



GEO, NOTIFICATION, MEDIA



ИЛЬЯ МЕДЖИДОВ



ИЛЬЯ МЕДЖИДОВ

CEO в RageMarket

 medzhidov@ragemarket.ru

ПЛАН ЗАНЯТИЯ

1. Geolocation

2. Notification

3. Media

HTTPS VS LOCALHOST

Обратите внимание, API, которое мы будем с вами сегодня рассматривать, будет работать только по HTTPS или на localhost/127.0.0.1 (в целях тестирования).

Geolocation

ЗАДАЧА

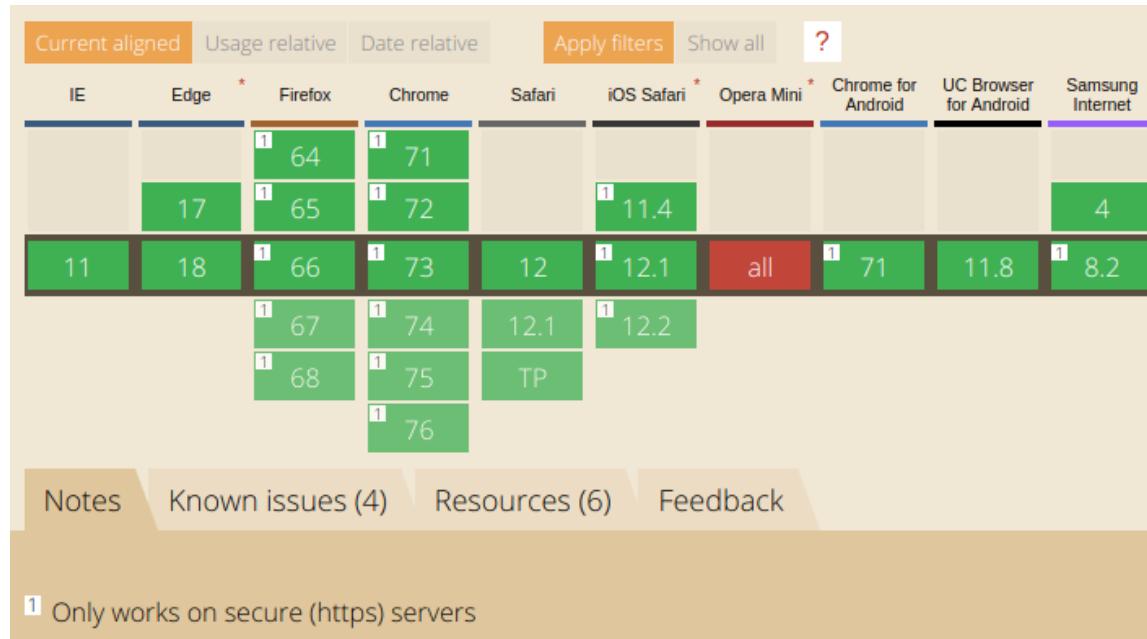
Мы делаем портал для большой компании, у которой есть по несколько филиалов в крупных городах. Но, при этом, филиалы есть не в каждом городе.

Заказчик хочет, чтобы при входе посетителя на сайт автоматически определялся ближайший филиал, исходя из геолокации пользователя. Если же она недоступна, то достаточно показать стандартную всплывашку «Выберите город».

Geolocation

Посмотрим, какие решения есть для этого:

1. Geolocation API.
2. Провайдеры API вроде Yandex Maps или Google Maps Platform.



Рассмотрим первый вариант и поговорим о втором.

navigator.geolocation

Проверяем, доступно ли API в браузере:

```
if (navigator.geolocation) {  
    // доступно  
}
```

Geolocation

Свойство `navigator.geolocation` соответствует интерфейсу `Geolocation`:

```
1 interface Geolocation {
2     void getCurrentPosition(
3         PositionCallback successCallback,
4         optional PositionErrorCallback errorCallback,
5         optional PositionOptions options
6     );
7     long watchPosition(
8         PositionCallback successCallback,
9         optional PositionErrorCallback errorCallback,
10        optional PositionOptions options
11    );
12    void clearWatch(long watchId);
13 }
```

Geolocation

Таким образом:

1. Мы можем попытаться получить текущую позицию:

`getCurrentPosition`.

2. Подписаться на получение позиции: `watchPosition`.

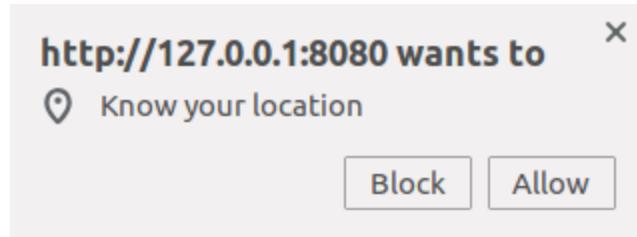
3. Отписаться от получения позиции (если мы на неё подписались):

`clearWatch`.

Первые два метода принимают callback'и на успешные, и неуспешные уведомления.

ПОЛУЧАЕМ КООРДИНАТЫ

В браузере мы увидим запрос на подтверждение:



ПОЛУЧАЕМ КООРДИНАТЫ

После чего нам в callback придёт объект интерфейса `Position`, в котором и будут необходимые координаты и точность:

```
1 interface Position {
2     readonly attribute Coordinates coords;
3     readonly attribute DOMTimeStamp timestamp;
4 };
5 interface Coordinates {
6     readonly attribute double latitude;
7     readonly attribute double longitude;
8     readonly attribute double? altitude;
9     readonly attribute double accuracy;
10    readonly attribute double? altitudeAccuracy;
11    readonly attribute double? heading;
12    readonly attribute double? speed;
13};
```

ПОЛУЧАЕМ КООРДИНАТЫ

```
const {latitude, longitude, accuracy} = position.coords;
```

Координаты приходят в градусах, а точность в метрах. На базе этого, по соответствующим формулам, уже можно решать задачу поиска ближайшего города/филиала.

Ключевые моменты:

1. Мы работаем на геоиде, поэтому формулы должны считать по поверхности, а не «сквозь землю».
2. Обязательно учитывайте ошибку и давайте возможность пользователю изменить то значение, которое вы выбрали по умолчанию.

ОШИБКА

Пользователь может не дать своего разрешения на передачу информации о геолокации, либо она может быть недоступна. Тогда вызовется errorCallback с аргументом `PositionError`:

```
1 interface PositionError {
2     const unsigned short PERMISSION_DENIED = 1;
3     const unsigned short POSITION_UNAVAILABLE = 2;
4     const unsigned short TIMEOUT = 3;
5     readonly attribute unsigned short code; //код ошибки см. выше
6     readonly attribute DOMString message; //текст ошибки
7 }
```

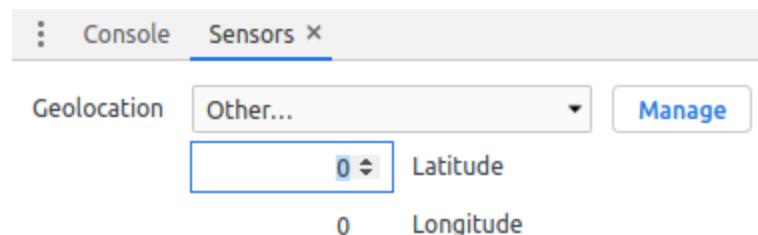
ТОЧНОСТЬ

Мы можем использовать последний (необязательный) параметр `PositionOptions`, для указания настроек получения позиции:

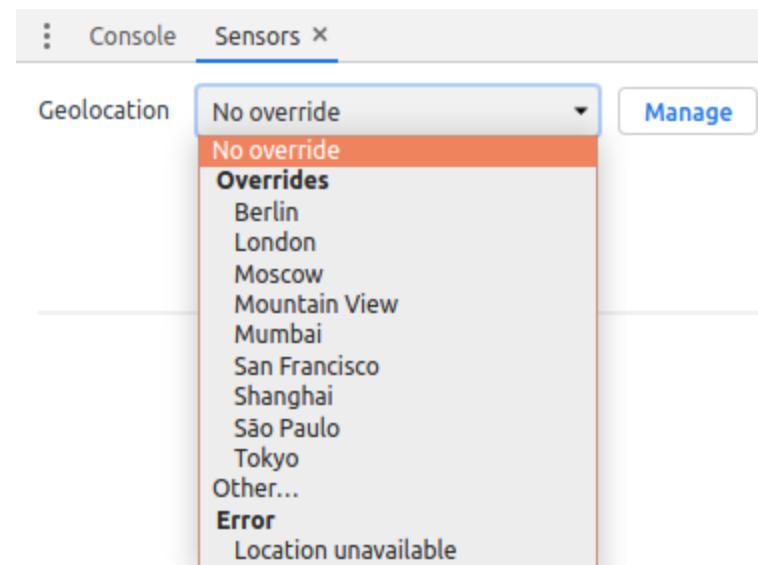
```
1 | dictionary PositionOptions {
2 |     boolean enableHighAccuracy = false;
3 |         // высокая точность
4 |     [Clamp] unsigned long timeout = 0xFFFFFFFF;
5 |         // через сколько ms вызовется errorCallback
6 |     [Clamp] unsigned long maximumAge = 0;
7 |         // сколько кэшировать в ms
8 | };
```

ТЕСТИРОВАНИЕ

Developer Tools позволяет нам как задать определённые координаты для эмуляции:



Так и сэмулировать ошибку:



ОГРАНИЧЕНИЯ

Geolocation API – это здорово, но:

1. Нужно вникать в «геометрию» и самостоятельно писать алгоритмы расчёта.
2. Мы считаем по «минимальному» расстоянию, не учитывая рельеф, реки, дороги, границы, океаны и т.д.
3. Если мы захотим отобразить ближайший объект и пользователя на карте, то это будет сложновато сделать*.

* Сделать это можно, дав дизайнеру задание нарисовать нужную картинку и пересчитав координаты в позицию элемента на этой картинке.

ПРОВАЙДЕРЫ API

Ключевых провайдеров, предоставляющих API для работы с картами, мы укажем два:

- [Google Maps Platform](#)
- [Yandex Maps](#)

ПРОВАЙДЕРЫ API

Рассмотрение данных провайдеров выходит за рамки нашего курса, мы лишь отметим несколько ключевых моментов:

1. API достаточно часто обновляется.
2. API требует API KEY (для этого вам необходимо создать учётную запись на этих сервисах и в настройках включить API).
3. Провайдеры всё чаще тарифицируют API по количеству запросов, поэтому будьте аккуратны.

Notification

ЗАДАЧА

Ваша компания разрабатывает систему мониторинга с веб-интерфейсом, и в качестве реализации новой функциональности поступила задача **alerting** 'а – уведомления о важных событиях по различным каналам:

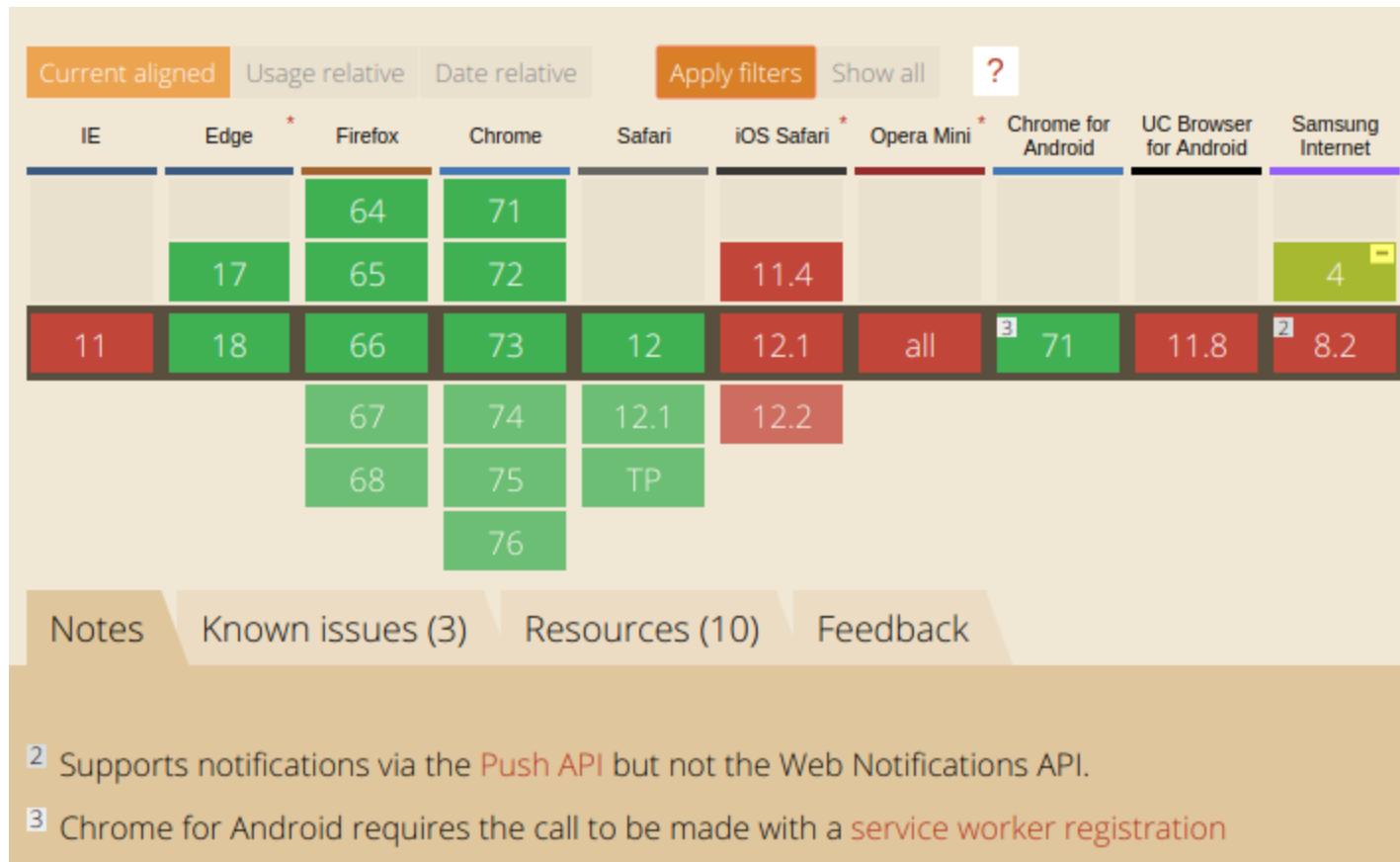
- SMS;
- мессенджеры.

И, что самое важное, в рамках реализации этой задачи потребовали отображать уведомления на рабочем столе операторов (т.е. выйти за пределы окна браузера).

Посмотрим, какие решения нам предлагаются для этого.

NOTIFICATION API

Notification API предоставляет нам API, позволяющие отображать подобные уведомления на десктопных и мобильных браузерах:



PERMISSIONS

Поскольку уведомления могут быть достаточно назойливыми, создатели предусмотрели механизм запроса разрешений на показ (и он отличается от `Geolocation`):

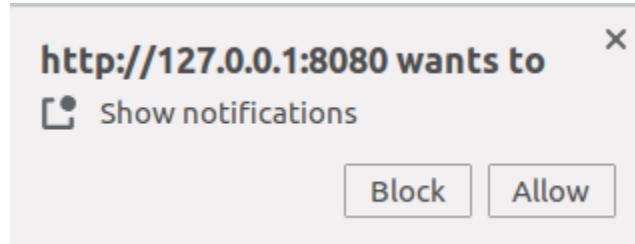
```
1 | (async() => {
2 |   if (!window.Notification) {
3 |     // didn't support notification
4 |     return;
5 |   }
6 |
7 |   const permission = await Notification.requestPermission();
8 | })();
```

ИНТЕРФЕЙС Notification

```
1 interface Notification : EventTarget {  
2     static readonly attribute NotificationPermission permission;  
3     [Exposed=Window] static Promise<NotificationPermission> requestPermission(  
4         optional NotificationPermissionCallback deprecatedCallback  
5     );  
6  
7     attribute EventHandler onclick;  
8     attribute EventHandler onshow;  
9     attribute EventHandler onerror;  
10    attribute EventHandler onclose;  
11    ...  
12    void close();  
13 }
```

Т.е. `requestPermission` возвращает нам `Promise`, который разрешится `NotificationPermission`.

NotificationPermission



```
1 enum NotificationPermission {  
2     "default",  
3         // пользователь не сделал выбор (не можем показывать)  
4     "denied",  
5         // пользователь запретил (не можем показывать)  
6     "granted"  
7         // пользователь разрешил (можем показывать)  
8 };
```

RequestPermission

С `requestPermission` связано несколько особенностей:

1. При достаточно частых запросах браузеры просто блокируют следующие запросы.
2. Если пользователь отклонил (`denied`), то следующие запросы даже не будут отображаться.

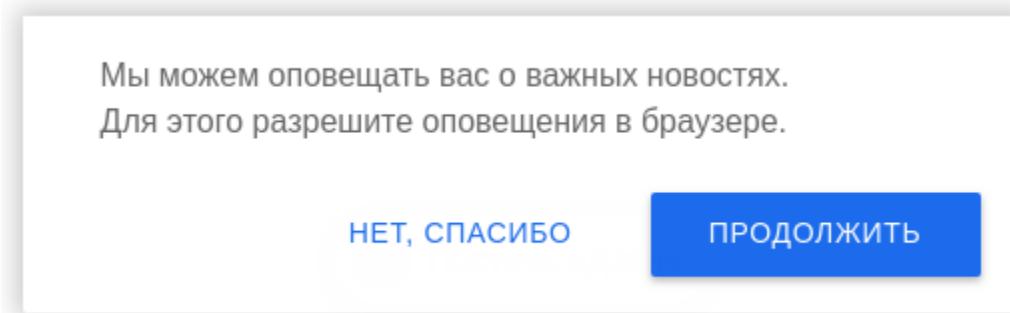
RequestPermission

Кроме того, если пользователь уже разрешил (granted), то нам и запрашивать нет смысла. Переделаем, исходя из этого, наш код:

```
1 | (async() => {
2 |   if (!window.Notification) {
3 |     // didn't support notification
4 |     return;
5 |   }
6 |   if (Notification.permission === 'granted') {
7 |     // TODO: show notification
8 |     return;
9 |   }
10 |   if (Notification.permission === 'default') {
11 |     const permission = await Notification.requestPermission();
12 |     if (permission === 'granted') {
13 |       // TODO: show
14 |     }
15 |   }
16 | })();
```

Notification

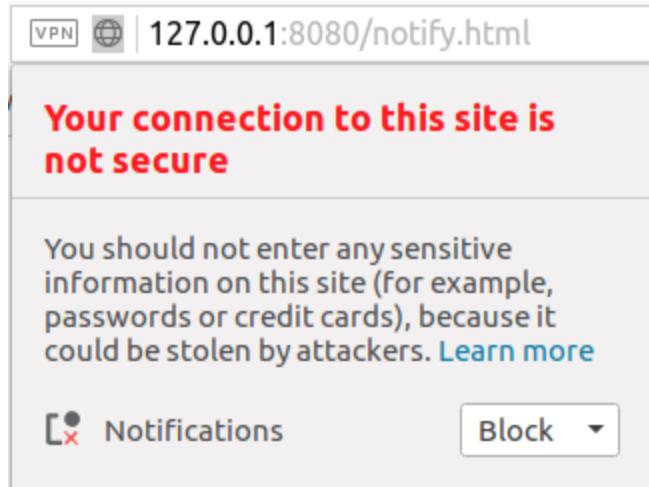
Стоп-стоп, но вы скажете, что на большинстве сайтов выпадает другое окошко, например, вот такое:



На самом деле, это просто DOM-элемент, который позволяет «не попадать на блокировку», если пользователь откажется. Т.е. `requestPermission` будет делаться только тогда, когда пользователь нажмёт «Продолжить».

ТЕСТИРОВАНИЕ

При тестировании разных сценариев `default`, `granted`, `denied` нужно ещё уметь и обратно «включать» необходимое состояние:



ПОКАЗ УВЕДОМЛЕНИЙ

Показ уведомлений производится путём создания объекта `Notification`. Конструктор выглядит следующим образом:

```
1 | Constructor(DOMString title, optional NotificationOptions options);
```

```
1 | dictionary NotificationOptions {
2 |   NotificationDirection dir = "auto";
3 |   DOMString lang = "";
4 |   DOMString body = "";
5 |   DOMString tag = "";
6 |   USVString image;
7 |   USVString icon;
8 |   USVString badge;
9 |   VibratePattern vibrate;
10 |  DOMTimeStamp timestamp;
11 |  boolean renotify = false;
12 |  boolean silent = false;
13 |  boolean requireInteraction = false;
14 |  any data = null;
15 |  sequence<NotificationAction> actions = [];
16 |};
```

ПОКАЗ УВЕДОМЛЕНИЙ

Большинство опций говорят сами за себя, мы рассмотрим самые интересные:

```
1 const notification = new Notification('Sample notification', {  
2   body: 'Long long long notification body text',  
3   image: '/img/js.png',  
4   icon: '/img/netology.png',  
5 });
```

PLATFORM SPECIFIC

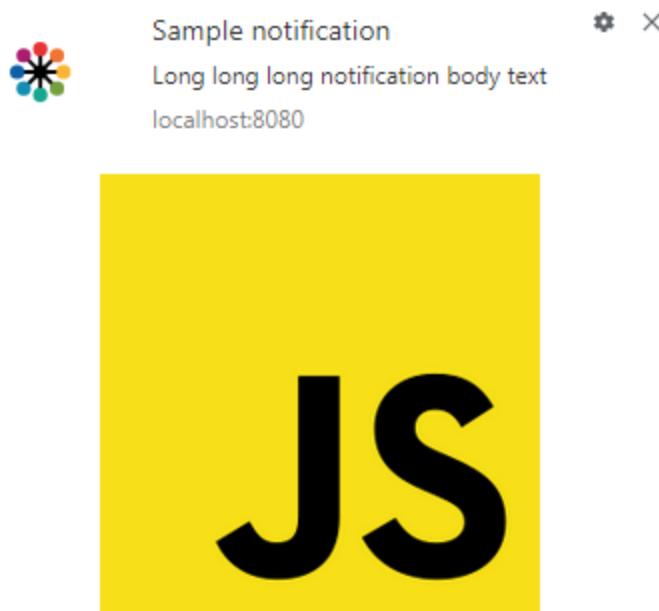
Внешний вид уведомлений будет очень сильно зависеть от платформы и браузера.

Windows Chrome:



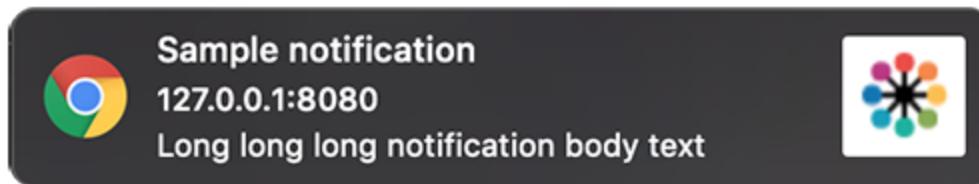
PLATFORM SPECIFIC

Windows FF:

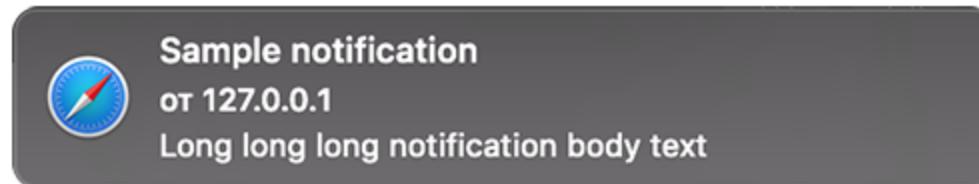


PLATFORM SPECIFIC

MacOS Chrome:

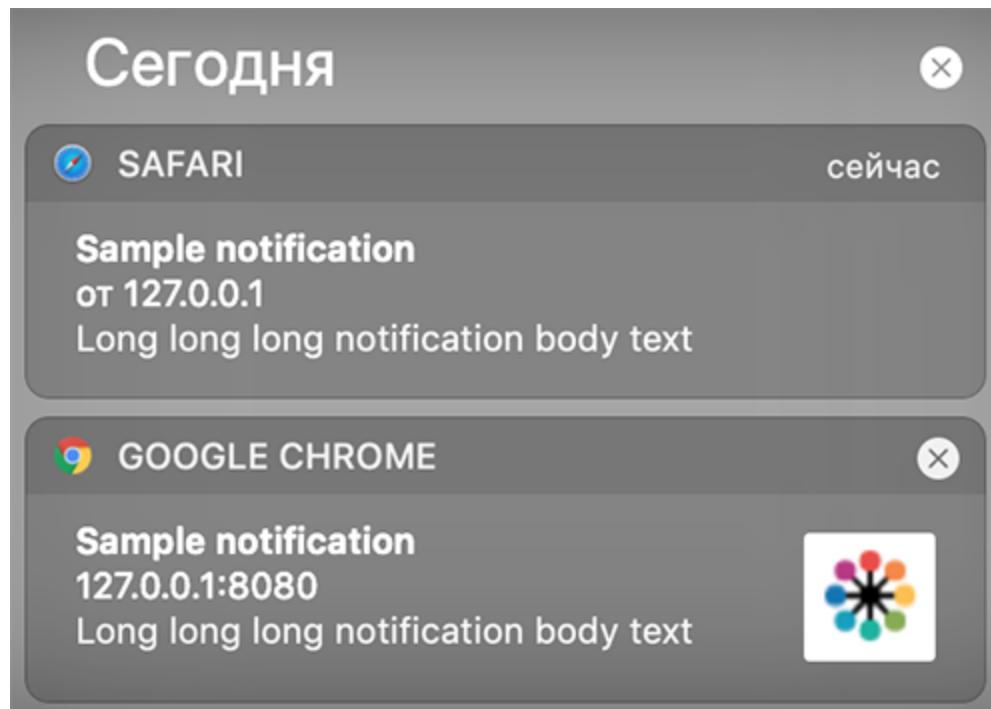


MacOS Safari:



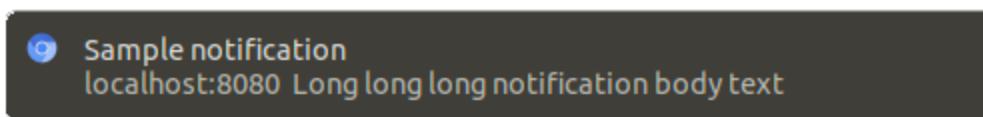
PLATFORM SPECIFIC

Кроме того, появятся в центре уведомлений:

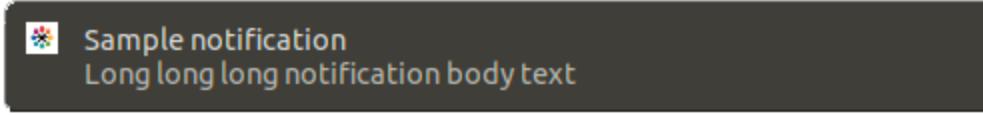


PLATFORM SPECIFIC

Ubuntu Chromium:



Ubuntu FF:



ВЗАИМОДЕЙСТВИЕ

Как видно при запуске, уведомление появляется и пропадает спустя несколько секунд. При этом пользователь может с этим уведомлением вообще никак не взаимодействовать.

Рассмотрим, как мы можем повлиять на это:

```
1 const notification = new Notification('Sample notification', {  
2   body: 'Long long long notification body text',  
3   image: '/img/js.png',  
4   icon: '/img/netology.png',  
5   requireInteractions: true, // <- требуем взаимодействия  
6 });
```

ВЗАИМОДЕЙСТВИЕ

Теперь уведомление не пропадает до тех пор, пока пользователь с ним не взаимодействует:

- либо закроет;
- либо кликнет на теле уведомления (при этом оно так же закроется).

Мы можем реагировать на эти события либо через `EventHandler`'ы, либо через `EventListener`'ы.

ВЗАЙМОДЕЙСТВИЕ

Давайте вспомним лекцию про события и попробуем это сделать через `EventHandler`'ы:

```
1 const notification = new Notification('Sample notification', {  
2   ...  
3   requireInteraction: true,  
4 });  
5  
6 notification.onclick = () => {  
7   console.log('click');  
8 }  
9  
10 notification.onclose = () => {  
11   console.log('closed');  
12 }
```

TAG

Если выводить уведомления одно за другим, в зависимости от платформы, они будут либо выстраиваться в стек, либо заменять одно на другое. При этом «старые» будут закрываться.

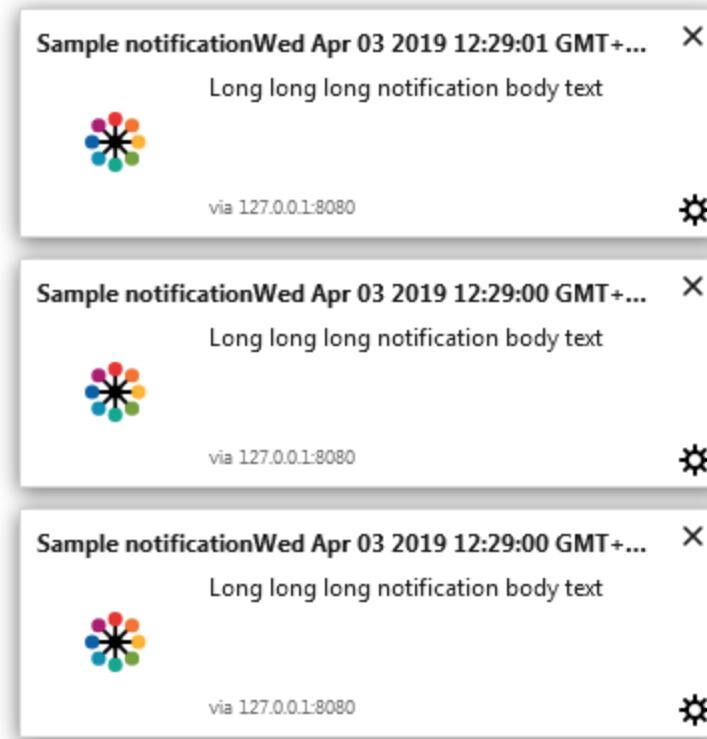
Строковое значение `tag` позволяет «обновлять» существующее уведомление вместо показа нового:

```
1 | const notification =
2 | new Notification('Sample notification' + new Date(), {
3 |   ...
4 |   tag: 'fixed',
5 |   requireInteraction: true,
6 | });

```

TAG

Без `tag` уведомления будут выстраиваться так (а с `tag` уведомление будет только одно):



Естественно, если установить другой `tag`, то он не будет замещать уведомление.

ИТОГ

Таким образом, останется лишь привязать данную функциональность к нашей логике – на данный момент для этого нужен сервер и периодические запросы через `setTimeout / setInterval`, например:

```
1 if (Notification.permission === 'granted') {
2   const interval = 5000;
3   setInterval(async () => {
4     try {
5       const response = await fetch(...);
6       const data = await response.json();
7
8       data.forEach((o) => {
9         const notification = new Notification(o.title, {
10           body: o.body,
11           icon: o.icon,
12           requireInteraction: true,
13         });
14
15         notification.onclick = () => {
16           console.log('click');
17         }
18
19         notification.onclose = () => {
20           console.log('closed');
21         }
22       });
23     } catch (e) {
24       console.error(e);
25     }
26   }, interval)
27 }
```

ИТОГ

Позже мы научимся использовать другие технологии для получения информации о событиях на сервере.

НЮАНСЫ

При использовании кода выше важно понимать, что сервер не должен возвращать больше нескольких уведомлений (в зависимости от платформы лучше не возвращать больше 1) и параллельно нужно дублировать их визуально в DOM.

ВЗАИМОДЕЙСТВИЕ

В принципе, это всё, что нам доступно в плане отображения контента и взаимодействия с ним. Мы не можем добавить собственные кнопки `actions`, если работаем из Window. Эта возможность будет доступна только из Service Worker'ов, которые мы будем проходить позже.

MEDIA

ЗАДАЧА

Мы делаем веб-мессенджер, и одна из ключевых фишек – давать возможность записывать пользователю аудио и видео сообщения для дальнейшей отправки.

MEDIA CAPTURE & STREAMS

Спецификация [Media Capture & Streams](#) предлагает нам ряд API для взаимодействия с микрофоном и видео-камерой устройства для получения звука и изображения.

Начнём решать наши задачи последовательно:

1. Запись аудио.
2. Запись видео.

MEDIADEVICES

Первое, с чего нужно начать, это объект `MediaDevices`, который является входной точкой для API:

```
1 interface MediaDevices : EventTarget {  
2     attribute EventHandler ondevicechange;  
3     Promise<sequence<MediaDeviceInfo>> enumerateDevices();  
4     Promise<MediaStream> getUserMedia(MediaStreamConstraints constraints);  
5 };
```

Метод `getUserMedia` позволяет получить ссылку на `MediaStream`, при этом мы можем указать, требования к устройству:

```
1 dictionary MediaStreamConstraints {  
2     (boolean or MediaTrackConstraints) video = false;  
3     (boolean or MediaTrackConstraints) audio = false;  
4 };
```

MEDIASTREAM

Интерфейс, представляющий группу `MediaStreamTrack`'ов. Очень упрощённо можно считать объектом, который позволяет сгруппировать несколько треков, для того, чтобы затем использовать в медиа-элементах (например, видео или аудио).

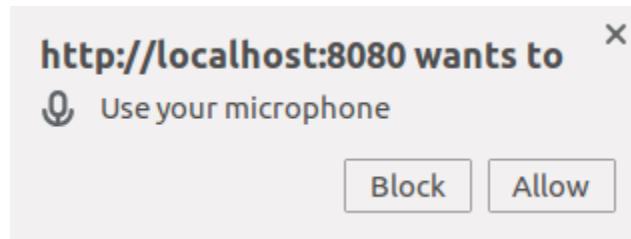
```
1 interface MediaStream : EventTarget {  
2     readonly attribute DOMString id;  
3     sequence<MediaStreamTrack> getAudioTracks();  
4     sequence<MediaStreamTrack> getVideoTracks();  
5     sequence<MediaStreamTrack> getTracks();  
6     ...  
7 };
```

ДУБЛИРОВАНИЕ ЗВУКА

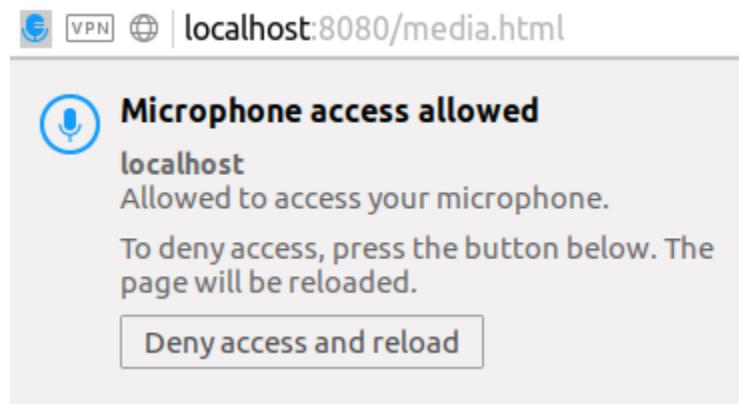
```
1 | <audio id="audio" controls>
2
3 | 1(async() => {
4 |   if (!navigator.mediaDevices) {
5 |     return;
6 |   }
7 |   try {
8 |     const audio = document.querySelector('#audio');
9 |     const stream = await navigator.mediaDevices.getUserMedia({
10 |       audio: true,
11 |       video: false,
12 |     });
13 |     audio.srcObject = stream;
14 |     audio.play();
15 |
16 |     setTimeout(() => {
17 |       stream.getTracks().forEach(track => track.stop());
18 |       audio.srcObject = null;
19 |     }, 5000)
20 |   } catch (e) {
21 |     console.error(e);
22 |   }
23 | })()
```

PERMISSIONS

Поскольку микрофон достаточно конфиденциальное устройство, браузер запросит у пользователя разрешение на его использование:



При этом пользователь может и отменить выданное разрешение:



РЕЗУЛЬТАТ

В итоге получили возможность дублировать звук в элементе `audio`. Но нам же нужно скачивать или загружать. Воспользуемся API `MediaRecorder`, которое позволяет манипулировать треками на высоком уровне:

```
if (!window.MediaRecorder) {  
    return;  
}  
}
```

ЗАПИСЬ ЗВУКА

```
1 const audio = document.querySelector('#audio');
2 const stream = await navigator.mediaDevices.getUserMedia({
3     audio: true,
4     video: false,
5 });
6 const recorder = new MediaRecorder(stream);
7 const chunks = [];
8 recorder.addEventListener('start', (evt) => {
9     console.log('recording started');
10});
11 recorder.addEventListener("dataavailable", (evt) => {
12     console.log('data available');
13     chunks.push(evt.data);
14});
15 recorder.addEventListener('stop', (evt) => {
16     console.log('recording stopped');
17     const blob = new Blob(chunks);
18     audio.src = URL.createObjectURL(blob);
19 })
20 recorder.start();
21 setTimeout(() => {
22     recorder.stop();
23     stream.getTracks().forEach(track => track.stop());
24 }, 5000)
```

ЗАПИСЬ ЗВУКА

Поскольку в обработчике `stop` мы получили `Blob`, то это открывает для нас все двери – от загрузки на сервер до локального скачивания.

ДУБЛИРОВАНИЕ ВИДЕО

С видео код повторится один в один, за исключением того, что вместо тега `audio` мы будем использовать тег `video` и в `permission`'ах запрашивать его тоже:

ДУБЛИРОВАНИЕ ВИДЕО

```
1 | <video id="video" controls>
2
3 | 1 (async() => {
4 |   if (navigator.mediaDevices) {
5 |     try {
6 |       const video = document.querySelector('#video');
7 |
8 |       const stream = await navigator.mediaDevices.getUserMedia({
9 |         audio: true,
10 |         video: true,
11 |       });
12 |       video.srcObject = stream;
13 |       video.play();
14 |
15 |       setTimeout(() => {
16 |         stream.getTracks().forEach(track => track.stop());
17 |         video.srcObject = null;
18 |       }, 5000)
19 |     } catch (e) {
20 |       console.error(e);
21 |     }
22 |   })
23 | })()
```

ЗАПИСЬ ВИДЕО

```
1 const video = document.querySelector('#video');
2 const stream = await navigator.mediaDevices.getUserMedia({
3     audio: true,
4     video: true,
5 });
6 const recorder = new MediaRecorder(stream);
7 const chunks = [];
8 recorder.addEventListener('start', (evt) => {
9     console.log('recording started');
10});
11 recorder.addEventListener("dataavailable", (evt) => {
12     console.log('data available');
13     chunks.push(evt.data);
14});
15 recorder.addEventListener('stop', (evt) => {
16     console.log('recording stopped');
17     const blob = new Blob(chunks);
18     video.src = URL.createObjectURL(blob);
19 })
20 recorder.start();
21 setTimeout(() => {
22     recorder.stop();
23     stream.getTracks().forEach(track => track.stop());
24 }, 5000)
```

ПОДДЕРЖКА

Ещё раз обращаем ваше внимание, что данное API работает не везде, поэтому нужно проверять в коде доступность тех или иных объектов. В частности, `MediaRecorder` без дополнительных настроек будет работать в Chrome и Firefox.

Пользователям же остальных браузеров вы должны писать «радостное» сообщение о том, что функция записи аудио-видео вашего мессенджера в их браузере не поддерживается.

ИТОГИ

Сегодня мы с вами рассмотрели достаточно много важных вещей, а именно:

- Geolocation;
- Notification;
- Media.



Задавайте вопросы и напишите отзыв о лекции!

ИЛЬЯ МЕДЖИДОВ

 medzhidov@ragemarket.ru