Урок 7

JavaScript на сервере

Создание простого сервера с помощью платформы Node.js.

Изначально JavaScript был простым языком для написания небольших скриптов. По задумке авторов, он должен был подойти тем, кто никогда ранее не занимался программированием. Простота и низкий порог входа сделали JavaScript очень популярным. Со временем появилась идея использовать его не только для работы в браузере. Сейчас на JavaScript можно писать код, работающий на сервере, создавать мобильные приложения и даже программировать роботов.

Для написания серверного JavaScript чаще всего используется платформа Node.js. Она появилась в 2009 году, и в её основе лежит очень быстрый движок V8 от Google. Ключевые преимущества Node.js – высокая скорость работы, простота и огромное комьюнити разработчиков.

Установка и запуск

Чтобы установить Node.js, нужно скачать дистрибутив с официального сайта https://nodejs.org/. На главной странице доступны две версии – Current и LTS. Current содержит все последние изменения и самые новые функции, но может быть нестабильна. LTS, или Long Term Support, – стабильная версия, которая подходит для решения большинства задач. Если вы устанавливаете Node.js не для экспериментов с самым свежим функционалом, то вам подойдёт LTS.

Node.js устанавливается как обычная программа, после чего в терминале становится доступна команда **node**.

```
$ node
>
```

После запуска вы попадаете в рабочую среду, где можно работать с JavaScript так же, как в консоли браузера:

```
$ node
> var a = 5;
> console.log(a);
5
```

Чтобы выйти, нужно дважды нажать **Ctrl+C**.

Можно написать код в отдельном файле и запустить его с помощью Node.js. Создадим файл **script.js** и напишем что-нибудь простое:

```
//script.js

var a = 5;
console.log(a);
```

Теперь запустим его с помощью команды **node**. Имя файла указывается через пробел:

```
$ node script.js
5
```

NPM

Вместе с Node.js в систему устанавливается Node Package Manager, или NPM. С помощью NPM можно устанавливать сторонние пакеты – библиотеки, фреймворки и компоненты, написанные другими разработчиками.

Чтобы установить пакет, нужно указать его имя после команды **npm install**:

```
$ npm install moment
```

Пакет устанавливается в папку **node_modules**, которая создаётся в той папке, откуда была вызвана команда **npm install**. Пакет не будет доступен за пределами папки.

Можно установить пакет так, чтобы он был доступен отовсюду. Для этого пакет устанавливается глобально с помощью флага **-g**:

```
$ npm i gulp -g
```

Список установленных пакетов удобно хранить в одном файле. Он называется **package.json**. Его можно создать вручную, но лучше и проще воспользоваться командой **init**:

```
$ npm init
```

После ввода команды npm задаст несколько вопросов и на основе ответов создаст **package.json**:

```
{
  "name": "geekbrains",
  "version": "1.0.0",
  "description": "",
  "author": "",
  "license": "ISC"
}
```

Чтобы при установке пакета сразу добавить его в **package.json**, установите пакет с флагом **--save**:

```
$ npm install moment --save
```

Название и версия пакета добавляются в поле **dependencies**:

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

```
  "license": "ISC",
  "dependencies": {
    "moment": "^2.22.2"
  }
}
```

Если в папке уже есть **package.json** и вы хотите установить все указанные в нём пакеты, то запустите команду **install** без указания имени пакета:

```
$ npm install
```

Установленный пакет можно подключить в файле скрипта **Node.js** через **require**:

```
// script.js
const moment = require('moment');
```

Теперь в переменной **moment** будет доступна библиотека **Moment.js**.

Список всех существующих пакетов можно посмотреть на официальном сайте https://www.npmjs.com/.

node-static

С помощью Node.js можно поднять простой сервер, который отдаёт статичные данные. Для этого воспользуемся пакетом **node-static**. Установим его локально:

```
$ npm install node-static
```

Создадим файл **server.js** и напишем в нём такой код:

```
const http = require('http');
const static = require('node-static');

const file = new static.Server('.');

http.createServer((req, res) => {
  file.serve(req, res);
}).listen(3000);
```

То есть мы создаём на 3000 порту http-сервер, все запросы которого будут обрабатываться модулем **node-static**. В качестве аргумента **static.Server()** передаём папку, из которой сервер будет брать файлы. Запись **'.'** означает папку, в которой лежит сам файл **server.js**.

Создадим файл **index.html** и пропишем в нём какую-нибудь простенькую разметку:

```
//index.html
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>I am static HTML page</h1>
</body>
</html>
```

Сервер работает локально на 3000 порту. Если мы перейдём в браузере по адресу **localhost:3000**, то увидим страницу **index.html**.

Работа с файлами

Для работы с файлами в Node.js есть встроенный модуль **fs**. Он уже входит в состав платформы, и его не нужно устанавливать отдельно. Создадим файл **script.js** и подключим в нём модуль **fs**:

```
//script.js
const fs = require('fs');
```

Теперь создадим файл, который будет читать сервер. Мы будем работать с JSON-файлом. Назовём его **data.json** и добавим в него немного данных:

```
{
  "first": "one",
  "second": 2
}
```

Теперь прочитаем этот файл и выведем результат в консоль. За чтение файлов отвечает метод **readFile()**. Он принимает несколько аргументов: путь к файлу, кодировка, колбэк-функция. Путь к файлу укажем относительно файла, в котором вызывается метод, т. е. **script.js**. Он и **data.json** лежат в одной папке, поэтому путь будет **./data.json**. В качестве кодировки укажем **UTF-8**. Если этого не сделать, **fs** не поймёт, что перед ним текст. Колбэк-функция нужна потому, что чтение файла – асинхронная операция, платформе нужно время, чтобы найти файл, обратиться к нему и считать содержимое. Посмотрим, что получилось:

```
const fs = require('fs');

fs.readFile('./data.json', 'utf-8', (err, data) => {

});
```

Колбэк-функция возвращает два аргумента: информация об ошибке и содержимое файла. Если ошибки нет, то в первом аргументе придёт null. Теперь считанное содержимое файла можно превратить в объект JavaScript:

```
const fs = require('fs');

fs.readFile('./data.json', 'utf-8', (err, data) => {
```

```
  if (!err) {
    const obj = JSON.parse(data);
  }
});
```

Теперь попробуем изменить объект и переписать файл. Для записи данных в файл используется метод **writeFile()**. Его параметры: путь к файлу, строка с данными и колбэк-функция. Кодировка по умолчанию – **UTF-8**, дополнительно устанавливать её не нужно:

```
const fs = require('fs');

fs.readFile('./data.json', 'utf-8', (err, data) => {
  const obj = JSON.parse(data);
  obj.third = 'THREE';

  fs.writeFile('./data.json', JSON.stringify(obj), (err) => {

  })
});
```

Если теперь открыть файл **data.json**, то можно увидеть, что в нём появилось новое поле:

```
{"first":"one","second":2,"third":"THREE"}
```

express

Express – фреймворк для создания веб-приложений на Node.js. С его помощью можно создать полноценный сервер, умеющий принимать запросы, обрабатывать их, обращаться к базе данных и т. д. Express – довольно большой фреймворк, и охватить все его особенности в рамках этого курса невозможно. Поэтому рассмотрим только самые базовые функции.

Установить express можно через NPM:

```
$ npm install express
```

Создадим файл **server.js** и напишем в нём следующий код:

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

app.listen(3000, () => {
  console.log('server is running on port 3000!');
});
```

Здесь мы сначала подключаем модуль **express**. Запись **const app = express();** записывает в переменную **app** объект, который содержит основные методы **express**. Метод **listen** принимает на вход номер порта, на котором создаётся сервер, и колбэк, который срабатывает после его запуска. Запустим сервер:

```
$ node server.js
```

В консоли появилось сообщение **«server is running on port 3000!»**. Теперь мы можем зайти на сайт через браузер, набрав в адресной строке **localhost:3000**. Пока что браузеру нечего отобразить, поэтому создадим на сервере html-страницу и научим сервер её отдавать. Создадим файл **index.html** в той же папке что и **server.js**.

```
//index.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>I am HTML page from Express</h1>
</body>
</html>
```

Теперь объясним серверу, что нужно отдавать статичные файлы, лежащие в определённой папке, – примерно так же, как в случае с **node-static**. За это отвечает метод **express.static()**, который в качестве параметра принимает путь к папке со статичными файлами. В нашем случае это текущая папка:

```
app.use(express.static('.'));
```

Теперь ко всем файлам можно обратиться через браузер по адресу **localhost:3000/имя_файла**. Если не указать имя файла, откроется **index.html**.

Далее научим сервер обрабатывать http-запросы. Для GET-запросов используется метод **app.get()**, для POST-запросов – **app.post()** и т. д. Эти методы принимают два параметра: роут, на который приходит запрос, и колбэк.

```
app.get('/data', (req, res) => {
  console.log('data');
});
```

Эта запись означает, что когда кто-нибудь обратится к серверу по **адрес_сервера/data**, сработает обработчик, и в консоли сервера появится текст «data».

Аргументы колбэк-функции **req** и **res** – это специальные объекты, в которых хранится информация о запросе (request) и ответе (response). Из объекта **req** можно получить, например, заголовки, тело POST-запроса и т. д. С помощью **res** можно, например, отправить ответ. Для этого используется метод **send()**:

```
app.get('/data', (req, res) => {
  res.send('data');
});
```

Теперь, когда браузер пришлёт запрос на **/data**, в ответ ему вернётся строка 'data'.

Практика

Создадим настоящее API для нашего интернет-магазина. Хранить информацию будем в двух JSON-файлах: **catalog.json** – список товаров, **cart.json** – корзина. Поскольку API теперь будет реальное, нужно разделить типы запросов на GET и POST. GET будем использовать для получения списка товаров и содержимого корзины, POST – для помещения товара в корзину и удаления товара из корзины. Прежде всего добавим в приложение метод **makePOSTRequest**:

```
...
methods: {
  makeGETRequest(url, callback) {
    ...
  },
  makePOSTRequest(url, data, callback) {
    let xhr;

    if (window.XMLHttpRequest) {
      xhr = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
      xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }

    xhr.onreadystatechange = function () {
      if (xhr.readyState === 4) {
        callback(xhr.responseText);
      }
    }

    xhr.open('POST', url, true);
    xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8');

    xhr.send(data);
  }
},
...
```

Теперь создадим для нашего проекта файл **server.js**. Это будет обычное приложение на **express**, аналогичное тому, что мы рассмотрели в прошлой главе:

```
const express = require('express');
const fs = require('fs');

const app = express();

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

app.listen(3000, function() {
  console.log('server is running on port 3000!');
});
```

Теперь добавим два JSON-файла. **catalog.json** сразу заполним какими-нибудь товарами, а в **cart.js** запишем пустой массив:

```
// catalog.json
[
  { "product_name": "Мышка", "price": 100 },
  { "product_name": "Клавиатура", "price": 200 },
  { "product_name": "Колонки", "price": 130 },
  { "product_name": "Наушники", "price": 150 },
  { "product_name": "Ноутбук", "price": 400 },
  { "product_name": "Коврик", "price": 150 },
  { "product_name": "Кресло", "price": 160 }
]
```

```
// cart.json
[]
```

Теперь научим наш сервер слушать GET-запросы на **/catalogData**. Если такой запрос пришёл, то читаем файл **catalog.json** и отправляем обратно содержимое:

```
const express = require('express');
const fs = require('fs');

const app = express();

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

app.get('/catalogData', (req, res) => {
  fs.readFile('catalog.json', 'utf8', (err, data) => {
    res.send(data);
  });
});

app.listen(3000, function() {
  console.log('server is running on port 3000!');
});
```

И изменим адрес запроса в методе **mounted()** нашего приложения:

```
...
mounted() {
  this.makeGETRequest(`/catalogData`, (goods) => {
    this.goods = JSON.parse(goods);
    this.filteredGoods = JSON.parse(goods);
  });
}
...
```

Обратите внимание, что не нужно указывать полный адрес сервера с API, поскольку и файл, из которого посылается запрос, и API находятся на одном сервере.

Добавим в API обработку добавления товара в корзину. Сперва получим содержимое корзины:

```
app.post('/addToCart', (req, res) => {
  fs.readFile('cart.json', 'utf8', (err, data) => {
    const cart = JSON.parse(data);
  });
});
```

Теперь нужно научить сервер смотреть содержимое POST-запроса. Для этого нужно подключить встроенный в Node.js модуль **body-parser**:

```
const express = require('express');
const bodyParser = require('body-parser');
const fs = require('fs');

const app = express();

app.use(bodyParser.json()); // Указываем, что содержимое - JSON
...
```

Готово. Теперь тело POST-запроса можно получить в **req.body**:

```
app.post('/addToCart', (req, res) => {
  fs.readFile('cart.json', 'utf8', (err, data) => {
    const cart = JSON.parse(data);
    const item = req.body;
  });
});
```

Теперь добавляем файл в корзину и добавляем в файл следующий фрагмент:

```
app.post('/addToCart', (req, res) => {
  fs.readFile('cart.json', 'utf8', (err, data) => {
    const cart = JSON.parse(data);
    const item = req.body;

    cart.push(item);

    fs.writeFile('cart.json', JSON.stringify(cart), (err) => {
      console.log('done');
    });
  });
});
```

Поскольку браузер будет ждать ответа от сервера, чтобы завершить запрос, будем посылать объект с полем **result**. В этом поле будем записывать 1, если всё прошло без ошибок, и 0, если были ошибки и

записать не удалось:

```
app.post('/addToCart', (req, res) => {
  fs.readFile('cart.json', 'utf8', (err, data) => {
    if (err) {
      res.send('{"result": 0}');
    } else {
      const cart = JSON.parse(data);
      const item = req.body;

      cart.push(item);

      fs.writeFile('cart.json', JSON.stringify(cart), (err) => {
        if (err) {
          res.send('{"result": 0}');
        } else {
          res.send('{"result": 1}');
        }
      });
    }
  });
});
```

Практическое задание

1. Привязать добавление товара в корзину к реальному API.
2. Добавить API для удаления товара из корзины.
3. * Добавить файл **stats.json**, в котором будет храниться статистика действий пользователя с корзиной. В файле должны быть поля с названием действия (добавлено/удалено), названием товара, с которым производилось действие и временем, когда оно было совершено.

Дополнительные материалы

1. Видеокурс по оскак новам Node.js.
2. Руководства по Node.js.

Используемая литература

1. Официальная документация Node.js.