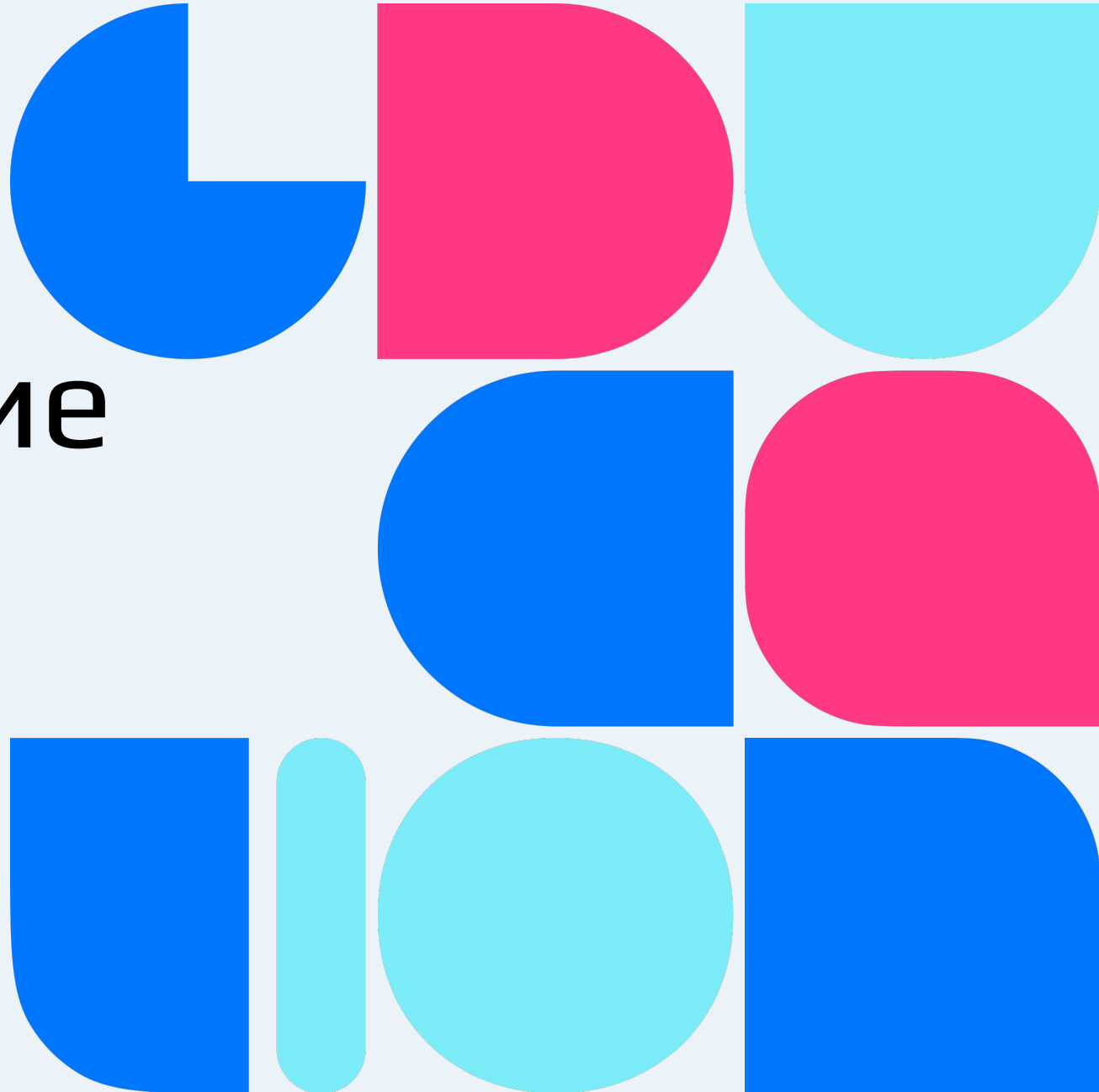
education

Лекция 7
Взаимодействие с сервером

Мартин Комитски



План на сегодня

- Что нужно сделать дома (ДЗ)
- Архитектура Web приложения
- Компьютерные сети
- Сетевые фишки в DOM
- Политики
- Взаимодействие клиента с backend
- RTU
- Инструменты разработчика

Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях

ДЗ

ДЗ?

- Backend дадим
- Proxy

ДЗ7 (коротко)

- Создать “общий” чат
- Связать ваш frontend с имеющимися API на backend

Как это сделать в коде

Функциональный компонент

```
1.  import { useEffect, useState } from React;
2.
3.  const SomeFunctionalComponent = () => {
4.    const [messages, setMessages] = useState([])
5.    useEffect(() => {
6.      fetch(`${API_URL}`)
7.        .then(res => res.json())
8.        .then(data => {
9.          console.log(data);
10.         setMessages(data.messages)
11.       });
12.   }, []);
13. }
```

Классовый компонент

```
1.  class SomeComponent extends React.Component {
2.    state = {
3.      // state declaration
4.      messages = []
5.    }
6.
7.    componentDidMount () => {
8.      this.getMessages()
9.    }
10.
11.   getMessages = () => {
12.     fetch(`${API_URL}`)
13.       .then(res => res.json())
14.       .then(data => {
15.         console.log(data);
16.         this.setState({
17.           messages: data.messages
18.         });
19.       });
```

Обновление в “реальном времени”

```
1.  const pollItems = () => {
2.    fetch(`${API_URL}?key=value&another_key=another_value`)
3.      .then(resp => resp.json())
4.      .then(data => console.log(data));
5.  }
6.
7.  const t = setInterval(() => pollItems(), 3000);
8.
9.  // clearInterval(t)
```

Сервер с данными (NodeJS)

Репозиторий: https://github.com/education-vk-company/tt-front-server

Роутинг: https://github.com/education-vk-company/tt-front-server/blob/master/server.js

Репозиторий 2: https://github.com/education-vk-company/vk-edu-messenger-backend

Роутинг 2: https://github.com/education-vk-company/vk-edu-messenger-backend/blob/main/application/urls.py

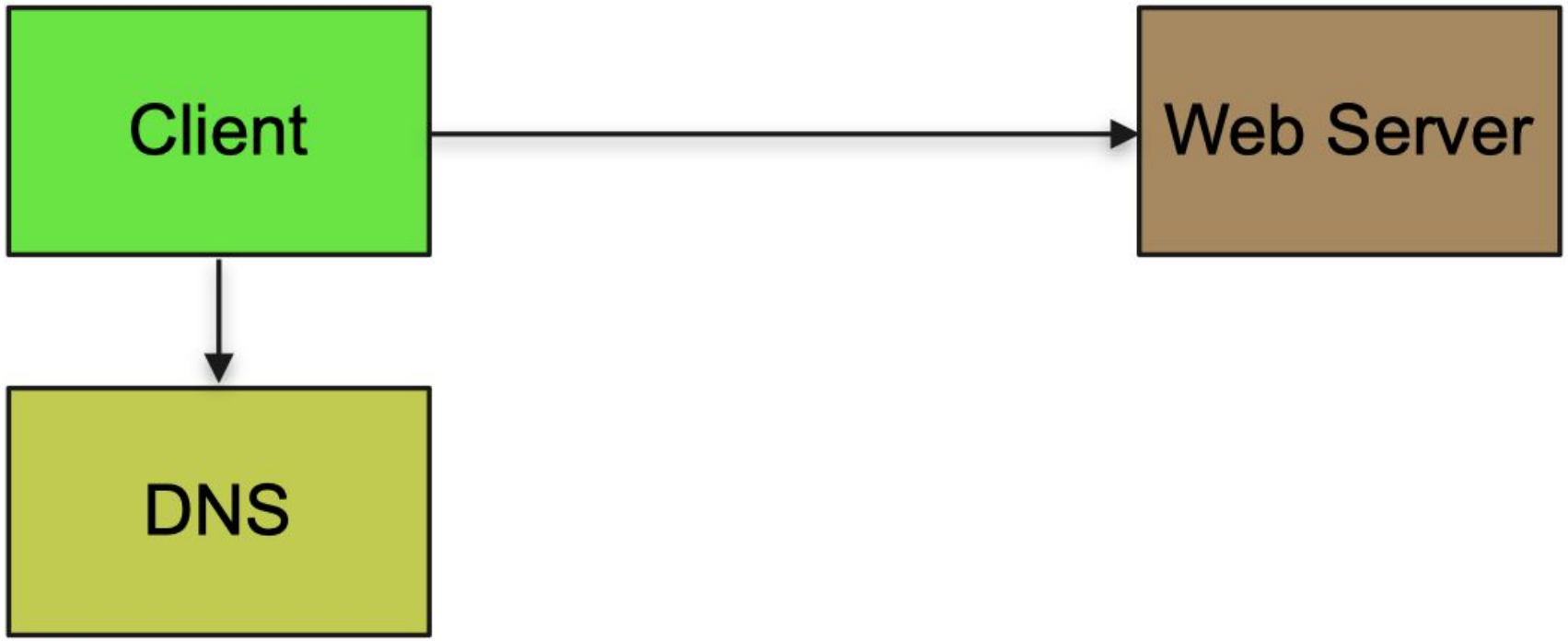
При выполнении ДЗ

- Можно обращаться к локальному бекенду до тех пор, пока он не будет задеплоен
- При сдаче ДЗ приложить gif/видео с обновлением сообщений в общем чате

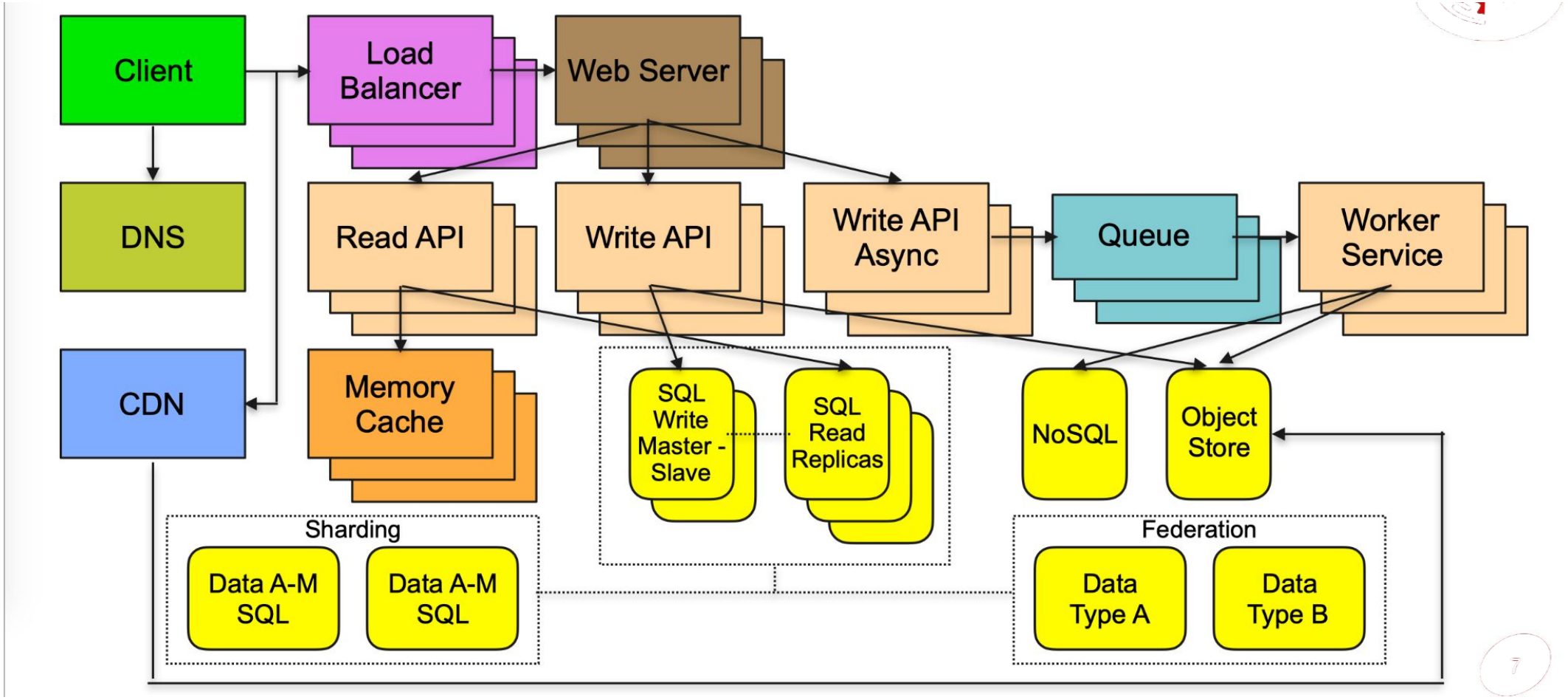
Архитектура веб приложения

Глазами обычного фронтендера

Глазами опытного фронтендера



Глазами опытного фронтендера



Архитектура более подробно

Читать подробнее:

- The System Design Primer
- Основы для начинающих разработчиков

Компьютерные сети

Компьютерные сети. Модель OSI

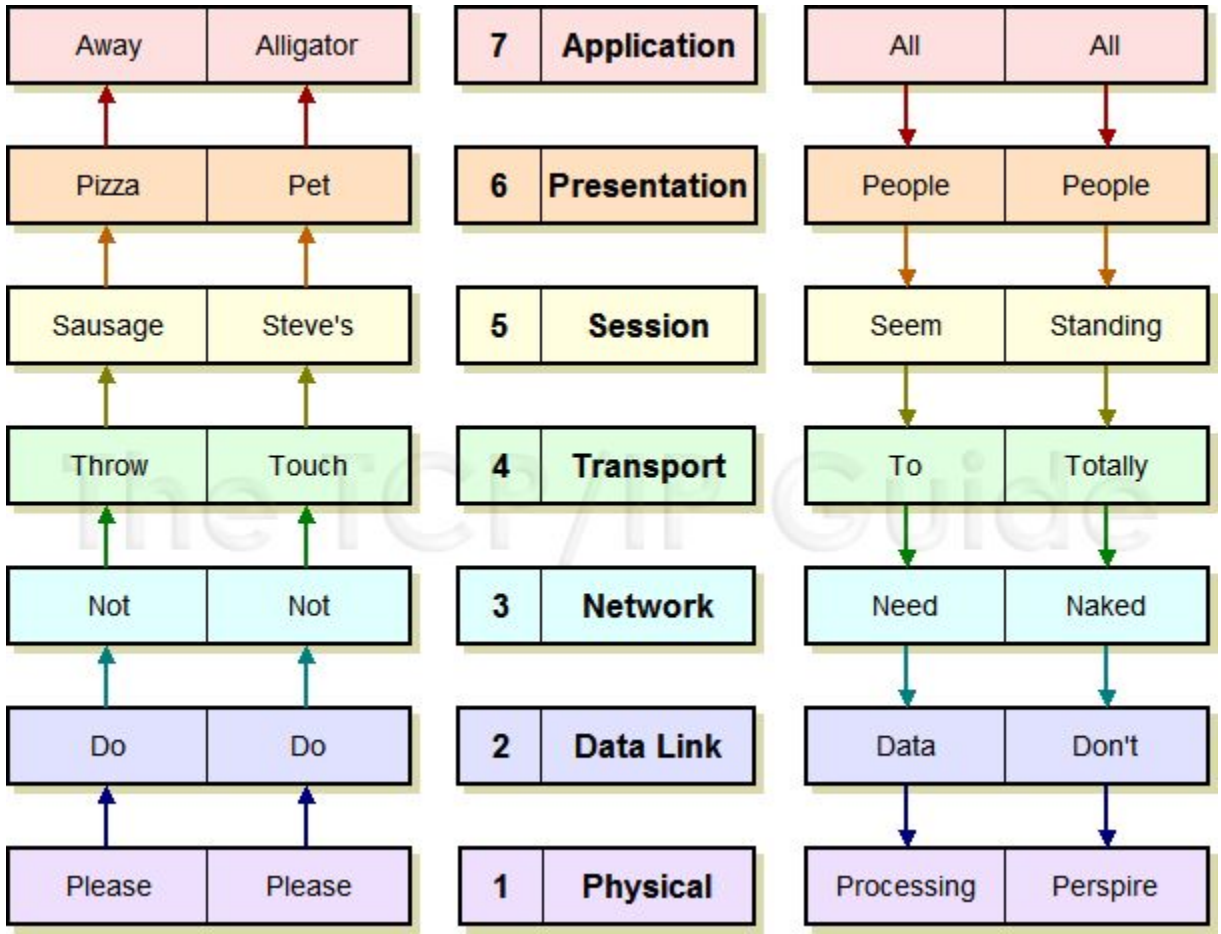
Open Systems Interconnection

Модель OSI – позволяет различным сетевым устройствам взаимодействовать друг с другом.

Модель определяет различные уровни взаимодействия систем.

Каждый уровень выполняет определённые функции при таком взаимодействии.

Компьютерные сети. Модель OSI



Компьютерные сети. Модель TCP/IP

Transmission Control Protocol

RFC 793: https://tools.ietf.org/html/rfc793

Протокол **TCP** – абстракция надежной передачи данных в сети (ненадежной среде)

Internet Protocol

Протокол **IP** – объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных между любыми узлами сети через произвольное число промежуточных узлов (маршрутизаторов)

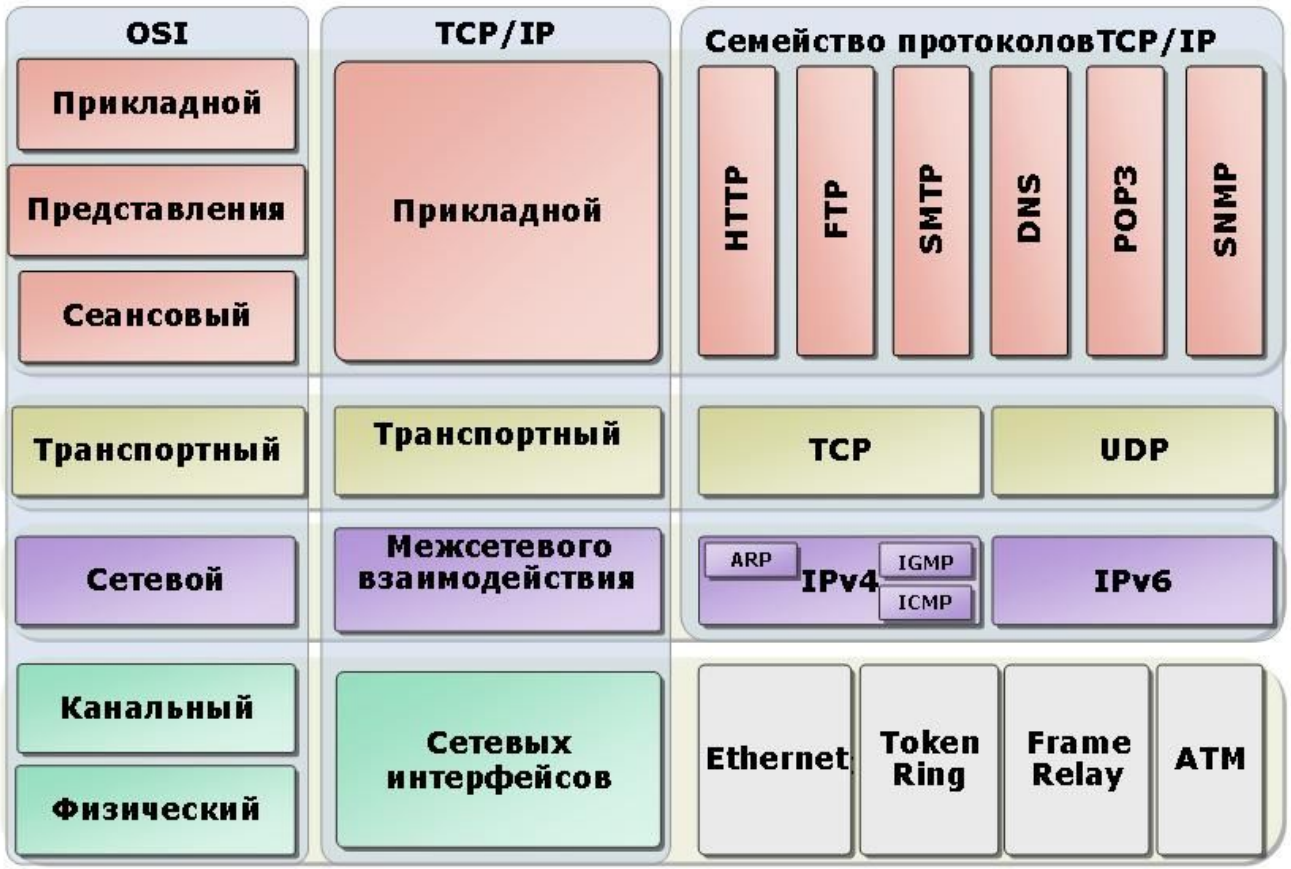
RFC791: https://tools.ietf.org/html/rfc791

Модель TCP/IP описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи).

Компьютерные сети. Модель OSI

Уровень	Физический	Канальный	Сетевой	Транспортный	Сеансовый	Представления	Прикладной
Описание	Передача бинарных данных (кабель, радиочастота)	Физическая адресация между двумя объектами	Адресация, определение маршрута, контроль траффика	Надежная передача данных между несколькими участниками	Поддержание сеанса связи	Сжатие, шифрование, представление данных	Высоко-уровневые апи
Протоколы	802.15 809.11 GSM	802.3 PPP ARP	IP RIP OSPF	TCP UDP	L2TP SOCKS RPC	MIME TLS	HTTP WebSocket

Компьютерные сети. Модели OSI, TCP/IP



Компьютерные сети. TCP и UDP

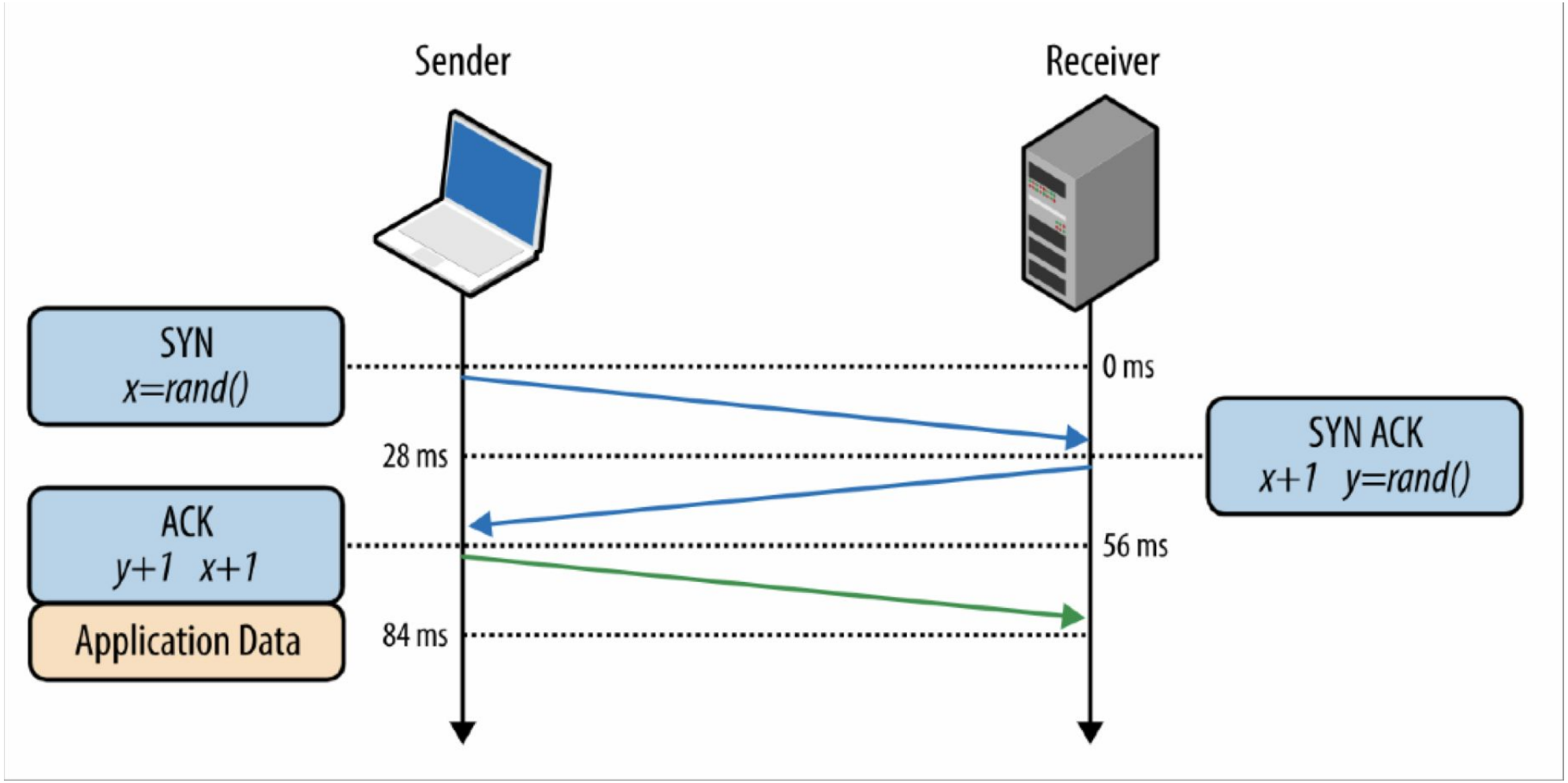
Знаю анекдот про UDP, но не факт, что он до вас дойдет...

А еще знаю анекдот про TCP. Если он до вас не дойдет, я повторю его снова.

TCP

UDP

Компьютерные сети. TCP



Компьютерные сети. SSL/TLS

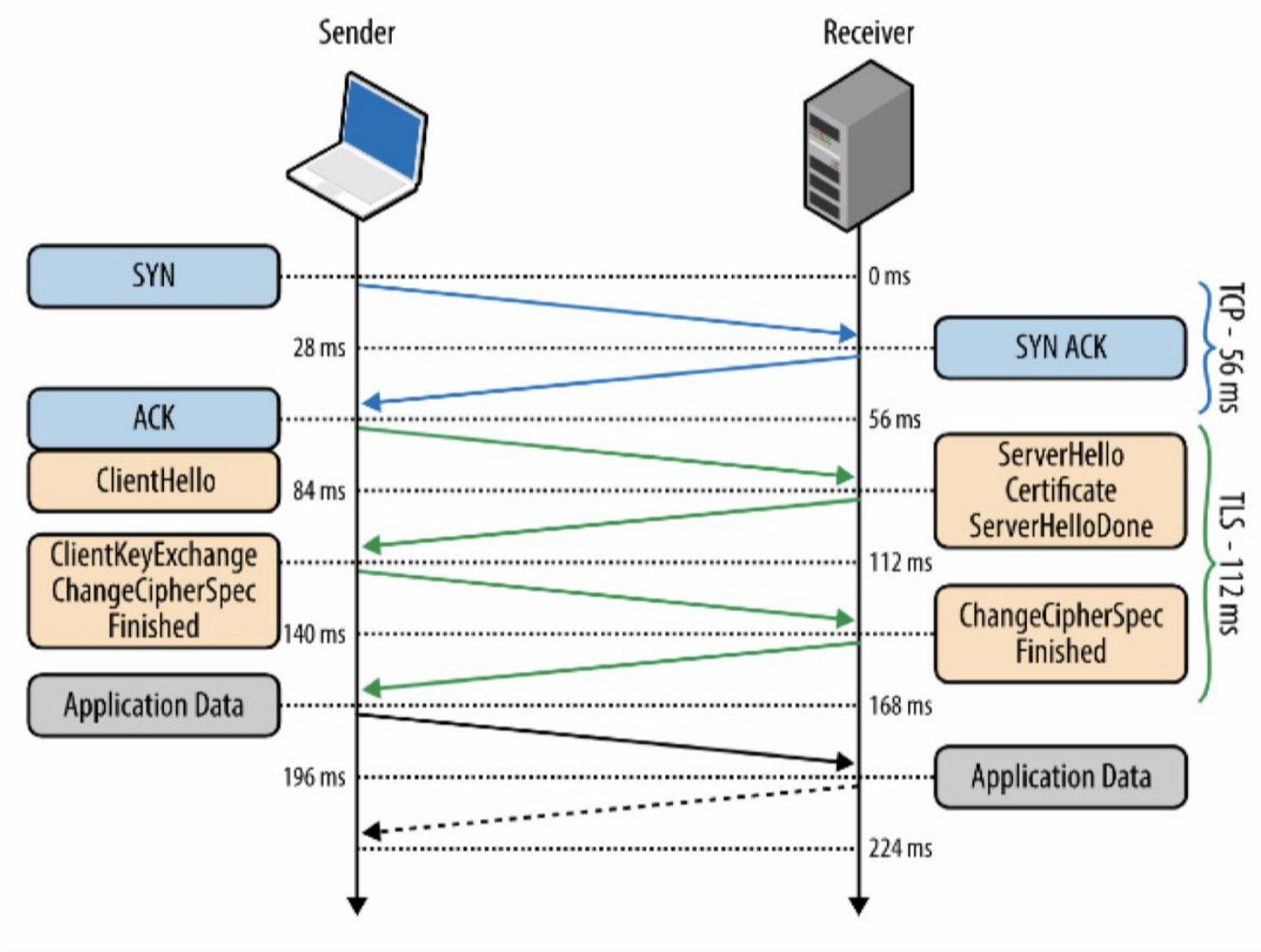
TLS – Transport Layer Security. Протокол защиты транспортного уровня, как и его предшественник **SSL**.

SSL – Secure Sockets Layer. Слой защищённых сокетов.

SSL/TLS – криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети Интернет.

RFC 5256: https://tools.ietf.org/html/rfc5246

Компьютерные сети. TLS handshake



Компьютерные сети. HTTP

HTTP 1.1 RFC 7231: https://tools.ietf.org/html/rfc7231

HTTP/2 RFC 7540: https://tools.ietf.org/html/rfc7540

HTTP/3 RFC 9114: https://tools.ietf.org/html/rfc9114

HTTP – HyperText Transfer Protocol. Протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

Компьютерные сети. HTTP

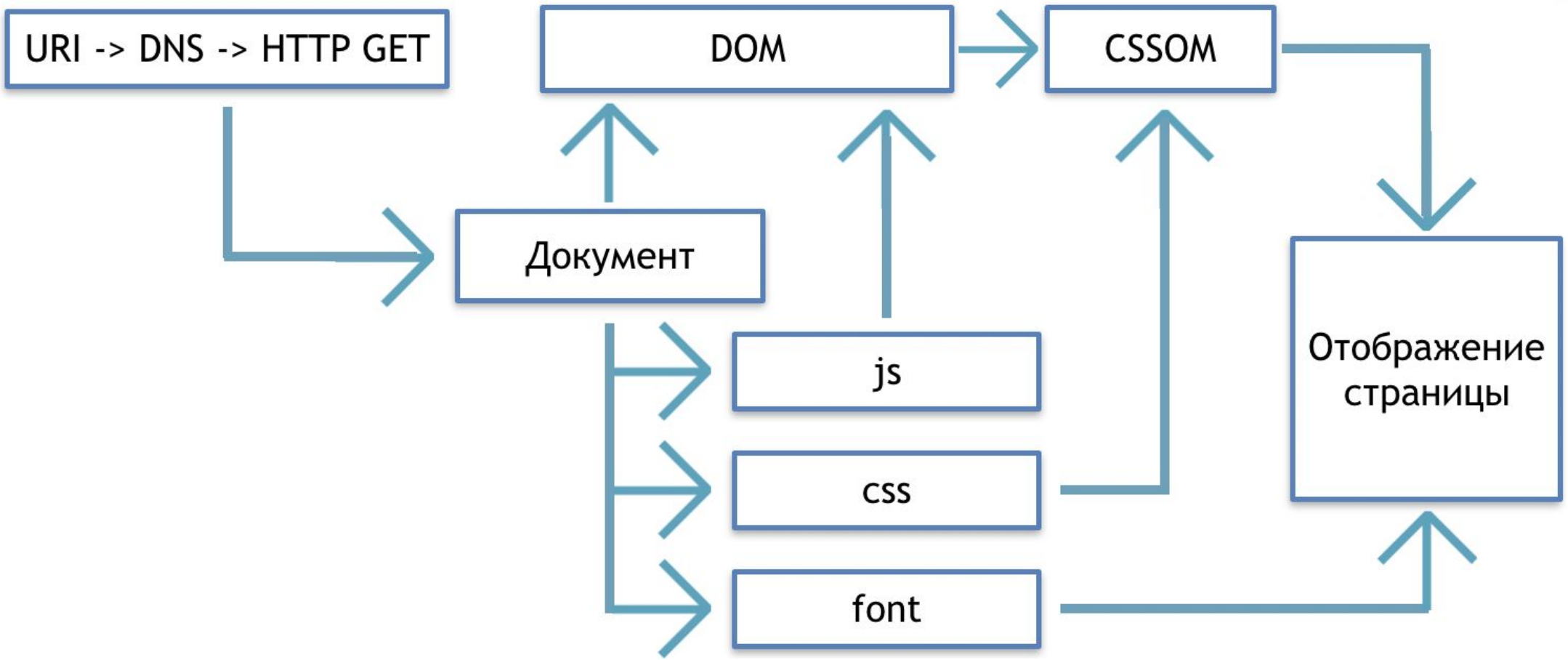
- Сократить количество запросов к DNS
- Сократить количество TCP соединений
- Сократить количество редиректов
- Снизить RTT
- Избавиться от ненужных ресурсов
- Кэшировать ресурсы на клиенте
- Сжимать данные при передаче
- Не запрашивать ненужные ресурсы
- Утилизировать обработку запросов и ответов
- Использовать оптимизации соответствующие протоколу (http 1.1 / http/2 / http/3)
- Помнить про мобильные сети и устройства

Компьютерные сети?

Вопросы?

Сетевые фишки в DOM

Сетевые фишки в DOM. Порядок загрузки страницы



Сетевые фишки в DOM. Порядок загрузки страницы

Читать больше: Analyzing Critical Rendering Path Performance

Сетевые фишки в DOM. Включение ресурсов

```
<script async />  <!-- не блокирует DOM, выполнится как можно скорее -->
<script defer />  <!-- выполнится после построения DOM →

<script>
  var script = document.createElement('script');
  script.addEventListener('load', (event) => {});
  script.src = '/dist/main.js';
  document.head.appendChild(script);
</script>
```

Script Tag - async & defer: https://stackoverflow.com/a/39711009/3984110

Сетевые фишки в DOM. Включение ресурсов

```
<link href="/dist/touch.css" rel="stylesheet" media="(max-width: 320px)">
<link href="landscape.css" as="style" rel="preload" media="(orientation: landscape)">

@import "common.css" screen;
@import url('landscape.css') screen and (orientation:landscape);
```

Сетевые фишки в DOM. Приоритет и предзагрузка

```
1.  НИЗКИЙ ПРИОРИТЕТ, Ресурсы, которые понадобятся позже
2.  <link rel="prefetch">
3.
4.  ВЫСОКИЙ ПРИОРИТЕТ, Ресурсы, которые нужны заранее
5.  <link rel="preload" as="script">
6.
7.  RESOURCE HINTS
8.  <link rel="dns-prefetch" href="https://otvet.mail.ru>"
9.  <link rel="preconnect" href="https://auto.mail.ru/external-static">
10. <link rel="prefetch" href="https://pets.mail.ru/news" as "document">
11. <link rel="prefetch" href="https://pets.mail.ru/news/?page=2" as "document">
12. <link rel="prerender" href="https://pets.mail.ru/news/?page=2">
```

Читать больше:

- Preconnect, prerender, prefetch
- w3.org: resource hints
- Preload, prefetch and other tags, Перевод статьи на хабре

Сетевые фишки в DOM. События загрузки

DOMContentLoaded

- Блокирующие ресурсы загружены
- Построение DOM завершено
- document.addEventListener("DOMContentLoaded", (event) => {console.log('DOMContentLoaded is done')});

https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event

Сетевые фишки в DOM. События загрузки

load

- Все внешние ресурсы загружены
- Построение CSSOM завершено
- window.addEventListener("load", (event) => {console.log('Load is done')});

https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

Сетевые фишки в DOM. События загрузки

unload/beforeUnload

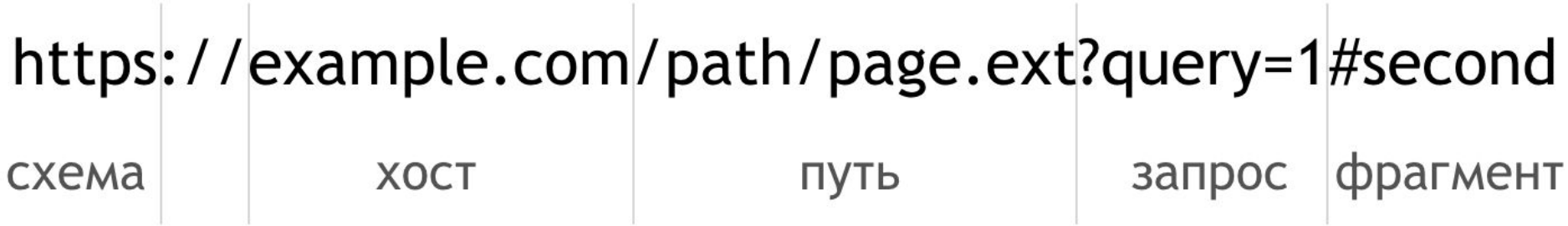
```
1. window.addEventListener("beforeunload", (event) => {
2.   event.preventDefault();
3.   event.returnValue = '';
4. });
```

https://developer.mozilla.org/en-US/docs/Web/API/Window/beforeunload_event

https://developer.mozilla.org/en-US/docs/Web/API/Window/unload_event

Сетевые фишки в DOM. URI

Uniform Resource Identifier



1. window.location
2. window.URL
3.
4. Object.getOwnPropertyNames(window.location)

protocol, hostname, pathname, search, hash, port, host, searchParams, password, username, href

Сетевые фишки в DOM. URI

1. const myUrl = new URL('https://example.com/path/page.ext?query=1#second');
2. window.location.assign(myUrl); // переход на новую страницу

Сетевые фишки в DOM?

Вопросы?

Перерыв! (10 минут)

Препод (с)

Политики

Политики. Same-origin policy

Same-origin policy – политика одинакового источника определяет, как документ или скрипт, загруженный из одного источника (origin), может взаимодействовать с ресурсом из другого источника. Это помогает изолировать потенциально вредоносные документы, снижая количество возможных векторов атак.

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Политики. Определение Origin

Две страницы имеют одинаковый origin, если протокол, порт и хост одинаковы для обеих страниц.

- https – protocol
- mail.ru – host
- 443 – port

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Success	
`http://store.company.com/dir/inner/another.html`	Success	
`https://store.company.com/secure.html`	Failure	Different protocol
`http://store.company.com:81/dir/etc.html`	Failure	Different port
`http://news.company.com/dir/other.html`	Failure	Different host

Политики. Изменение origin

1. document.domain = "company.com";

Deprecated по причине потенциальных уязвимостей и нарушения логики Same-origin policy.

Политики. Разные источники

1. Можно отправлять запросы и встраивать контент
2. Нельзя программно читать ответы
3. Нет доступа к DOM и свойству location

Политики. Iframe

1. <iframe src="https://www.w3schools.com" title="W3Schools Free Online Web Tutorials"></iframe>

https://www.w3schools.com/tags/tag_iframe.ASP

Заголовок: X-Frame-Options

- X-Frame-Options: deny
- X-Frame-Options: allow-from https://example.com/

Политики. CORS

CORS – Cross-Origin Resource Sharing

Техника управления доступом к ресурсам из внешних источников.

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Политики. CORS

- Access-Control-Allow-*
- Access-Control-Request-*
- Access-Control-Allow-Origin: https://example.com

Политики. CORS. Credentials

Реквизиты доступа не будут отправлены по-умолчанию.

Они будут отправлены только с разрешения источника.

Политики. CORS. Credentials

- Cookie
- Authorization
- TLS-сертификат

Особенности:

- Ответ на простой запрос нельзя прочесть
- Целевой запрос не будет отправлен, если в preflight нет разрешения

What exactly does the Access-Control-Allow-Credentials header do?

Политики. CORS. Простые запросы

Методы

- GET
- POST
- HEAD

Политики. CORS. Основные заголовки запросов

https://fetch.spec.whatwg.org/#cors-safelisted-request-header

- Accept
- Accept-Language
- Content-Language
- Content-Type
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain

Политики. CORS. Запрос и ответ

1. GET /public HTTP/1.1
2. Host: my-example.com
3. Origin: https://example.com

1. HTTP/1.1 200 OK
2. Access-Control-Allow-Origin: *

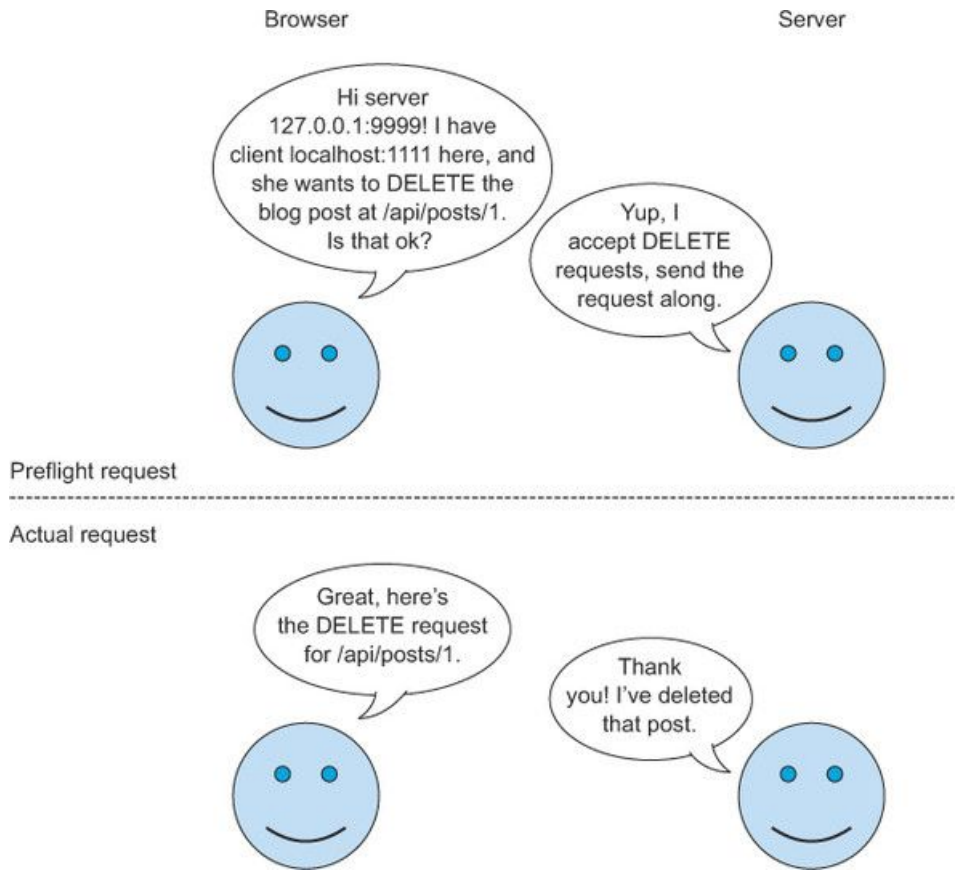
Политики. CORS. Сложные запросы. Preflight

Все запросы, в которых есть заголовки, которые отсутствуют в списке cors-safelisted-request-header, будут считаться сложными и будут инициировать т.н. предзапрос.

Условия сложности:

- Отличается метод
- Присутствуют другие заголовки
- Другой Content-Type

Политики. CORS. Сложные запросы. Preflight



Политики. CORS. Preflight запрос

1. OPTIONS /public HTTP/1.1
2. Host: my-example.com
3. Access-Control-Request-Method: POST
4. Access-Control-Request-Headers: X-CSRF-Token, Content-Type

1. HTTP/1.1 200 OK
2. Access-Control-Allow-Origin: https://example.com
3. Access-Control-Allow-Methods: POST, GET, OPTIONS
4. Access-Control-Allow-Headers: X-CSRF-Token, Content-Type
5. Access-Control-Max-Age: 600

Политики. CORS. Основной запрос

1. POST /public HTTP/1.1
2. Host: my-example.com
3. X-CSRF-Token: some-value
4. Origin: https://example.com

1. HTTP/1.1 200 OK
2. Access-Control-Allow-Origin: https://example.com

Политики?

Вопросы?

Взаимодействие клиента с backend

Взаимодействие клиента с backend. Выполнение запросов

	Встроенное поведение	**Обработка результата**	**Отмена запроса**
Переход между страницами	Да	Нет *	Нет
Отправка формы	Да	Нет *	Нет
Добавление ресурса	Да *	Да *	Нет
XHR/Fetch	Нет	Да	Да *

Взаимодействие клиента с backend. AJAX и XHR

X

Взаимодействие клиента с backend. AJAX и XHR

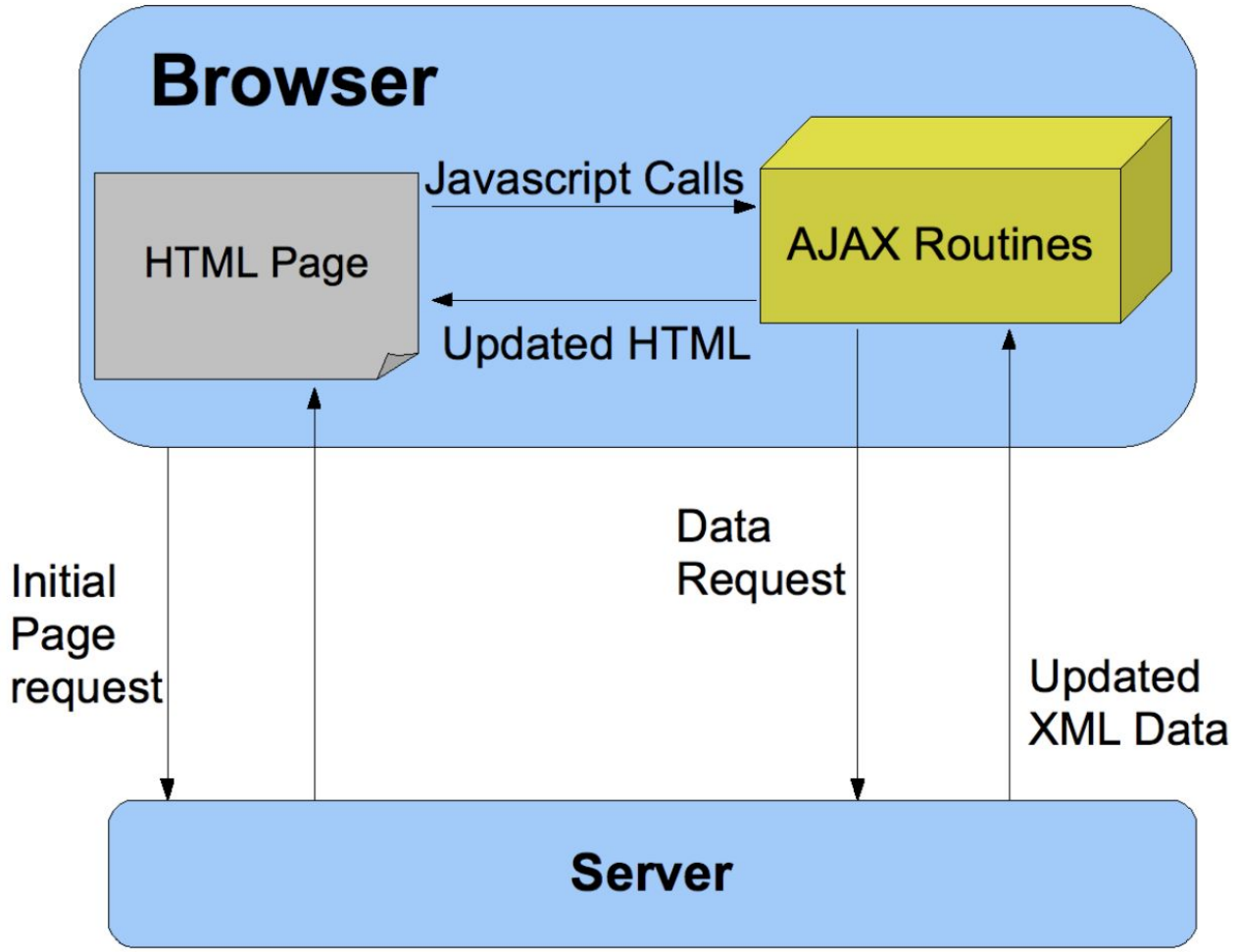
AJAX – Asynchronous JavaScript And XML

XHR – XMLHttpRequest

XMLHttpRequest и AJAX – «исторические» названия

https://www.w3schools.com/JS/js_ajax_intro.asp

Взаимодействие клиента с backend. AJAX и XHR



Взаимодействие клиента с backend. AJAX и XHR

```
1.  const myRequest = new XMLHttpRequest;
2.  myRequest.addEventListener('readystatechange', (event) => {
3.    if (myRequest.readyState !== XMLHttpRequest.DONE) {
4.      return;
5.    }
6.    if (myRequest.status === 200) {
7.      console.log(JSON.parse(myRequest.responseText));
8.    } else {
9.      console.log(myRequest.responseText);
10.   }
11. });
12. myRequest.open('GET', '/test.json', true);
13. myRequest.send();
```

Взаимодействие клиента с backend. AJAX и XHR

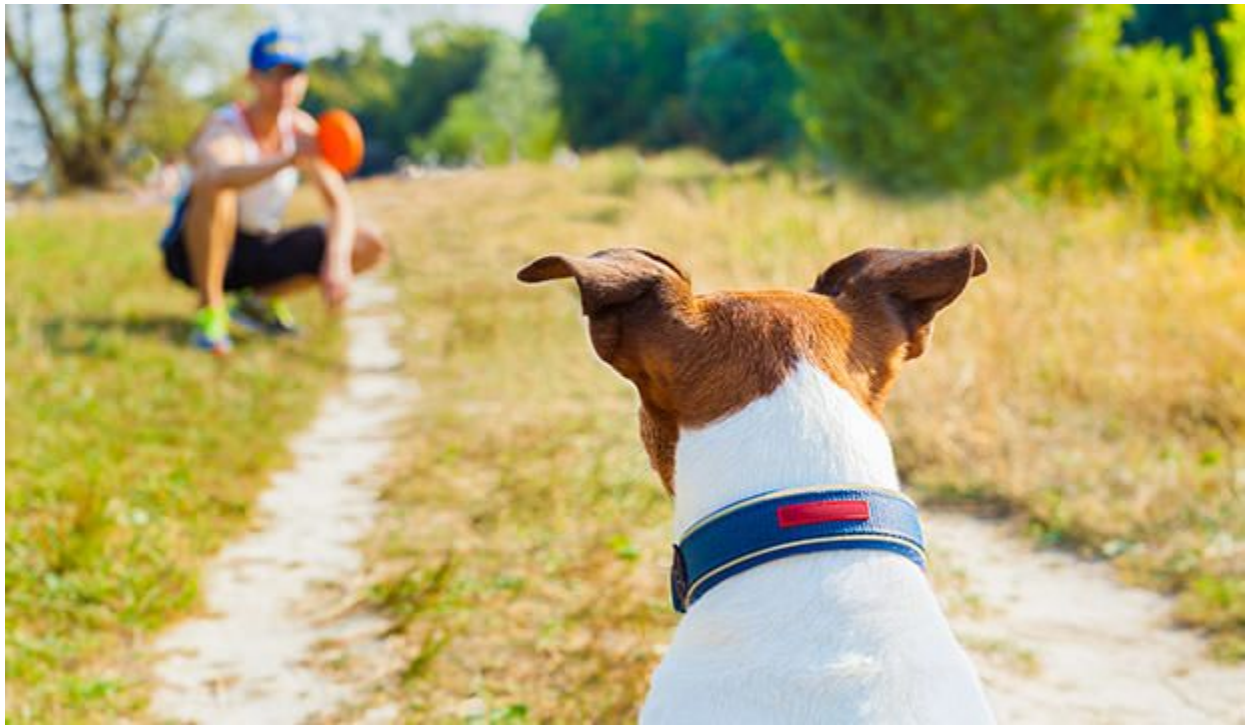
```
const myRequest = new XMLHttpRequest;
XMLHttpRequest.prototype.readyState;
// 0  UNSENT
// 1  OPENED
// 2  HEADERS_RECEIVED
// 3  LOADING
// 4  DONE

XMLHttpRequest.prototype.onreadystatechange = () => {};

XMLHttpRequest.prototype.response/responseText

XMLHttpRequest.prototype.status/statusText

```

Взаимодействие клиента с backend. AJAX и XHR

Читать больше про XHR:

- javascript.info: XMLHttpRequest
- MDN: XMLHttpRequest
- MDN: Using XMLHttpRequest

Взаимодействие клиента с backend. Fetch



Взаимодействие клиента с backend. Fetch

- Считается удобной заменой XHR
- Возвращает Promise
- Отдельные объекты запроса, ответа и заголовков
- 404, 500, etc – вернут fulfilled Promise
- В качестве аргумента принимает строку или объект Request

```
// fetch(input[, init]);
const myRequest = fetch('/test.json');
myRequest.then(
    response => response.ok && response.json()
).then(console.log);
```

Взаимодействие клиента с backend. Fetch

Объект инициализации

1. method, body, headers
2. mode, credentials
3. cache, redirect

Взаимодействие клиента с backend. Fetch

```
// Пример отправки POST запроса:
postData('http://example.com/answer', {answer: 42})
  .then(data => console.log(JSON.stringify(data))) // JSON-строка полученная после вызова `response.json()`
  .catch(error => console.error(error));

function postData(url = '', data = {}) {
  // Значения по умолчанию обозначены знаком *
    return fetch(url, {
        method: 'POST',     // *GET, POST, PUT, DELETE, etc.
        mode: 'cors',       // no-cors, cors, *same-origin
        cache: 'no-cache',  // *default, no-cache, reload, force-cache, only-if-cached
        credentials: 'same-origin', // include, *same-origin, omit
        headers: {
            'Content-Type': 'application/json',
            // 'Content-Type': 'application/x-www-form-urlencoded',
        },
        redirect: 'follow', // manual, *follow, error
        referrer: 'no-referrer',        // no-referrer, *client
        body: JSON.stringify(data),     // тип данных в body должен соответствовать значению заголовка "Content-Type"
    })
    .then(response => response.json()); // парсит JSON ответ в Javascript объект
}
```

Взаимодействие клиента с backend?

Вопросы?

Real Time Updates

RTU. Polling

Запрос с клиента на сервер каждые N секунд.

Ответ с сервера возвращается сразу.

Ниже пример реализации polling для получения сообщений из общего чата.

```
const pollItems = () => {
  fetch('https://tt-front.vercel.app/messages/')
    .then(resp => resp.json())
    .then(data => console.log(data));
}

const t = setInterval(() => pollItems(), 1000);

// clearInterval(t)

```

RTU. Long polling

Запрос с клиента на сервер, где:

- Сервер отвечает только когда будет, чем ответить
- После получения ответа от сервера выполняется новый запрос

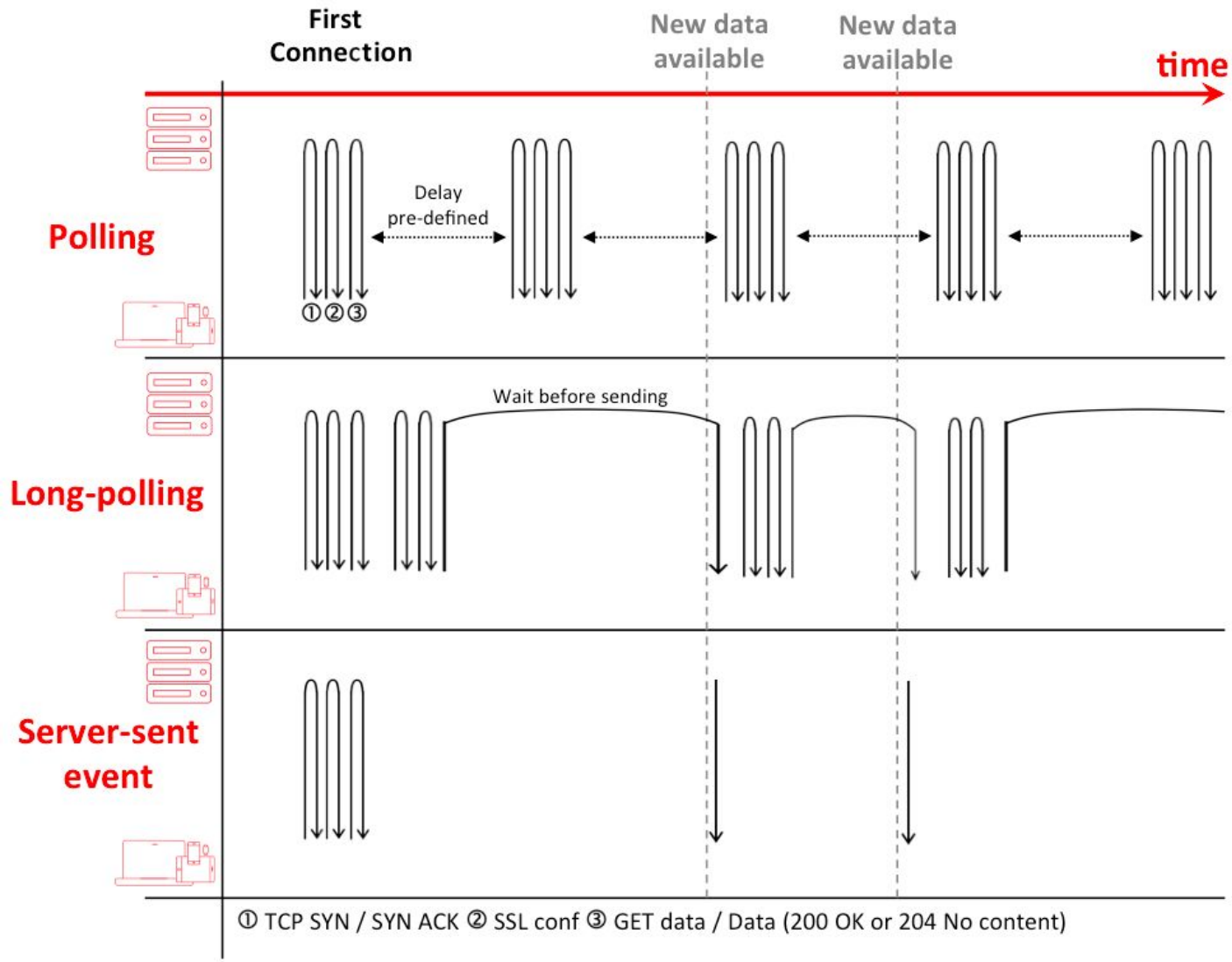
RTU. SSE

Server Sent Events / EventSource

Клиент устанавливает соединение с сервером.

Сервер отправляет данные по установленному соединению.

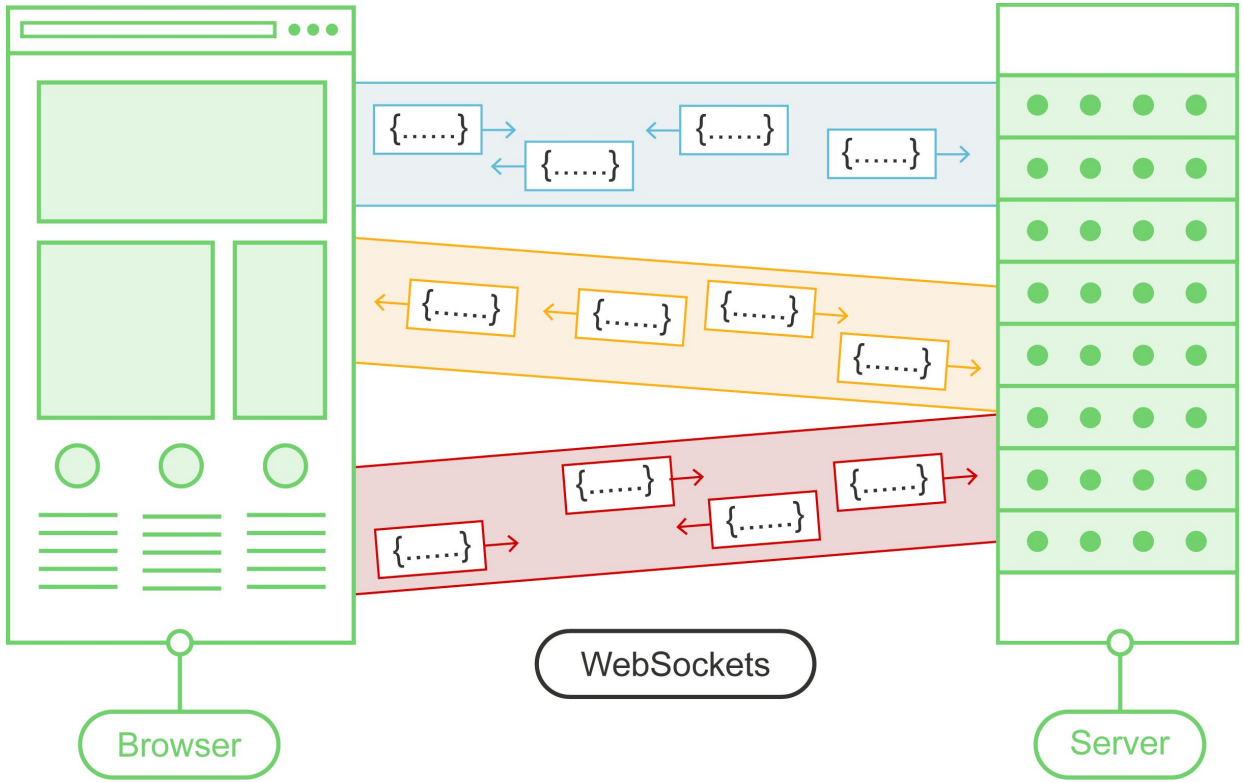
RTU. Polling/Long polling/SSE



RTU. Websocket

Клиент устанавливает соединение с сервером.

И клиент, и сервер могут отправлять данные друг другу.



RTU

What are Long-Polling, Websockets, Server-Sent Events (SSE) and Comet?

https://stackoverflow.com/a/12855533/3984110

Инструменты разработчика

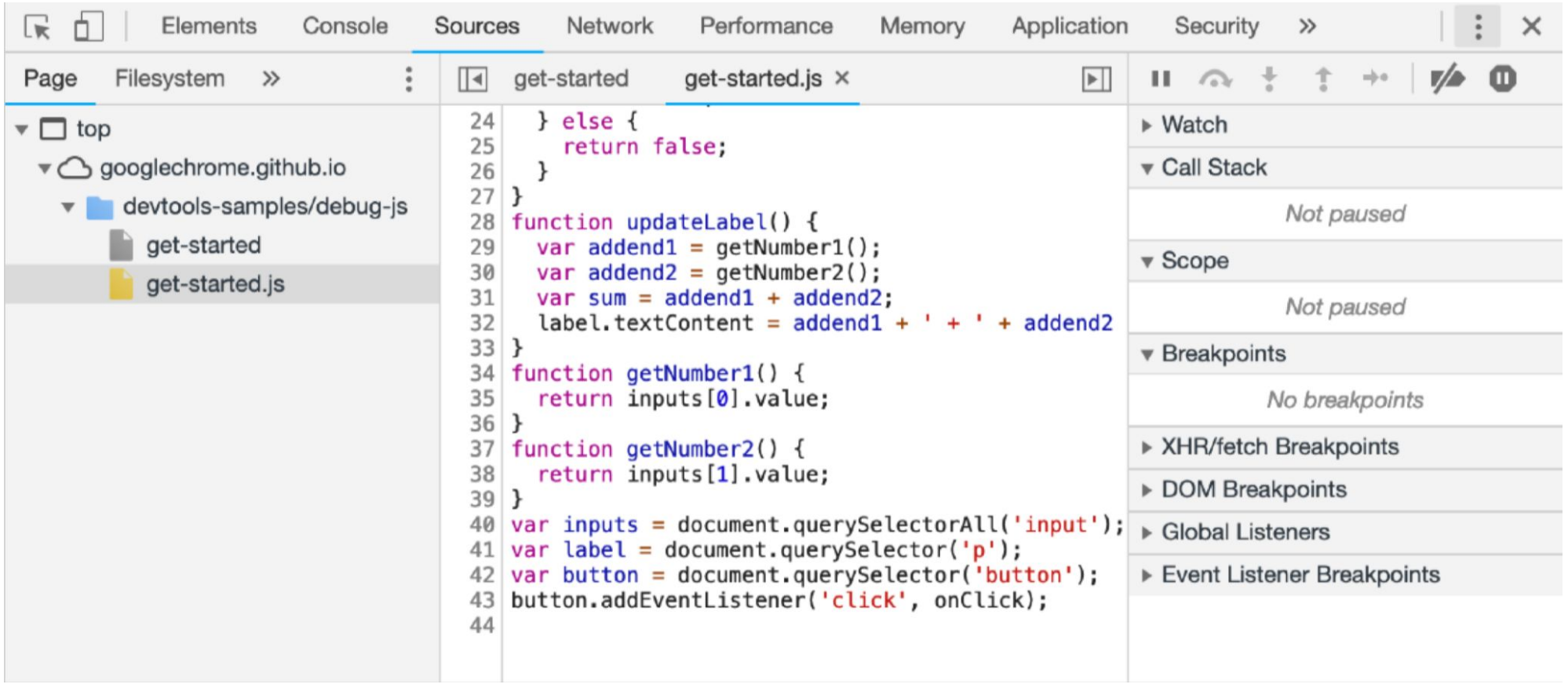
Инструменты разработчика

Ресурс для испытаний:

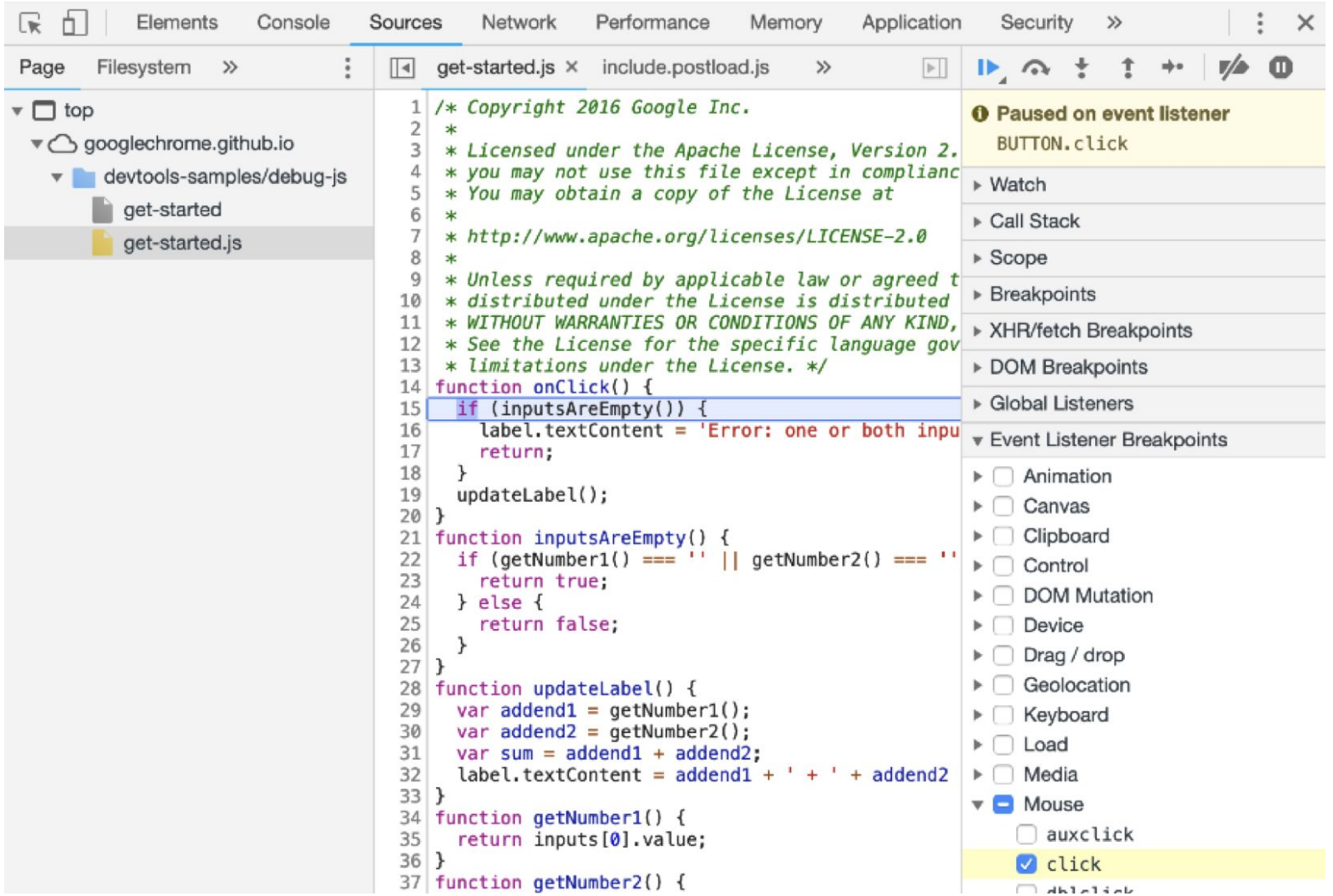
- https://googlechrome.github.io/devtools-samples/debug-js/get-started

Можно использовать cmd+s / ctrl+s для сохранений изменений

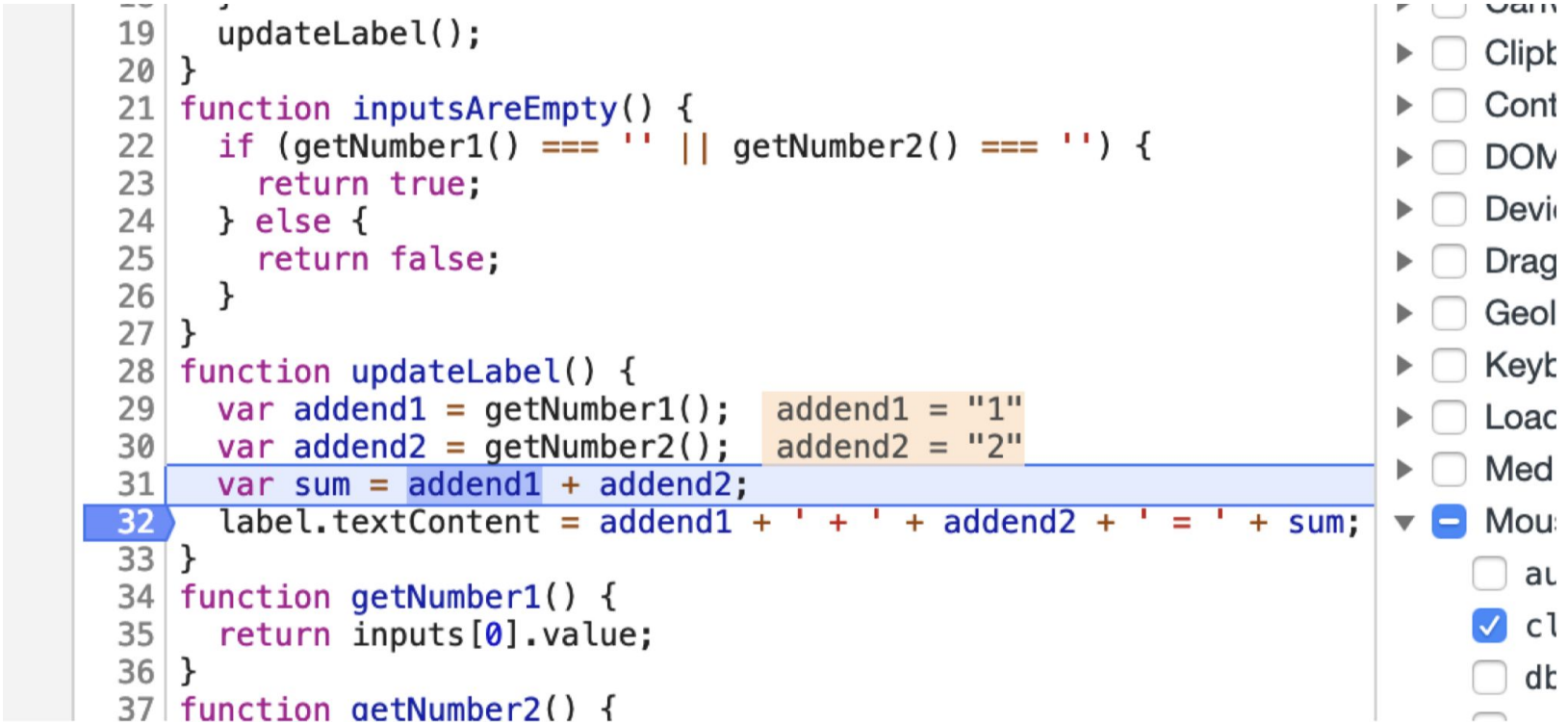
Инструменты разработчика



Инструменты разработчика



Инструменты разработчика



Инструменты разработчика

Инструменты разработчика

- Вкладка Network
- Показать SSE

Инструменты разработчика?

Вопросы?

Домашнее задание №7

1. Создать отдельный "общий" чат
2. Передавать новые сообщения на backend
3. Получать новые сообщения с backend в режиме реального времени (RTU)
4. Можно использовать любую из технологий для RTU (polling, sse, ws)

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

7 ноября

Мем дня (если до вас еще не дошло)

Спасибо за внимание!

education

Пока!

Присоединяйтесь к сообществу про образование в VK

- VK Образование

education