**Event Loop і асинхронність:**

1. <https://docs.google.com/presentation/d/1CX9BamJ02FSdqahSBPfaIILhXbNISRjcs_B7nB59bpo/edit#slide=id.p>
2. <https://developer.mozilla.org/ru/docs/Web/JavaScript/EventLoop>
3. <https://medium.com/@pavelbely/javascript-event-loop-%D0%B2-%D0%BA%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0%D1%85-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1-a19e4d99f242>
4. <https://habr.com/ru/company/ruvds/blog/340508/>

**JavaScrip-движок** - зчитує змінні і об’явлені функції і записує їх в глобальну пам’ять. У кожного движка є стек викликів **call-stack.** В **call-stack** елементи можуть добавлятися зверху, але не можуть видалятися із структури, доки над ними є інші елементи. Загалом движок займається:

1. Поміщення в глобальну пам’ять змінних і об’явлених функцій.
2. Відправляє виклики кожної функції в стек викликів
3. Створює глобальний контекст виконання в якому виконуються глобальні функції
4. Створює багато маленьких локальних контекстів (якщо є внутрішні змінні чи вложені функції)

**Однопоточність** – наявність одного стека при якому методи виконуються за принципом останній зайшов – перший вийшов. JS однопоточна мова програмування.

**Ассинхронність** – метод виконання коду при якому асинхронні дії виконуються по черзі після спустошення call stack (виконання основного коду ) до того перебуваючи в **callback queue**

**event loop** – цикл, провіряє чи пустий стек виклику(call stack), якщо в call stack пустий, то асинхронні функції, які зберігались в call queue переносяться в call stack для виконання після відпрацювання основного коду.

**AJAX** – термін, який має на меті динамічну зміну сторінки при ‘спілкування’ з сервером, яке не приводить до перезагрузки сторінки

1. <https://habr.com/ru/post/120917/>
2. <https://developer.mozilla.org/ru/docs/Web/Guide/AJAX>
3. <https://docs.google.com/presentation/d/1zk7gYssA9iWoTB9mkxH98Dc1y5xuiapbBnG0BhBTuys/edit#slide=id.g5dbb9b3eea_2_65> - робота мережі і ajax

**Використання даної технології:**

1. const **xhr** = **new** **XMLHttpRequest()**; - отримуємо об’єкт для формування запитів

**Налаштування запитів**

1. **xhr**.**open**(mehod, url, asinc, user, password)
   1. “(GET/POST…) – метод запиту”,
   2. “URL” – шлях надсилання запиту,
   3. asinc – булева змінна, що вказує, чи потрібно запит опрацьовувати асинхронно,
   4. user, password – Дані користувача і пароль для входу, якщо потрібно
2. **xhr**.**send**([body]) – метод для відкриття з’єднання і відправки запиту. [body] – опціональний параметр, для get не потрібен
3. **xhr**.**abort**() – метод для скасування виконання запиту
4. **xhr.setRequestHeader**('Content-Type', 'image/jpeg'); - методи для встановлення заголовків (додаткових параметрів для сервера)
5. **xhr.getResponseHeader**('Content-Type') – метод отримання заголовків
6. **xhr.getAllResponseHeaders()** – метод для отримання усіх заголовків
7. **xhr.status – повертає статус виконання запиту**
8. **xhr.readyState() –** Статус опрацювання запиту

0 (uninitialized)

1 (loading)

2 (loaded)

3 (interactive)

4 (complete)

1. **xhr.statusText – повертає текст статусу виконання запиту**
2. **xhr.responseText –** тіло відповіді, те що ми попросили у сервера
3. **xhr.addEventListener**(**“load”**, () => {}); - подія для прослуховування відповіді від сервера
4. **xhr.timeout** = ms **–** метод для задання часу очікування відповіді від сервера**.** Коли час вичерпається – з’єднання із сервером буде розірвано
5. **xhr.addEventListener**(**“timeout”**, () => {}); - метод для прослушки моменту розриву з’єднання із сервером

**Типи подій для роботи із сервером:**

1. **loadstart** – запит початий;

2. **progress** – браузе получить ще один пакет данних (responseText);

3. **abort** – запит був відхилений методом xhr.abort();

4. **error** – відбулась помилка;

5. **load** – запит був успішно завершений;

6. **timeout** – запит був вечерпаний по вказаному часу;

7. **loadend** – запит був закінчений (успішно чи не успішно).

**CORS**

<https://medium.com/@baphemot/understanding-cors-18ad6b478e2b>

**CORS** - це механізм, який спрямований на те, щоб дозволити запити, зроблені від вашого імені, і в той же час блокувати деякі запити, зроблені шахрайським JS, і спрацьовує кожен раз, коли ви робите запит HTTP:

Доступ здійснюється лише завдяки заголовкам і певним значенням в них:

**Cors заголовки:**

**Access-Control-Allow-Origin**

Це заголовок призначений для повернення сервером і вказує, яким клієнтським доменам дозволено отримувати доступ до його ресурсів. Значення може бути:

* \* - дозволити будь-який домен
* повноцінне доменне ім’я (наприклад, https://example.com)

## Access-Control-Allow-Credentials

Цей заголовок повинен бути присутнім у відповіді, лише якщо ваш сервер підтримує автентифікацію за допомогою файлів cookie. Єдине допустиме значення для цього випадку true.

## Access-Control-Allow-Header Надає розділений комами список значень заголовків запитів, які сервер готовий підтримувати. Якщо ви використовуєте власні заголовки (наприклад, x-authentication-token, вам потрібно повернути його у цій відповіді заголовка ACA на виклик OPTIONS, інакше запит буде заблоковано.

## Access-Control-Expose-Headers

Аналогічно, ця відповідь повинна містити список заголовків, які будуть присутні у фактичній відповіді на дзвінок і повинні бути доступними для клієнта. Усі інші заголовки будуть обмежені

**Access-Control-Allow-Methods**  
Список розділених комами дієслів типу запиту HTTP (наприклад, GET, POST), які сервер готовий підтримати.

## Origin

Цей заголовок є частиною запиту, який робить клієнт, і міститиме домен, з якого запускається програма. З міркувань безпеки браузери не дозволять вам перезаписати це значення.

## CallBackHells

## CallbackHells – назва способу використання функцій js. Замість повернути результат одразу, використовують зворотні виклики, які займають деякий час або відбудеться в майбутньому. Проблематика такого методу полягає у тому, що виконання не відбувається зверху вних, а пригає в залежності від виконання. Для поправки даної ситупції потрібно притримуватись даних вимог:

## Тримати код не глибоким

## Створюйте модулі – багаторазові методи перемістити у модулі для використання у любій частині коду

## Обробляйте кожну помилку

**Promises, fetch, asinc/await**

1. <https://habr.com/ru/company/ruvds/blog/340508/>
2. <https://docs.google.com/presentation/d/1gNQuKKYSp3Yx7fpFI1jkXwNatv733oMNMAOik3E7sNo/edit#slide=id.g5dba56d409_2_92>
3. https://stasonmars.ru/javascript/polnoe-ponimanie-syncronnogo-i-asyncronnogo-javascript-s-async-await/

**Fatch**

<https://learn.javascript.ru/fetch>

<https://developer.mozilla.org/ru/docs/Web/API/Fetch_API/Using_Fetch>

<https://habr.com/ru/post/252941/>

let promise = fetch(url, [options])

* **url** – URL для відправки запиту
* **options** – додаткові параметри: метод , заголовки і так далі.

Для встановлення заголовка в  fetch ми можемо використовувати опцію **headers**.

Без параметрів це простий GET-запит, викачує контент за адресою url.

Браузер відразу ж починає запит і повертає промис, який використовує зовнішній код для отримання результату.

**Методи для об’єкта response:**

* **response**.**status** – HTTP-код відповді,
* **response**.**ok** – true, якщо статус відповіді в діапазоні 200-299.
* **response**.**headers** – об’єкт з HTTP-заголовками.

**Методы для получения тела ответа:**

* **response.text()** – повертає результат, як звичайни текст
* **response.json()** – перетворює резкльтат в JSON фортмат,
* **response.formData()** – повертає об’єкт, як FormData (кодіровка form/multipart),
* **response.blob()** – повертає об’єкт, як [Blob](https://learn.javascript.ru/blob) (бінарні дані з типом),
* **response.arrayBuffer()** – повертає відповідь, як [ArrayBuffer](https://learn.javascript.ru/arraybuffer-binary-arrays) (низькорівневі бінарні дані)

**Опції Fetch, які ми вивчили на даний момент:**

* **method**– HTTP-метод,
* **headers**– об’єкт із заголовками для запиту (не всі заголовки дозволені),
* **body**– дані для відправки (тіло запиту) в вигляді текста, FormData, BufferSource, Blob чи UrlSearchParams.

**Async / Await**

1. <https://habr.com/ru/company/ruvds/blog/340508/>
2. <https://stasonmars.ru/javascript/vse-chto-nuzhno-znat-ob-async-await-tsikly-kontrol-potokov-ogranicheniya/>
3. <https://stasonmars.ru/javascript/polnoe-ponimanie-syncronnogo-i-asyncronnogo-javascript-s-async-await/>

**Asinc** - перед методом змусить його завжди повертати **promise.**  Якщо така функція повертає значення, яке не є об'єктом Promise. Завдяки чому ми можемо не прописувати в даному методі обготку Promise, але отримаємо його на виході і використовуватимемо його методи. Асинхронні функції оголошують, користуючись ключовим словом **async**.

**Await** – заморожує код до виконання асинхронної дії, задається будь-чому. Ключове слово **await** можна використовувати тільки в функціях, оголошених з ключовим словом **async**. Умовно все, до чого застосовується дана приставка – обгортається в Promise.resolve().then(…)

**Переваги використання asinc/await**

Кожен раз, коли ми прописуємо до асинхронного методу **asinc/await** – ми позбавляємося потреби прописувати **asinc/await.**

Конструкція **async / await** дозволяє обробляти синхронні і асинхронні помилки з використанням одних і тих же механізмів.

Полегшує написання умов при використанні **asinc/await** це робиться із використанням меншого об’єму коду.

На відміну від **async / await**, стек помилок, повернутий з ланцюжка Promises, не містить відомостей про точне місце, в якому сталася помилка. А при використанні **asinc/await** інформація про помилку вкаже більш точні дані щодо помилки.

З використанням **async / await** можна переходити за викликами, в яких використовується ключове слово **await** так, ніби це - звичайні синхронні операції.

**Методи**

* 1. await timeoutPromise(1000); - створює паузу в асинхронних методах