фаза 2 · неделя 1 · день 1

HTTP, Express.js

ELBRUS/BOOTCAMP

План

1. Протокол HTTP
   1.1. HTTP в составе TCP/IP
   1.2. HTTP запрос и его структура
   1.3. HTTP ответ и его структура
2. Express.js
   2.1. Установка и подключение
   2.2. Базовые методы
   2.3. Обработчики запросов
   2.4. Виды ответов

HTTP

- HyperText Transfer Protocol, HTTP — протокол передачи гипертекста
- Изначально применялся для передачи HTML, на текущий момент стал более функциональным в применении

HTTP: версия 1.1

Версии протокола:

- 0.9, 1.0, 1.1 — текстовые протоколы
- HTTP/2 и HTTP/3 (готовятся к выходу) — бинарные

Мы будем изучать версию 1.1, вся логика будет строиться на передаче текста

HTTP в TCP/IP, application layer

TCP/IP (Transmission Control Protocol/Internet Protocol) это сетевая модель состоящая из уровней:

- Прикладной уровень (Application Layer)
  - интернет браузер для протокола HTTP/HTTPS (порты: 80/443)
  - FTP-клиент для протокола FTP (порты: 20/21)
  - почтовая программа для протокола SMTP (порт: 25)
  - SSH для безопасного соединения с удалённой машиной (порт: 22)
- Транспортный уровень (Transport Layer)
- Межсетевой уровень (Internet Layer),
- Канальный уровень (Network Access Layer)

HTTP: URI, URL, URN

`HTTP-ресурс` (веб-ресурс) — любой ресурс, доступный в WWW

`URI` — унифицированный идентификатор ресурса

`URL` — унифицированный определитель местонахождения ресурса

`URN` — унифицированное имя ресурса

Каждый URL - это URI, но не наоборот.
Любой URI, в котором указан протокол - это URL (http, ftp, mailto и др.)

HTTP: URI, URL, URN. Пример

Имя и адрес ресурса в сети, включает в себя URL и URN

URI: `https://somesite.ru/images/logo.png`

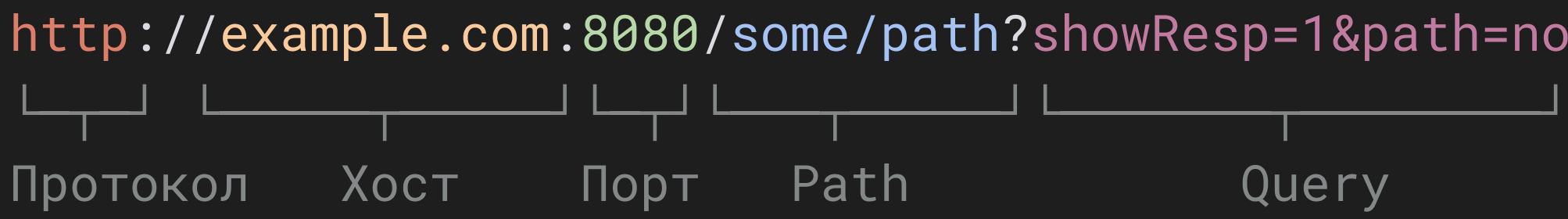
Адрес ресурса в сети, определяет местонахождение и способ обращения к нему

URL — `https://somesite.ru`

Имя ресурса в сети, определяет только название ресурса

URN — `images/logo.png`

HTTP: анатомия URL



HTTP: запрос

HTTP-запрос (request)

Сообщение, отправленное клиентом на сервер.

Состоит из:

1. стартовой строки
2. HTTP заголовков
3. тела запроса (может быть пустым)

HTTP: запрос. Пример

```
GET /students HTTP/1.1     — стартовая строка
Host: school.example       ┐
Accept-Language: ru-ru     ├ HTTP заголовки
User-Agent: Mozilla/5.0    ┘
(пустая строка)
(тело запроса — пустое или с данными)
```

HTTP: ответ

HTTP-ответ (response)

Сообщение, отправленное сервером в ответ на запрос клиента.

Состоит из:

1. стартовой строки / строка статуса
2. HTTP заголовков
3. тела ответа (может быть пустым)

HTTP: ответ. Пример

```
HTTP/1.1 200 OK  — строка статуса (стартовая строка)
X-Powered-By: Express                ┐
Content-Type: application/json       ┴ HTTP заголовки
(пустая строка)
{"students":[{"id":1,"name":"Пётр — тело ответа
Петров","phone":"70001112233"}]}
```

HTTP: коды ответа

Коды ответа поделены на группы по смыслу:

1** информационные коды (например, смена протокола с HTTP на WS)

2** всё хорошо, запрос выполнен успешно

3** переадресация, сервер просит перейти на другую страницу

4** ошибка со стороны клиента (запрос некорректного адреса и т. д.)

5** ошибка со стороны сервера (сервер в ошибке, ошибка в БД и т. д.)

HTTP: типы передаваемого контента

Типы контента или иначе MIME-типы, их очень много, вот часть популярных:

- `text/plain` — простой текст
- `text/html` — HTML-документ
- `application/json` — данные в JSON-формате
- `application/xml` — XML-документ
- `multipart/form-data` — бинарные данные

Express

- Фреймворк web-приложений для Node.js
- Де-факто является стандартным каркасом для написания серверного кода или API

Express: установка и подключение

Как и любой другой npm пакет `Express` устанавливается через терминал с помощью команды: `npm i express`

После установки появляется возможность подключить модуль через импорт:

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

Express: базовые переменные

```
// переменная app формируется в сущность с методами для формирования серверного приложения
const app = express();

// переменная PORT хранит значение порта, который является частью URL, часто пишут в верхнем регистре, как отсылка к переменным окружения (регистр не влияет на функционал)
const PORT = 3000;
```

Express: базовые методы

```
// метод listen() запускает работу сервера с прослушиванием заданного порта
// когда слушатель фиксирует запрос, сервер начинает проверку наличия обработчика
app.listen(PORT, () => console.log('Server started'));
```

Express: обработчики запросов

```
app.get('/', (req, res) => {
  // тело обработчика в котором должен формироваться ответ
  // параметры req (request) и res (response) отвечают за
запрос и ответ
  // благодаря им формируется клиент-серверное взаимодействие
  // золотое правило для обработчика 1 запрос = 1 ответ
});
```

Express: обработчики запросов

```
// обработчик GET-запроса, который формирует ответ на клиент
app.get('/', (req, res) => {
  // формирование ответа в виде строки
  res.send('Hello, world');

  // ещё одно формирование ответа в виде строки (так делать нельзя)
  // приводит ошибке сервера: "Cannot set headers after they are sent to the client"
  res.send('More!');
});
```

Express: виды ответов

```
res.send(text)                  // послать текст с кодом 200 + завершить ответ
res.sendFile(path)              // послать файл с кодом 200 + завершить ответ
res.json({ user: 'Anna' })      // послать json с кодом 200 + завершить ответ
res.end()                       // завершить ответ
res.status(403)                 // установить статус код, но НЕ завершить ответ

// установить статус код 500, послать json + завершить ответ
res.status(500).json({ error: 'message' })

res.status(404).end()           // установить статус код + завершить ответ
res.redirect('/other-route')    // переадресовать клиента + завершить ответ
res.write(data)                 // отправка части ответа, но НЕ завершает ответ
```

Express: применение

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

const app = express();
const PORT = 3000;

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

app.listen(PORT, () => console.log('Server started'));
```