Что такое кэш?

Кэш - это буфер между браузером и интернетом, в котором сохраняются посещённые пользователем страницы. Вместо того, чтобы скачивать их из интернета, браузер может «достать» страницы из кэша, что значительно сокращает скорость загрузки страниц. Проблема может возникнуть, если на сервере страница обновится, а браузер продолжает подгружать старую версию из кэша.

Для чего нужен кэш?

Идею кэширования можно отлично понять путём сравнения, скажем, с магазином овощей. Продавец, чтобы ускорить обслуживание покупателей, держит часть наиболее ходовой продукции в холодильнике под прилавком, и вместо того, чтобы каждый раз бегать на склад за капустой, достаёт из имеющихся запасов под рукой.

Так и ваш браузер: вместо того, чтобы каждый раз, когда вы заходите на Яндекс, грузить с сервера этой славной компании заново все картинки, держит их в кэше и достаёт каждый раз, когда вы снова заходите на сайт Яндекса.

Back/Forward Cache — механизм кеширования страниц в браузере

В Google Chrome, если вы открываете и закрываете страницу, она немедленно **выгружается для очистки памяти**. В течение почти двух лет Google тестировал кеш-память обратной перемотки, чтобы быстрее загружать ранее открытые страницы, когда пользователи нажимают кнопку "назад" или "вперед".
В прошлом году Google представила новую функцию в Chrome для Android под названием **Back-forward cache (кэш назад и вперед - BFCache)**, которая была направлена на повышение производительности при использовании кнопок "**Назад**" и "**Вперед**". Кэширование назад и вперед сохраняет страницы в памяти на некоторое время после того, как вы вернетесь, поэтому, если вам нужно вернуться к самой последней странице, время загрузки будет мгновенно.
Google Chrome 87, выпущенный в ноябре 2020 года, добавили флаг функции для ручного включения обратного кэширования (Back-forward cache) на настольных платформах, таких как Windows, Linux и macOS. До этого функция была доступна только на Android. Google опубликовал страницу "Intent to Experiment" в группах Google, объяснив, что эта функция теперь будет протестирована на небольшом количестве настольных пользователей Chrome.

После включения функции BFCache, при посещении страницы и использования навигации вперед/назад, эта страница **должна** будет мгновенно открываться повторно, а **не перезагружать** ресурсы снова. Эта функция пытается заставить кнопки Google Chrome "Назад" и "Вперед" работать почти мгновенно.
Google не упомянул, почему эта функция так долго оставалась только мобильной, но задержка могла быть связана с использованием памяти. Chrome часто критиковали за высокое использование памяти на настольных платформах, и "замораживание" большего количества страниц, вероятно, не поможет

Данный механизм позволяет производить навигацию по посещенным страницам крайне быстро. Состояние кеша остается неизменным, пока действует сессия браузера (пока пользователь не закроет закладку, или браузер).

Cookies — это небольшие текстовые файлы у нас на компьютерах, в которых хранится информация о наших предыдущих действиях на сайтах. Это небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Кроме входов в аккаунты они умеют запоминать:

- предпочтения пользователей, например, язык, валюту или размер шрифта.
- товары, которые мы просматривали или добавили в корзину;
- текст, который мы вводили на сайте раньше;
- IP-адрес и местоположение пользователя;
- дату и время посещения сайта;
- версию ОС и браузера;
- клики и переходы.

Самая большая проблема при использовании cookie в качестве локального хранилища заключается в том, что:

- в каждом из них можно храниться максимум 4 Кбайт данных (этого может быть недостаточно для работы веб-приложения);
- cookie должны отправляться в обоих направлениях при каждой перезагрузке страницы, что может существенно замедлять работу веб-сайта;
- куки также включаются в каждый HTTP-запрос при передаче данных через Интернет в незашифрованном виде.

Веб-хранилище (Web Storage, DOM *Document Object Model*-хранилище) - это набор механизмов, которые используются для хранения данных локально в веб-браузере пользователя в виде пары ключ/значение. Другими словами, это специально отведенное место в браузере (похоже на небольшую базу данных), в котором мы можем работать с данными - записывать, читать и удалять их. Интернет-хранилище представляет собой постоянное хранилище данных, похожее на куки, но со значительно расширенной ёмкостью и без хранения информации в заголовке запроса HTTP.

С появлением HTML5 мы получили доступ к более объемному веб-хранилищу (между 5 и 10 МБайт на каждый домен), которое сохраняет информацию между загрузками страницы и посещениями сайта (даже после выключения и включения компьютера). Кроме того, данные локального хранилища не отправляются на сервер при каждой загрузке страницы, в связи с чем ускоряется работа веб-приложения.

Существуют два основных типа веб-хранилища:

1. сессионное хранилище (sessionStorage);
2. локальное хранилище (localStorage);
3. В них можно хранить ТОЛЬКО СТРОКОВЫЕ ДАННЫЕ для ключей и их значений.
4. Разберем каждый из них подробнее.

Что можно хранить и для чего использовать?

Локальное хранилище часто используется для сохранения настроек пользователя на сайте (выбор темы оформления, вид отображения информации), чтобы при следующем заходе эти данные уже были применены к его профилю и повторно не вводились.
Поскольку мы работаем с данными, то можем сохранять в sessionStorage и localStorage все, что связано с идентификацией пользователя - User-Agent, местоположение, IP-адрес, уникальный идентификатор пользователя (Client ID), значение сгенерированного промокода от виджета на сайте, источник перехода, список товаров, добавленных в корзину, пользовательские данные при неудачной попытке отправки на сервер и другие параметры.

Где можно увидеть эти хранилища?

Например, в Google Chrome для этого необходимо открыть «Инструменты разработчика», перейти на вкладку «Application». Здесь они находятся на левой панели в разделе «Storage». При выборе источника вы увидите какие данные содержатся соответственно в sessionStorage и localStorage.

Использует или нет сайт встроенное хранилище данных, можно увидеть в консоли разработчика, во вкладке *Application / Local Storage*, предварительно нажав F12. На скриншоте у сайта *booking.com*, мы видим какие-то данные в виде характерной для массивов записи ('key': 'value').

LocalStorage - это локальное хранилище данных прямо в браузере клиента. Иногда нужно между перезагрузками страниц, сохранять пользовательские настройки, чаще всего для элементов формы, корзины с товарами. Чтобы при повторном заходе на сайт, пользователю не нужно было бы все заново настраивать.

Например, пользователь установил настройки в магазине и при повторном заходе, ему уже не нужно этого делать заново. Если посетитель не досмотрел видео до конца, то продолжить просмотр он может с того самого места, где остановился. Использование технологии localStorage, даёт нам такие возможности.

Локальное хранилище позволяет хранить данные для каждого домена в течение неограниченного времени (после закрытия вкладки браузера, самого браузера, перезагрузки компьютера), а точнее пока пользователь самостоятельно не удалит данные.

В отличие от sessionStorage оно сохраняет свои данные для всего приложения, т.е. вашего сайта. С целью безопасности браузер прикрепляет записи к домену сайта.

Но там можно хранить секретный токен со сроком действия или JSON Web Token (но здесь cookie здесь будет более уместно).

Вы спросите, почему в localStorage? У localStorage нет времени очистки. Потенциально эта информация будет храниться в браузере вечно. Пользователь тоже может ее удалить, но было бы неразумно ожидать от него таких странных вещей.

Увы, у этой функции есть определенные ограничения. Возможность записи в localStorage доступна не всегда. Например, iOS Safari вернёт ошибку при попытке записать в localStorage, если бразуер находится в приватном режиме

Хранение информации в этих объектах осуществляется в формате «ключ-значение». Ключ и значение – это всегда строки.

Т.е., по сути, каждое хранилище представляет собой вот такой объект:

```
{
  'key1': 'value1',
  'key2': 'value2',
  'key3': 'value3',
  ...
}
```

Если в качестве значения указать не строку, а какой-то другой тип данных, например, объект, то он будет, перед тем как туда записан, автоматически преобразован в строку (т.е. так как будто мы для него явно вызвали метод `toString()`)

Каждый домен имеет доступ к своему хранилищу данных localStorage. Например, localStorage, используемый для, https://loungefm.com.ua является отдельным от localStorage, используемым для https://lux.fm/ Субдомены (поддомены) и различные протоколы HTTP (HTTP и HTTPS) имеют независимые друг от друга хранилища данных. Например, localStorage https://ksendzov.com используется полностью отдельно от http://ksendzov.com
Некоторые браузеры блокируют localStorage в режиме инкогнито. localStorage работает даже с отключенными cookie.

`SessionStorage` хранит данные только во время текущей сессии (для вкладки, пока она открыта). Закрытие вкладки или браузера приводит к очищению этих данных. При этом данные сохраняются при обновлении страницы или отображение в этой вкладке другой страницы из этого же источника. . Это означает, что с помощью JavaScript нельзя получить доступ к данным другой вкладки даже если она имеет тот же источник.

Объект `sessionStorage` используется гораздо реже, чем `localStorage`.

в данном случае, сессия, к которой мы обращаемся — это конкретная вкладка браузера, а не браузер или окно.
Свойства и методы такие же, но есть существенные ограничения:

- `sessionStorage` существует только в рамках текущей вкладки браузера.
  - Другая вкладка с той же страницей будет иметь другое хранилище.
- Данные продолжают существовать после перезагрузки страницы, но не после закрытия/открытия вкладки.

Объём данных больше чем в Cookie.

Не поддерживается старыми браузерами, например, IE 7 и ниже.

Поэтому не стоит использовать этот тип хранилища для работы с долгосрочными данными. Например, с сессиями аутентификации. Когда вы входите в Facebook, Facebook запоминает вас, несмотря на то, что вы закрыли страницу и открыли её снова. Здесь используются Cookies, а не Session Storage.

Волатильность хранилища – характеристика способа хранения, отражающая способность хранилища сохранять данные в течение длительного времени. Волатильность в переводе с английского языка означает изменчивость. Термин является полной противоположностью понятиям стабильность и постоянство.

sessionStore – очень волатильное хранилище. Оно не хранится долго. К нему можно относиться, как к текстовому документу без кнопки “Сохранить”. Все в нем хранящееся превратится в прах, стоит вам закрыть вкладку браузера. Ну, почти. Чисто технически кое-какой JavaScript может перед этим отослать данные на удаленный сервер, который запомнит их для вас.

localStorage и sessionStorage не подходят для сохранения конфиденциальных данных, необходимых в клиентских скриптах между страницами (например, предпочтения, результаты в играх). Данные, хранящиеся в localStorage и sessionStorage, могут быть легко прочитаны или изменены из клиента/браузера, поэтому не следует полагаться на хранение конфиденциальных или связанных с безопасностью данных в приложениях.

Свойство sessionStorage позволяет получить доступ к объекту хранилища сеанса. Он похож на localStorage, но разница в том, что данные, хранящиеся в localStorage, не имеют настройки времени истечения срока действия, а данные, хранящиеся в sessionStorage, будут очищены после завершения сеанса страницы. Сеанс страницы поддерживается, пока браузер открыт, и исходный сеанс страницы будет сохранен при перезагрузке или восстановлении страницы. Открытие страницы в новой вкладке или окне инициирует новый сеанс, который отличается от того, как работают файлы cookie сеанса.

Печеньки

Теперь, к самой сладкой части. Cookies стали привычным объектом ограничений законов о приватности и используются на всех сайтах с авторизацией по логину и паролю. Cookies тоже являются domain specific, то есть привязаны только к конкретному домену, могут храниться очень долго. Но, в отличие от прошлых хранилищ, они отправляются с каждым HTTP/HTTPS запросом на сервер. Их можно читать и записывать со стороны сервера. При записи они будут добавлены в HTTP ответ, чтобы браузер пользователя их запомнил.

Как работают cookies

Когда мы совершаем на сайте какое-то действие, например, добавляем товар в корзину или вводим детали входа в аккаунт, сервер записывает эту информацию в куки и отправляет браузеру вместе со страницей. Когда мы переходим на другую страницу сайта или заходим на него через время, браузер отправляет куки обратно.

Куки бывают временными и постоянными. Постоянные куки остаются на компьютере, когда мы закрываем вкладку с сайтом, а временные удаляются. Какие именно куки использовать на конкретном сайте — временные или постоянные — решает его разработчик. Именно поэтому на одних сайтах мы не выходим из аккаунтов, даже когда заходим на них раз спустя несколько дней, а на других вводим пароль заново, хотя отошли от компьютера на пять минут.

Cookies и безопасность

Сами по себе куки не опасны — это обычные текстовые файлы. Они не могут запускать процессы на компьютере и вообще взаимодействовать с операционной системой. Но их могут попытаться перехватить или украсть, чтобы отследить ваши предыдущие действия в сети или входить в ваши аккаунты без авторизации.
Обычно информацию, которую записывают в куки, зашифровывают перед отправкой, а сами куки передают по HTTPS-протоколу. Это помогает защитить пользовательские данные, но за внедрение шифрования и безопасную отправку отвечает разработчик сайта. Посетителям остаётся только надеяться, что всё настроили грамотно. Со своей стороны пользователь может только запретить браузеру использовать куки или время от времени чистить их самостоятельно.
Совсем отключать куки — не всегда хорошая идея. Например, все интернет-магазины работают с помощью куки. Если запретить браузеру их использовать, сервер не сможет запомнить, что именно вы добавили в корзину. Чистить куки вручную практичнее, но придётся каждый раз заново настраивать внешний вид сайта и входить в аккаунты.
Другая причина периодически чистить куки — место на компьютере. Со временем куки накапливаются и могут занимать гигабайты на жестком диске. Поэтому раз в несколько месяцев полезно удалить куки хотя бы для тех сайтов, на которые вы заходите редко. Расскажем, как сделать это в популярных браузерах.

Session Cookies / *Сессионные cookie*

также известные как *временные cookie*, существуют только во временной памяти, пока пользователь находится на странице веб-сайта. Браузеры обычно удаляют сессионные cookie после того, как пользователь закрывает окно браузера[13]. В отличие от других типов cookie, сессионные cookie не имеют истечения срока действия, и поэтому браузеры понимают их как сессионные.

First-Party Cookies / Постоянные cookie

Вместо того, чтобы удаляться после закрытия браузера, как это делают сессионные cookie, *постоянные cookie-файлы* удаляются в определённую дату или через определённый промежуток времени. Это означает, что информация о cookie будет передаваться на сервер каждый раз, когда пользователь посещает веб-сайт, которому эти cookie принадлежат. По этой причине постоянные cookie иногда называются *следящие cookie*, поскольку они могут использоваться рекламодателями для записи о предпочтениях пользователя в течение длительного периода времени. Однако, они также могут использоваться и в «мирных» целях, например, чтобы избежать повторного ввода данных при каждом посещении сайта.

Third-Party Cookies

Third-party cookies are the bad guys. They are the reason that cookies have such a bad reputation among internet users.

Обычно атрибут домена cookie совпадает с доменом, который отображается в адресной строке веб-браузера. Это называется первый файл cookie. Однако сторонний файл cookie принадлежит домену, отличному от того, который указан в адресной строке. Этот тип файлов cookie обычно появляется, когда веб-страницы содержат контент с внешних веб-сайтов, например, рекламные баннеры. Это открывает возможности для отслеживания истории посещений пользователя и часто используется рекламодателями для предоставления релевантной рекламы каждому пользователю.

В качестве примера предположим, что пользователь посещает www.example.org. Этот веб-сайт содержит рекламу от ad.foxytracking.com, которая при загрузке устанавливает файл cookie, принадлежащий домену рекламы (ad.foxytracking.com). Затем пользователь посещает другой веб-сайт www.foo.com, который также содержит рекламу от ad.foxytracking.com и устанавливает файл cookie, принадлежащий этому домену (ad.foxytracking.com). В конце концов, оба этих cookie будут отправлены рекламодателю при загрузке их рекламы или посещении их веб-сайта. Затем рекламодатель может использовать эти cookie для создания истории просмотров пользователя на всех веб-сайтах, на которых размещена реклама этого рекламодателя.

Большинство современных веб-браузеров содержит настройки конфиденциальности, которые могут блокировать сторонние cookie.

Супер-cookie — это cookie-файл с источником домена верхнего уровня (например, .ru) или общедоступным суффиксом (например, .co.uk). Обычные cookie, напротив, имеют происхождение от конкретного доменного имени, например example.com.

Супер-cookie могут быть потенциальной проблемой безопасности и поэтому часто блокируются веб-браузерами. Если браузер разблокирует вредоносный веб-сайт, злоумышленник может установить супер-cookie и потенциально нарушить или выдать себя за законные запросы пользователей на другой веб-сайт, который использует тот же домен верхнего уровня или общедоступный суффикс, что и вредоносный веб-сайт. Например, супер-cookie с происхождением .com может злонамеренно повлиять на запрос к example.com, даже если файл cookie не был создан с сайта example.com. Это может быть использовано для подделки логинов или изменения информации пользователя.

Zombie Cookies

Поскольку cookie можно очень легко удалить из браузера, программисты ищут способы идентифицировать пользователей даже после полной очистки истории браузера. Одним из таких решений являются зомби-cookie (или *evercookie*, или *persistent cookie*) — неудаляемые или трудно удаляемые cookie, которые можно восстановить в браузере с помощью JavaScript. Это возможно потому, что для хранения куки сайт одновременно использует все доступные хранилища браузера (*HTTP ETag, Session Storage, Local Storage, Indexed DB*), в том числе и хранилища приложений, таких как Flash Player (*Local Shared Objects*), Microsoft Silverlight (*Isolated Storage*) и Java (*Java persistence API*). Когда программа обнаруживает отсутствие в браузере cookie-файла, информация о котором присутствует в других хранилищах, она тут же восстанавливает его на место и тем самым идентифицирует пользователя для сайта.

Заключение — Разница между куки и локальным хранилищем

sessionStorage vs localStorage vs cookies

Нагляднее всего продемонстрировать отличия между sessionStorage, localStorage и cookies с помощью таблицы:

	Cookies	sessionStorage	localStorage
Размер	4 Кбайт	5 Мбайт	до 10 Мбайт
Поддержка браузера	HTML4/HTML5	HTML5	HTML5
Отправка при запросе	Да	Нет	Нет
Срок жизни	Настраивается вручную	При закрытии вкладки браузера	Не удаляется
Место хранения	Браузер и сервер	Только в браузере	Только в браузере
Доступность	Из любого окна	Из того же окна	Из любого окна

Отличие cookies от sessionStorage и localStorage

- Размер: куки ограничены 4 Кбайт, sesionStorage - 5 МБайт, а localStorage - от 5 до 10 Мбайт; (этот размер поддерживается всеми основными веб-браузерами)
- Поддержка браузера: cookies были еще в стандарте HTML4, в то время как sessionStorage и localStorage появились в HTML5;
- Отправка при запросе: (куки в отличие от данных локального хранилища включаются в состав запроса при отправке его на сервер, а также сервер их может добавлять в ответ при отправке его клиенту; таким образом cookies являются частью HTTP-протокола, и увеличивают объём передаваемых данных от клиента серверу и обратно)
- ;можно передать куки при отправке запроса, а sessionStorage и localStorage нет;
- Срок жизни: для файлов cookie мы можем задать период жизни, данные в sessionStorage удаляются после закрытия вкладки браузера, а в localStorage они не удаляются "никогда";
- Место хранения: куки хранятся и в браузере, и на сервере, а sessionStorage и localStorage только локально в браузере пользователя;
- Доступность: данные о пользователе в cookies и localStorage доступны из любого окна, при условии, что вы осуществляется заход под теми же данными (например, авторизуетесь на сайте), в то время как sessionStorage работает только из того же окна, а изменение вкладки браузера приведет к удалению информации.

- по необходимости информирования пользователей Евросоюза (при использовании cookies сайт в ЕС должен получать на это разрешение от пользователей; для данных локального хранилища это не требуется);
- по назначению (куки в основном используются для управления сеансом, персонализации и отслеживания действий пользователя, в то время как localStorage применяется в качестве обычного локального хранилища информации на компьютере пользователя).

Безопасность

Вопрос безопасности — решающий фактор, поэтому поговорим о нем более подробно.
Итак, localStorage НЕБЕЗОПАСЕН! Совсем! Каждый, кто использует его для хранения конфиденциальных данных, поступает неправильно.

Давайте разберемся, что понимается под конфиденциальными данными:
— Идентификаторы пользователей
— Идентификаторы сессий
— JWT (JSON Web Token)
— Персональная информация
— Информация о кредитной карте
— API-ключи
— Любая другая информация, которую вы бы не стали публиковать публично

LocalStorage разрабатывался не как безопасный механизм хранения данных в браузере, а как простое хранилище ключей и значений с целью облегчения создания небольших сайтов/веб-приложений. И все. Что, по вашему, самое опасное в мире? Верно! JavaScript. Поэтому, когда захотите сохранить что-нибудь важное в локальном хранилище, представьте, что хотите спрятать секретнейшую информацию в самом ненадежном сейфе на планете. Не лучшая идея.
На самом деле проблема заключается в межсайтовом скриптинге (XSS). если хакер сможет запустить JavaScript-код на вашем сайте, то он запросто вытащит всю информацию из localStorage и отправит её на свой сервер, тем самым заполучив, например, данные о пользовательской сессии.

Про токены

Отдельно стоит разъяснить ситуацию с JSON Web Token (JWT). Многие разработчики, которые хранят JWT в localStorage, не понимают, что это, по сути, то же самое, что и имя пользователя / пароль.

Если хакеры скопируют эти токены, то смогут отправлять запросы на ваш сервер, и вы об этом никогда не узнаете. Поэтому обращайтесь с ними так же, как с данными кредитной карты или паролем, а именно — не храните в localStorage, вопреки тысячам туториалов, видео на Youtube и даже курсам программирования в университетах. Это неправильно! Если кто-то советует вам хранить токены в локальном хранлище для аутентификации, покажите им эту статью.

Краткий обзор Access token и Refresh token
Токены доступа (Access token) обычно являются недолговечными токенами JWT, подписанными сервером и включенными в каждый HTTP-запрос к серверу для авторизации запроса.
Токены обновления (Refresh token) обычно представляют собой долговечные зашифрованные токенами JWT, хранящиеся в базе данных, которые используются для получения нового токена доступа по истечении срока его действия.
Где лучше хранить свои токены?
Существует два распространенных способа хранения токенов: в localStorage и в cookie. Есть много споров о том, какой из них лучше, но большинство людей склоняются к cookie из-за большей безопасности.
Давайте рассмотрим это сравнение чуть более подробнее.

Local Storage

Плюсы: Это удобно.

- Это чистый JavaScript и с ним удобней работать.
- Минусы: Такой способ уязвим к XSS атакам.

Атака XSS происходит, когда злоумышленник может запустить свой скрипт JavaScript на вашем сайте во время посещения его другим пользователем. Это означает, что злоумышленник сможет получит доступ к токену доступа, который вы сохранили в localStorage.

Cookies

Плюсы: Файл cookie может быть не доступен через JavaScript, поэтому он не так уязвим для атак XSS, как localStorage.

- Если вы используете флаги httpOnly, то cookie будут не доступны из JavaScript. Это означает, что даже если злоумышленник сможет запустить JS на сайте, он не сможет прочитать ваш токен доступа.
- Cookie автоматически отправляется в каждом HTTP-запросе на ваш сервер.

Минусы: В зависимости от варианта использования иногда вы не сможете хранить свои токены в файлах cookie..

- Размер файлов cookie ограничен 4 КБ, поэтому, если вы используете большие токены JWT, сохранение в файле cookie станет не возможным.

Итак мы уже сказали что local storage уязвим, так как он легко доступен с помощью JavaScript, и злоумышленник может получить access token и использовать его. Однако, хотя cookie с httpOnly недоступны из JavaScript, это не означает, что с помощью cookie вы полностью защищены от XSS атак.

Если злоумышленник может запустить JavaScript в вашем приложении, он все равно сможет отправить HTTP-запрос на ваш сервер получив таким образом все токены. Но для злоумышленника это менее удобно, потому что он не сможет прочитать содержимое токена, хотя это и не всегда нужно.

Конфиденциальные данные

Для хранения таких данных единственным верным решением является сессия на стороне сервера. Алгоритм следующий:

- Когда пользователь логинится на вашем сайте, нужно создать уникальный индентификатор сессии и сохранить в криптографически зашифрованный куки-файл.
- Убедитесь, что у cookie-библиотеки, которую использует ваш фреймворк, в настройках включено “httpOnly”. Это сделает невозможным просмотр куки-файлов браузером, что необходимо для их безопасного использования на стороне сервера.