Лекция 5

Node.js

Поиск сервера

Работу сайта обеспечивает сервер. У сервера есть адрес - **IP-адрес.**

Откуда браузеру взять IP-адрес сервера?

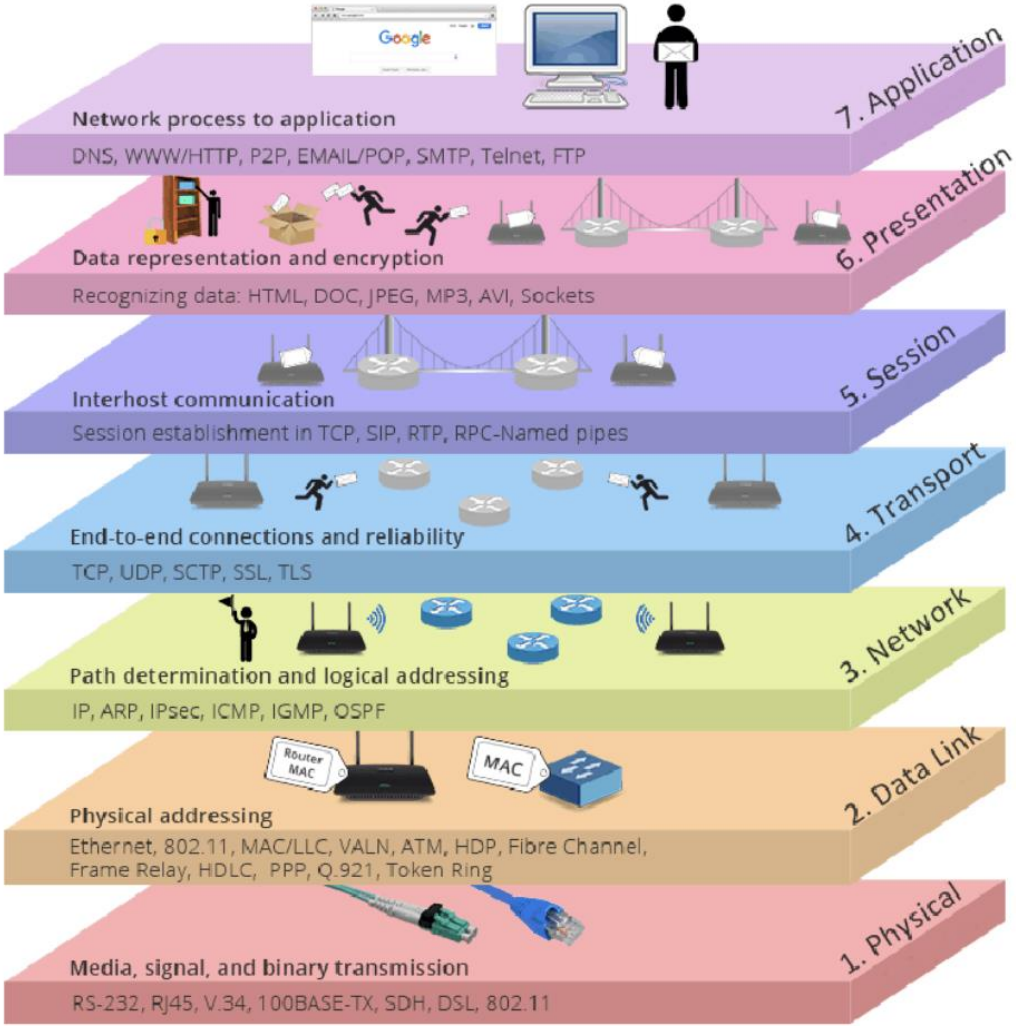
Такая информация хранится в распределенной системе серверов — **DNS** (**Domain Name System**). Система работает как общая «контактная книга», хранящаяся на распределенных серверах и устройствах в интернете.

Подробнее тут https://vc.ru/selectel/76371-chto-proishodit-kogda-polzovatel-nabiraet-v-brauzere-adres-sayta

TCP соединение

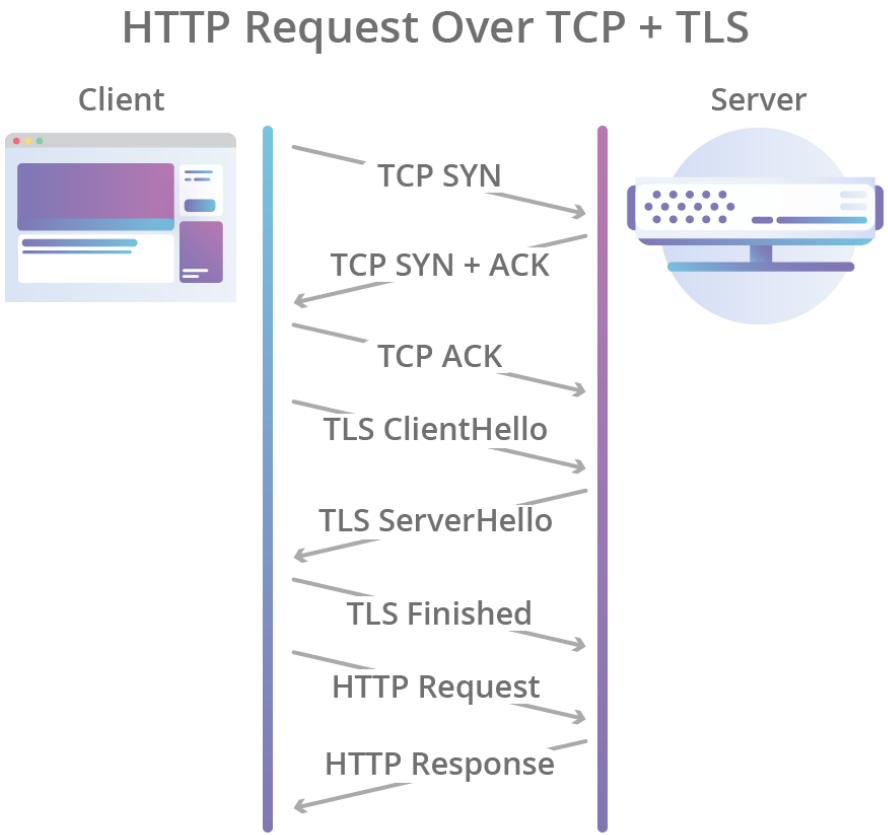
Как только браузер узнал IP-адрес нужного сервера, он пытается установить с ним соединение. В большинстве случаев для этого используется специальный протокол — TCP.

Для установления TCP соединения используется система трех рукопожатий.



Тройное рукопожатие

- Устройство пользователя отправляет специальный запрос на установку соединения с сервером — называется SYN-пакет.
- Сервер в ответ отправляет запрос с подтверждением получения SYN-пакета — называется SYN/ACK-пакет.
- В конце устройство пользователя при получении SYN/ACK-пакета отправляет пакет с подтверждением — ACK-пакет. В этот момент соединение считается установленным.

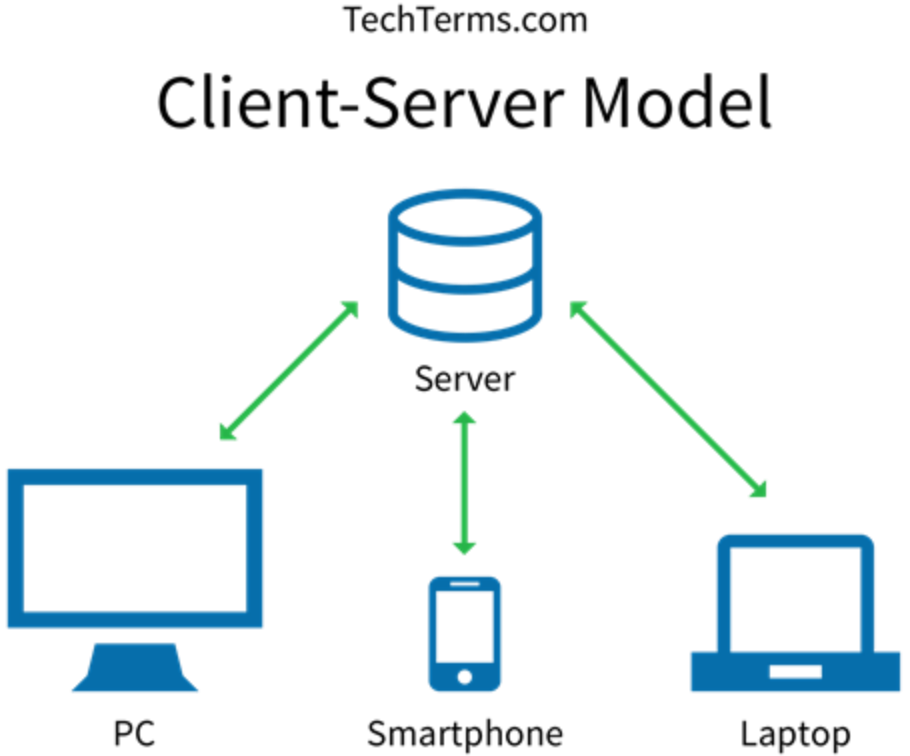


Остальные шаги

1. Браузер отправляет HTTP-запрос по установленному соединению;
2. Сервер получает запрос и обрабатывает его (бизнес-логика, рендер, проксирование и тд);
3. Сервер отправляет ответ браузера. Это может быть любой (!) ресурс;
4. Браузер обрабатывает полученный ответ и «рисует» веб-страницу;

Клиент-сервер

Клиент – серверная модель — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами



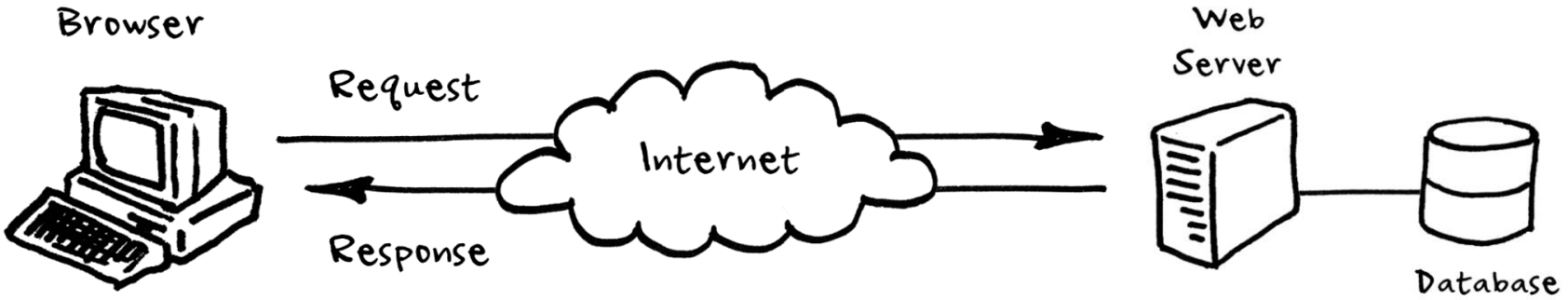
Подробнее тут: https://habr.com/ru/post/495698/

Статические веб-страницы

- Frontend
  a. Отображение статических HTML-страниц (HTML, CSS);
  b. Контент на страницах, переходы по гиперссылкам
- Backend
  a. Хранение статических документов и отдача по запросу по протоколу HTTP

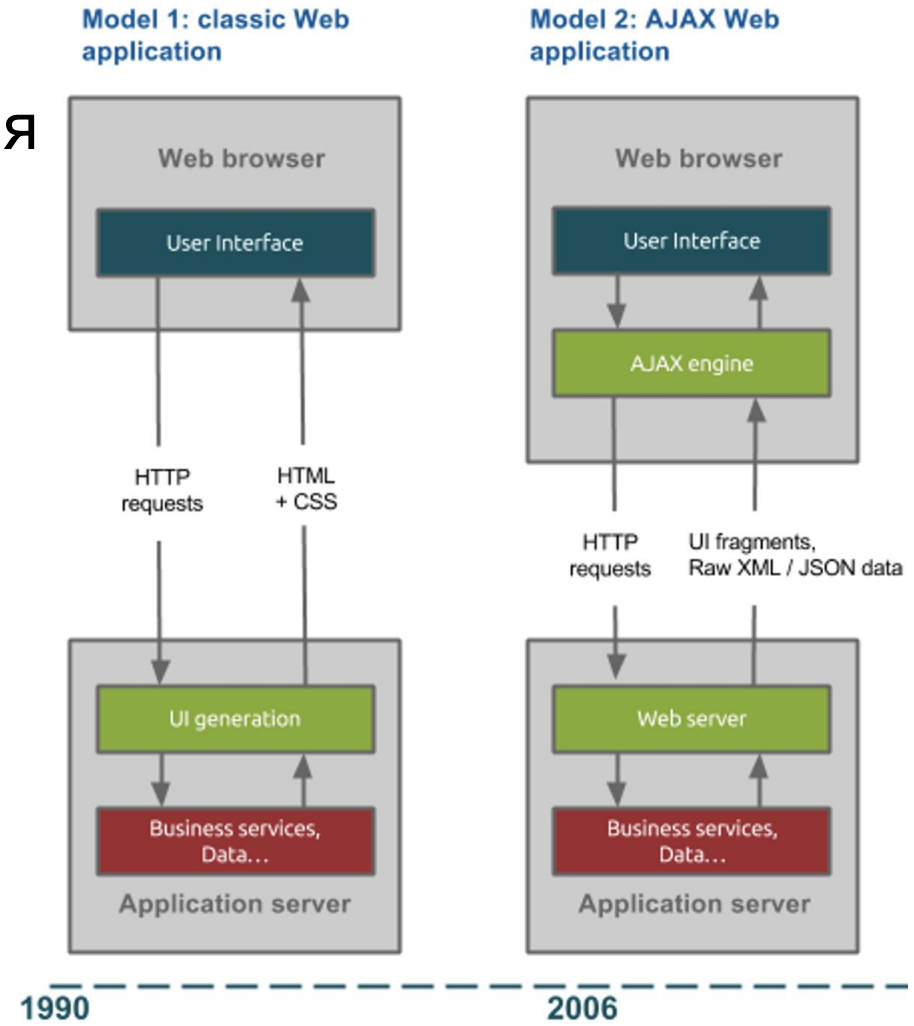
SSR (server side rendering) – динамические документы

- Frontend
  a. Отображение статических HTML-страниц (HTML, CSS)
  b. Контент на страницах, переходы по гиперссылкам
  c. Взаимодействие с сервисом посредством форм
- Backend
  a. Хранение статических документов и отдача по запросу по протоколу HTTP
  b. Обработка пользовательских запросов и генерация динамических страниц
  c. Хранение данных в базе данных



Веб-сервисы, веб-приложения

- Frontend
  a. Хранение и доступ к статическому контенту (файлы стилей, скрипты)
  b. Генерация и отображение пользовательского интерфейса
  c. Взаимодействие с пользователем и выполнение запросов к API
  d. Обновление пользовательского интерфейса в ответ на действия пользователя
- Backend
  a. Реализация публичного API
  b. Хранение данных в базе данных и работа с ними



REST – Передача состояния

Мы должны с каждым запросом передать всю информацию (состояние) для сервера, чтобы он мог выполнить наше действие

Нам все равно что было раньше, каждый раз мы должны передать всю информацию

Что такое Node.js?

Node.js - среда выполнения javascript основанная на браузерном движке Google V8.

Google

- веб-приложения (бэкенд);
- десктопные приложения (Electron.js);
- микроконтроллеры (Espurino), ссылка;
- различные консольные утилиты;
- машинное обучение (TensorFlow.js), ссылка;

Преимущества

Node.js - это платформа для разработки очень быстрого сервера. По производительности спокойно может сравниться с Java, Kotlin и Golang.

При этом все вышеперечисленные языки имею вложенные механизмы параллелизма (корутины, потоки и тд), а Node.js однопоточный.

Движок V8 из коробки поддерживает исключительно асинхронное API, т.к. изначально был предназначен для работы в браузере на клиенте.
Это значит, что ни одна из read/write операций не блокирует основной поток исполнения (спасибо Event Loop).

Поднять простой веб-сервер на Node.js - это 15 строк кода (с натяжкой).

При этом, т.к. мы пишем на javascript, мы можем использовать различные парадигмы для написания кода.

Отсутствие типизации

Javascript - нестрого типизированный язык.

Риск написать плохой код на JavaScript настолько высок, что сравнить его можно, наверно, только с Python. Обилие поддерживаемых парадигм, нестрогая типизация, сложная асинхронность, лексическое окружение, замыкания.

```
1 true + false
2 12 / "6"
3 "number" + 15 + 3
4 15 + 3 + "number"
5 [1] > null
6 "foo" + + "bar"
7 'true' == true
8 false == 'false'
9 null == ''
10 !!"false" == !!"true"
11 [‘x’] == ‘x’
12 [] + null + 1
13 0 || "0" && {}
14 [1,2,3] == [1,2,3]
15 {}+[]+{}+[1]
16 !+[]+[]+![]
17 new Date(0) - 0
18 new Date(0) + 0
19
```

Другие проблемы разработки на JS

- отсутствие многопоточки;
- медленные вычисления;
- плохое ООП, несмотря на его наличие;
- скудные коллекции (по сравнению с java, но великолепные по-сравнению с Go)

RunTime-ошибки - код запустился, но упал во время исполнения.

Такая возможность есть из-за того, что javascript интерпретируемый язык программирования.

Как использовать Node.js

1. Создаем файл <название>.js
2. Пишем туда код на javascript
3. Запускаем командой `node <название>.js`
4. Обращаемся по HTTP

```
node server.js
```

```
Сервер запущен по адресу http://localhost:8000
```

```
curl http://localhost:8000
```

```
JavaScript - классный язык программирования!
```

```
// /server.js
const http = require("http");

const HOST = 'localhost';
const PORT = 8000;

const handler = (req, res) => {
    res.writeHead(200);
    res.end('JavaScript - классный язык программирования!');
};

const server = http.createServer(handler);
server.listen(PORT, HOST, () => {
    console.log(`Сервер запущен по адресу http://${host}:${port}`);
});
```

Стандартная библиотека

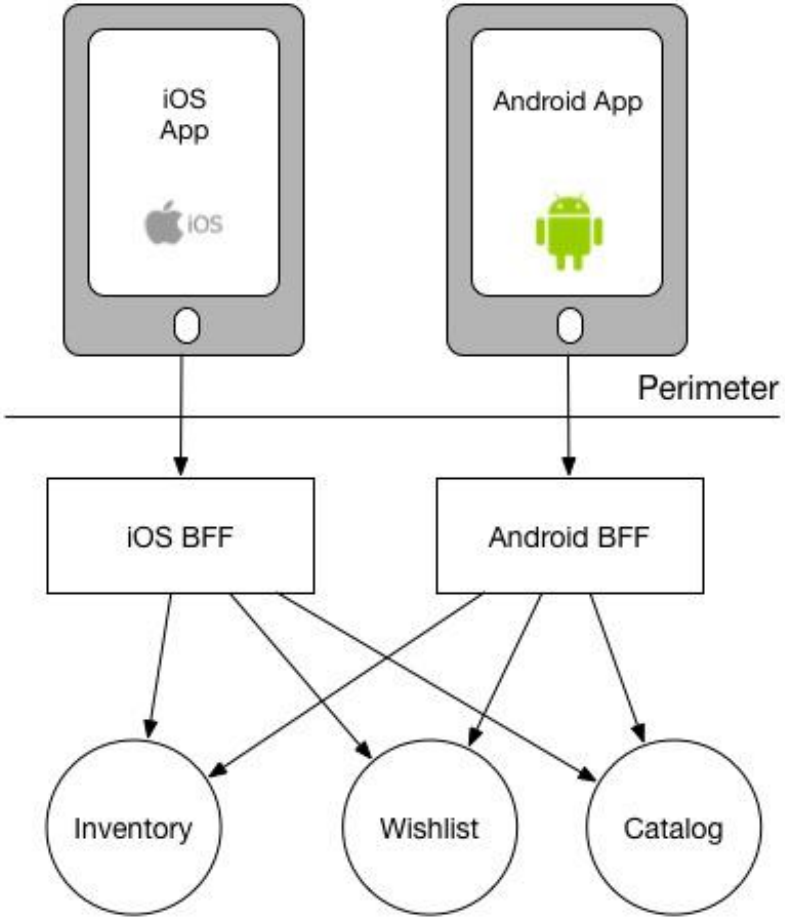
- `events` - встроенный EventEmitter
- `fs` - работа с файловой системой
- `os` - работа с операционной системой
- `http` - основной модуль для создания http-серверов
- `net` - низкоуровневое сетевое соединение
- `path` - для работы с путями в директориях
- и еще много-много, почитать можно здесь

```
const {writeFileSync, readFileSync} = require('fs');
const path = require('path');

function doMusic() {
        return 'listen to radiohead, guys';
}

writeFileSync(path.join(__dirname, 'wisdom-book.txt'), doMusic());
```

Так как используют Node.js ?

Прежде всего для серверной разработки. Например:

- SSR - серверный рендеринг для SPA
- BFF - бэкенд-прокси для агрегации данных
- Full-Backend API - полноценный бэкенд
- различные микросервисы со специфической логикой

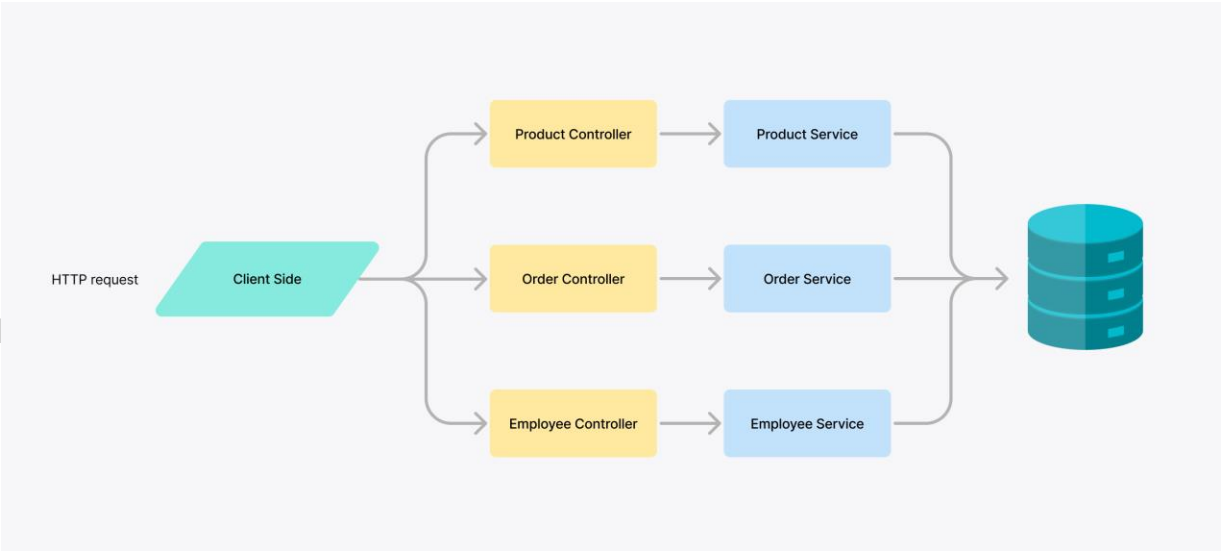


Nest.js

Nest.js - это фреймворк для создания эффективных, расширяемых серверных приложений. ОБЯЗАТЕЛЬНО на TypeScript

Какие преимущества дает Nest.js

- готовую архитектуру;
- готовые решения для middleware;
- удобные механизмы валидации;
- возможность grpc почти из коробки;
- dependency injection
- java-подобное приложение.



TypeScript (vs JS)

```ts
function sum(a: number, b: number): number {
    retun a + b
}
```

```js
function sum(a, b) {
    retun a + b
}
```

```ts
interface Person {
    name: string;
    age: number;
}
function meet(person: Person) {
    return `Привет, я ${person.name}, мне ${parson.age}`;
}
const user = { name: "Jane", age: 21 };
console.log(meet(user));
```

Преимущества TypeScript

TypeScript — язык программирования, представленный Microsoft в 2012 году. TypeScript является **обратно совместимым** с JavaScript и компилируется в последний. TypeScript отличается от JavaScript возможностью явного статического назначения типов, а также поддержкой подключения модулей

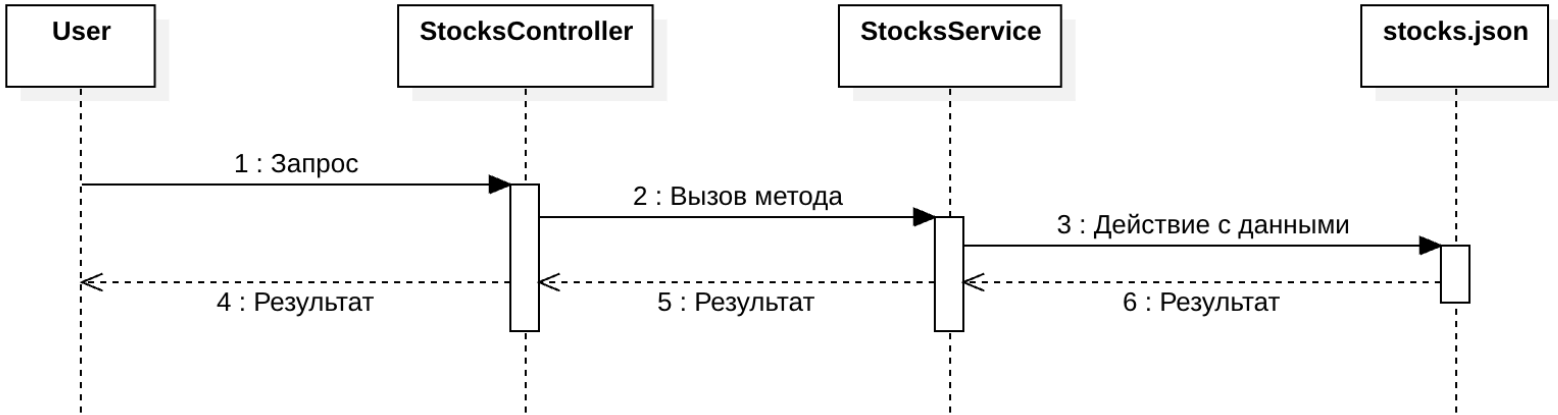
- Аннотации типов и проверка их согласования на этапе компиляции
- Интерфейсы, кортежи, декораторы свойств и методов, расширенные возможности ООП
- TypeScript — **надмножество JavaScript**, поэтому любой код на JavaScript будет выполнен и в TypeScript
- Широкая поддержка IDE и адекватный автокомплит
- Поддержка ES6-модулей из коробки

Аннотации типов

```
1 const valid = true;
2 const count = 42;
3 const name = 'Jon Snow';
4
5 const values: number[] = [1, 2, 3, 4, 5];
6 const tuple: [string, number] = ['Mean of life', 42];
7
8 enum Color {Red, Green, Blue};
9 const c: Color = Color.Green;
10
```

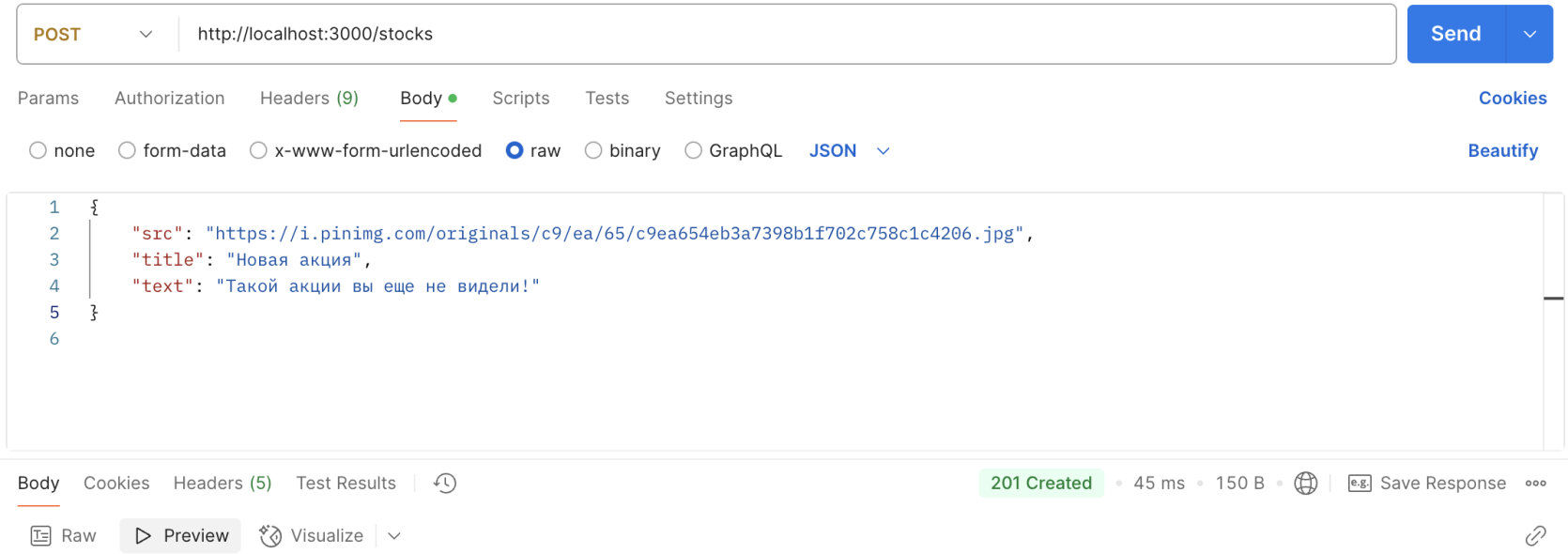
Задание в лабораторной

- GET /stocks/ - получение всех карточек
- POST /stocks - создание новой карточки
- GET /stocks/:id - получение карточки по ID
- PATCH /stocks/:id - обновление карточки по ID
- DELETE /stocks/:id - удаление карточки по ID



Postman

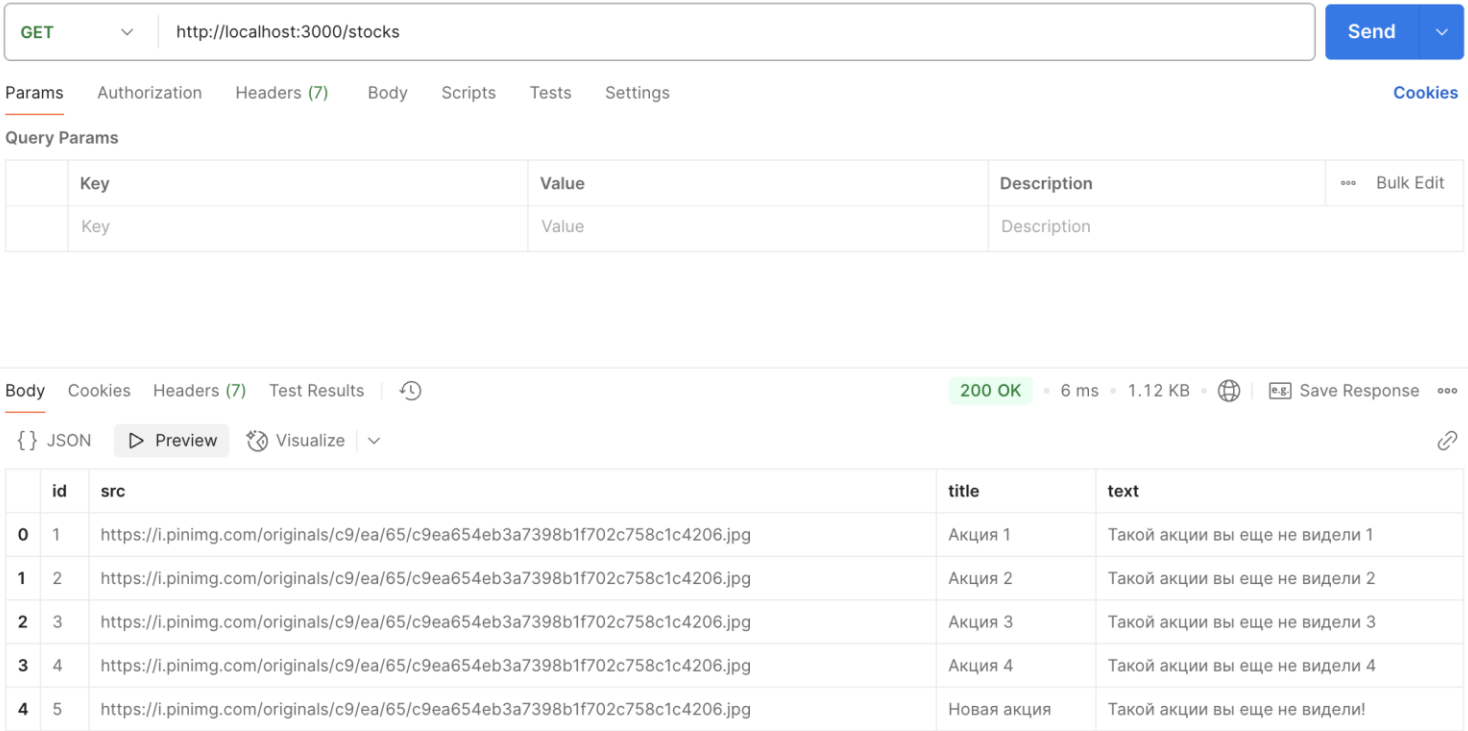
- Для тестирования методов веб-сервиса используем программу Postman
- Для создания новой записи мы должны написать и передать JSON на сервер



Postman

4 записи об акциях вначале

В конце пятая - наша новая акция



	id	src	title	text
0	1	https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg	Акция 1	Такой акции вы еще не видели 1
1	2	https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg	Акция 2	Такой акции вы еще не видели 2
2	3	https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg	Акция 3	Такой акции вы еще не видели 3
3	4	https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg	Акция 4	Такой акции вы еще не видели 4
4	5	https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg	Новая акция	Такой акции вы еще не видели!

Наши первоначальные данные - mock

У нас данные хранятся просто в массиве без базы данных

Сформируем начальные записи, которые будут отображаться в начале

```
[
    {
        "id": 1,
        "src": "https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg",
        "title": "Акция 1",
        "text": "Такой акции вы еще не видели 1"
    },
    {
        "id": 2,
        "src": "https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg",
        "title": "Акция 2",
        "text": "Такой акции вы еще не видели 2"
    },
    {
        "id": 3,
        "src": "https://i.pinimg.com/originals/c9/ea/65/c9ea654eb3a7398b1f702c758c1c4206.jpg",
        "title": "Акция 3",
        "text": "Такой акции вы еще не видели 3"
    },
```

Модель из MVC – Сервис в NestJS

Все данные у нас хранятся в массиве – это наша модель

Массив хранится в оперативной памяти, поэтому мы все данные сохраняем в файл на жесткий диск

Модель это не только сам массив, но и методы с логикой по которой мы получаем и изменяем данные

```
@Injectable()
export class StocksService {
  constructor(private fileService: FileService<Stock[]>) {}

  create(createStockDto: CreateStockDto) {
    const stocks = this.fileService.read();

    // для простоты новый id = текущее количество карточек + 1
    const stock = { ...createStockDto, id: stocks.length + 1 };

    this.fileService.add(stock);
  }

  findAll(): Stock[] {
    const stocks = this.fileService.read();

    return stocks;
  }
```

Контроллер – обработчики HTTP запросов

- Для все наших методов (комбинации url и HTTP метода) мы указываем код который будет выполняться
- Каждый контроллер предоставляет данные через модель (сервис NestJS)

```
@Controller('stocks')
export class StocksController {
    constructor(private readonly stocksService: StocksService) {}

    @Post()
    create(@Body() createStockDto: CreateStockDto) {
        return this.stocksService.create(createStockDto);
    }

    @Get()
    findAll() {
        return this.stocksService.findAll();
    }

    @Get(':id')
    findOne(@Param('id') id: string) {
        return this.stocksService.findOne(+id);
    }
```