Лабораторная работа

Разработка на Node.js довольно увлекательный процесс. Он состоит из тех же деталей, что и обычная разработка, плюс один маленький нюанс: разработчику никто не навязывает, как он должен структурировать проект, все в его руках.

Простая программа на Node

Загрузите версию **9.4.0** с сайта https://nodejs.org. Установите, следуя инструкциям установщика. Можете выбрать каталог, например `C:\nodejs`. Создайте папку под свои проекты. **Сохраняйте каждый файл!**

Первый скрипт

Создайте в любом текстовом редакторе (кроме Блокнот, если вы выбрали его - не жалуйтесь на проблемы с кодировкой!) файл с именем `hello.js`. Напишите туда следующий текст:

```
let name = "John Doe"
console.log(`Привет, ${name}`)
```

Объясняю построчно:

1. Объявляем переменную `name`
2. И выводим в стандартный поток (log) строку. **ВНИМАНИЕ!** Это **НЕ кавычки**! Это символ backtick (находится на букве ё!) для быстрой интерполяции строки

И запустите его, используя командную строку: `node hello.js`

Простой HTTP сервер

Ну вот, с азами вы познакомились. Остальное изучим по ходу. Начнем с простого HTTP сервера. Пускай и звучит это довольно сложно, на деле ничего такого нет. Главное внимательно читать далее.

Так что такое HTTP сервер? Это сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. В нашем случае будут только HTML страницы. Воспользуемся встроенным в Node пакетом `http`. Создайте файл `http_server.js` и напишите текст программы:

```
const http = require('http')
let port = 5279 // напишите любой порт > 1024

let requestHandler = (request, response) => {
        console.log(request.url)
        response.writeHead(200, {
                'Content-Type': 'text/html; charset=utf-8'
        })
        response.write(`Я веб-сервер, работаю на порту ${port}!`)
        response.end()
}

const server = http.createServer(requestHandler)

server.listen(port, (err) => {
        if(err) {
                return console.log('Ой, произошло что-то ужасное! =(', err)
        }
        console.log(`Я простой веб-сервер, работаю на порту ${port}`)
})
```

Запустите, используя команду `node http_server.js` и перейдите по адресу http://127.0.0.1:5279 (не забудьте указать порт, который прописали)

Теперь по порядку:

- Строка 1 добавляем модуль `http` из стандартной библиотеки Node
- Строка 2 задает порт для нашего сервера. Рекомендую установить его самостоятельно, и не забудьте, что нужно использовать порт не менее 1024
- Строки 4-11 задают функцию, которая будет отслеживать запросы к веб-серверу. `response` - все, что связано с ответом сервера а `request` содержит необходимые функции для работы с запросом. `response.writeHead` записывает **заголовки**, которые мы передаем веб-браузеру. В нашем случае мы говорим, что хотим отобразить HTML документ с кодировкой utf-8. `response.write` пишет в текущий поток сервера необходимые данные. Попробуйте написать еще HTML код таким образом. `response.end` завершает отправку запроса.
- Строка 13 создает веб-сервер с нашей функцией
- Строки 15-21 создают слушатель (listener) для веб-сервера. Мы создаем веб-сервер с портом `port` и вторым аргументом передаем inline-функцию, которая, как и в `requestHandler`, выполняет определенные действия. Единственный ее аргумент - переменная, содержащая сообщение об ошибке. Если она возникла, мы вернем результат и выйдем из потока. Если ошибки нет - работаем дальше.

HTTP с батарейками: Express.js

Несомненно, подход, описанный выше самый быстрый в плане потребления памяти. Он не обрабатывает лишних данных и не создает дополнительных расширений для приложения.

Конечно, его можно использовать для разработки боевых проектов. Правда, если вы собрались так сделать, прошу обратиться к психиатру.

В современном мире такой подход неприемлим: он трудоемок и часто допускает повторения. К счастью, у нас есть Express!

Express дополняет HTTP и делает его более более доступным для разработчика. Давайте определимся с некоторыми терминами:

1. *Middleware* (или *промежуточное ПО*) - слой или комплекс технологического программного обеспечения для обеспечения взаимодействия между различными приложениями, системами, компонентами. В случае с Express - middleware выполняет функцию посредника между библиотекой и приложением, дающая доступ к основному слою приложения и HTTP. Хороший пример middleware - подключение системы визуализации Handlebars(см. ниже)
2. *Система визуализации* (или *template engine*) - программное обеспечение, позволяющее использовать html-шаблоны для генерации конечных html-страниц. Основная цель использования шаблонизаторов — это отделение представления данных от исполняемого кода. Примеры в Node.js: Pug, Jade, Handlebars, Hogan.js, EJS.

Создание проекта

Для проекта нам потребуется отдельная директория. Например, создайте директорию `start-express` и переключитесь на нее в командной строке. Далее выполните команду `npm init` и жмите enter до тех пор, пока снова не увидите приглашение командной строки. В директории `start-express` будет создан файл `package.json` примерно следующего содержания (если сильно хочется, можете создать его вручную):

```
{
  "name": "start-express",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Далее работаем именно в **этой** директории. Выполните команду:

```
npm i -S express hbs morgan winston
```

- Express - это фреймворк, на котором мы и будем писать приложение
- Hbs - это сокращение от Handlebars - системы визуализации
- Morgan, Winston - библиотеки для логирования нашего приложения

Простое веб-приложение

Создайте файл `index.js` и напишите в нем следущий код:

```
const express = require('express')
const app = express()

let port = 8787

app.get('/', (req, res) => {
        res.send('Добро пожаловать в Express!')
})

app.listen(port, () => console.log(`Работает! Порт ${port}`))
```

Как видите, мы сократили код веб-сервера. На строке 1 мы подключаем express. Строка 2 создает экземпляр express. Затем создаем переменную, в которую поместим порт сервера.

А дальше - особая магия Express! У HTTP 1.1, как вы знаете, есть 4 глагола запроса: GET, POST, PUT, DELETE. Так вот, Express умеет понимать их простым написанием "пути" (route). Например:

```
// POST
app.post(...)
// PUT
app.put(...)
// DELETE
app.delete(...)
```

Таким образом, у разработчика экономится куча времени и сил.

Первый аргумент в `app.get(...)` - путь (route). Например `app.get('/hi')` будет отображать что-то по адресу http://127.0.0.1:8787/hi

После объявления путей мы запускаем веб-сервер. Этот запуск абсолютно идентичен прошлому примеру. Только теперь мы используем экземпляр Express.

Handlebars

Давайте добавим нашему приложению немного удобства: добавим систему визуализации или шаблонизатор. После `let port = ...` добавьте следующие строки:

```
app.set('view engine', 'hbs')
app.set('views', __dirname + '/views')
app.set('view options', { layout: 'layout' })
```

- 1 строка - добавляем middleware функцией `set`
- 2 строка - указываем, что все представления (знакомое слово?) находятся в папке `./views`
  - ./ - это текущая директория
  - `__dirname` - магическая переменная Node, указывающая на текущую директорию
- 3 строка - укажем, какой из файлов использовать как общий макет (layout)

Теперь создадим представления. В папке проекта создаем папку `views` и там 2 файла:

1. `layout.hbs` - макет
2. `index.hbs` - представление для главной страницы

Наполним содержимым файл `layout.hbs`:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>App</title>
	</head>
	<body>
		{{{body}}}
	</body>
</html>
```

В Handlebars переменные заключаются в фигурные скобки: `{{ var_name }}`. В нашем случае Handlebars передает переменную `body` в наш шаблон. Но так как по умолчанию HTML код превращается в мнемоники, мы оборачиваем `body` в тройные фигурные скобки.

Файл `index.hbs` будет просто здороваться с пользователем:

```
<h1>Привет, {{username}}!</h1>
```

А так же изменим `app.get('/' ...)`:

```
app.get('/', (req, res) => {
	res.render('index', {
		username: req.query.username === undefined ? 'Никто' : req.query.username
	})
})
```

Как вы, наверное, догадались - мы заменили `send` на `render`. `render` ищет указанный шаблон и вставляет его в общий, все переменные и другие управляющие конструкции переводит в HTML результат. Второй аргумент функции - список переменных, которые будут доступны в шаблоне типа `{ ключ: значение }`. Доступ осуществляется по ключу.

Теперь, если зайти на http://127.0.0.1:8787/?username=jon мы получим "Привет, jon!", а если просто на http://127.0.0.1:8787 то получим ответ "Привет, Никто!".

Задание: создайте еще 3 страницы и наполните их содержимым: "О сайте (путь /about)", "Обратная связь (путь /feedback)" и "Новости (путь /news)". Второй аргумент у функции `render` можно не писать. Будьте внимательны!

Покажите результаты вашей работы преподавателю.