Лекция 6
SPA

Дмитрий Зайцев

образование

План на сегодня

- Введение
- Теория
  - MPA
  - SPA
  - SPA vs. MPA
  - SSR
  - SEO
  - Fetch
  - Short polling
  - Long polling
  - Websocket
  - SSE
  - History API
  - React-router
- Практика

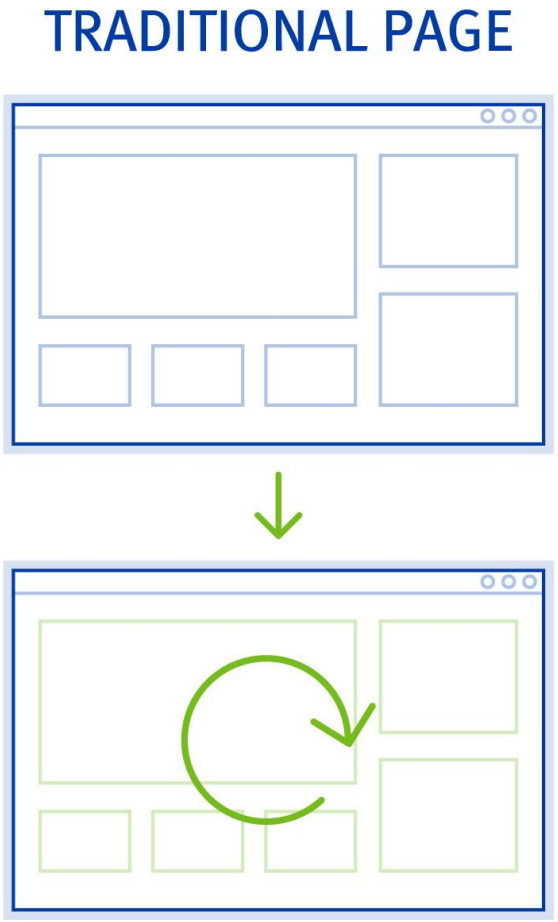
Минутка бюрократии

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

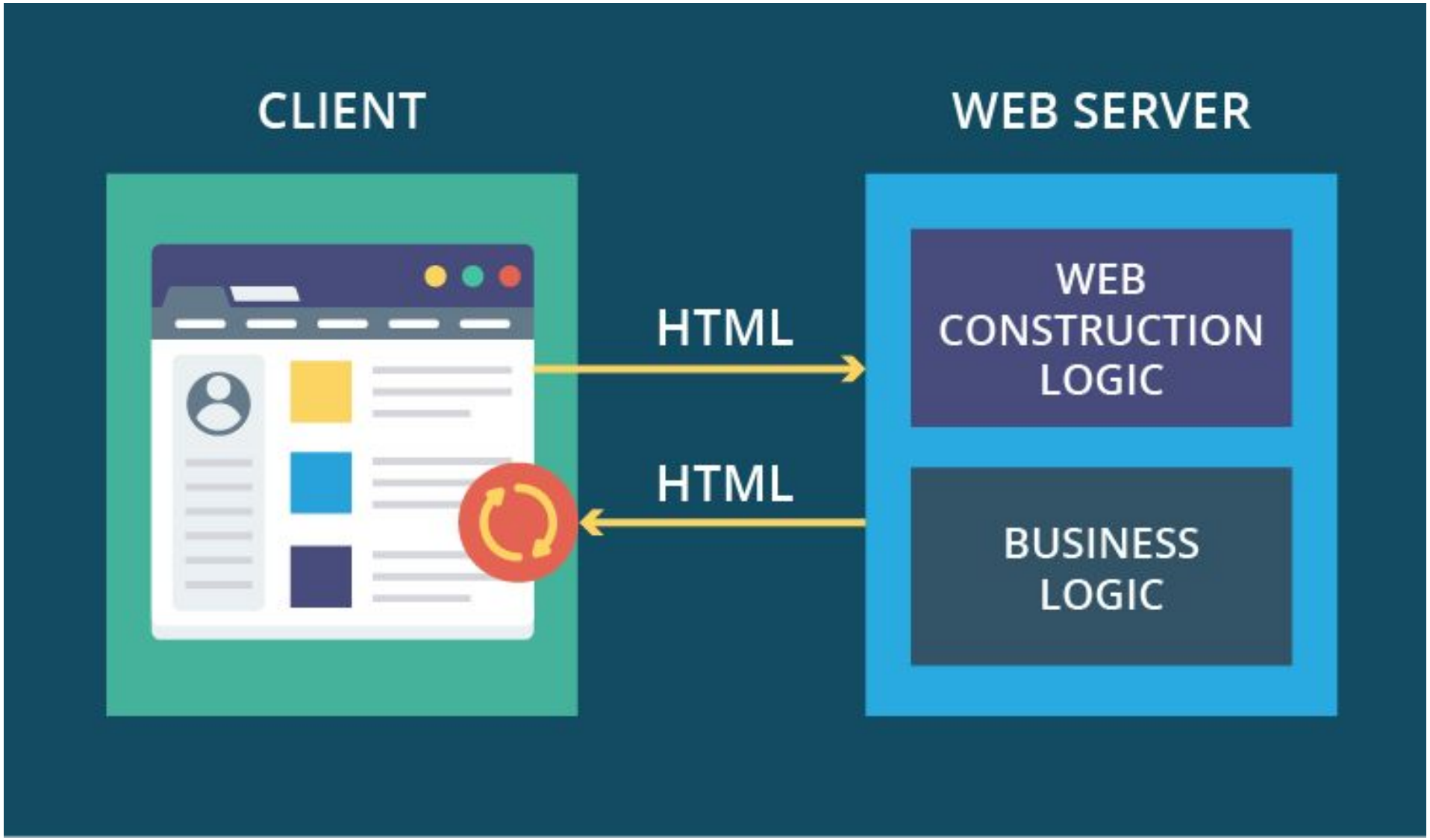
Как было раньше

MPA

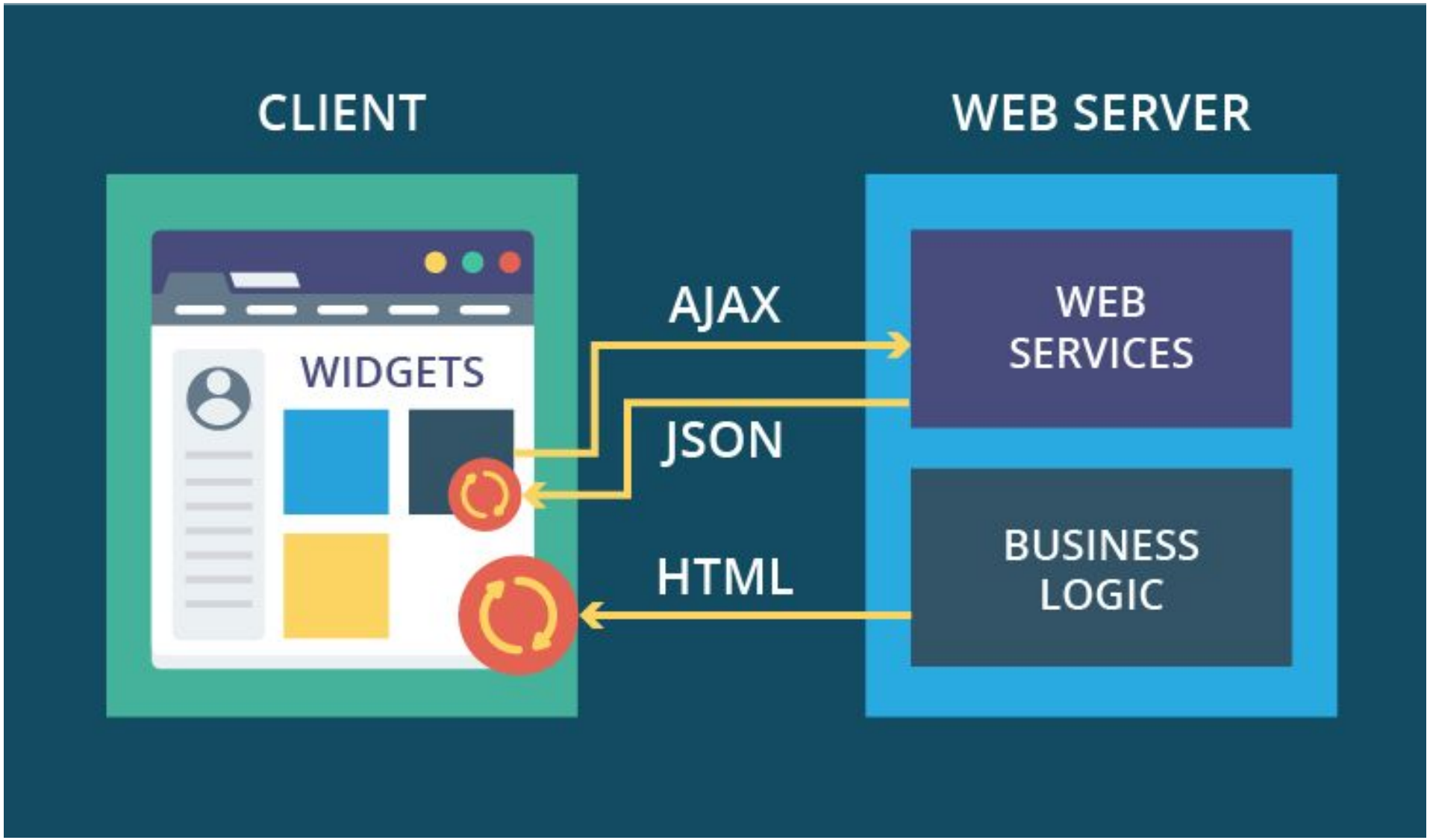
MPA (Multiple Page Application) - *многостраничное приложение (website).*



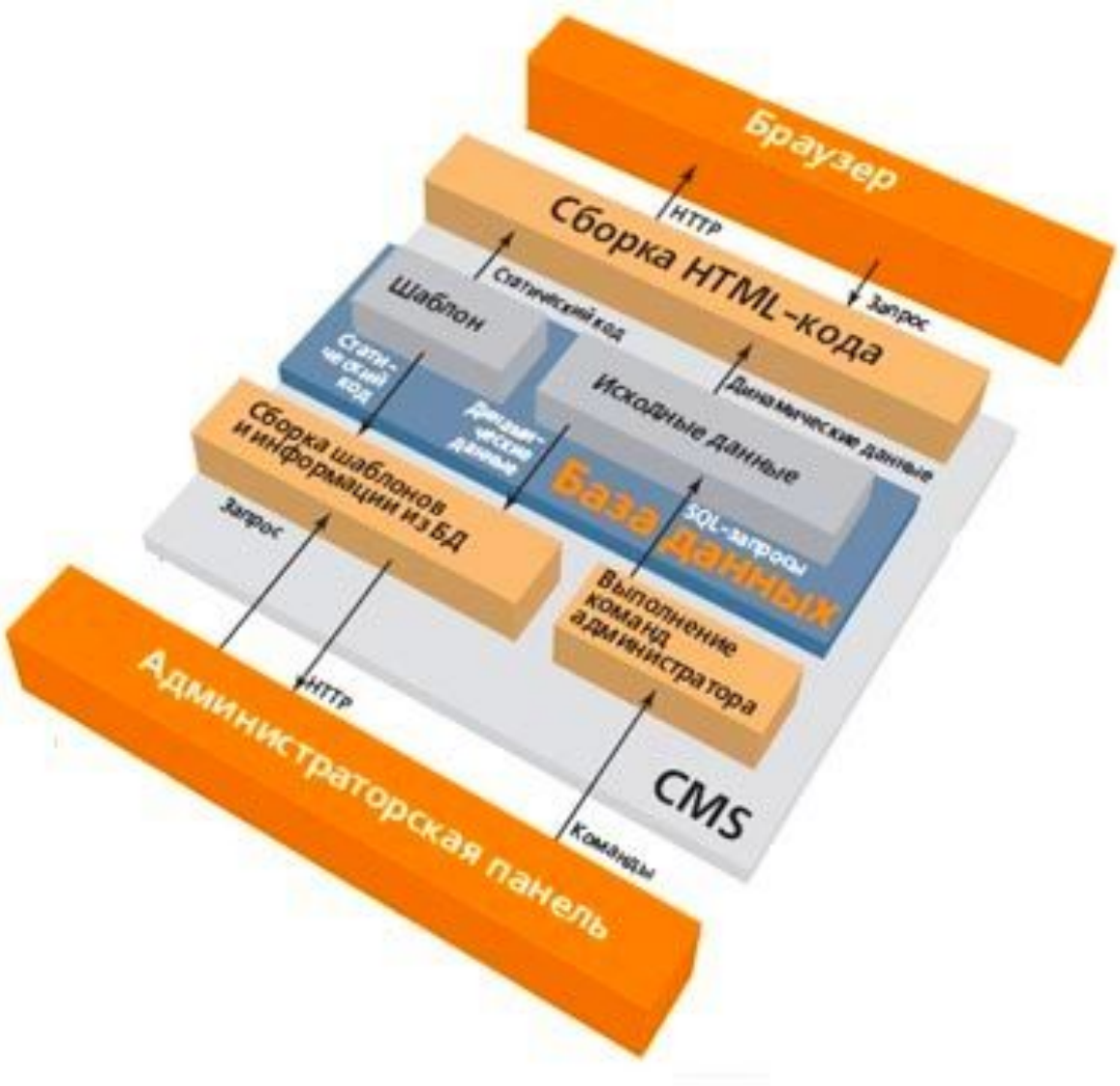
MPA. Архитектура старого HTML сайта



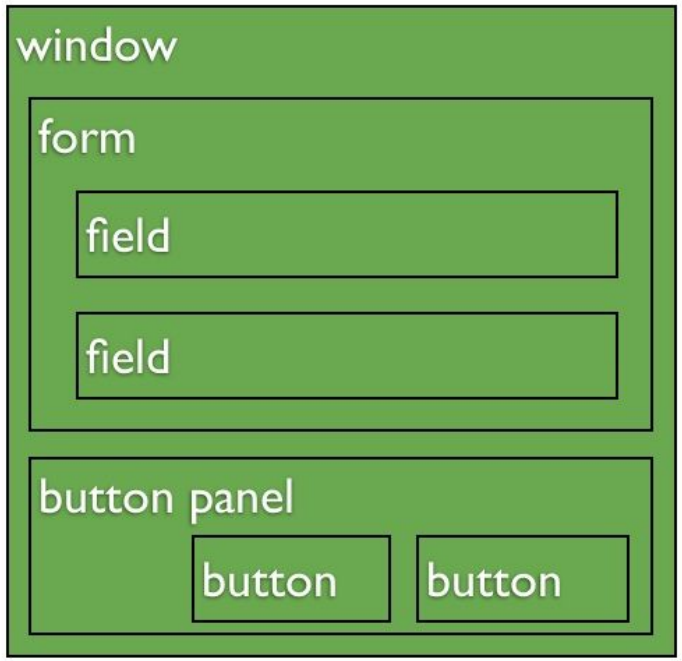
MPA. Архитектура чуть менее старого сайта с виджетами



MPA. Архитектура. CMS



MPA. Подход к построению верстки. Монолит



MPA. Плюсы и минусы

Плюсы

- Лучший выбор для простого старта
- Безопасность (сокрытие опасных и важных данных)
- SEO из коробки
- Могут работать без JS
- Стоимость разработки
- Множество готовых решений

Минусы

- Скорость работы приложения
- Скорость разработки
- Сложность
- Монолитность
- Обслуживание и обновление
- Высокая связность бекенда с фронтендом
- Легко начать делать плохие решения и костыли

MPA. Примеры фреймворков и CMS

Фреймворки и CMS, которые работают по принципу MPA

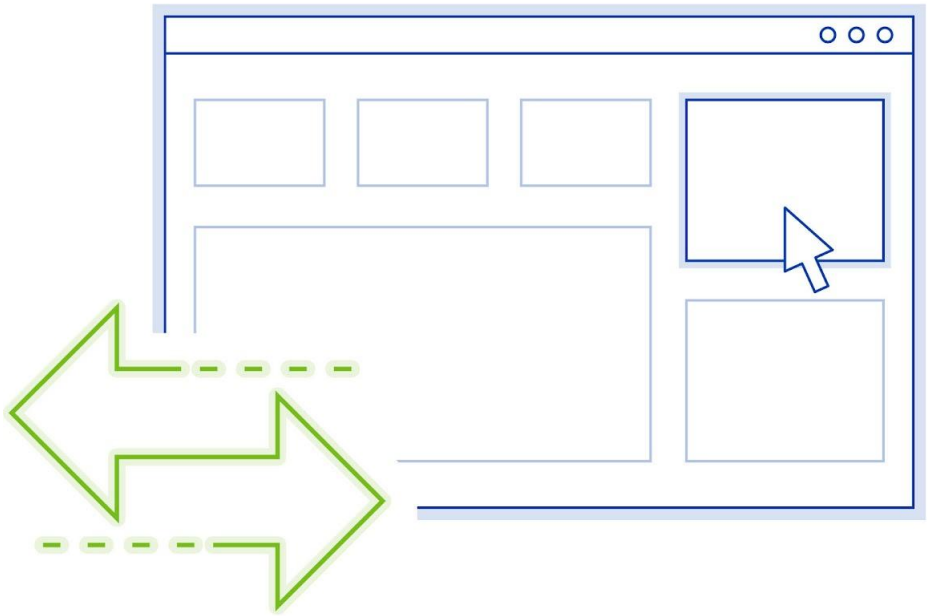
- Wordpress
- Joomla
- Drupal
- Django CMS
- Bitrix

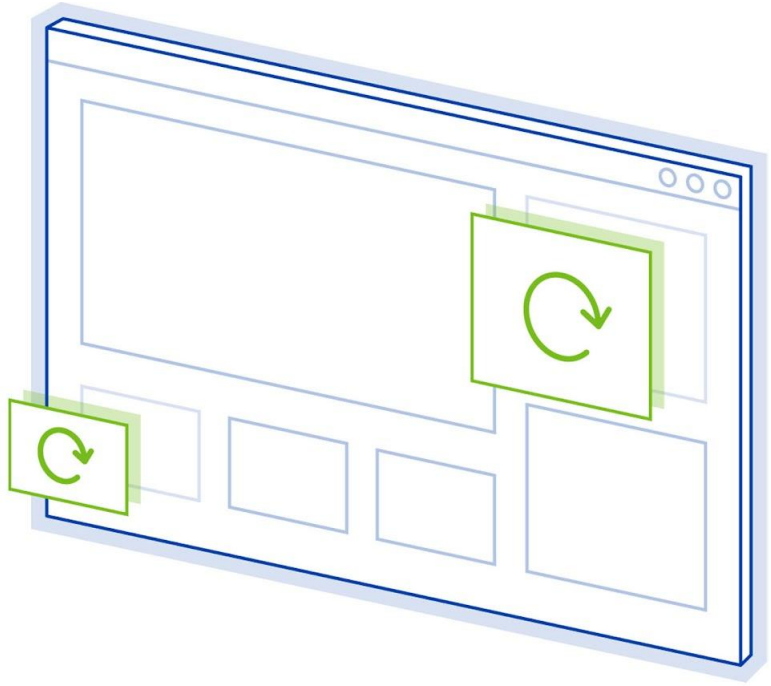
Как стало сейчас

SPA

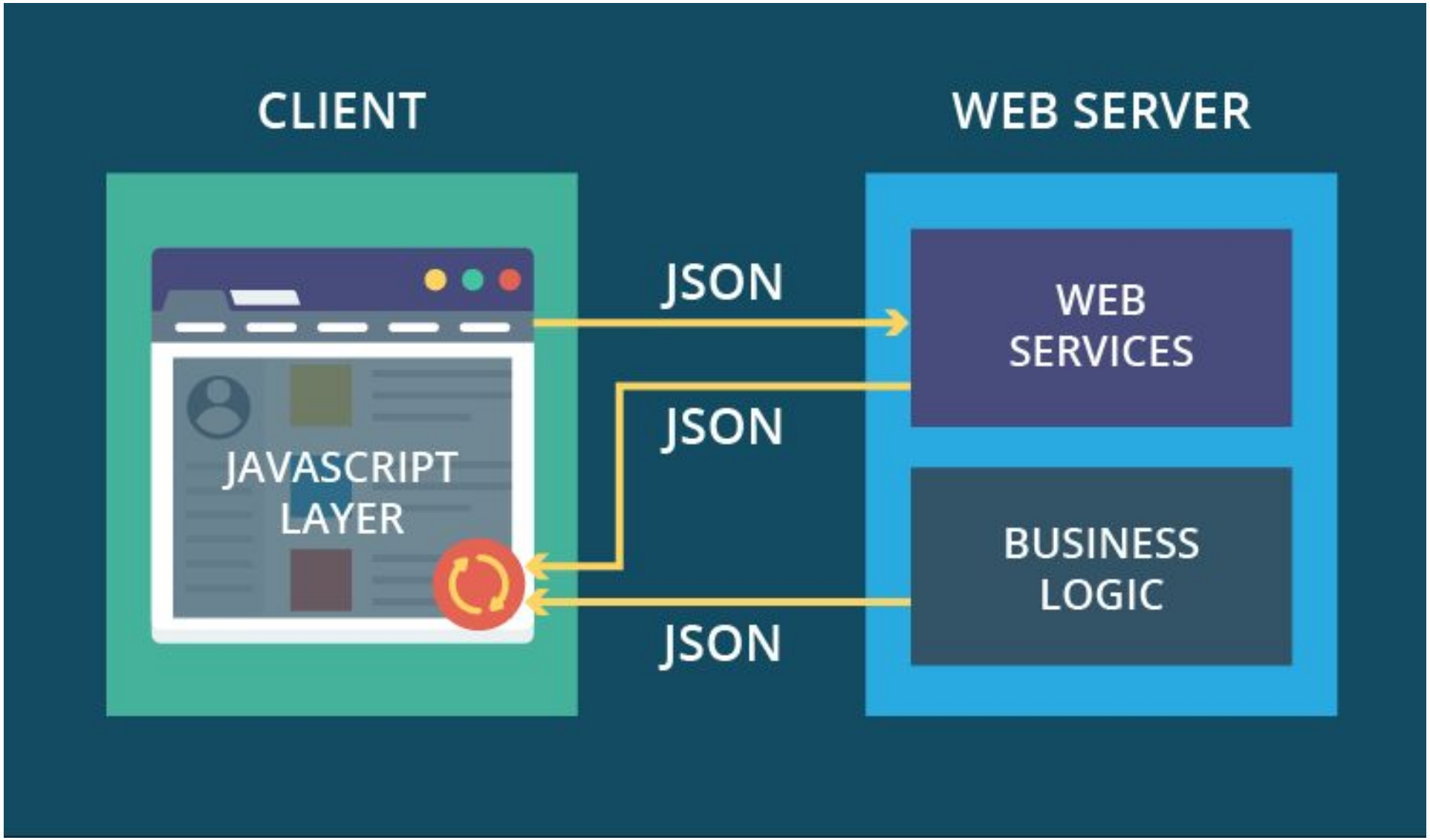
SPA

SPA (Single Page Application) - *одностраничное приложение (website).*

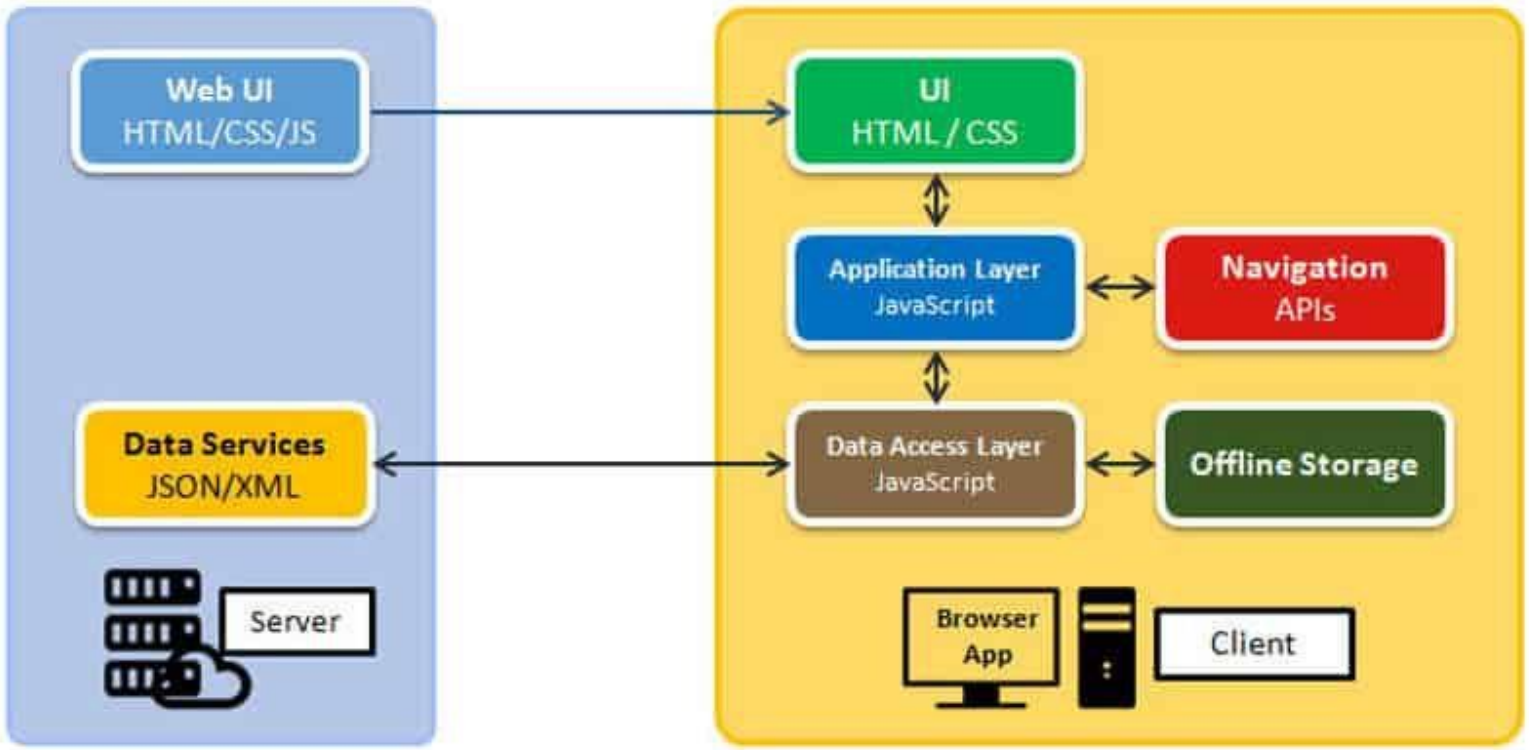




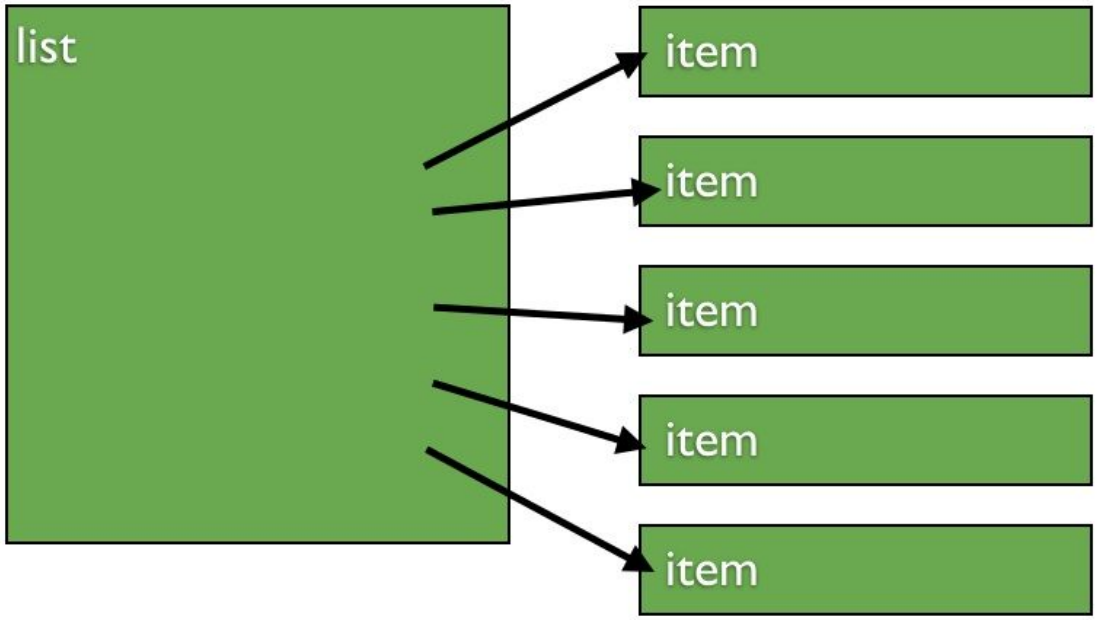
SPA. Архитектура

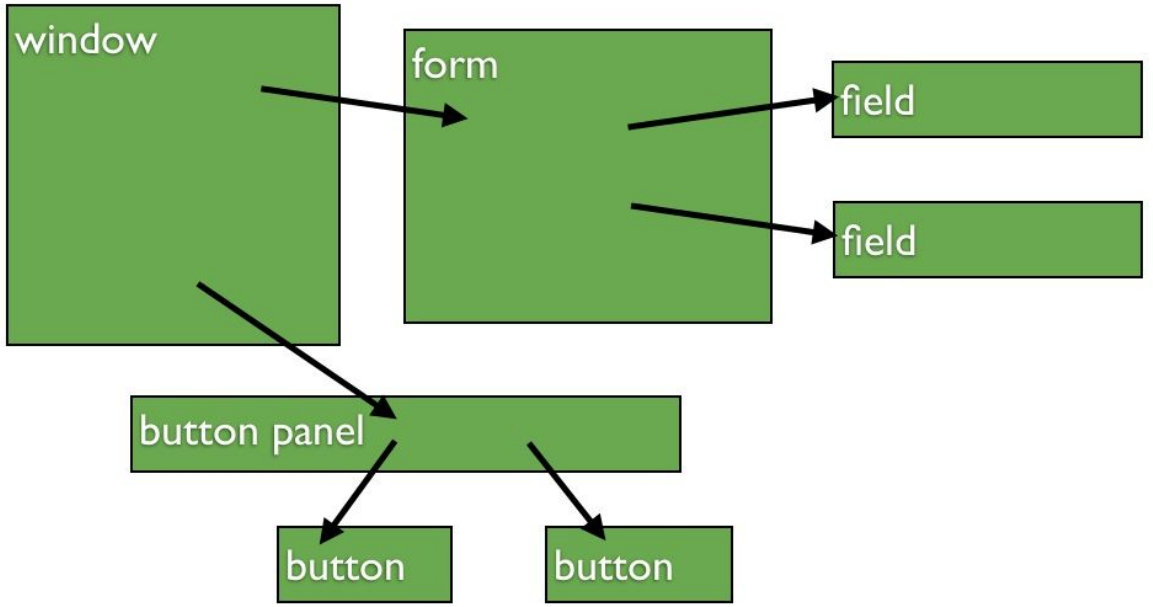


SPA. Архитектура



SPA. Подход к построению верстки. Компоненты





SPA. Компоненты

- Компоненты могут быть достаточно сложны внутри, но они должны быть просты для использования снаружи
- Компонентом может быть вообще всё что угодно, что выполняет какую-то функцию в вашем приложении

Виды компонентов:

- **Dummy-компоненты** — компоненты, которые либо вообще не содержат никакой логики (чисто визуальные компоненты), либо содержат логику, которая глубоко инкапсулирована внутри компонента
- **Smart-компоненты** — компоненты, которые управляют множеством других компонентов, содержат в себе бизнес-логику и хранят какое-то состояние

SPA. Плюсы и минусы

Плюсы

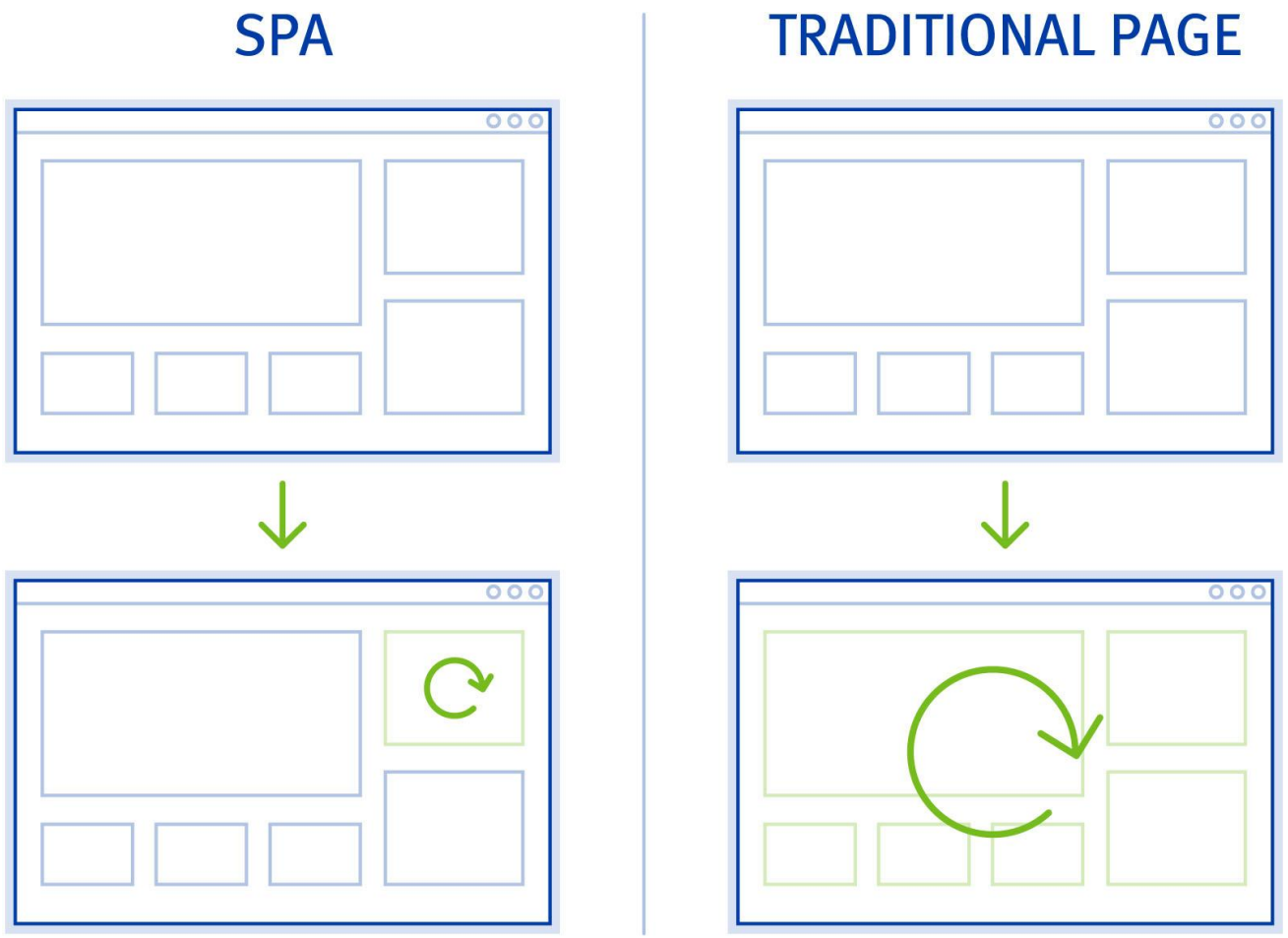
- Скорость работы приложения
- Экономия трафика
- Возможности кэширования
- Возможность работы оффлайн
- Простота отладки
- Низкая связность бэкенда с фронтендом
- Возможность сделать SPA мобильным приложением
- Кроссплатформенность (отличающиеся части в зависимости от UA)

Минусы

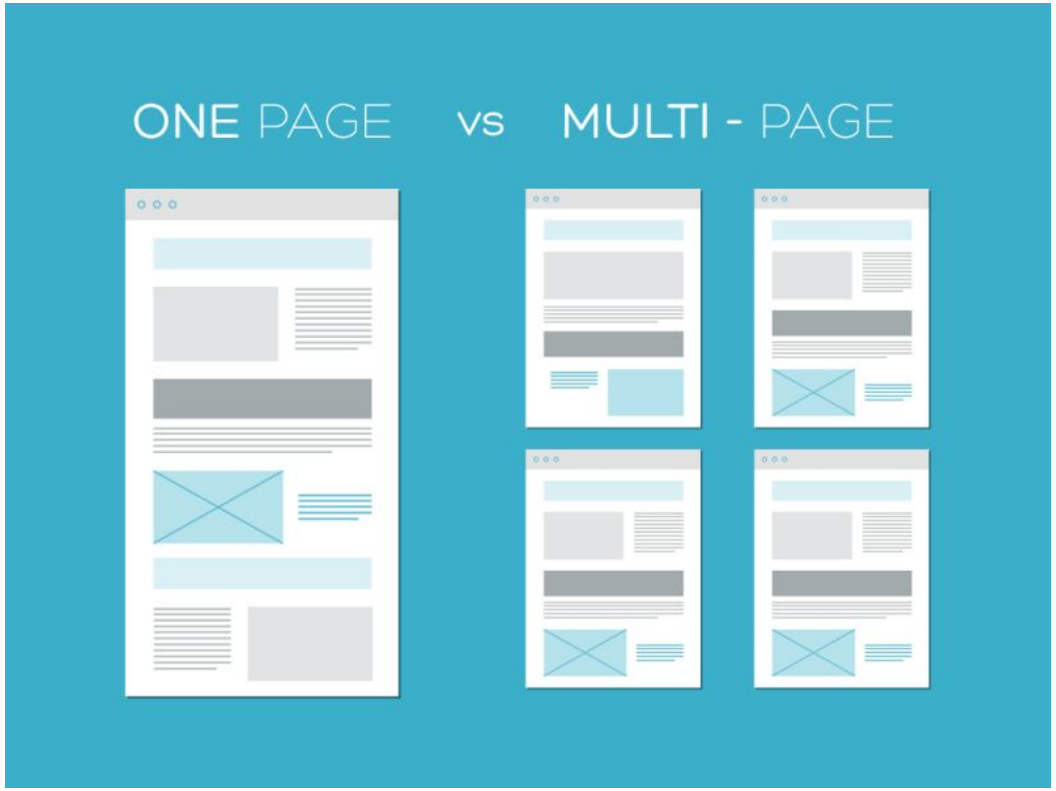
- Отсутствие SEO
- Нужно самому имитировать историю переходов
- Нужно сильнее следить за безопасностью и изначальными данными
- Разработка
- Возможность утечек памяти на клиенте

SPA vs MPA

SPA vs MPA



SPA vs MPA



Что выбрать?

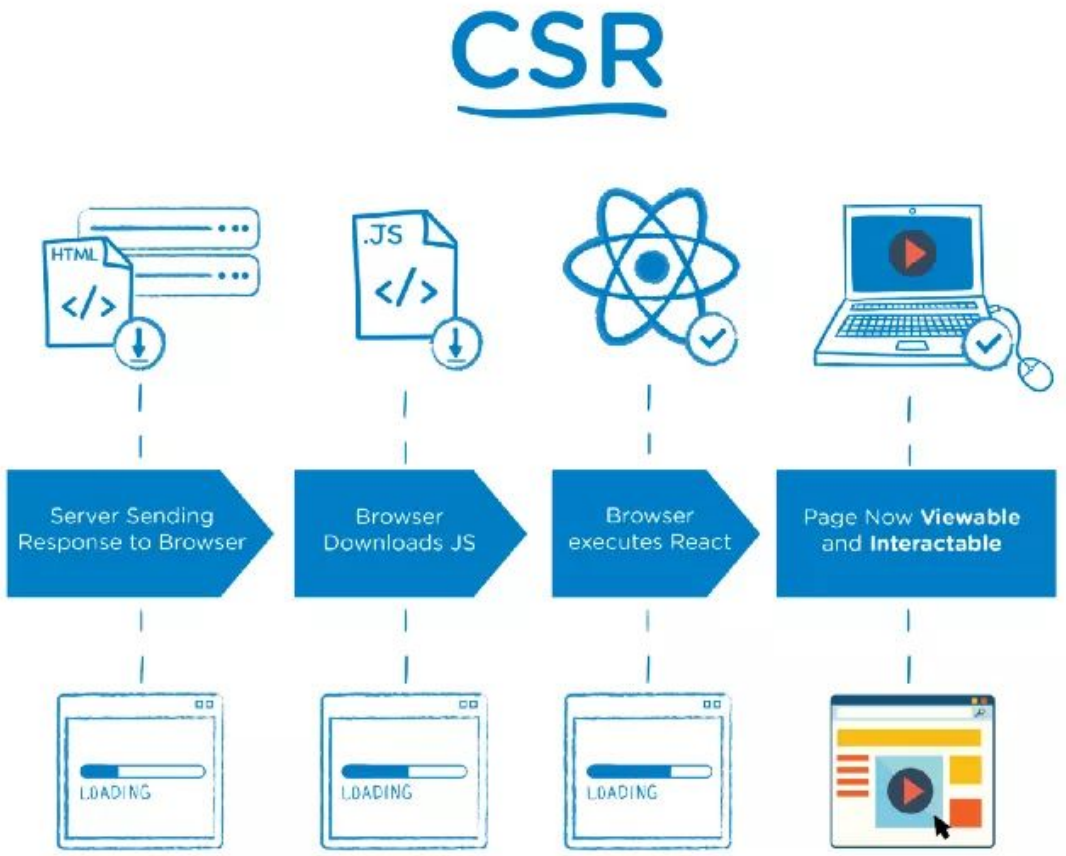
Нужно оценить:

- Цель проекта
- Специфику проекта
- Бюджет
- Квалификацию разработчиков
- Сроки
- Тенденции в разработке

Как становится сейчас и как будет дальше

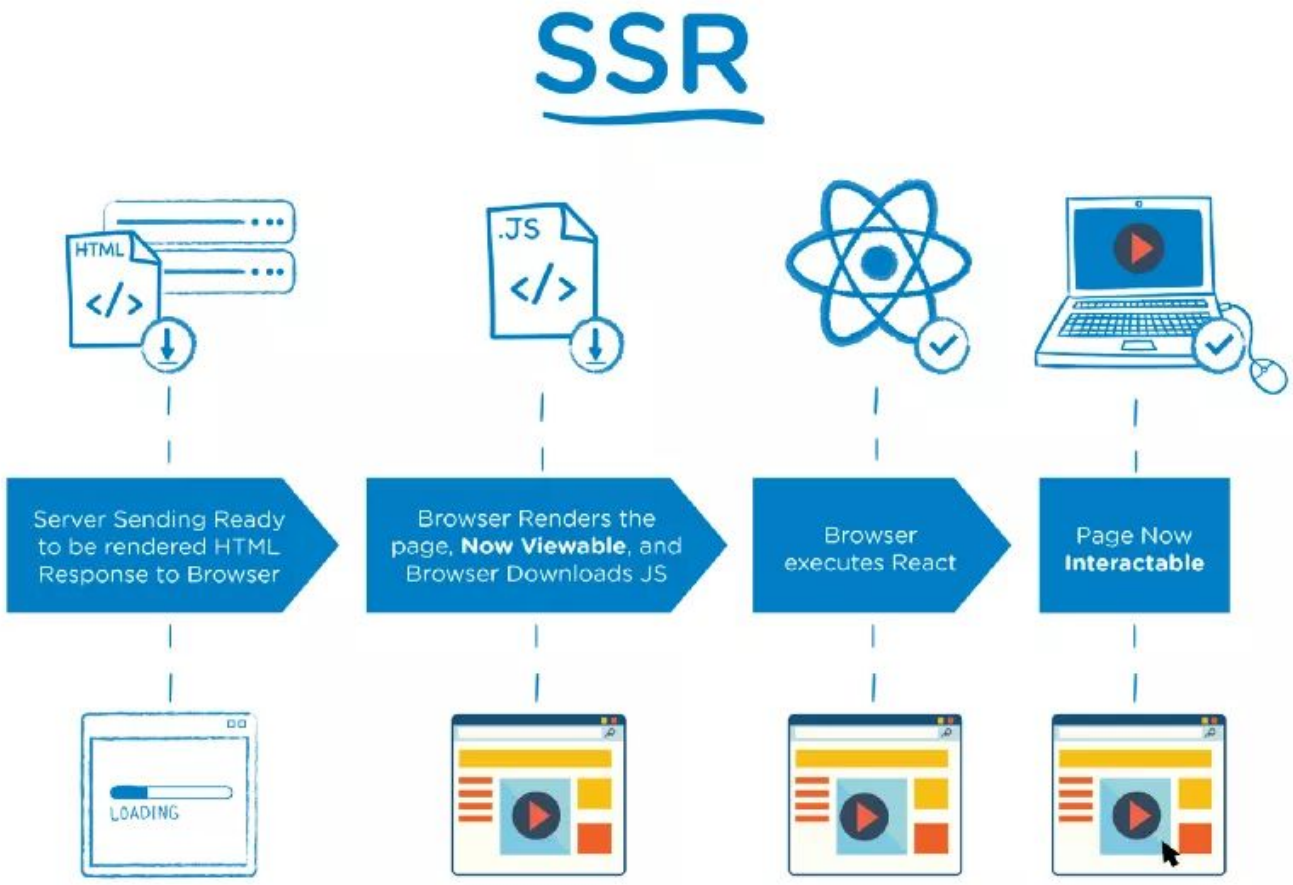
CSR

CSR (Client Side Rendering) - *отрисовка динамического HTML на фронтенде.*



SSR

SSR (Server Side Rendering) - *отрисовка динамического HTML на бекенде.*



SEO

SEO (Search Engine Optimization, поисковая оптимизация) – *это всестороннее развитие и продвижение сайта для его выхода на первые позиции в результатах выдачи поисковых систем (SERPs) по выбранным запросам с целью увеличения посещаемости и дальнейшего получения дохода.*

SEO, как работает?

Виды коммуникаций с сервером

Fetch

Fetch API предоставляет интерфейс JavaScript для работы с запросами и ответами *HTTP*. Он также предоставляет глобальный метод ***fetch()***, который позволяет легко и логично получать ресурсы по сети асинхронно.

- ***Promise*** возвращаемый вызовом ***fetch()*** не перейдет в состояние "отклонено" из-за ответа *HTTP*, который считается ошибкой, даже если ответ *HTTP* 404 или 500. Вместо этого, он будет выполнен нормально (с значением false в статусе ok) и будет отклонён только при сбое сети или если что-то помешало запросу выполниться.

- По умолчанию, ***fetch*** не будет отправлять или получать cookie файлы с сервера, в результате чего запросы будут осуществляться без проверки подлинности, что приведёт к неаутентифицированным запросам, если сайт полагается на проверку пользовательской сессии (для отправки cookie файлов в аргументе *init options* должно быть задано значение свойства *credentials* отличное от значения по умолчанию *omit*).

Fetch, пример

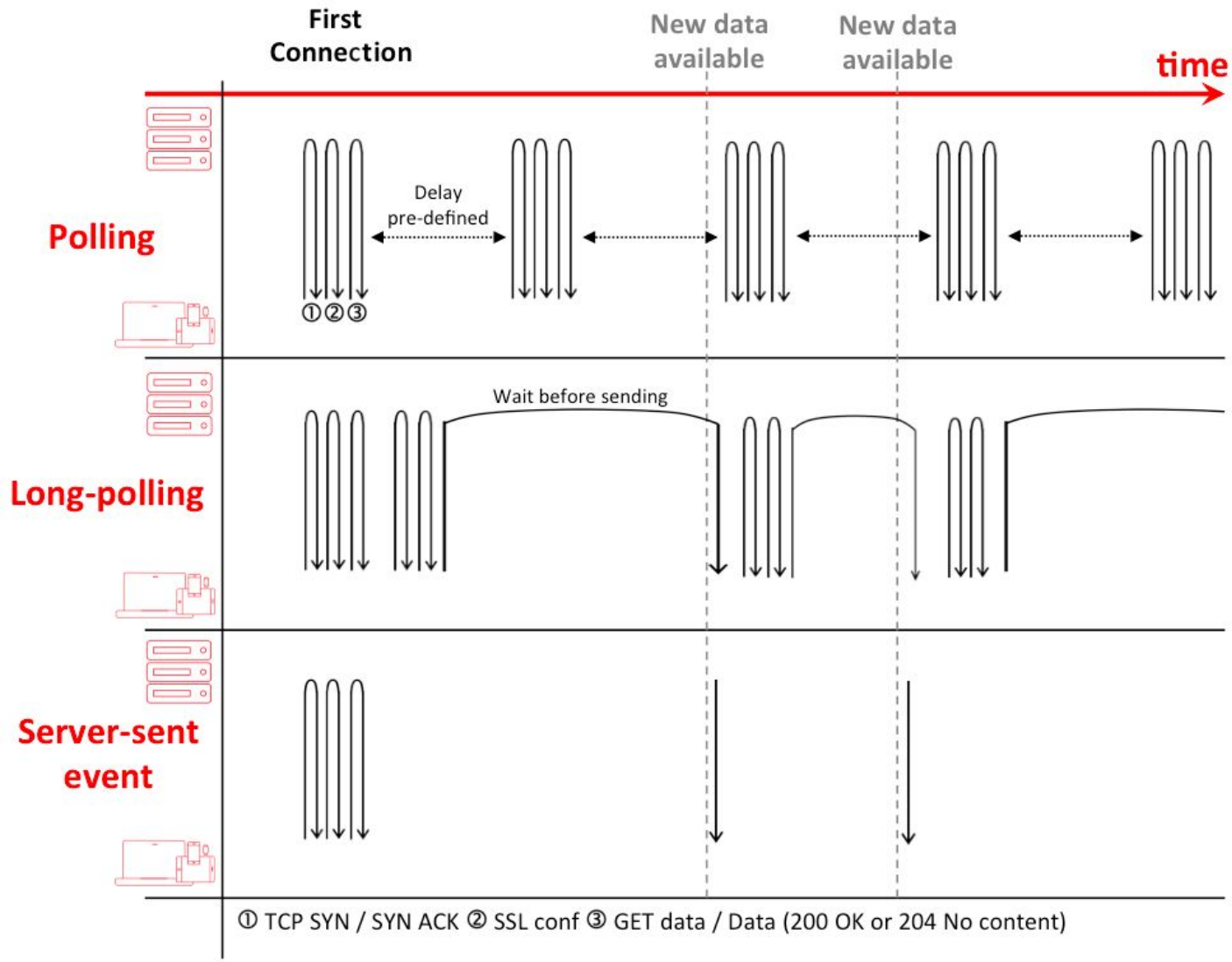
```
// Пример отправки POST запроса:
postData('http://example.com/answer', {answer: 42})
  .then(data => console.log(JSON.stringify(data))) // JSON-строка полученная после вызова `response.json()`
  .catch(error => console.error(error));

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
}
```

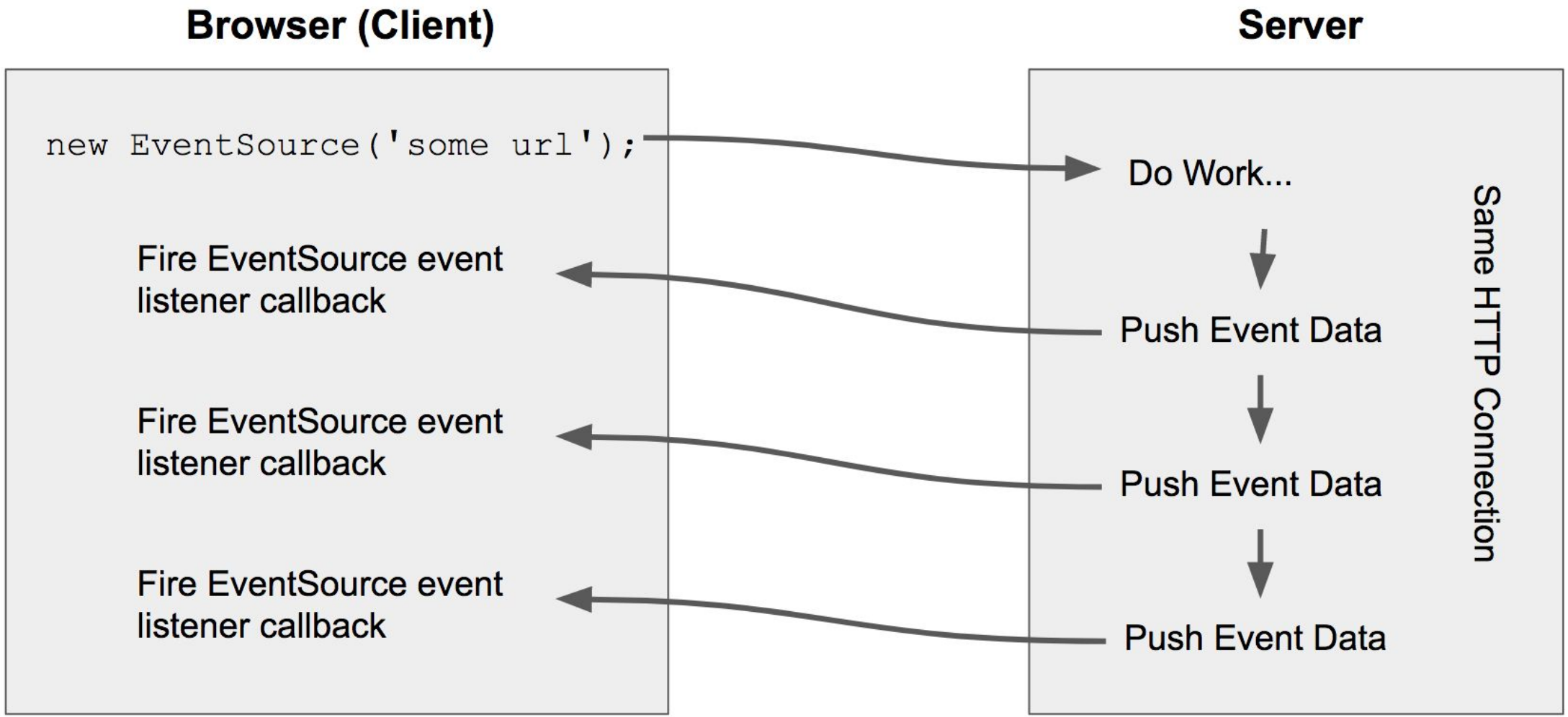
Виды коммуникаций с серверами

- Polling (client pull)
- Long polling (client pull)
- Websocket (server push)
- SSE (server push)

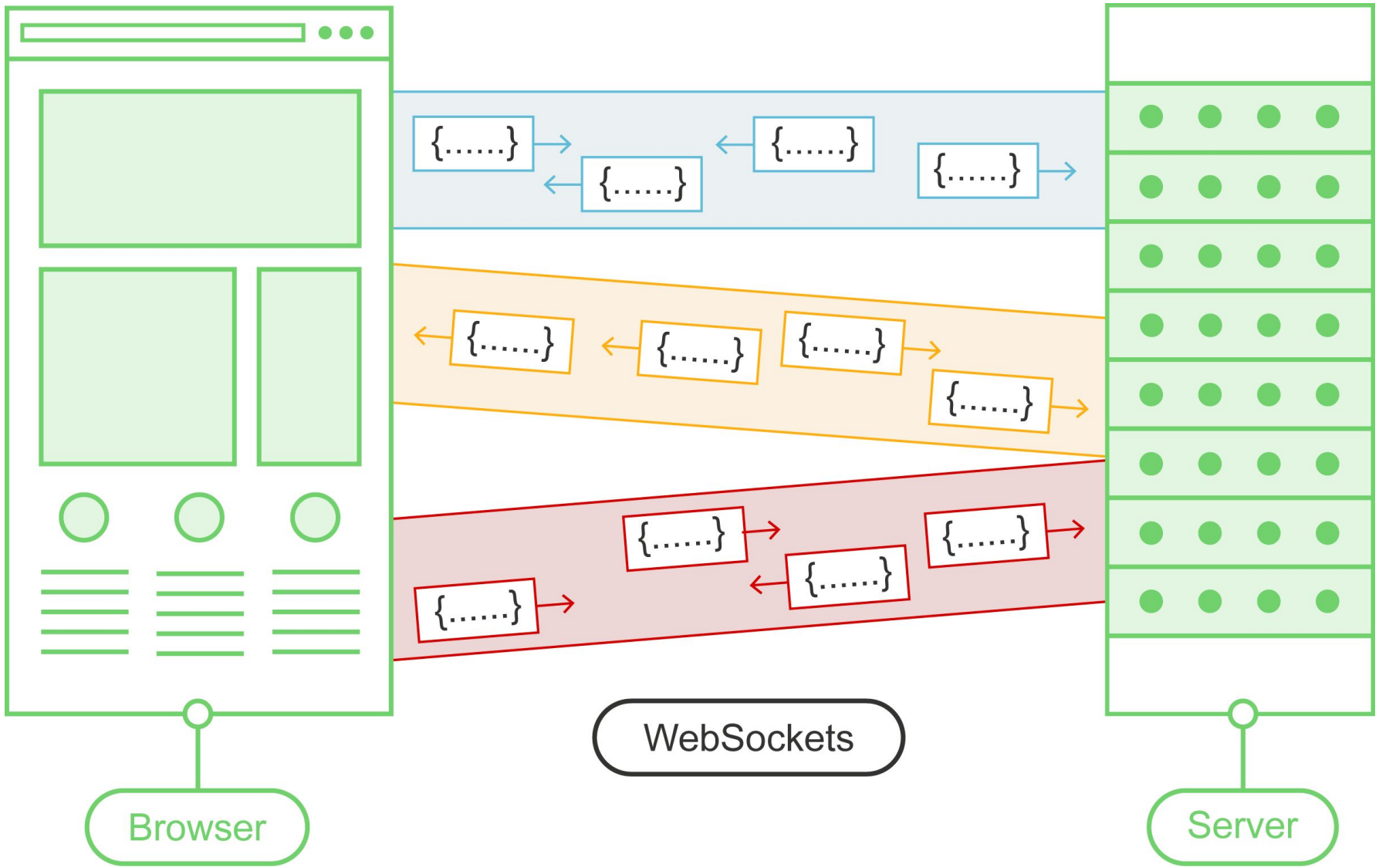
Polling/Long polling/SSE



SSE


Browser (Client)
Server
new EventSource('some url');
Do Work...
Fire EventSource event listener callback
Push Event Data
Fire EventSource event listener callback
Push Event Data
Fire EventSource event listener callback
Push Event Data
Same HTTP Connection

Websocket



Вопросы?

Вопросы?

Перерыв! (10 минут)

Препод (с)

Практика

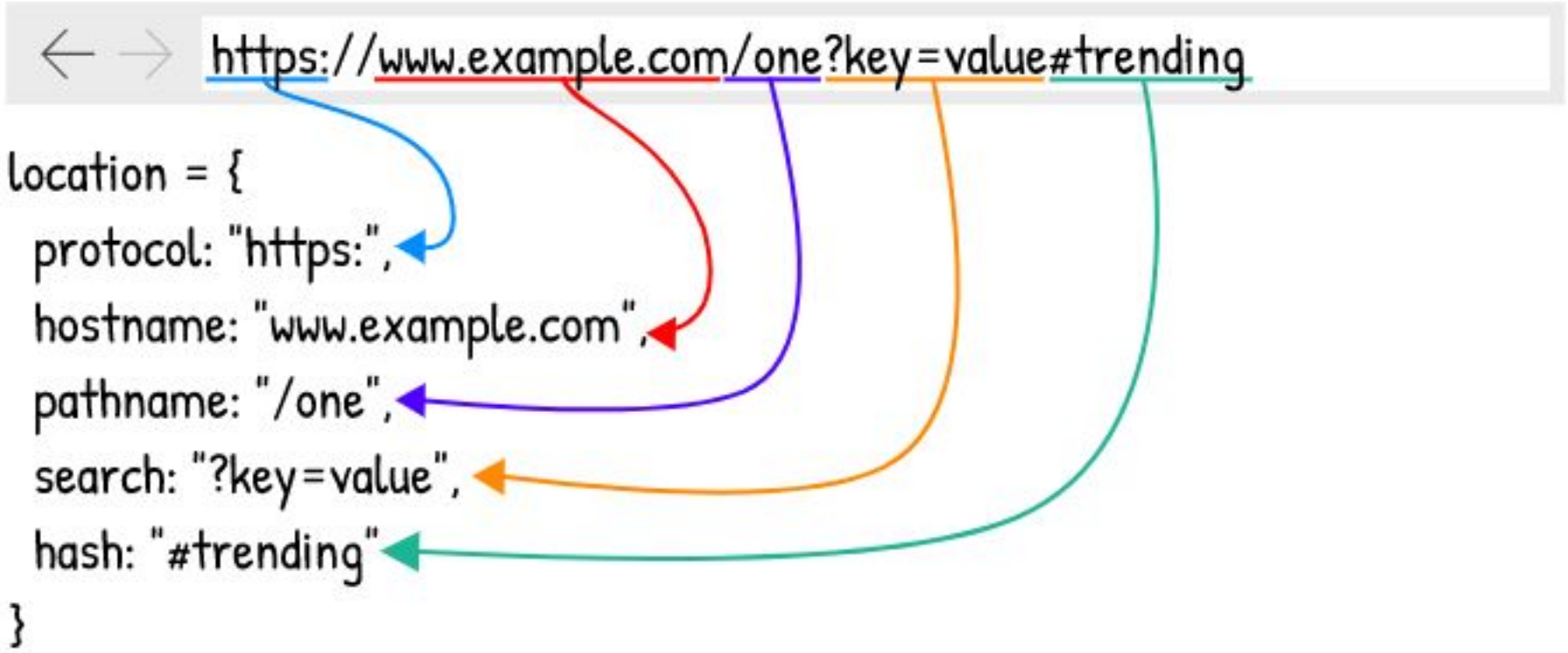
HISTORY API

History API

History API — браузерное API, позволяет манипулировать историей браузера в пределах сессии, а именно историей о посещённых страницах в пределах вкладки или фрейма, загруженного внутри страницы. Позволяет перемещаться по истории переходов, а также управлять содержимым адресной строки браузера

History API

Свойства объекта **window.location**



History API

Методы объекта **window.history**

```
window.history.back();
```

```
window.history.forward();
```

```
window.history.go();
```

```
window.history.pushState();
```

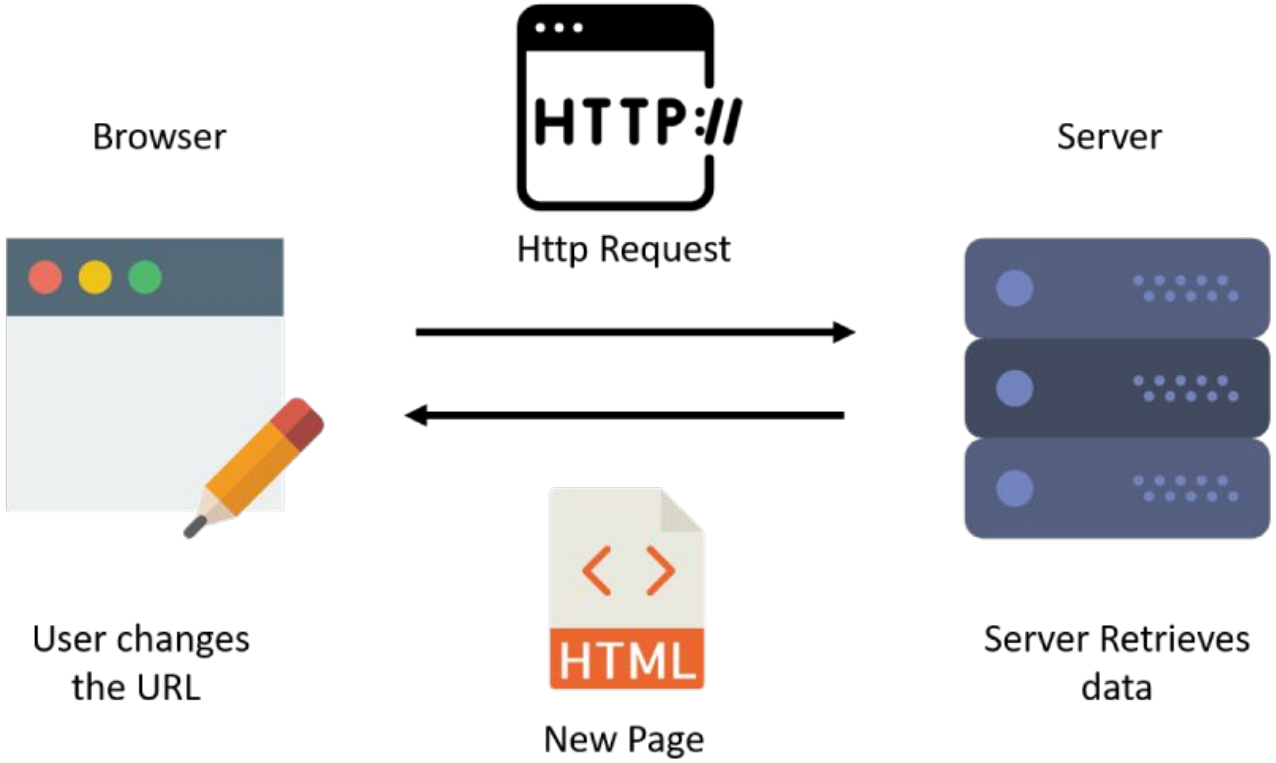
```
window.history.replaceState();
```

react-router

react-router

react-router - это библиотека, включающая в себя набор навигационных компонентов и логику, которая следит за состоянием ***HistoryAPI***. С её помощью можно построить всю маршрутизацию ***react*** приложения.

Server-side маршрутизация:



SPA. Полезные ссылки

- polling-vs-sse-vs-websocket-how-to-choose-the-right-one
- next-js-vs-create-react-app
- single-page-apps-or-multiple-page-apps-whats-better-for-web-development
- single-page-application-vs-multiple-page-application
- single-page-applications
- enlightenment-of-single-page-website-single-page-wordpress-themes.html
- single-page-application-vs-multi-page-application
- Что такое SEO
- History API
- Working with History API
- React Router
- https://medium.com/webbdev/react-9cf527ed63a7
- Fetch API
- Using Fetch

Домашнее задание №6

1. Закрепить знания про SPA
2. Изучить react-router
3. Настроить клиентскую маршрутизацию для своих страниц
4. Сверстать страницу профиля

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

Срок сдачи

17 ноября

Спасибо за внимание!

образование

Пока!

Присоединяйтесь к сообществам про образование в VK

- VK Джуниор
- VK Образование

образование