



Progressive Web Apps

Борис Ребров



ЦРИТО
Центр Развития
ИТ-Образования



Минутка бюрократии



- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях
- ДЗ!



Что такое PWA



- Инициатива, активно продвигаемая Google
<http://bit.ly/pwa-main>
- Использует ряд web-технологий для создания более функциональных приложений
- Сокращает разрыв между web и «нативными» приложениями



Преимущества PWA



- **Надежность**

Приложения доступны в любое время, даже когда девайс не подключен к сети

- **Скорость**

Страницы приложения открываются максимально быстро, в том числе, за счет кеширования контента

- **Вовлечение**

Приложение становится "ближе" к пользователю, за счет установки на home screen, полноэкранного режима работы и push-нотификаций

- **Ограниченнная поддержка**

Зачем нужны PWA



- Время, проведенное пользователями мобильных девайсов делится на 87% и 13% для приложений и web.
- MAU пятиста наиболее популярных сайтов более чем вдвое превышает аналогичный показатель для пятиста наиболее популярных приложений.
- Количество установок новых приложений

0	1	2	3	4+
51 %	13 %	11 %	8 %	17 %

- ~80% времени пользователи проводят в трех "любимых" приложениях, до 50% – в одном из них.

Зачем нужны PWA



- Доступность
- Большой охват
- Кроссплатформенность
- Консистентный пользовательский опыт
- Работа оффлайн*
- Функциональность
- Установка на девайс*
- Дистрибуция через магазин*

PWA формальные признаки



- Приложение работает по HTTPS
- Страницы приложения содержат тег `<link />` ссылающийся на корректный Web App Manifest
- Зарегистрированы Service Workers, контролирующие страницы приложения
- <http://bit.ly/pwa-checklist>
- <http://bit.ly/lighthouse-util>

Пример



<http://bit.ly/my-pwa-example>



Web App Manifest



- JSON с метаданными приложения

```
1. {  
2.   "short_name": "Maps",  
3.   "name": "Google Maps",  
4.   "icons": [  
5.     {  
6.       "src": "/images/icons-192.png",  
7.       "type": "image/png",  
8.       "sizes": "192x192"  
9.     },  
10.    {  
11.      "src": "/images/icons-512.png",  
12.      "type": "image/png",  
13.      "sizes": "512x512"  
14.    }  
15.  ],  
16.  "start_url": "/maps/?source=pwa",  
17.  "background_color": "#3367D6",  
18.  "display": "standalone",  
19.  "scope": "/maps/",  
20.  "theme_color": "#3367D6"  
21. }  
22.
```

```
1. <link rel="manifest" href="/manifest.json">
```

<http://bit.ly/web-app-manifest>

Service Worker



- Кеширование контента
- Push-уведомления
- Фоновая синхронизация

```
1. let register = navigator.serviceWorker.register(scriptURL, {scope: '/app'});
2. navigator.serviceWorker.getRegistration('/app').then(registration => {
3.   registration.unregister();
4. }
```

- Все API асинхронные
- Один воркер для одного scope
- Scope не может быть выше чем расположения воркера
- Отладка через <chrome://inspect#service-workers>
- Не забывайте убивать старые воркеры на localhost ;)

Service Worker – жизненный цикл



- Запрос на регистрацию (js)
- Загрузка ([https](https://))
- Установка (событие install)
- Активация (событие active)
- Ожидание (обработка событий)
- Периодическая загрузка обновлений ([https](https://))
- Активация новой версии (событие active)
- Ожидание (обработка событий)
- ExtendableEvent.waitUntil([promise](#)) в событиях install и active

Service Worker – кеширование контента



- CacheStorage (не только в Service Worker)
 - global.caches
 - Хранит именованные наборы кешей (Cache)
 - Хранит записи до их явного удаления*

```
1. self.addEventListener('install', (event) => event.waitUntil(  
2.   caches.open(cacheName).then((cache) => cache.addAll(  
3.     [  
4.       '/css/bootstrap.css',  
5.       '/css/main.css',  
6.       '/js/bootstrap.min.js',  
7.       '/js/jquery.min.js',  
8.       '/offline.html'  
9.     ]  
10. ))));  
11.
```

*Браузер может удалять кеш при достижении квоты

Service Worker – кеширование контента



- ПОИСК В КОНКРЕТНОМ КЕШЕ

```
1. self.addEventListener('fetch', (event) => event.respondWith(  
2.   caches.open('static')  
3.     .then((cache) => cache.match(event.request))  
4.     .then((response) => response || fetch(event.request))  
5.     .then((response) => {  
6.       cache.put(event.request, response.clone());  
7.       return response;  
8.     }));  
9.
```

- ПОИСК В ЛЮБОМ ИЗ КЕШЕЙ

```
1. self.addEventListener('fetch', (event) => event.respondWith(  
2.   caches.match(event.request)  
3.     .then((response) => response || fetch(event.request))  
4.     .then((response) => {  
5.       ...  
6.     }));
```

Стратегии кеширования



- cache first + fallback

```
1. self.addEventListener('fetch', (event) => event.respondWith(  
2.   caches.match(event.request)  
3.     .then((response) => response || fetch(event.request))  
4.     .catch(() => caches.match('/offline.html'))  
5. );
```

- network first

```
1. self.addEventListener('fetch', (event) => event.respondWith(  
2.   fetch(event.request).catch(() => caches.match(event.request))  
3. ));
```

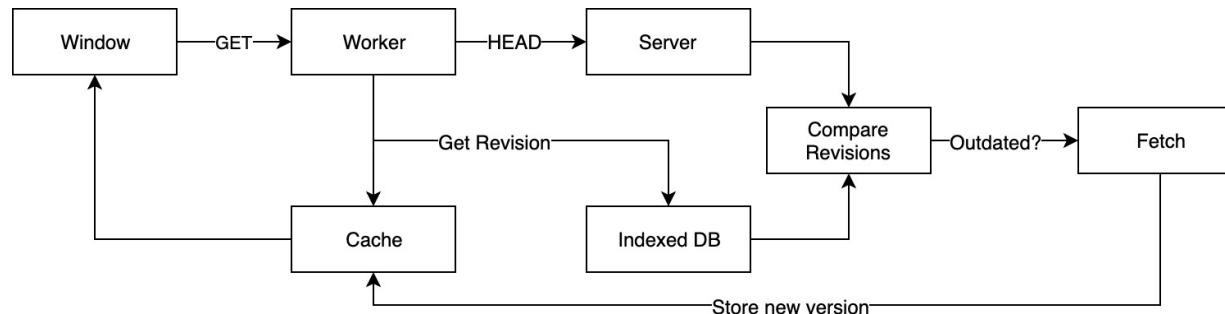
fetch



- Заголовки кеширования HTTP не учитываются
- Событие `activate` подходит для удаления кеша старой версии

```
1. self.addEventListener('activate', (event) => event.waitUntil(  
2.   caches.keys().then((cacheNames) => Promise.all(  
3.     cacheNames.map((cacheName) => caches.delete(cacheName))  
4.   )  
5. );
```

- Версионирование кэшей



Workbox

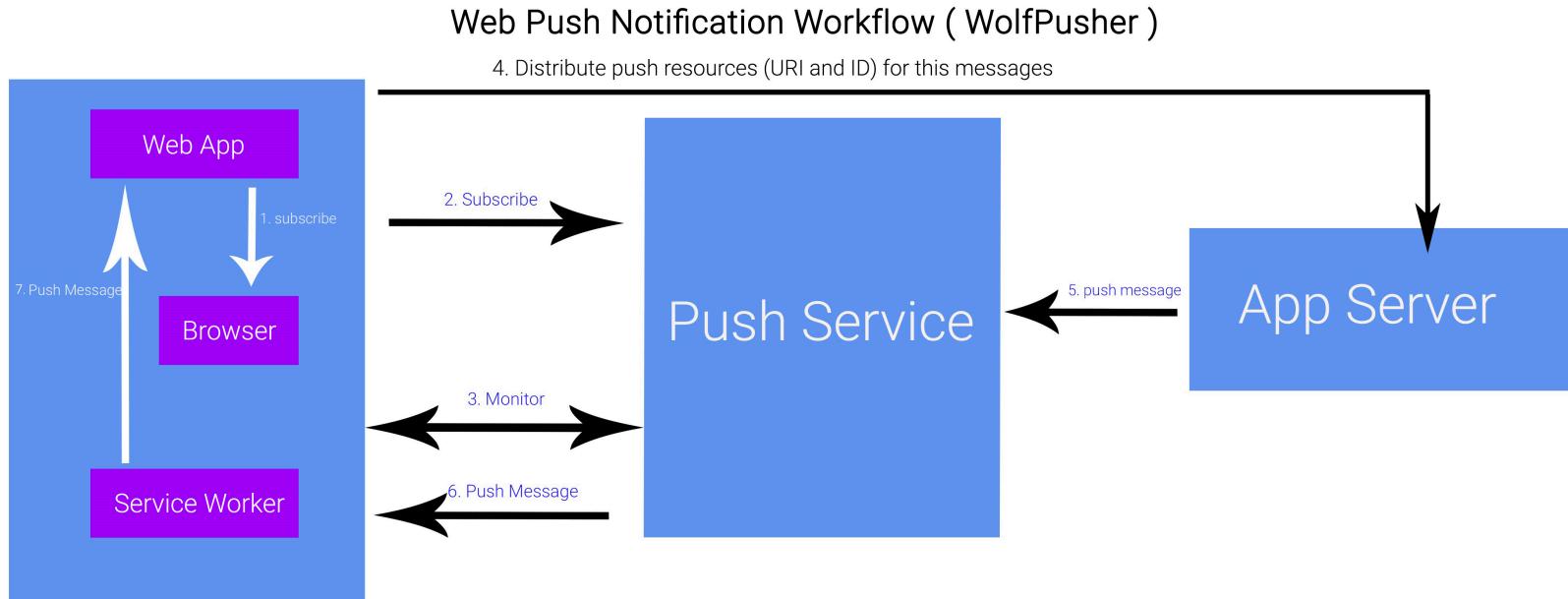


- Кеширование, инвалидация, обработка роутинга
- workbox-webpack-plugin

```
1. const {GenerateSW} = require('workbox-webpack-plugin');
2. module.exports = {
3.   plugins: [
4.     // Other plugins...
5.     new GenerateSW(options)
6.   ]
7. };

1. const {InjectManifest} = require('workbox-webpack-plugin');
2. module.exports = {
3.   plugins: [
4.     // Other plugins...
5.     new InjectManifest(options)
6.   ]
7. };
8.
```

Push Notifications



Push Notifications – подписка



- Подписка уникальна для воркера

```
1. swRegistration.pushManager.subscribe({  
2.   userVisibleOnly: true  
3. }).then(subscription => { ... }).catch(error => { ... });
```

- Параметры подписки могут устаревать и требуют периодического обновления

```
1. swRegistration.pushManager.getSubscription()  
2.   .then(subscription => sendToServer(subscription));
```

- В объекте подписки содержится поле endpoint (часть из которого – push token)
- Запросы к push-серверу содержат токены целевых пользователей
- У разных вендоров разные push-серверы (сюрприз)

Push Notifications – обработка пуша



- Поступление сообщения порождает событие «push» в воркере
- Сообщение может содержать payload или быть триггером для его получения с сервера

```
1. self.addEventListener('push', event => {
2.   event.waitUntil(
3.     self.registration.showNotification('Title', options)
4.   );
5. });
6.
7. self.addEventListener('notificationclick', (event) => {
8.   const payload = event.notification;
9.   const url = (payload.data || {}).url || '/';
10.  event.notification.close();
11.  event.waitUntil(clients.openWindow(url));
12.});
```

Firebase Cloud Messaging



- ApplicationKey + Sender ID, необходимые для подписки и рассылки уведомлений в Chrome

```
1. {
2.   "gcm_sender_id": "103953800507"
3. }
```

- JS SDK

```
1. const messaging = firebase.messaging();
2. messaging.requestPermissions()
3.   .then(() => registerSw('/url-to-sw.js'))
4.   .then(registration => messaging.useServiceWorker(registration))
5.   .then(() => messaging.getToken())
6.   .then((token) => { ... });
```

Push Notifications – отправка



- SERVER_KEY нужен только для CHROME
1. curl "ENDPOINT_URL" --request POST --header "TTL: 60" --header "Content-Length: 0" --header "Authorization: key=SERVER_KEY"
 - Отправка payload
 1. curl -X POST -H "Authorization: key=SERVER_KEY" -H "Content-Type: application/json" -d '{"notification": {"title": "Test"}, "to": :token}' "https://fcm.googleapis.com/fcm/send";

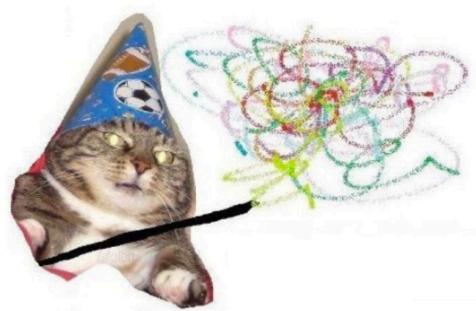
Домашнее задание № 3



- Реализовать в вашем приложении подписку на Push-уведомления и их отображение
- Реализовать кеширование по стратегии cache-first с использованием webpack-workbox-plugin

Срок сдачи

- до 14 марта





Спасибо за внимание!