Курс

Тестировщик ПО

Блок6

Тестирование API

бруноям

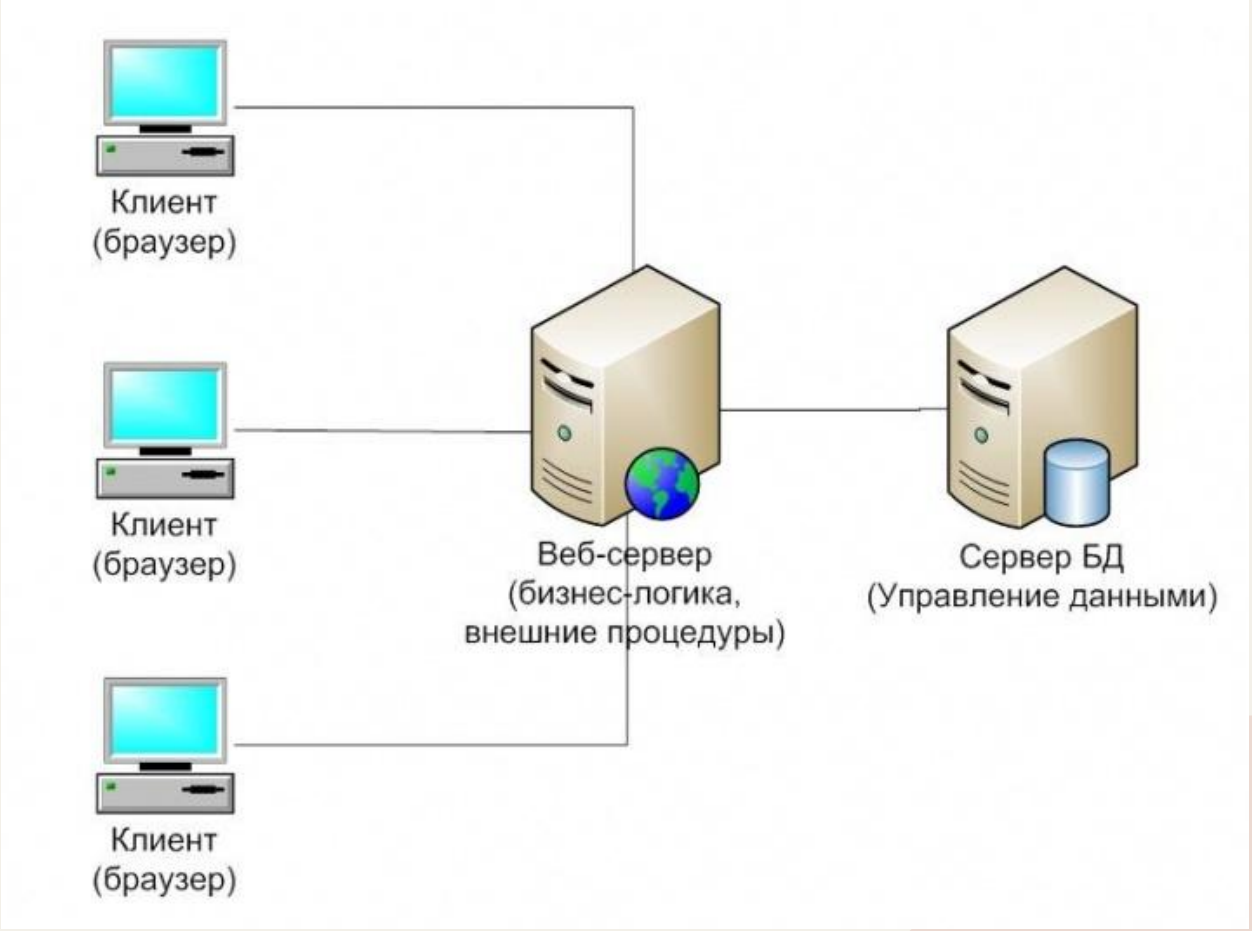
Структура:

- Понятие клиент-серверная архитектура
- HTTP(s)-протокол
- Модели сетей
- HTTP(s)
- API
- SOAP и REST
- SOAP UI/POSTMAN/SWAGGER
- DevTools

Клиент-серверная архитектура

Архитектура в которой сетевая нагрузка распределяется между поставщиками услуг (серверами) и заказчиками услуг (клиентами).

Клиент-сервер = программное обеспечение

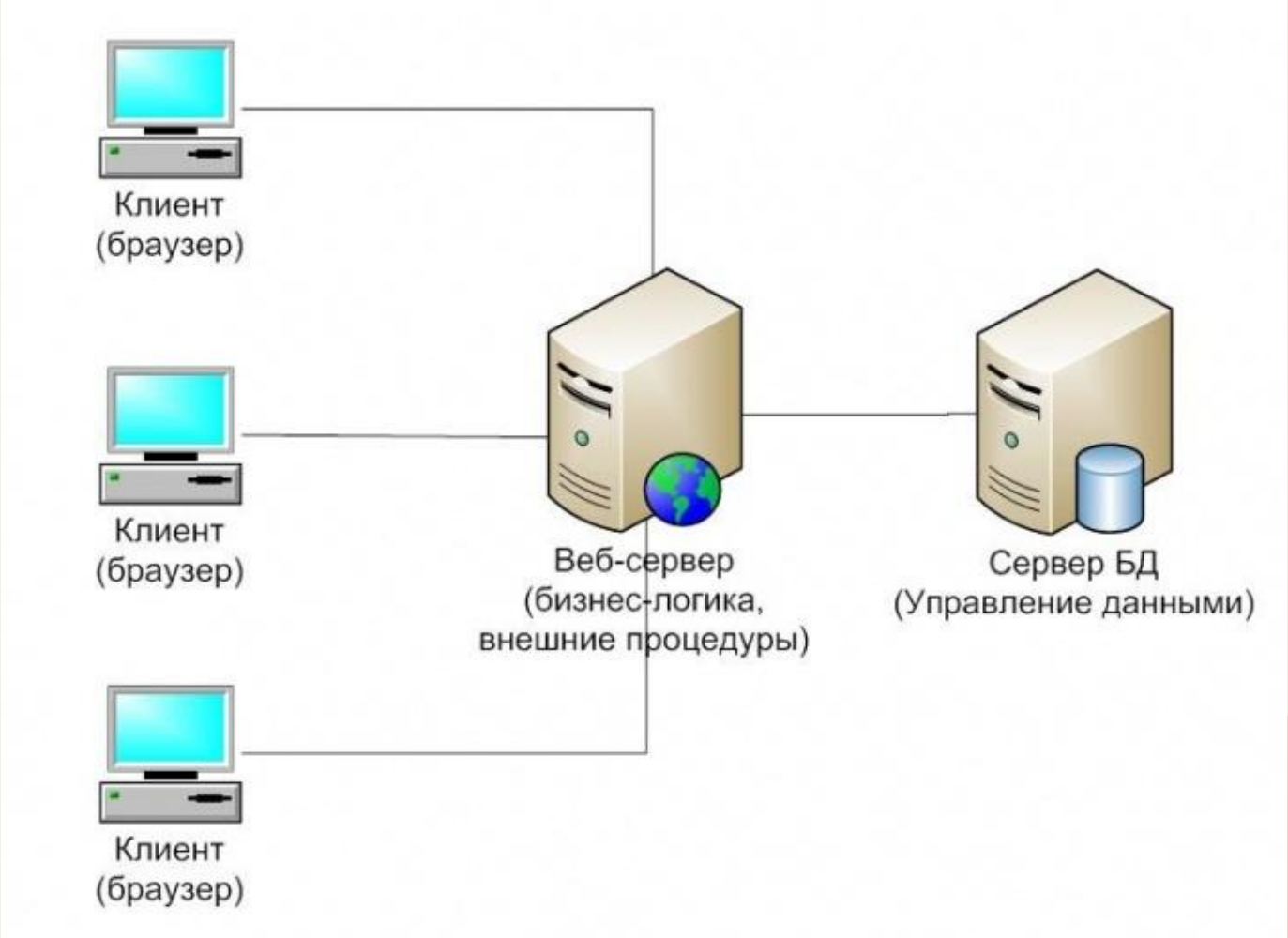


Сценарий 1 (двухуровневая архитектура)

1. Клиент отправляет HTTP request (запрос)
2. Сервер обрабатывает
3. Сервер отправляет HTTP response (ответ) на клиент

Сценарий 2 (трехуровневая архитектура)

1. Клиент отправляет HTTP request (запрос)
2. Сервер отправляет запрос на БД
3. БД отправляет ответ на сервер
4. Сервер отправляет HTTP response (ответ) на клиент



Плюсы:

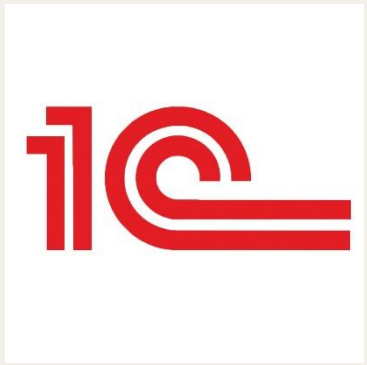
- Отсутствует дублирование кода программы сервера программами клиента
- Требования компьютера снижается
- Все данные хранятся на сервера
- Контроль прав доступов

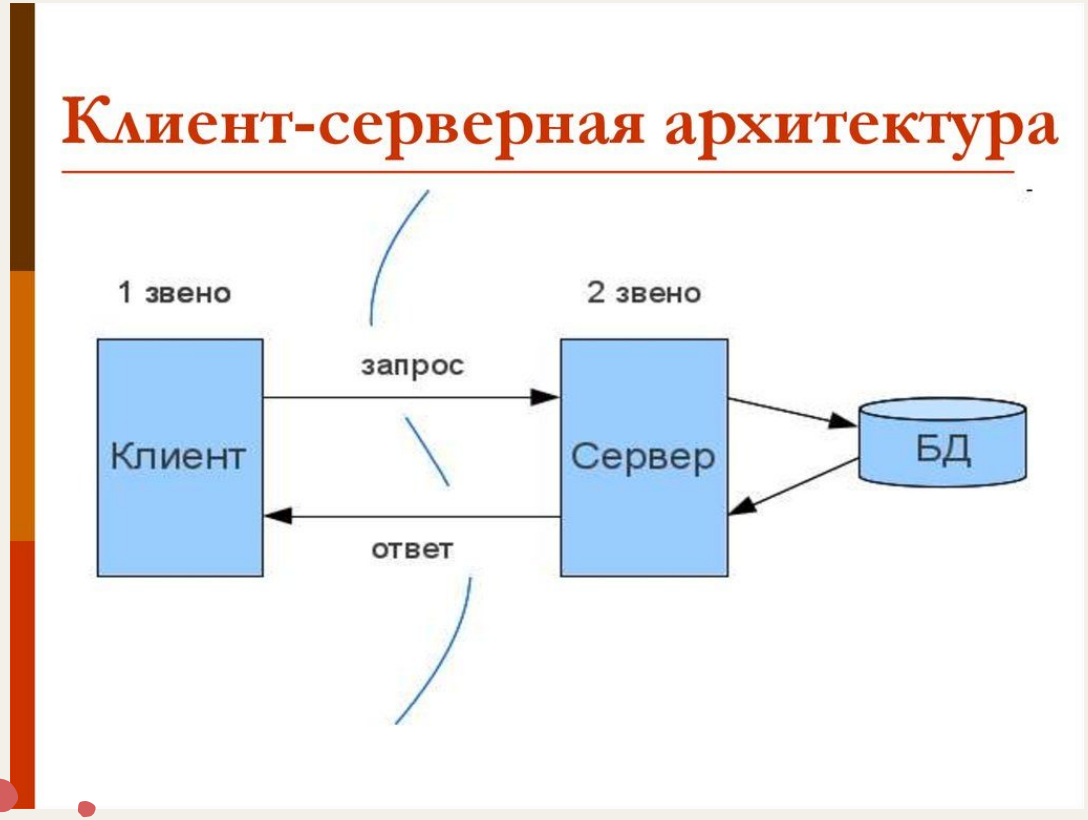
«Тонкий» клиент

Вся логика «лежит» сервере

«Толстый» клиент

Вся логика «лежит» в клиенте



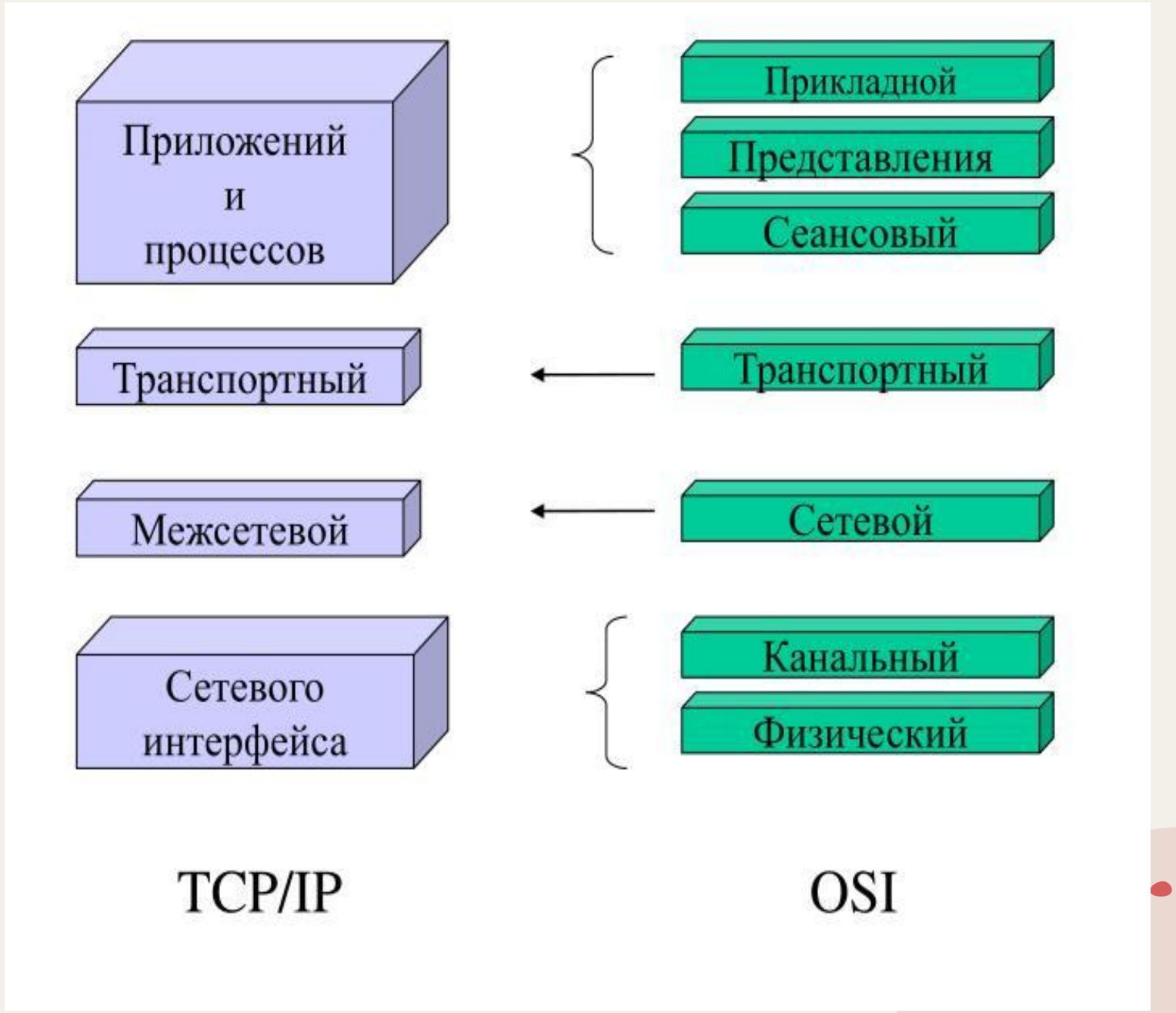


HTTP(s)-протокол:

HTTP(s) - Hypertext Transfer Protocol (secure) - протокол правил передачи гипертекста (защищенный).

Протокол – набор правил передачи информации

Модели сетей



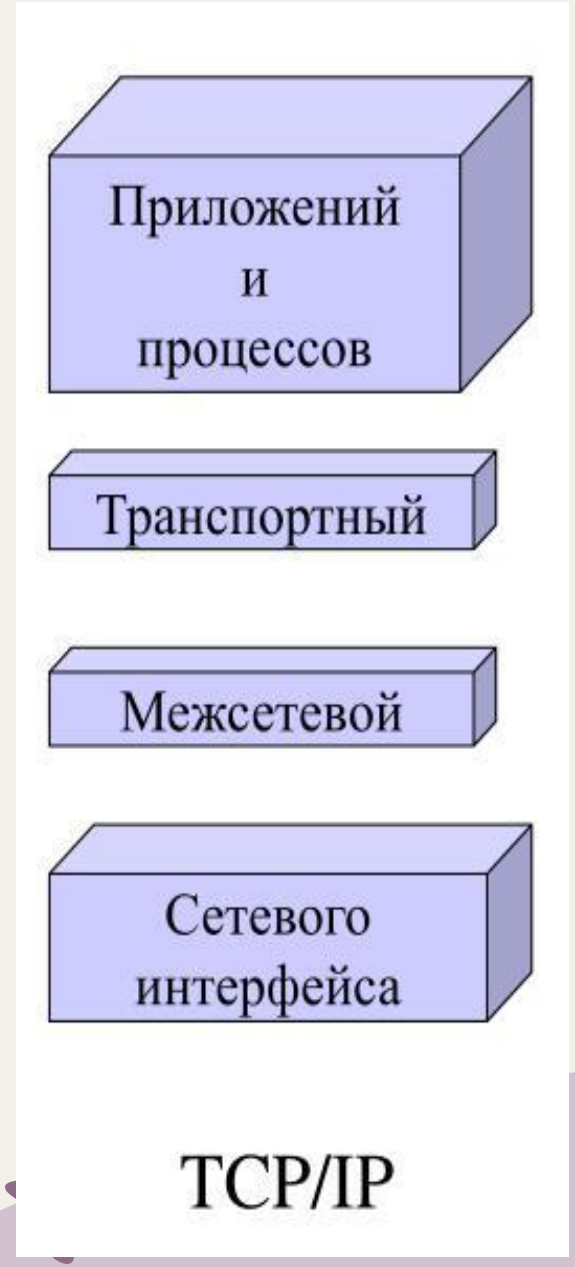
УРОВНИ TCP/IP

1 уровень
Сетевой интерфейс (Network Interface)
Это аппаратный уровень, на котором работают сетевые карты, коммутаторы, повторители, концентраторы.

2 уровень
Межсетевой (Network)
Интернет состоит из множества локальных сетей, объединённых между собой маршрутизаторами (физ.устройства) Здесь реализуется **ip**-адресация и маршрутизация, за счет введения ip-адресов.

3 уровень
Транспортный (Transport)
Здесь также работают 2 протокола: **TCP и UDP**. А в качестве служебной информации выступают сетевые порты.

4 уровень
Прикладной уровень (Application)
Протокол HTTP(S)
Здесь работают приложения, установленные на компьютере, телефоне или сервере. Здесь происходит согласование данных (шифрование, сжатие, выбор формата данных и выбор кодеков) и установка сеансов связи.



HTTP протокол:

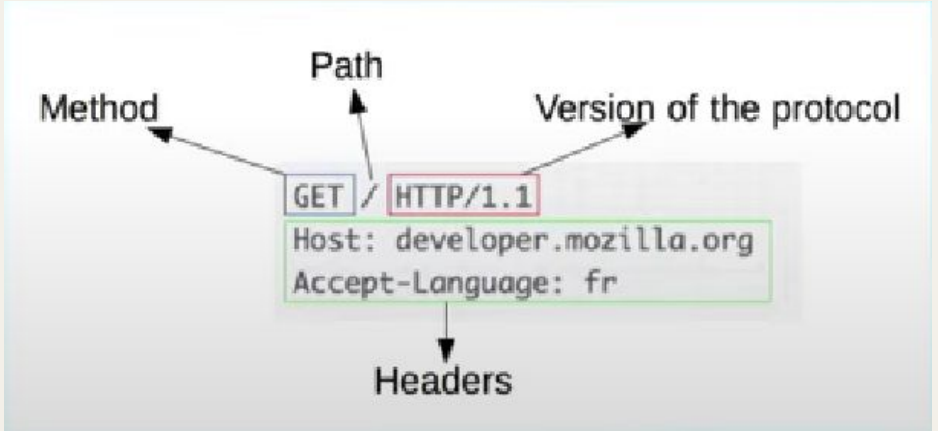
- Headers: Служебная информация
- Payload: Основная часть

- Request
- Response

https://developer.mozilla.org/ru/docs/Web/HTTP/Headers

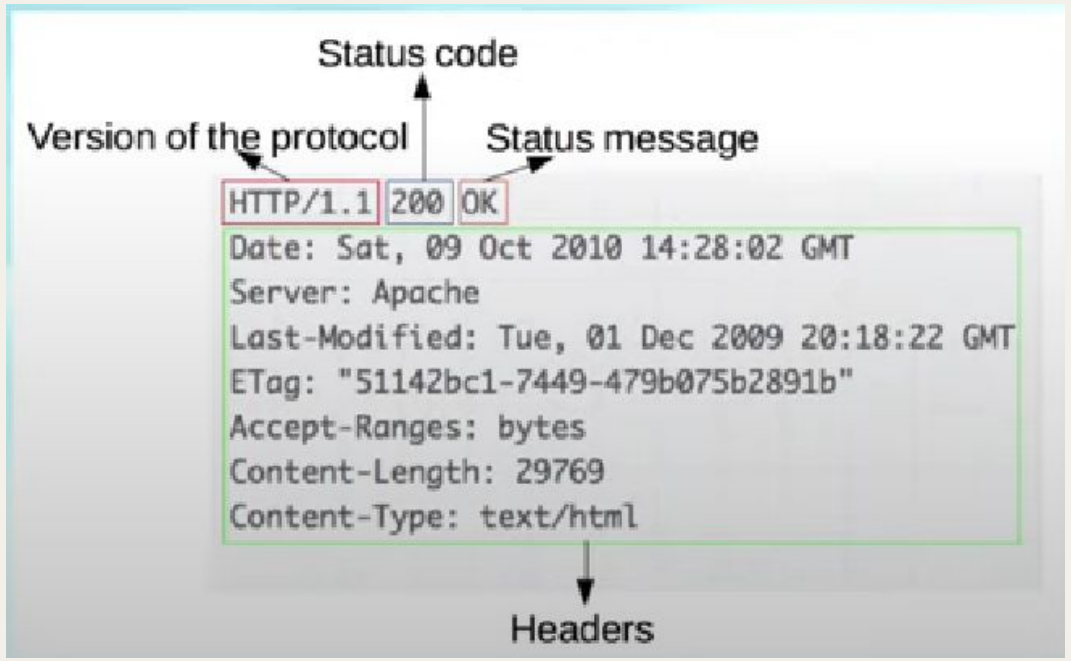
HTTP(S) request

1. Метод
2. Версия прокола
3. Host машина: сервер
4. Header: служебная информация
5. URL к какому ресурсу обращаемся



HTTP(S) response

1. Версия прокола
2. Статус код
3. Статус сообщение
4. Header: служебная информация



Типы методов:

1. **GET** - получение информации
2. **POST** - отправление информации
3. **PUT** - добавление/изменение
4. **DELETE** - удаление

CRUD-операции.

Акроним, обозначающий 4 базовые функции, используемые при работе с базами данных: CREATE, READ, UPDATE, DELETE.

Как работают методы:

1. **GET**

GET /book/ - получить список всех книг

GET /book/3/ - получить книгу №3

2. **POST** - отправление информации

POST /book/3 - изменить книгу. Данные в теле запроса

3. **PUT** - добавление/изменение

PUT /book/ - Добавить книгу. Данные про книгу в теле запроса

4. **DELETE** - удаление

DELETE /book/3 - Удалить книгу №3.

Коды ошибок:

https://httpstatuses.com/

1 - информационные

2 – успех

3 – перенаправление

4 - ошибки клиента

5 - ошибки сервера

Версии HTTP(S) протокола

HTTP/0.9, HTTP/1.0 – устаревшие

HTTP/1.1 – самая популярная версия

HTTP/2 – более новая, но еще набирает популярность

HTTP/3 –самая новая версия. Пока не набрала популярность, но может встречаться

API. Что такое:

API (Application Programming Interface) - программный интерфейс приложения. Набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

API = Набор функций (правил нет, каждый разработчик составляет сам).

Можно сгруппировать по:

- Функциональности (апи платежных сервисов)
- Заказчику
- Итд

API:

ТЕСТИРОВАНИЕ API = ТЕСТИРОВАНИЕ **через** API

ТЕСТИРОВАНИЕ API - тестирование того, как одна программа общается с другой по какому-то протоколу передачи данных. Чаще всего http(s).

REST и SOAP:

Самые популярные архитектурные стили API:

- SOAP (*Simple Object Access Protocol*) - протокол обмена структурированными сообщениями в распределённой вычислительной среде.
- REST (***Re****presentational* ***S****tate* ***T****ransfer*) - архитектурный стиль взаимодействия компонентов распределённого приложения в сети

Устанавливаем SOAP UI, Postman.

Трогаем руками ;)

SOAP

- SOAP (*Simple Object Access Protocol*) - протокол обмена структурированными сообщениями в распределённой вычислительной среде.

Обмен сообщениями **ТОЛЬКО** в формате XML (Похож на HTML)

WSDL – описывает структуру для WEB-сервиса

Является устаревшим (медленный, много правил, долгая разработка)

Работают только с HTTP

WSDL

WSDL — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

WSDL

```
<!-- Abstract type -->
<types>
    <xs:schema>
        <xs:element name="request"> ... </xs:element>
        <xs:element name="response"> ... </xs:element>
    </xs:schema>
</types>

<!-- Abstract interfaces -->
<interface name="Interface1">
    <fault name="Error1" element="tns:response"/>
    <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
        <input messageLabel="In" element="tns:request"/>
        <output messageLabel="Out" element="tns:response"/>
    </operation>
</interface>
```

REST

- REST (***Re**presentational **S**tate **T**ransfer)* - архитектурный стиль взаимодействия компонентов распределённого приложения в сети

Нет жестких правил

Используют JSON

Практически все сейчас работают с REST

Работают только с HTTP

JSON

JSON - JavaScript Object Notation - формат данных в виде "ключ-значение".

Правила:

1. Данные хранятся в виде пар "ключ-значение"
2. Данные разделены запятыми
3. Объекты находятся внутри фигурных скобок
4. Массив внутри квадратных скобок

JSON example

```
{
        "firstName": "John",
        "lastName": "Smith",
        "age": 25,
        "address": {
                "streetAddress": "21 2nd Street",
                "city": "New York",
                "state": "NY",
                "postalCode": 10021
        },
        "phoneNumbers": [
                {
                        "type": "home",
                        "number": "212 555-1234"
                },
                {
                        "type": "fax",
                        "number": "646 555-4567"
                }
        ]
}
```

Практика

http://users.bugred.ru/

- Скачиваем SOUP UI
- Скачиваем Postman

Практика

https://petstore.swagger.io/

- Трогаем Сваггер Добавить новое животное в каталог, загрузить изображение

Практика

http://users.bugred.ru/

- Зарегистрироваться, создать компанию

https://petstore.swagger.io/

- Трогаем Сваггер

Зачем тестировать API:

- Можно начать тестировать раньше
- Фронта/UI вообще нет
- Экономия времени
- Локализация дефектов (на чьей стороне проблемы клиент-сервер)
- Автоматизация

Чек-лист тестирования API:

1. Отправить запрос из документации
2. Тестирование бизнес-логики
3. Тестирование ошибок
4. Отправка запроса с “плохим” JSON/XML
5. Поменять параметры местами
6. Регистрозависимость

Чек-лист тестирования API:

https://developer.chrome.com/docs/devtools/

DevTools

Вкладка Elements (Элементы)

Вкладка Console (Консоль)

Вкладка Sources (Источники)

Вкладка Network (Сеть)

Вкладка Perfomance (Производительность)

Вкладка Security (Безопасность)

Практическое задание:

1. На сайте школы Бруноям перейти в Курс Тестировщик ПО и найти запрос, который загружает видео.
Дернуть этот метод отдельно в Postman/строке браузера

Домашка:

Задание 1 (normal)

На сайте **users.bugred** зарегистрируйте юзера без использования GUI. Потом с помощью Postman добавьте юзеру 1 компанию и 3 задачи. Приложите ссылку на юзера и JSON-файлы, которые вы использовали для отправки запроса через Postman.

В https://petstore.swagger.io/ пройти 4 любых запросов (на выбор)

Задание 2 (hard)

Выполните все запросы на сайте users.bugred.ru . Всю информацию берите из документации.

В https://petstore.swagger.io/ пройти все запросы