавлены в ответ, который отправляется и контроль можно возвращать браузеру. (Ответ сформирован и готов для отправки в браузер)

***res.write('<head><link rel="stylesheet" href="#"></head>');*** // передаем подключение стилей в браузер

***res.write('<h1>Hello world!</h1>');***

***res.write('<p>Hy! My name is Oleh</p>');***

***const data = JSON.stringify([*** //создали небольшой массив, в котором 2 объекта с именем и возрастом, который "оборачиваем" в ***JSON.stringify***, т.к. по результату нам должен прийти ***JSON***-формат

***{ name: 'Tommy', age: 35 },***

***{ name: 'Arthur', age: 40 },***

***])***

// ***res.end();***

***res.end(data);*** //поскольку мы передаем данные, а не разметку, то созданный JSON передаем напрямую в метод .***end***

});

// указываем порт, который будет "слушать" сервер. По стандарту это порт 3000. Для прослушивания применяется метод .***listen***. Он принимает 3 аргумента: 1 - порт, 2 - имя хоста (***localhost*** - значение по умолчанию), 3 - ***callback***, внутри которого мы добавим консоль для вывода информации о том, что слушаем определенный порт, а также если произойдет ошибка - мы будем о ней знать

***server.listen(PORT, 'localhost', (error) => {***

***error ? console.log(error) : console.log('listening port ${PORT}');*** // ошибка ? "Да" : "Нет" // условие ? что делать если "Да" : что делать если "Нет"

***})***;

**Node.js #8 Создание сервера (Create Server) с помощью http-модуля**

***const http = require("http");*** // импортируем модуль ***http*** в наш ***js*** файл

***const { request } = require("https");***

***const PORT = 3000;*** // Сделали минимальный рефакторинг - вынесли PORT в отдельную константу и обновили эти данные в методе .***listen*** (находится ниже)

// создаем сервер, используя встроенный метод .***createServer***. В качестве аргумента метод принимает ***callback***-функцию, которая будет вызываться каждый раз, когда к серверу идет какое-то обращение

***const server = http.createServer((req, res) => {***

// сама функция принимает 2 аргумента - это объекты запроса и ответа (отклика). ***request*** - хранит информацию, которую мы можем использовать в нашем сервере (в будущем мы будем брать ***url***-адресс, который приходит в запросе и на основании него отправлять нужные данные или страницу), а ***response*** - объект который мы будем формировать и отправлять в браузер

***console.log("Server request");*** // каждый раз когда к серверу идет обращение - выводим сообщение в консоль

***console.log(req.url, req.method);*** // выведет информацию о запрашиваемом пути и методе

***// console.log(req);*** // увидим в консоли что из себя представляет весь объект ***request***

// Для передачи вспомогательной информации, с которой мы можем взаимодействовать в браузере, служит ХЭДЭР ответа. Чтобы в него записать какие-то данные используем метод .***setHeader***

***// res.setHeader('Content-Type', 'text/plain');*** //передаем информацию о возвращаемом типе контента, т.е. обычный текст

***// res.setHeader('Content-Type', 'text/html');*** //передаем информацию о возвращаемом типе контента, т.е. текст ***html***

***res.setHeader("Content-Type", "application/json");*** ///передаем информацию о возвращаемом типе контента, т.е. формат ***application/json***

// Для возврата серверу что-то в ответ используем объект ***response***. Здесь есть ДВА основных метода: ***.write*** - с его помощью записываем какой-то ответ. Использовать метод .***write*** можно сколько угодно раз, главное, чтобы это было ДО метода .***end***

// ***.end*** - сообщает о том, что все нужные данные были добавлены в ответ, который отправляется и контроль можно возвращать браузеру. (Ответ сформирован и готов для отправки в браузер)

***// res.write('<head><link rel="stylesheet" href="#"></head>');*** // передаем подключение стилей в браузер

***// res.write('<h1>Hello world!</h1>');***

***// res.write('<p>Hy! My name is Oleh</p>');***

***const data = JSON.stringify([***

//создали небольшой массив, в котором 2 объекта с именем и возрастом, который "оборачиваем" в ***JSON.stringify***, т.к. по результату нам должен прийти JSON-формат

***{ name: "Tommy", age: 35 },***

***{ name: "Arthur", age: 40 },***

***]);***

***// res.end();***

***res.end(data);*** //поскольку мы передаем данные, а не разметку, то созданный JSON передаем напрямую в метод ***.end***

});

// указываем порт, который будет "слушать" сервер. По стандарту это порт 3000. Для прослушивания применяется метод ***.listen***. Он принимает 3 аргумента: 1 - порт, 2 - имя хоста (***localhost*** - значение по умолчанию), 3 - ***callback***, внутри которого мы добавим консоль для вывода информации о том, что слушаем определенный порт, а также если произойдет ошибка - мы будем о ней знать

***server.listen(PORT, "localhost", (error) => {***

// указываем порт, который будет "слушать" наш сервер.

***error ? console.log(error) : console.log("listening port ${PORT}");*** // ошибка ? "Да" : "Нет" // условие ? что делать если "Да" : что делать если "Нет"

});

**Node.js #9 Создание базового роутинга (Create Base Routing)**

**09-0app.js**

// Серверный РОУТИНГ или маршрутизация - логика, которая позволяет пользователю передвигаться по сайту.

//Она может быть нативная (когда в ***html***-документ добавляем ссылку на внутреннюю страницу и при клике пользователь получает к ней доступ) и эмулированная (использование ***Singl Page Application*** - вместо стандартного перехода на новую страницу будет выполнен рендеринг нового компонента, но визуально это будет выглядеть именно как переход на новую страницу)

//Задача: создать серверный роутинг, при котором при переходе по ***url***-адресу будем возвращать новую разметку (т.е. одну из статичных страниц)

***const http = require("http");*** // импортируем ***http***-модуль в наш ***js***-файл

***const fs = require("fs");*** // подключаем модуль файловой системы для работы с файлами, которые мы будет возвращать в браузер

***const path = require("path");*** // подключаем модуль ***path*** для помощи в формировании корректного пути

***const PORT = 3000;***

***const server = http.createServer((req, res) => {***

***console.log("Server request");***

***res.setHeader("Content-Type", "text/html");*** //поскольку работать будем с разметкой, то Content-Type, определяем как html

// для примера: при обращении на корневой роут (обычный /) будем возвращать сраницу ***index.html***

// примерно это механизм мы будем использовать для нашего роутинга, т.е. определяем какой ***url***-адрес прилетает в запросе, и в зависимости от этого возвращаем нужную страницу

***if ((req.url = "/")) {***

// создаем условие в котором будем проверять адрес приходящего запроса

***fs.readFile("./views/index.html", (err, data) => {***

// читаем файл index.html и после прочтения внутри ***callback***-функции вернуть его в ответ

***if (err) {***

// кейс обработки ошибки: если что-то пойдет не так, мы получим в терминале сообщение об ошибке и остановим чтение файла

***console.log(err);***

***res.end();*** // записывает и в кейсе ошибки и ответа. ВСЕГДА нужно завершать ответ, чтобы вернуть контроль браузеру

***} else {***

// в случае успеха в ответ записываем прочитанную разметку и возвращаем ее браузеру

***res.write(data);***

***res.end();*** // записывает и в кейсе ошибки и ответа. ВСЕГДА нужно завершать ответ, чтобы вернуть контроль браузеру

***}***

***});*** // далее, используя файловую систему нам нужно прочитать файл index.html и после прочтения, внутри ***callback***-функции, вернуть его в ответ

***}***

***});***

***server.listen(PORT, "localhost", (error) => {***

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

***});***

**Node.js #9 Создание базового роутинга (Create Base Routing)**

**(продолжение)**

**09-1app.js**

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

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

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

***const PORT = 3000;***

***const server = http.createServer((req, res) => {***

***console.log("Server request");***

***res.setHeader("Content-Type", "text/html");*** //поскольку работать будем с разметкой, то ***Content-Type***, определяем как ***html***

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

// поскольку основная задача это построение пути до файла, её мы можем инкапсулировать. Для этого внутри функции воспользуемся ***path.resolve***. Первым аргументом передадим ***\_\_dirname*** (при определении пути до файла применяется явное определение с помощью этого глобального объекта, для получения пути до исполняемого скрипта), дальше указываем папку ***'views'***, и имя файла ***`${page}.html`***. На выходе функции получаем простую строку. Модуль ***Path*** кроссплатформенный, т.к. различает использование ***\*** и ***/*** и передает нужное значение

***let basePath = "";***

// создаем условие ***swith - case***, принимать оно будет ***url***-адрес запроса, а дальше в каждом ***case***, в переменную ***Path*** на основании ***url***-адреса, мы будем присваивать путь до страницы

***switch (req.url) {***

***case "/":***

***case "/home":*** // пример множественных путей для одной страницы

***case "/index.html":*** // пример множественных путей для одной страницы

***basePath = createPath("index");*** // вызывает страницу index

***res.statusCode = 200;*** // создаем успешный код возврата - обращаемся к объекту ответа и определяем его статус-код. Фактически это можно не делать, т.к. если сервер что-то возвращает, то статус-код присваивается автоматически

***break;***

// Создание ре-директа.

// добавим логику: если пользователь переходит на старый роут, он автоматически будет перенаправлен на актуальный. Для этого используется статус-код и метод .***setHeader***:

***case "/about-us":*** // Создали ***case***. предположим это старый адрес контактов

***res.statusCode = 301;*** // ставим статус-код = 301, для того, чтобы дать понять, что ре-директ контролируемый

***res.setHeader("Location", "/contacts");*** // устанавливаем хедер ответа, в котором определяем ***Location*** и присваиваем значение пути на который должден произойти ре-директ

***res.end();*** // поскольку в этом случае пути до файла мы не создаем, нам нужно отправить сформированный ответ с помощью ***res.end()***;

***break;***

***case "/contacts":***

***basePath = createPath("contacts");***

***res.statusCode = 200;***

***break;***

***default:***

***basePath = createPath("error");*** // вызывает страницу ошибки, т.е. если пользователь введет не существующий путь, то он автоматически будет отправлен на страницу ошибки

***res.statusCode = 404;*** // ввели информацию о клиентской ошибке, а именно - пользователь ввел некорректные данные и получил ошибку о несуществующей странице

***break;***

***}***

***fs.readFile(basePath, (err, data) => {***

***if (err) {***

// кейс обработки ошибки: если что-то пойдет не так, мы получим в терминале сообщение об ошибке и остановим чтение файла

***console.log(err);***

***res.statusCode = 500;*** // т.к. ошибка происходит при парсинге файла, следовательно это какая-то внутренняя серверная ошибка

***res.end();*** // записывает и в кейсе ошибки и ответа

***} else {***

// в случае успеха в ответ записываем прочитанную разметку и возвращаем ее браузеру

***res.write(data);***

***res.end();***

***}***

***});*** // далее, используя файловую систему нам нужно прочитать файл ***index.html*** и после прочтения, внутри ***callback***-функции, вернуть его в ответ

***});***

***server.listen(PORT, "localhost", (error) => {***

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

***});***