Манипулирование историей веб-обозревателя и создание одностраничных приложений

Кулаков Кирилл Александрович

История браузера

- HTML5 History API
  - **window.history.state** Возвращает текущий объект истории. С помощью этого объекта можно получить данные, которые передаются методами pushState и replaceState путем доступа к свойствам этого объекта.
  - **window.history.length** Свойство lenght показывает количество записей в истории.
  - **window.history.go(n)** Переход к определенной позиции в истории, в качестве аргумента передается смещение относительно текущий позиции. Этот метод существовал до появления HTML5.
  - **window.history.back()** Переход к предыдущему элементу в истории, идентично вызову go(- 1).
  - **window.history.forward()** Переход к следующему элементу в истории, идентичный вызову go(1).

История браузера

- HTML5 History API
  - **window.history.pushState(data, title [, url])** Добавляет новый элемент в историю. Метод принимает три параметра:
    - Данные состояния истории. Эти данные можно получить затем в обработчике события popstate. Если дополнительные данные не требуются можно передавать null;
    - Заголовок страницы, который отобразится в окне браузер, так же можно передавать null;
    - URL, который должен отображаться в адресной строке.
  - Пример:

    history.pushState({param: 'Value'}, '', 'myurl.html');
  - **window.history.replaceState(data, title [, url])** изменение записи в истории браузера
  - **popstate** Это событие срабатывает при переход от одного элемента истории к другому. При этом history.pushState() и history.replaceState() не приводят к вызову этого события. Только нажатие кнопок вперед/назад в браузере, либо вызов history.back() или аналогичной функции в Javascript.
  - Пример:

    window.addEventListener('popstate', function(e) {

    // код обработчика события

    });

Подходы к разработке веб приложений

- Одностраничные веб приложения (SPA) — это одностраничное веб-приложение, которое загружается на одну HTML-страницу.
- Многостраничные веб приложения (MPA) — это многостраничные приложения, которые работают по традиционной схеме.
- Прогрессивные веб приложения (PWA) — взаимодействуют с пользователем, как приложение. Они могут устанавливаться на главный экран смартфона, отправлять push-уведомления и работать в офлайн-режиме.

Одностраничные приложения

- SPA или Single Page Application — это одностраничное веб-приложение, которое загружается на одну HTML-страницу.
- Благодаря динамическому обновлению с помощью JavaScript, во время использования не нужно перезагружать или подгружать дополнительные страницы.
- На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки нужные элементы просто подгружаются.
- В процессе работы пользователю может показаться, что он запустил не веб-сайт, а десктопное приложение, так как оно мгновенно реагирует на все его действия, без задержек и «подвисаний».
- Такого эффекта удается добиться с помощью продвинутых фреймворков JavaScript: Angular, React, Ember, Meteor, Knockout.
- Примеры динамических приложений: Gmail, Google Maps, Facebok, GitHub, Meduza.

Одностраничные приложения

- Преимущества
  - Высокая скорость — все ресурсы загружаются за одну сессию, а во время действий на странице данные просто меняются, что очень экономит время;
  - гибкость и отзывчивость пользовательского интерфейса — за счет того, что веб-страница всего одна, проще построить насыщенный интерфейс, хранить сведения о сеансе, управлять состояниями представлений и анимацией;
  - упрощенная разработка — код можно начинать писать с файла file://URL, не используя сервер, не нужен отдельный код для рендера страницы на стороне сервера;
  - кэширование данных — приложение отправляет всего один запрос, собирает данные, а после этого может функционировать в offline-режиме.

Одностраничные приложения

- Недостатки
  - Seo оптимизация требует решений в виде серверного рендеринга — из-за того, что контент загружается при помощи технологии AJAX, которая подразумевает динамическое изменение содержания станицы, а для оптимизации важна устойчивость;
  - нагрузка на браузер — из-за того, что клиентские фреймворки тяжелые, они довольно долго загружаются;
  - необходима поддержка JavaScript — без JS нельзя полноценно пользоваться полным функционалом приложения;
  - утечка памяти в Java Script — из-за плохой защиты, SPA больше подвержена действиям злоумышленников и утечке памяти.

Многостраничные приложения

- MPA или Multi Page Application — это многостраничные приложения, которые работают по традиционной схеме.
- Это означает, что при каждом незначительном изменении данных или загрузке новой информации страница обновляется.
- Такие приложения тяжелее, чем одностраничные, поэтому их использование целесообразно только в тех случаях, когда нужно отобразить большое количество контента.

Многостраничные приложения

- Преимущества
  - Простая SEO оптимизация — можно оптимизировать каждую из страниц приложения под нужные ключевые запросы;
  - привычность для пользователей — за счет простого интерфейса и классической навигации.

Многостраничные приложения

- Недостатки
  - Тесная связь между бакендом и фронтендом, поэтому их не получается развивать параллельно; сложная разработка — требуют использования фреймворков как на стороне клиента, так и на стороне сервера, что увеличивает сроки и бюджет разработки.
  - сложная разработка — требуют использования фреймворков как на стороне клиента, так и на стороне сервера, что увеличивает сроки и бюджет разработки.

Прогрессивные приложения

- Прогрессивные приложения или Progressive Web Application взаимодействуют с пользователем, как приложение.
- Они могут устанавливаться на главный экран смартфона, отправлять push-уведомления и работать в офлайн-режиме.
- Пример: Google Docs.

Прогрессивные приложения

- Преимущества
  - Кроссплатформенность — могут работать сразу с несколькими операционными системами;
  - высокая скорость работы и возможность запуска и отображения данных в офлайн-режиме с моментальной загрузкой;
  - высокая скорость установки;
  - быстрая разработка — для создания PWA, не нужен отдельный сайт, достаточно изменить уже существующий.
- Недостатки
  - Не все браузеры поддерживают основные функции таких приложений (например, Firefox и Edge).

Насыщенное интернет-приложение

- это веб-приложение, загружаемое пользователем через интернет, предназначенное для выполнения функций традиционных настольных приложений и работающее на устройстве пользователя (не на сервере).
- RIA состоит из двух частей: клиентской и серверной;
- серверная часть RIA выполняется на сервере, может хранить информацию, необходимую для работы приложения, может заниматься обслуживанием запросов, поступающих от клиентской части RIA;
- клиентская часть RIA выполняется на компьютере пользователя, занимается рисованием интерфейса пользователя, выполняет запросы пользователя, при необходимости может отправлять запросы серверной части RIA;
- клиентская часть RIA выполняется в безопасной среде, называемой «песочницей» (англ. sandbox), и не требует установки дополнительного ПО.

Насыщенное интернет-приложение

- Преимущества:
  - возможность использования в UI стандартных для ОС элементов управления (например, использование ползунка для изменения данных);
  - возможность использования типовых действий для взаимодействия с другими программами (например, drag-and-drop, копирование данных в буфер обмена);
  - возможность выполнения вычислений на устройстве пользователя (без отправки личных данных пользователя на сервер (например, ипотечный калькулятор));
  - гибкие возможности по построению UI (например, валидация введённых пользователем данных в процессе ввода без отправки запросов серверу (интерактивность));
  - возможность продолжения работы приложения после отправки запроса серверу (асинхронность);
  - возможность загрузки данных с сервера до того, как пользователь запросит данные (например, в Google Maps фрагменты карты, расположенные рядом с фрагментом, на который смотрит пользователь, загружаются заранее);
  - возможность снижения нагрузки на сервер (в случае выполнения вычислений на клиенте), и, следовательно, возможность повышения количества сессий, обрабатываемых сервером одновременно (без замены «железа»).

Насыщенное интернет-приложение

- Недостатки:
  - отсутствие доступа к ресурсам ОС (так как веб-приложение выполняется в «песочнице»). Если права на доступ к ресурсам некорректны, RIA могут работать неправильно;
  - для запуска веб-приложения может потребоваться выполнение кода, написанного на скриптовом языке (например, на JavaScript); если пользователь отключит возможность выполнения кода, RIA может работать неправильно или может вообще не работать;
  - низкая скорость работы многоплатформенных веб-приложений.
  - необходимость установки «движка клиента»;
  - продолжительное время загрузки веб-приложения. Клиент каждый раз загружает с сервера клиентскую часть RIA.
  - утрата целостности. Интерфейс DOM для X/HTML делает возможным создание RIA, но это не даёт никаких гарантий, что оно будет работать корректно.
  - невозможность индексирования веб-приложения поисковыми системами. Поисковые системы могут оказаться не в состоянии проиндексировать содержимое RIA.
  - зависимость от подключения к интернету. RIA, созданные для замены настольных приложений, должны позволять пользователям подключаться к сети по необходимости, например, не должны терять работоспособность при перемещении пользователя между зонами покрытия беспроводных сетей.

Server side rendering

- Server side rendering (SSR) — генерация страницы с содержимым на стороне сервера
  - При первом запросе на стороне сервера делаются все необходимые запросы к API для получения данных + "раскрывается" вся html
  - Дальнейшая работа приложения выполняется как SPA
  - Производительность при первой инициализации сайта на стороне клиента (без использования SSR) будет уступать многостраничным (MPA) приложением
  - Дальнейшая работа SPA намного быстрее MPA
  - Если использовать SPA + SSR, то MPA приложения проигрывают по производительности практически во всех аспектах.

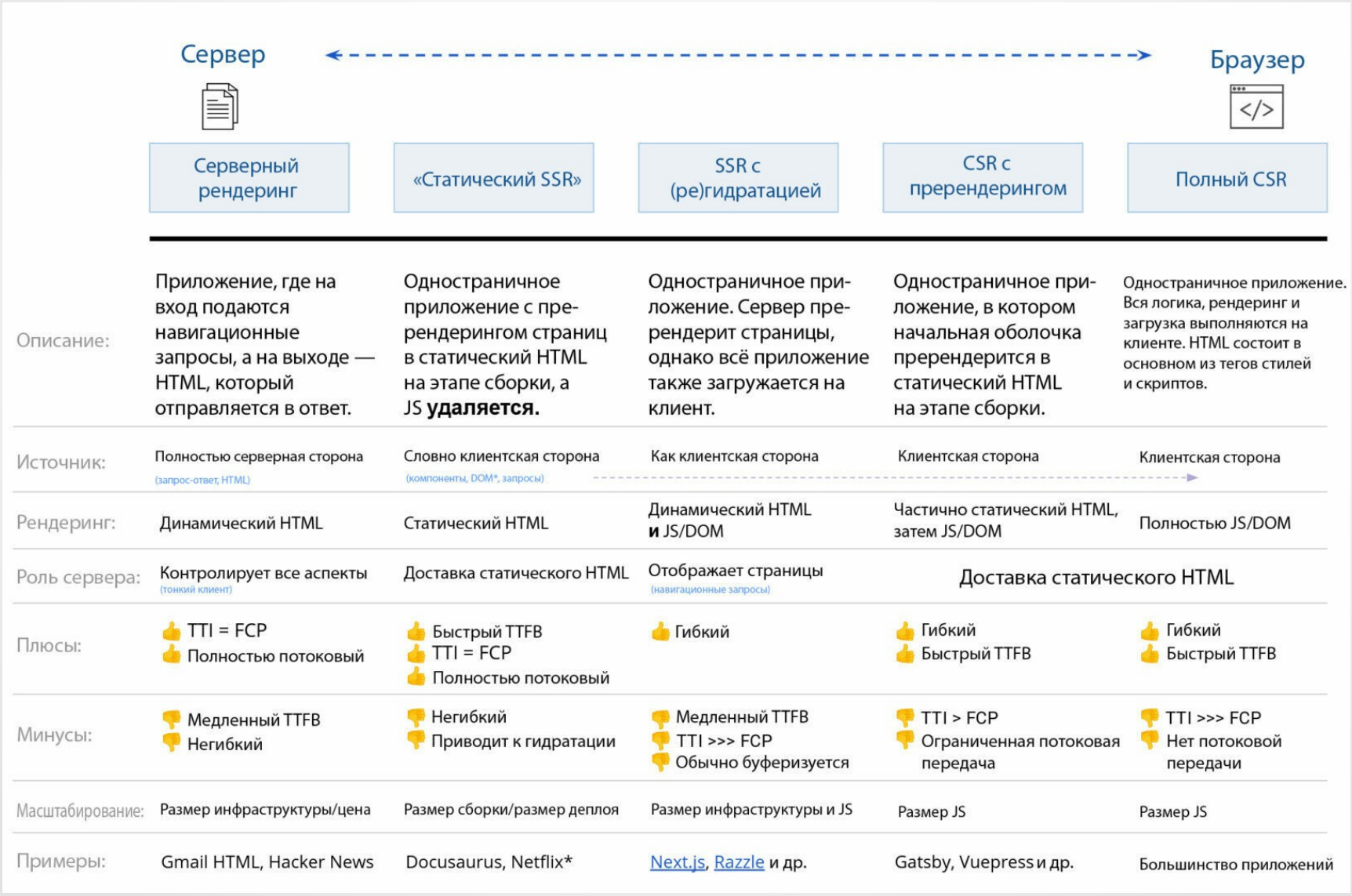
Метрики загрузки страниц приложения

- Базовые
  - **DOM Content Loaded (DCL).** Время, когда страница загрузилась, а скрипты только начали выполняться.
  - **Loaded.** Время, когда страница полностью загружена и с ней можно взаимодействовать.
- Дополнительные
  - **Первая отрисовка (FP).** Это время, спустя которое пользователь видит белую пустую страницу в браузере впервые. Любое первоначальное изменение на экране считается первой отрисовкой.
  - **Первая отрисовка контента (FCP).** Время, спустя которое на экране появляется первый контент. Это может быть часть панели поиска, фрагмент текста или изображения. Нормативные значения такие (согласно Google):
    - высокая скорость — менее 1 с;
    - средняя скорость — от 1 до 2,5 с;
    - низкая скорость — более 2,5 с.
  - **Первая значимая отрисовка (FMP).** Время, спустя которое весь основной контент появляется на экране.

Метрики загрузки страниц приложения

- **Время загрузки для взаимодействия (TTI).** Метрика показывает, сколько времени требуется странице, чтобы стать доступной к взаимодействию (интерактивной).
- **Первая задержка ввода (FID).** Это период времени от первого взаимодействия пользователя с вашим сайтом (клик по ссылке, кнопке, использование элемента управления на основе JavaScript) до реакции браузера на это взаимодействие. Нормативные значения:
  - высокая скорость — менее 50 мс;
  - средняя скорость — от 50 до 250 мс;
  - низкая скорость — более 250 мс.
- **Индекс скорости (SI).** Показывает, насколько быстро страница заполняется видимым контентом.
- **Визуально готово (VR).** Эта метрика в анализе производительности страницы показывает время, когда страница на 100 % загружена и готова к использованию.

Server side rendering



Пример SPA приложения

- Макет сайта:

```
<body>
    <div class="wrapper">
        <header class="header">
            <div class="header__content">
                <h1 class="header__title" id="title">Main</div>
            </div>
        </header>
        <main class="main">
            <div class="main__content" id="body">
                <p> этот текст показывается при загрузке </p>
                <a href="/Articles/1" class="link link_internal">Article 1</a><br>
                <a href="/Articles/2" class="link link_internal">Article 2</a><br>
            </div>
        </main>
    </div>
    <script type="text/javascript" src="/spa/scripts/app.js"></script>
</body>
```

Пример SPA приложения

- Скрипт:

```
var links = null; //Создаём переменную, в которой будут храниться ссылки
var loaded = true; //Переменная, которая обозначает, загрузилась ли страница
var data = //Данные о странице
{
    title: "",
    body: "",
    link: ""
};
var page = //Элементы, текст в которых будет меняться
{
    title: document.getElementById("title"),
    body: document.getElementById("body")
};
//По умолчанию в макете содержится контент для главной страницы.
//Но если пользователь перейдёт по ссылке, которая ведёт на какую-нибудь статью, он увидит не то, что ожидает.
//Поэтому нужно проверить, на какой странице находится пользователь, и загрузить релевантные данные.
OnLoad();
```

Пример SPA приложения

- Скрипт:

```
function OnLoad() {
    var link = window.location.pathname; //Ссылка страницы без домена
    // сайт находится по ссылке http://localhost/spa, поэтому мне нужно обрезать часть с spa/
    var href = link.replace("spa/", "");
    LinkClick(href);
}
function InitLinks() {
    links = document.getElementsByClassName("link_internal"); //Находим все ссылки на странице
    for (var i = 0; i < links.length; i++) {
      //Отключаем событие по умолчанию и вызываем функцию LinkClick
      links[i].addEventListener("click", function (e) {
          e.preventDefault();
          LinkClick(e.target.getAttribute("href"));
          return false;
      });
    }
}
```

Пример SPA приложения

- Скрипт:

```
function LinkClick(href) {
    var props = href.split("/"); //Получаем параметры из ссылки. 1 - раздел, 2 - идентификатор


    switch(props[1]) {
      case "Main":
      SendRequest("?page=main", href); //Отправляем запрос на сервер
      break;


      case "Articles":
      if(props.length == 3 && !isNaN(props[2]) && Number(props[2]) > 0) //Проверяем валидность идентификатора и тоже отправляем запрос
      {
      SendRequest("?page=articles&id=" + props[2], href);
      }
      break;
    }
}
```

Пример SPA приложения

- Скрипт:

```
function SendRequest(query, link) {
    var xhr = new XMLHttpRequest(); //Создаём объект для отправки запроса
    xhr.open("GET", "/spa/core.php" + query, true); //Открываем соединение
    xhr.onreadystatechange = function() { //Указываем, что делать, когда будет получен ответ от сервера
      if (xhr.readyState != 4) return; //Если это не тот ответ, который нам нужен, ничего не делаем
      //Иначе говорим, что сайт загрузился
      loaded = true;
      if (xhr.status == 200) { //Если ошибок нет, то получаем данные
          GetData(JSON.parse(xhr.responseText), link);
      } else { //Иначе выводим сообщение об ошибке
          alert("Loading error! Try again later.");
          console.log(xhr.status + ": " + xhr.statusText);
      }
    }
    loaded = false; //Говорим, что идёт загрузка
    //Устанавливаем таймер, который покажет сообщение о загрузке, если она не завершится через 2 секунды
    setTimeout(ShowLoading, 2000);
    xhr.send(); //Отправляем запрос
}
```

Пример SPA приложения

- Скрипт:

```
function GetData(response, link) { //Получаем данные
    Data = {
      title: response.title,
      body: response.body,
      link: link
    };
    UpdatePage(); //Обновляем контент на странице
}
function ShowLoading() {
    if(!loaded) { //Если страница ещё не загрузилась, то выводим сообщение о загрузке
      page.body.innerHTML = "Loading...";
    }
}
function UpdatePage() { //Обновление контента
    page.title.innerText = data.title;
    page.body.innerHTML = data.body;
    document.title = data.title;
    window.history.pushState(data.body, data.title, "/spa" + data.link); //Меняем ссылку
    InitLinks(); //Инициализируем новые ссылки
}
```