JavaScript данные и фреймы

Фреймы и окна;
Хранение данных в браузере.

Фреймы и окна

- Открытие окон и методы window
- Общение между окнами

Открытие окон и методы window

Всплывающее окно («попап» – от англ. Popup window) – один из древнейших способов показать пользователю ещё один документ.

window.open('https://javascript.info/')

… и откроется новое окно с указанным URL. Большинство современных браузеров по умолчанию будут открывать новую вкладку вместо отдельного окна.

Блокировка попапов

В прошлом злонамеренные сайты заваливали посетителей всплывающими окнами. Такие страницы могли открывать сотни попапов с рекламой. Поэтому теперь большинство браузеров пытаются заблокировать всплывающие окна, чтобы защитить пользователя.

Всплывающее окно блокируется в том случае, если вызов window.open произошёл не в результате действия посетителя (например, события onclick).

```
// попап заблокирован
window.open('https://javascript.info');

// попап будет показан
button.onclick = () => {
  window.open('https://javascript.info');
};
```

Полный синтаксис window.open

Синтаксис открытия нового окна: ***window.open(url, name, params)***:

- url - URL для загрузки в новом окне.
- name - Имя нового окна. У каждого окна есть свойство window.name, в котором можно задавать, какое окно использовать для попапа. Таким образом, если уже существует окно с заданным именем – указанный в параметрах URL откроется в нем, в противном случае откроется новое окно.
- params - Строка параметров для нового окна. Содержит настройки, разделённые запятыми. Важно помнить, что в данной строке не должно быть пробелов.

Параметры в строке params:

Позиция окна:

- left/top (числа) – координаты верхнего левого угла нового окна на экране. Существует ограничение: новое окно не может быть позиционирован вне видимой области экрана.
- width/height (числа) – ширина и высота нового окна. Существуют ограничения на минимальную высоту и ширину, которые делают невозможным создание невидимого окна.

Панели окна:

- menubar (yes/no) – позволяет отобразить или скрыть меню браузера в новом окне.
- toolbar (yes/no) – позволяет отобразить или скрыть панель навигации браузера (кнопки вперёд, назад, перезагрузки страницы) нового окна.
- location (yes/no) – позволяет отобразить или скрыть адресную строку нового окна. Firefox и IE не позволяют скрывать эту панель по умолчанию.
- status (yes/no) – позволяет отобразить или скрыть строку состояния. Как и с адресной строкой, большинство браузеров будут принудительно показывать её.
- resizable (yes/no) – позволяет отключить возможность изменения размера нового окна. Не рекомендуется.
- scrollbars (yes/no) – позволяет отключить полосы прокрутки для нового окна. Не рекомендуется.

Доступ к попапу из основного окна

Вызов open возвращает ссылку на новое окно. Эта ссылка может быть использована для управления свойствами окна, например, изменения положения и др.

Политика одного источника

Окна имеют свободный доступ к содержимому друг друга только если они с одного источника (у них совпадают домен, протокол и порт (protocol://domain:port).

Иначе, например, если основное окно с site.com, а попап с gmail.com, это невозможно по соображениям пользовательской безопасности.

```
// ОБЯЗАТЕЛЬНО НУЖЕНА ИМИТАЦИЯ СЕРВЕРА (live server например)

const myPopupButton = document.querySelector('#myPopupButton');

const popupHandler = () => {
  let newWindow = open('/index3.html', 'example', 'width=300,height=300');
  newWindow.focus();

  newWindow.onload = function () {
    const testEl = document.createElement('div');
    testEl.innerText = 'test el';
    newWindow.document.body.append(testEl);
  };
};

myPopupButton.addEventListener('click', popupHandler);
```

Чтобы код работал локально требуется поднять локальный сервер, например с помощью плагина live-server в vs-code-е

Закрытие попапа

Чтобы закрыть окно: win.close()

Для проверки, закрыто ли окно: win.closed.

Технически метод close() доступен для любого окна, но window.close() будет игнорироваться большинством браузеров, если window не было создано с помощью window.open(). Так что он сработает только для попапов.

Если окно закрыто, то его свойство closed имеет значение true. Таким образом можно легко проверить, закрыт ли попап (или главное окно) или все еще открыт. Пользователь может закрыть его в любой момент, и наш код должен учитывать эту возможность.

```
let newWindow = open('/', 'example', 'width=300,height=300');

newWindow.onload = function() {
  newWindow.close();
  alert(newWindow.closed); // true
};
```

Прокрутка и изменение размеров

Методы для передвижения и изменения размеров окна:

win.moveBy(x,y)

Переместить окно относительно текущей позиции на x пикселей вправо и y пикселей вниз. Допустимы отрицательные значения (для перемещения окна влево и вверх).

win.moveTo(x,y)

Переместить окно на координаты экрана (x,y).

win.resizeBy(width,height)

Изменить размер окна на указанные значения width/height относительно текущего размера. Допустимы отрицательные значения.

win.resizeTo(width,height)

Изменить размер окна до указанных значений.

Также существует событие ***window.onresize***.

Только попапы

Чтобы предотвратить возможные злоупотребления, браузер обычно блокирует эти методы. Они гарантированно работают только с попапами, которые мы открыли сами и у которых нет дополнительных вкладок.

Нельзя свернуть/развернуть окно

Методами JavaScript нельзя свернуть или развернуть («максимизировать») окно на весь экран. За это отвечают функции уровня операционной системы, и они скрыты от фронтенд-разработчиков.

Методы перемещения и изменения размера окна не работают для свернутых и развёрнутых на весь экран окон.

Прокрутка окна

win.scrollBy(x,y)

Прокрутить окно на x пикселей вправо и y пикселей вниз относительно текущей прокрутки. Допустимы отрицательные значения.

win.scrollTo(x,y)

Прокрутить окно до заданных координат (x,y).

elem.scrollIntoView(top = true)

Прокрутить окно так, чтобы elem для elem.scrollIntoView(false) появился вверху (по умолчанию) или внизу.

Также существует событие ***window.onscroll.***

Общение между окнами

Политика «Одинакового источника» (Same Origin) ограничивает доступ окон и фреймов друг к другу.

Идея заключается в том, что если у пользователя открыто две страницы: john-smith.com и gmail.com, то у скрипта со страницы john-smith.com не будет возможности прочитать письма из gmail.com. Таким образом, задача политики «Одинакового источника» – защитить данные пользователя от возможной кражи.

Политика "Одинакового источника"

Два URL имеют «одинаковый источник» в том случае, если они имеют совпадающие протокол, домен и порт.

Эти URL имеют одинаковый источник:

- http://site.com
- http://site.com/
- http://site.com/my/page.html

А эти – разные источники:

- http://www.site.com (другой домен: www. важен)
- http://site.org (другой домен: .org важен)
- https://site.com (другой протокол: https)
- http://site.com:8080 (другой порт: 8080)

Политика «Одинакового источника» говорит, что:

если у нас есть ссылка на другой объект window, например, на всплывающее окно, созданное с помощью window.open или на window из <iframe> и у этого окна тот же источник, то к нему будет полный доступ.

в противном случае, если у него другой источник, мы не сможем обращаться к его переменным, объекту document и так далее. Единственное исключение – объект location: его можно изменять (таким образом перенаправляя пользователя). Но нельзя читать location (нельзя узнать, где находится пользователь, чтобы не было никаких утечек информации).

Доступ к содержимому ифрейма

Внутри <iframe> находится по сути отдельное окно с собственными объектами document и window.

Мы можем обращаться к ним, используя свойства:

iframe.contentWindow ссылка на объект window внутри <iframe>.

iframe.contentDocument – ссылка на объект document внутри <iframe>, короткая запись для iframe.contentWindow.document.

```
<!-- ифрейм с того же сайта -->
<iframe src="/" id="iframe"></iframe>

<script>
  iframe.onload = function() {
    // делаем с ним что угодно
    iframe.contentDocument.body.prepend("Привет, мир!");
  };
</script>
```

`iframe.onload` и `iframe.contentWindow.onload`

Событие `iframe.onload` – по сути то же, что и `iframe.contentWindow.onload`. Оно сработает, когда встроенное окно полностью загрузится со всеми ресурсами.

...Но `iframe.onload` всегда доступно извне ифрейма, в то время как доступ к `iframe.contentWindow.onload` разрешён только из окна с тем же источником.

Коллекция window.frames

Другой способ получить объект window из <iframe> – забрать его из именованной коллекции window.frames:

По номеру: window.frames[0] – объект window для первого фрейма в документе.

По имени: window.frames.iframeName – объект window для фрейма со свойством name="iframeName".

Ифрейм может иметь другие ифреймы внутри. Таким образом, объекты window создают иерархию.

Навигация по ним выглядит так:

- window.frames – коллекция «дочерних» window (для вложенных фреймов).
- window.parent – ссылка на «родительский» (внешний) window.
- window.top – ссылка на самого верхнего родителя.

```
<iframe src="/" style="height:80px" name="win" id="iframe"></iframe>

<script>
  alert(iframe.contentWindow == frames[0]); // true
  alert(iframe.contentWindow == frames.win); // true
</script>
```

Атрибут ифрейма sandbox

Атрибут sandbox позволяет наложить ограничения на действия внутри <iframe>, чтобы предотвратить выполнение ненадёжного кода. Атрибут помещает ифрейм в «песочницу», отмечая его как имеющий другой источник и/или накладывая на него дополнительные ограничения.

Существует список «по умолчанию» ограничений, которые накладываются на <iframe sandbox src="...">. Их можно уменьшить, если указать в атрибуте список исключений (специальными ключевыми словами), которые не нужно применять, например: <iframe sandbox="allow-forms allow-popups">.

Вот список ограничений:

- allow-same-origin - "sandbox" принудительно устанавливает «другой источник» для ифрейма. Другими словами, он заставляет браузер воспринимать iframe, как пришедший из другого источника, даже если src содержит тот же сайт. Со всеми сопутствующими ограничениями для скриптов. Эта опция отключает это ограничение.
- allow-top-navigation -Позволяет ифрейму менять parent.location.
- allow-forms - Позволяет отправлять формы из ифрейма.
- allow-scripts - Позволяет запускать скрипты из ифрейма.
- allow-popups - Позволяет открывать всплывающие окна из ифрейма с помощью window.open.

Обмен сообщениями между окнами

Интерфейс postMessage позволяет окнам общаться между собой независимо от их происхождения.

Это способ обойти политику «Одинакового источника». Он позволяет обмениваться информацией, скажем john-smith.com и gmail.com, но только в том случае, если оба сайта согласны и вызывают соответствующие JavaScript-функции. Это делает общение безопасным для пользователя.

Интерфейс имеет две части:

- postMessage
- Событие message

postMessage

Окно, которое хочет отправить сообщение, должно вызвать метод postMessage окна получателя. Другими словами, если мы хотим отправить сообщение в окно win, тогда нам следует вызвать win.postMessage(data, targetOrigin).

Аргументы:

- data - Данные для отправки. Может быть любым объектом, данные клонируются с использованием «алгоритма структурированного клонирования». IE поддерживает только строки, поэтому мы должны использовать метод JSON.stringify на сложных объектах, чтобы поддержать этот браузер.
- targetOrigin - Определяет источник для окна-получателя, только окно с данного источника имеет право получить сообщение.

```
<iframe src="http://example.com" name="example">

<script>
  let win = window.frames.example;

  win.postMessage("message", "http://example.com");
</script>
```

ли мы не хотим проверять, то в `targetOrigin` можно указать `*`.

```
<iframe src="http://example.com" name="example">

<script>
  let win = window.frames.example;

  win.postMessage("message", "*");
</script>
```

Событие message

Чтобы получать сообщения, окно-получатель должно иметь обработчик события message (сообщение). Оно срабатывает, когда был вызван метод postMessage (и проверка targetOrigin пройдена успешно).

Объект события имеет специфические свойства:

- data - Данные из postMessage.
- origin - Источник отправителя, например, http://javascript.info.
- source - Ссылка на окно-отправитель. Можно сразу отправить что-то в ответ, вызвав source.postMessage(...).

Чтобы добавить обработчик, следует использовать метод addEventListener, короткий синтаксис window.onmessage не работает.

```
window.addEventListener("message", function(event) {
  if (event.origin != 'http://javascript.info') {
    // что-то пришло с неизвестного домена. Давайте проигнорируем это
    return;
  }

  alert( "received: " + event.data );

  // can message back using event.source.postMessage(...)
});
```

Хранение данных в браузере

- Куки, document.cookie
- LocalStorage & sessionStorage
- IndexedDB

Куки, document.cookie

Куки – это небольшие строки данных, которые хранятся непосредственно в браузере. Они являются частью HTTP-протокола.

Куки обычно устанавливаются веб-сервером при помощи заголовка Set-Cookie. Затем браузер будет автоматически добавлять их в (почти) каждый запрос на тот же домен при помощи заголовка Cookie.

Один из наиболее частых случаев использования куки – это аутентификация:

1. При входе на сайт сервер отсылает в ответ HTTP-заголовок Set-Cookie для того, чтобы установить куки со специальным уникальным идентификатором сессии («session identifier»).
2. Во время следующего запроса к этому же домену браузер посылает на сервер HTTP-заголовок Cookie.
3. Таким образом, сервер понимает, кто сделал запрос.

Мы также можем получить доступ к куки непосредственно из браузера, используя свойство document.cookie.

Чтение из document.cookie

document.cookie - возвращает (хранит) строку

Значение document.cookie состоит из пар ключ=значение, разделённых ;. Каждая пара представляет собой отдельное куки.

Чтобы найти определённое куки, достаточно разбить строку из document.cookie по ;, и затем найти нужный ключ. Для этого мы можем использовать как регулярные выражения, так и функции для обработки массивов.

Запись в document.cookie

Запись в document.cookie обновит только упом
все остальные.

Технически, и имя и значение куки могут состо
форматирования следует использовать встро

Ограничения

```
document.cookie = "user=John"; // обновляем только куки с именем 'user'
alert(document.cookie); // показываем все куки
```

```
// специальные символы (пробелы), требуется кодирование
let name = "my name";
let value = "John Smith"

// кодирует в my%20name=John%20Smith
document.cookie = encodeURIComponent(name) + '=' + encodeURIComponent(value);

alert(document.cookie); // ...; my%20name=John%20Smith
```

Существует несколько ограничений:

- После encodeURIComponent пара name=value не должна занимать более 4Кб. Таким образом, мы не можем хранить в куки большие данные.
- Общее количество куки на один домен ограничивается примерно 20+. Точное ограничение зависит от конкретного браузера.

path

path=/mypath

URL-префикс пути, куки будут доступны для страниц под этим путём. Должен быть абсолютным. По умолчанию используется текущий путь.

Если куки установлено с path=/admin, то оно будет доступно на страницах /admin и /admin/something, но не на страницах /home или /adminpage.

Как правило, указывают в качестве пути корень path=/, чтобы наше куки было доступно на всех страницах сайта.

domain

domain=site.com

Домен определяет, где доступен файл куки. Однако на практике существуют определённые ограничения. Мы не можем указать здесь какой угодно домен.

Нет никакого способа разрешить доступ к файлам куки из другого домена 2-го уровня, поэтому other.com никогда не получит куки, установленный по адресу site.com.

Это ограничение безопасности, позволяющее нам хранить конфиденциальные данные в файлах куки, которые должны быть доступны только на одном сайте.

По умолчанию куки доступны лишь тому домену, который его установил.

```
// если мы установим файл куки на веб-сайте site.com...
document.cookie = "user=John"

// ...мы не увидим его на forum.site.com
alert(document.cookie); // нет user
```

```
// находясь на странице site.com
// сделаем куки доступным для всех поддоменов *.site.com:
document.cookie = "user=John; domain=site.com"

// позже

// на forum.site.com
alert(document.cookie); // есть куки user=John
```

expires, max-age

По умолчанию, если куки не имеют ни одного из этих параметров, то они удалятся при закрытии браузера. Такие куки называются сессионными («session cookies»).

Чтобы помочь куки «пережить» закрытие браузера, мы можем установить значение опций expires или max-age.

expires

expires=Tue, 19 Jan 2038 03:14:07 GMT

Дата истечения срока действия куки, когда браузер удалит его автоматически.

Дата должна быть точно в этом формате, во временной зоне GMT. Мы можем использовать date.toUTCString, чтобы получить правильную дату. Например, мы можем установить срок действия куки на 1 день.

// +1 день от текущей даты

let date = new Date(Date.now() + 86400e3);

date = date.toUTCString();

document.cookie = "user=John; expires=" + date;

Если мы установим в expires прошедшую дату, то куки будет удалено.

```
// +1 день от текущей даты
let date = new Date(Date.now() + 86400e3);
date = date.toUTCString();
document.cookie = "user=John; expires=" + date;
```

max-age

max-age=3600

Альтернатива expires, определяет срок действия куки в секундах с текущего момента.

```
// куки будет удалено через 1 час
document.cookie = "user=John; max-age=3600";

// удалим куки (срок действия истекает прямо сейчас)
document.cookie = "user=John; max-age=0";
```

secure

Куки следует передавать только по HTTPS-протоколу.

По умолчанию куки, установленные сайтом http://site.com, также будут доступны на сайте https://site.com и наоборот.

То есть, куки, по умолчанию, опираются на доменное имя, они не обращают внимания на протоколы.

С этой настройкой, если куки будет установлено на сайте https://site.com, то оно не будет доступно на том же сайте с протоколом HTTP, как http://site.com. Таким образом, если в куки хранится конфиденциальная информация, которую не следует передавать по незашифрованному протоколу HTTP, то нужно установить этот флаг.

```
// предполагается, что сейчас мы на https://
// установим опцию secure для куки (куки доступно только через HTTPS)
document.cookie = "user=John; secure";
```

httpOnly

Эта настройка не имеет ничего общего с JavaScript, но мы должны упомянуть её для полноты изложения.

Веб-сервер использует заголовок Set-Cookie для установки куки. И он может установить настройку httpOnly.

Эта настройка запрещает любой доступ к куки из JavaScript. Мы не можем видеть такое куки или манипулировать им с помощью document.cookie.

Эта настройка используется в качестве меры предосторожности от определённых атак, когда хакер внедряет свой собственный JavaScript-код в страницу и ждёт, когда пользователь посетит её. Это вообще не должно быть возможным, хакер не должен быть в состоянии внедрить свой код на ваш сайт, но могут быть ошибки, которые позволят хакеру сделать это.

Обычно, если такое происходит, и пользователь заходит на страницу с JavaScript-кодом хакера, то этот код выполняется и получает доступ к document.cookie, и тем самым к куки пользователя, которые содержат аутентификационную информацию. Это плохо.

Но если куки имеет настройку httpOnly, то document.cookie не видит его, поэтому такое куки защищено.

LocalStorage, sessionStorage

Объекты веб-хранилища localStorage и sessionStorage позволяют хранить пары ключ/значение в браузере.

Что в них важно – данные, которые в них записаны, сохраняются после обновления страницы (в случае sessionStorage) и даже после перезапуска браузера (при использовании localStorage). Скоро мы это увидим.

Но ведь у нас уже есть куки. Зачем тогда эти объекты?

- В отличие от куки, объекты веб-хранилища не отправляются на сервер при каждом запросе. Именно поэтому мы можем хранить гораздо больше данных. Большинство современных браузеров могут выделить как минимум 5 мегабайтов данных (или больше), и этот размер можно поменять в настройках.
- Ещё одно отличие от куки – сервер не может манипулировать объектами хранилища через HTTP-заголовки. Всё делается при помощи JavaScript.
- Хранилище привязано к источнику (домен/протокол/порт). Это значит, что разные протоколы или поддомены определяют разные объекты хранилища, и они не могут получить доступ к данным друг друга.

LocalStorage, sessionStorage - как с ними работать?

Объекты хранилища localStorage и sessionStorage предоставляют одинаковые методы и свойства:

- setItem(key, value) – сохранить пару ключ/значение.
- getItem(key) – получить данные по ключу key.
- removeItem(key) – удалить данные с ключом key.
- clear() – удалить всё.
- key(index) – получить ключ на заданной позиции.
- length – количество элементов в хранилище.

Только строки

Если мы используем любой другой тип, например число или объект, то он автоматически преобразуется в строку.

Мы можем использовать JSON для хранения объектов.

```
localStorage.user = {name: "John"};
alert(localStorage.user); // [object Object]
```

```
localStorage.user = JSON.stringify({name: "John"});

// немного позже
let user = JSON.parse( localStorage.user );
alert( user.name ); // John
```

sessionStorage

Объект sessionStorage используется гораздо реже, чем localStorage.

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.
- Но оно разделяется между ифреймами на той же вкладке (при условии, что они из одного и того же источника).
- Данные продолжают существовать после перезагрузки страницы, но не после закрытия/открытия вкладки.

sessionStorage привязан не только к источнику, но и к вкладке браузера. Поэтому sessionStorage используется нечасто.

Событие storage

```
// срабатывает при обновлениях, сделанных в том же хранилище из других документов
window.onstorage = event => {
  // можно также использовать window.addEventListener('storage', event => {
  if (event.key != 'now') return;
  alert(event.key + ':' + event.newValue + " at " + event.url);
};

localStorage.setItem('now', Date.now());
```

Когда обновляются данные в localStorage или sessionStorage, генерируется событие storage со следующими свойствами:

- key – ключ, который обновился (null, если вызван .clear()).
- oldValue – старое значение (null, если ключ добавлен впервые).
- newValue – новое значение (null, если ключ был удалён).
- url – url документа, где произошло обновление.
- storageArea – объект localStorage или sessionStorage, где произошло обновление.

Важно: событие срабатывает на всех остальных объектах window, где доступно хранилище, кроме того окна, которое его вызвало.

IndexedDB

IndexedDB – это встроенная база данных, более мощная, чем localStorage.

- Хранит практически любые значения по ключам, несколько типов ключей
- Поддерживает транзакции для надёжности.
- Поддерживает запросы в диапазоне ключей и индексы.
- Позволяет хранить больше данных, чем localStorage.

Где хранятся данные?

Технически данные обычно хранятся в домашнем каталоге посетителя вместе с настройками браузера, расширениями и т.д.

У разных браузеров и пользователей на уровне ОС есть своё собственное независимое хранилище.

Основные моменты, которые нужно знать о IndexedDB

Браузерное хранилище: IndexedDB позволяет хранить данные на стороне клиента в браузере. Это хранилище полезно для приложений, которые должны работать в автономном режиме.

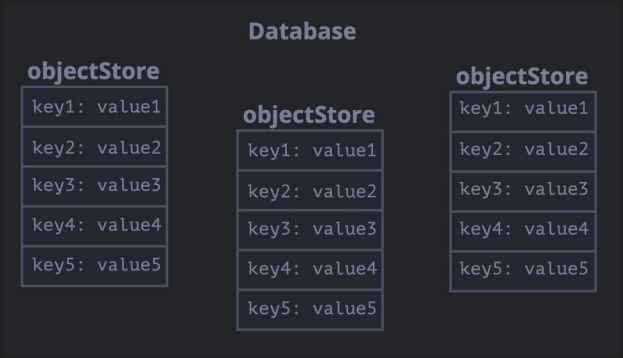
Объектное хранилище: Данные в IndexedDB хранятся как объекты. В отличие от реляционных баз данных, здесь нет таблиц, а есть хранилища объектов (object stores).

Ключи и индексы: Для быстрого поиска данных используются ключи и индексы. Каждый объект в хранилище имеет уникальный ключ.

Асинхронность: Все операции в IndexedDB асинхронны. Это значит, что выполнение запросов не блокирует основной поток выполнения кода, что особенно важно для работы в браузере.

Открытие базы данных

IndexedDB имеет встроенный механизм «версионирования схемы», который отсутствует в серверных базах данных.




```
let request = indexedDB.open("MyDatabase", 1);

request.onupgradeneeded = function(event) {
    let db = event.target.result;
    // Создание хранилища объектов (object store) с именем "customers"
    let objectStore = db.createObjectStore("customers", { keyPath: "id" });
    // Создание индекса по имени
    objectStore.createIndex("name", "name", { unique: false });
};

request.onsuccess = function(event) {
    console.log("Database opened successfully");
};

request.onerror = function(event) {
    console.error("Database error: " + event.target.errorCode);
};
```

Добавление данных

Все операции с данными в IndexedDB могут быть сделаны только внутри транзакций.

Производительность является причиной, почему транзакции необходимо помечать как readonly или readwrite.

Несколько readonly транзакций могут одновременно работать с одним и тем же хранилищем объектов, а readwrite транзакций – не могут. Транзакции типа readwrite «блокируют» хранилище для записи. Следующая такая транзакция должна дождаться выполнения предыдущей, перед тем как получит доступ к тому же самому хранилищу.

```
let db;
request.onsuccess = function(event) {
    db = event.target.result;

    let transaction = db.transaction(["customers"], "readwrite");
    let objectStore = transaction.objectStore("customers");

    let customer = { id: 1, name: "John Doe", email: "john.doe@example.com" };
    let addRequest = objectStore.add(customer);

    addRequest.onsuccess = function(event) {
        console.log("Customer added to the database");
    };

    addRequest.onerror = function(event) {
        console.error("Error adding customer: " + event.target.errorCode);
    };
};
```

Чтение данных

```
let transaction = db.transaction(["customers"], "readonly");
let objectStore = transaction.objectStore("customers");

let getRequest = objectStore.get(1);

getRequest.onsuccess = function(event) {
    if (getRequest.result) {
        console.log("Customer:", getRequest.result);
    } else {
        console.log("Customer not found in the database");
    }
};

getRequest.onerror = function(event) {
    console.error("Error getting customer: " + event.target.errorCode);
};
```

Удаление конкретной записи

Для удаления конкретной записи нужно создать транзакцию с правами на запись и использовать метод delete объекта хранилища (object store).

```
let db;
let request = indexedDB.open("MyDatabase", 1);

request.onsuccess = function(event) {
    db = event.target.result;

    let transaction = db.transaction(["customers"], "readwrite");
    let objectStore = transaction.objectStore("customers");

    let deleteRequest = objectStore.delete(1); // Удаление записи с ключом 1

    deleteRequest.onsuccess = function(event) {
        console.log("Customer deleted from the database");
    };

    deleteRequest.onerror = function(event) {
        console.error("Error deleting customer: " + event.target.errorCode);
    };
};

request.onerror = function(event) {
    console.error("Database error: " + event.target.errorCode);
};
```

Удаление всех записей

Для удаления всех записей из хранилища объектов можно использовать метод clear.

```
let db;
let request = indexedDB.open("MyDatabase", 1);

request.onsuccess = function(event) {
    db = event.target.result;

    let transaction = db.transaction(["customers"], "readwrite");
    let objectStore = transaction.objectStore("customers");

    let clearRequest = objectStore.clear(); // Удаление всех записей

    clearRequest.onsuccess = function(event) {
        console.log("All customers deleted from the database");
    };

    clearRequest.onerror = function(event) {
        console.error("Error clearing customers: " + event.target.errorCode);
    };
};

request.onerror = function(event) {
    console.error("Database error: " + event.target.errorCode);
};
```

Ресурсы

Пример popup - ТЫК

Открытие окон и методы window - ТЫК

Общение между окнами - ТЫК

Куки, document.cookie - ТЫК

LocalStorage, sessionStorage - ТЫК

IndexedDB - ТЫК