for (var i = 0; i < 10; i++) { setTimeout(function () { console.log(i); }, 100); }

for (var i = 0; i < 10; i++) {

(function (i) {

setTimeout(function () {

console.log(i);

}, 100);

})(i)

}

for (var i = 0; i < 10; i++) {

setTimeout(function (i) {

console.log(i);

}.bind(this, i), 100);

}

for (let i = 0; i < 10; i++) {

setTimeout(function () {

console.log(i);

}, 100);

}

Сортировка без sort

let a = [3, 2, 1, 1, 5, 4],

c = [];

for (; a.length;) { c.push(a.splice(a.indexOf(Math.min(...a)),1)[0])}

console.log(c)

let myFirstPromise = new Promise((resolve, reject) => {

// Мы вызываем resolve(...), когда асинхронная операция завершилась успешно, и reject(...), когда она не удалась.

// В этом примере мы используем setTimeout(...), чтобы симулировать асинхронный код.

// В реальности вы, скорее всего, будете использовать XHR, HTML5 API или что-то подобное.

setTimeout(function(){

resolve("Success!"); // Ура! Всё прошло хорошо!

}, 250);

});

**Возврат к Server Side Rendering**

Однако, в отличие от старого подхода, когда разметка генерировалась на сервере с помощью серверных языков программирования, например **PHP**, теперь в **Server Side Рендеринге**, как и в **CSR**, будут использоваться современные **JavaScript** библиотеки, например **React**.

Разница только в том, что приложение будет генерировать разметку с помощью **React** не на клиентской стороне, а на серверной.

Это решение было направлено на исправление существующих проблем с производительность и **SEO**. Всякий раз, когда поисковый робот запрашивает страницу, то он получит необходимый контент - для визуализации больше не требуется выполнение **JavaScript** на клиенте.

Теперь, когда в разработке упоминают термин **Server Side Rendering**, то ссылаются именно на этот сценарий.

**Более подробно о Server Side Rendering (SSR)**

Итак, рассмотрим плюсы и минусы **SSR**, действительно ли он настолько хорош?

**Плюсы**

* **SEO.** Возможность настроить поисковую оптимизацию.
* **First Contentful Paint.** Благодаря **Server Side Rendering**, страницы вашего сайта быстрее становятся доступными для взаимодействия. Имея производительный сервер, **SSR** может дать вам отличную оценку **First Contentful Paint**, что улучшает **UX** и, возможно, также **SEO** страницы.

На самом деле это все плюсы, давайте теперь рассмотрим минусы.

**Минусы**

* **Замедляется время перехода между страницами.** Да, для первоначального рендера **SSR** быстрее, чем **CSR**, но далее при работе с приложением вам приходится отрисовывать данные дважды, один раз на сервере и один раз на клиенте.
* **Более сложная разработка.** Написать приложение только на **React** в связке, например, с **Redux** значительно проще, чем добавить к этому стэку еще и технологию для **SSR**. Соответственно увеличивается количество кода, сложность разработки и появляются дополнительные библиотеки.
* **Кэширование.** Для частичного закрытия первого минуса и в дополнение ко второму, добавляется более сложное кэширование данных.
* **Стоимость.** В отличие от **CSR**, теперь нам 100% нужен сервер, да еще и более производительный, это стоит дороже.

Оказывается, у **SSR** огромное количество минусов, так что же делать? Рассмотрим еще один подход - **SSG**, может быть он окажется лучше?

**Static Site Generator**

**SSG** - это новый подход для веб-разработки, который позволяет вам создавать веб-сайты, страницы которых генерируются в статические файлы в момент сборки. Тем самым, когда браузер делает запрос, то сервер не генерирует разметку, как в случае **SSR**, а отдает уже готовую. Стоп, похоже мы вернулись к тому, с чего начинали, сервер снова отдает нам простой **HTML** и **CSS**, в чем же тогда отличия?

Разница в том, что вместо использования **HTML** и **CSS**, теперь мы используем современные инструменты, такие как **React**, **Vue** и **Angular**. То есть приложение, написанное на современных инструментах преобразуется в **HTML**, **CSS** и **JavaScript** с помощью транспиляторов, сборщиков пакетов и т.д.

Звучит отлично, но в чем же тогда подвох? Проблема в том, что **SSG** не подразумевает динамическую работу с сервером, т.е. вы не можете создать, например, корзину интернет-магазина при помощи **SSG**, т.к. отображаемая в корзине информация должна быть уникальна для каждого пользователя. В таком случае мы можем применить только **SSR** или **CSR**. А **статическая генерация сайтов** удобна, например, для страницы "Доставка" или для карточки товара, т.е. для той информации, которая идентична для каждого пользователя.

Чтобы более детально понять инструмент, рассмотрим его плюсы и минусы.

**Плюсы:**

* **SEO.** Как и в случае с **SSR**, поисковый робот получает всю необходимую информацию.
* **Производительность.** **SSG** обладает самой большой производительностью по сравнению с **SSR** и **CSR**, т.к. все данные уже сгенерированы при сборке.
* **Очень дешево.** Используя **статическую генерацию сайтов**, вам не нужен сервер, достаточно загрузить код своего сайта на github и собрать его, например, с помощью Netlify. В таком случае вам придется заплатить только за доменное имя.
* **Простота разработки.** Если сравнивать с **CSR**, то **SSG** несколько сложнее, т.к. требуется разобраться в тонкостях, но в целом, картинка слабо отличается от обычной разработки.
* **Безопасность.** У вашего статического сайта нет сервера, а значит у злоумышленников нет возможности получить доступ к вашей базе данных или панели администратора.

**Минусы:**

* **Нет панели администратора.** Плюс, как отсутствие уязвимостей, но минус, так как все изменения на сайте вам придется делать в редакторе кода, потом делать пуш в репозиторий, а далее собирать и деплоить ваш сайт.
* **SSG** запустить сложнее, чем **Wordpress** или другую **CMS.** При первоначальной настройке большинство **CMS** предоставляет огромное количество подсказок для развертывания сайта. В случае работы со **статическим генератором сайтов** вам придется изучать документацию.
* **Отображение только статических данных.** Вы не можете отобразить динамические данные посредством **SSG**, соответственно у вас нет возможности сделать достаточно сложное веб-приложение.

Хорошо, **SSG** звучит частично лучше чем **SSR**, но как же тогда разработать сложное веб-приложение с динамическими данными, используя современные инструменты и не потеряв в **SEO**?