The Technology Behind Progressive Web Apps
Service Workers in Detail

Erik Witt
ew@baqend.com
24.01.2018
@baqendcom
What we are going to cover.

**PWA**
- Core Features
- Building Blocks
- Implementation

**Service Worker**
- Lifecycle
- Network Interception
- Caching Strategy etc.

**Speed Kit**
- Cache Coherence
- Performance Measures
Why do(n’t) we love native apps?

Progressive Web Apps

seek to combine the great from native and web apps
What are Progressive Web Apps?
Progressive Web Apps (PWAs)

- Fast Loads through Caching
- Offline Mode (Synchronization)
- Add-to-Homescreen and Push Notifications
Try this:

www.baqend.com
Building Blocks of PWAs

1. Manifest

- PWAs are best practices and open web standards

2. Service Worker

- Progressively enhance when supported
Implementing PWAs

- PWAs are best practices and open web standards
- Progressively enhance when supported

1. **Manifest** declares Add-to-Homescreen:

```xml
<link rel="manifest" href="/manifest.json">
{
  "short_name": "Codetalks PWA",
  "icons": [
    {"src": "icon-1x.png", "type": "image/png", "sizes": "48x48"}
  ],
  "start_url": "index.html?launcher=true"
}
```
Implementing PWAs

- PWAs are best practices and open web standards
- Gracefully degrade when not supported

2. Service Workers for caching & offline mode:
Implementing PWAs

- PWAs are **best practices** and **open web standards**
- Progressively enhance the user experience

3. Add **Web Push** and **Background Sync**:

![Diagram showing Web Push and Background Sync](image)
Typical Architecture: **App Shell Model**

**App Shell:** HTML, JS, CSS, images with app logic & layout

**Content:** Fetched on demand & may change more often
What is the future of Progressive Web Apps?
The Future of **PWAs** is bright.

**Payment Request API**

- **Goal:** replace traditional **checkout** forms
- Just ~10 LOC to implement payment
- **Vendor- & Browser-Agnostic**
The Future of **PWAs** is bright.

---

**Credentials Management API**

1. Click *Sign-in* → Native Account Chooser

2. Credentials API **stores** information for future use

3. **Automatic** Sign-in afterwards
The Future of **PWAs** is bright.

**Web Speech API**
Native Speech Recognition in the Browser:

```javascript
annyang.addCommands({
  'Hello Code.talks': () => {
    console.log('Hello you.');
  }
});
```
The Future of **PWAs** is bright.

**Web Share API**

- **Share** site through native share sheet UI
- Service Worker can register as a **Share Target**
What are Service Workers?
What are **Service Workers**?

Programmable **Network Proxy**, running as a separate **Background Process**, without any **DOM Access**.
What do Service Workers do?

- **Cache** Data (CacheStorage)
- **Store** Data (IndexedDB)
- Receive **Push**
- Respond when **Offline**
What do Service Workers do?

- Intercept HTTP Requests
- Sync Data in Background
- Hide Flaky Connectivity from the User
Browser Support for **Service Workers**

Supported by **75%** of browsers.
Requires **TLS Encryption**.
Browser Support for Service Workers

**WebKit**

- **Service Workers**
  A method for browsers to run JavaScript in the background to handle network requests and manage cached responses. Service Workers offers a replacement for Application Cache.

  - **Reference**: w3c.github.io...
  - **Contact**: @bradeeoh - Brady Eidson

  **Safari**: In Development
  **Edge**: Implemented, but Toggled
How are **Service Workers** registered?

```javascript
navigator.serviceWorker.register('/sw.js');
```
How does the **Lifecycle** look like?

```javascript
self.addEventListener('install', (event) => {
  // Perform install steps
});

self.addEventListener('activate', (event) => {
  // Perform activate steps
});

self.addEventListener('fetch', (event) => {
  // React to fetch event
});
```

[Diagram showing the lifecycle states: Installing, Activated, Idle, Terminated, and Error with transitions between them.]

- **Installing**
- **Activated**
- **Idle**
- **Terminated**
- **Fetch**
- **Error**
How to **Communicate** with Service Workers?

---

**Client**

```javascript
const client = await clients.get('id');
client.postMessage(someJsonData);
```

**Server**

```javascript
self.addEventListener('push', (event) => {
  // Receive push notification
});
```

---

**Browser Tab**

```javascript
self.addEventListener('message', (event) => {
  // Receive message push
});
```

---

**Diagram**

The diagram illustrates the communication flow between a service worker and a browser tab. The service worker can send messages to the browser tab, which can then receive and respond to these messages. The diagram also shows how the service worker can receive push notifications from a (Web) Push Service.
Intercepting Network Requests

```javascript
self.addEventListener('fetch', (event) => {
  // React to fetch event
  const { url } = event.request;
  event.respondWith(async () => {
    const request = new Request(url.replace('.com', '.de'))
    const response = await fetch(request);
    const text = await response.text();
    const newText = text.replace('Goethe', 'Schiller');
    return new Response(newText, { status: 200 });
  })());
});
```

There is so much you can do:
- **Rewrite** Request
- **Change** Response
- **Concat** Responses
- **Cache** Responses
- **Serve** Cached Data
- ...
Service Worker Scope

Scope determines which requests go to Service Worker

// Default (and maximum) scope is location of Service Worker
// Gets all requests starting with '/path/'
navigator.serviceWorker.register('/path/sw.js');
// Scope option can further limit which requests got to Service Worker
// Gets all requests starting with '/path/subpath/'
navigator.serviceWorker.register('/path/sw.js', { scope: '/path/subpath/' });

// Widening the scope is NOT ALLOWED
// unless explicitly allowed by HTTP header 'Service-Worker-Allowed: /
// Gets all requests from the domain
navigator.serviceWorker.register('/path/sw.js', { scope: '/' });
Service Worker Persistence

IndexedDB
an actual database in the browser

- Stores Data **Persistently**
- Stores **Structured** Data
- Supports **Range Queries**
- Browser Support 94%
Service Worker Background Sync

One-off Sync

```javascript
// Register a sync at the Service Worker registration.
registration.sync.register('keyword')

// React to sync event in Service Worker
self.addEventListener('sync', (event) => {
  if (event.tag == 'keyword') {
    event.waitUntil(...);
  }
});
```

One-off Sync is
- executed when user is online
- retried when failed (exponential backoff)

Example Use Cases
- Save file when online again
- Send email when online again

Experimental
Service Worker Background Sync

Periodic Sync

// Registers a periodic sync
registration.periodicSync.register(options)

// Execute periodic sync in Service Worker
self.addEventListener('periodicsync', (event) => {
  if (event.registration.tag === 'keyword') {
    event.waitUntil(...);
  }
});

Periodic Sync is
• executed when online, according to period options
• retried when failed

Example Use Case
• Load updates to social media timeline when browser closed

Experimental
Service Worker Debugging
Service Worker Caching

Cache Storage
Stores Request/Response pairs

// Putting Request/Response pair in cache
const cache = await caches.open('name');
cache.put(request, response);

// Retrieving Response from cache
const response = await caches.match(request);
return response || fetch(request);

Cache Storage
- Programmatically managed
- Persistent and non-expiring
- Supports only HTTP
- Only caches GET requests (no HEAD)
How Resources get into the Cache

// Cache resource when fetched
self.addEventListener('fetch', (event) => {
  event.respondWith((async () => {
    const cache = await caches.open('name');
    const response = await fetch(request);
    await cache.put(request, response.clone());
    return response;
  })());
});

// Cache resources as install dependency
const urlsToCache = ['index.html', 'style.css', 'app.js'];

self.addEventListener('install', (event) => {
  event.waitUntil((async () => {
    const cache = await caches.open('name');
    await cache.addAll(urlsToCache);
  })());
});
Caching Strategies – Cache Only

Cache only strategy gets all requests from cache or fails.

- Fast responses (or none)
- Only for pre-cached requests
- Only for static resources
- Needs asset hashing or versioning of cache

```javascript
// Serve cached Response
self.addEventListener('fetch', (event) => {
    event.respondWith(
        caches.match(event.request)
    );
});
```
Caching Strategies – Cache, Network Fallback

This strategy gets request from cache and from network as fallback.

- Fast responses for cached resources, slow for others
- Only for static resources cacheable
- Needs asset hashing or versioning of cache

```javascript
// Serve cached Response or fallback to network
self.addEventListener('fetch', (event) => {
  event.respondWith(async () => {
    const response = await caches.match(request);
    return response || fetch(request);
  });
});
```
Caching Strategies – Network Only

This strategy gets request from network only.

- Slow responses
- Always up-to-date

```javascript
// Serve everything from network
self.addEventListener('fetch', (event) => {
    event.respondWith(async () => {
        return fetch(request)
    });
});
```
Caching Strategies – **Network, Cache Fallback**

This strategy gets request from network and from cache as fallback.

- Slow responses from network
- Effectively offline mode (better use `navigator.onLine`)
Caching Strategies – Cache, then Network

This strategy gets requests from cache first and from network in background.

- Fast initial response
- Uses a lot of bandwidth
- Used for static app shell or message inboxes

// Idea:
// Serve resource from cache
// In the background:
// * fetch resource from network
// * send new response to browser
// * apply changes to DOM or ask user to reload the page
Major Challenge: Cache Coherence

All caching strategies either serve outdated data or degrade performance.
What we do with Service Workers
Speed Kit
Turning Websites into Instantly-Loading Progressive Web Apps
What **Speed Kit** does.

**Faster**

**More Scalable**
What **Speed Kit** does.
What Speed Kit does.

Backed by 30 Man-Years of Research


F. Gessert, N. Ritter „Scalable Data Management: NoSQL Data Stores in Research and Practice“, 32nd IEEE International Conference on Data Engineering, ICDE, 2016.

How Speed Kit solves Cache Coherence

Has Time-to-Live (expiration)

False-Positive Rate: \( f \approx \left( 1 - \frac{\ln n}{m} \right)^k \)

Hash-Functions: \( k = \left\lfloor \ln(2) \cdot \frac{n}{m} \right\rfloor \)

With 20,000 entries and a 5% false positive rate: **11 Kbyte**

**Consistency**: \( \Delta \)-Atomicity, Read-Your-Writes, Monotonic Reads, Monotonic Writes, Causal Consistency

How Speed Kit solves Cache Coherence
Adding **Speed Kit** to a Site
1. Configure Domain

Set which URLs Baqend should accelerate.
2. Include Code Snippet

Add the Speed Kit Service Worker to the website.
3. Requests Accelerated

Speed Kit routes the requests through Baqend’s caches.
How it works under the hood

Website with Snippet → Speed Kit Service Worker

Requests → Fast Requests

Baqend Service

Pull → Push

3rd Party Services → other

Existing Backend
Now, we have a PWA, HTTP/2, CDNs, etc. How do we measure web performance?
# Page Speed Analyzer

[Go to Page](https://www.meetup.com/de-DE/Hamburg-Web-Performance-Group/events/245)

<table>
<thead>
<tr>
<th>Domains</th>
<th>Requests</th>
<th>Response Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>30</td>
<td>3.23 MB</td>
</tr>
</tbody>
</table>

## Your Website

<table>
<thead>
<tr>
<th>Speed Metric</th>
<th>Current Time (ms)</th>
<th>Improved Time (ms)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Index</td>
<td>1476</td>
<td>581</td>
<td>2.54x Faster</td>
</tr>
<tr>
<td>1st Meaningful Paint</td>
<td>1406</td>
<td>602</td>
<td>2.34x Faster</td>
</tr>
<tr>
<td>Time To First Byte</td>
<td>960</td>
<td>18</td>
<td>53.33x Faster</td>
</tr>
<tr>
<td>DOMContentLoaded</td>
<td>1610</td>
<td>492</td>
<td>3.27x Faster</td>
</tr>
<tr>
<td>FullyLoaded</td>
<td>3158</td>
<td>3975</td>
<td>0.79x</td>
</tr>
<tr>
<td>Last Visual Change</td>
<td>2.3s</td>
<td>1.3s</td>
<td>1.8x Faster</td>
</tr>
</tbody>
</table>

## Your Website with Speed Kit

<table>
<thead>
<tr>
<th>Speed Metric</th>
<th>Current Time (ms)</th>
<th>Improved Time (ms)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Index</td>
<td>1476</td>
<td>581</td>
<td>2.54x Faster</td>
</tr>
<tr>
<td>1st Meaningful Paint</td>
<td>1406</td>
<td>602</td>
<td>2.34x Faster</td>
</tr>
<tr>
<td>Time To First Byte</td>
<td>960</td>
<td>18</td>
<td>53.33x Faster</td>
</tr>
<tr>
<td>DOMContentLoaded</td>
<td>1610</td>
<td>492</td>
<td>3.27x Faster</td>
</tr>
<tr>
<td>FullyLoaded</td>
<td>3158</td>
<td>3975</td>
<td>0.79x</td>
</tr>
<tr>
<td>Last Visual Change</td>
<td>2.3s</td>
<td>1.3s</td>
<td>1.8x Faster</td>
</tr>
</tbody>
</table>
Measuring Web Performance

- **TTFB**
- **DOMContent Loaded**
- **Last Visual Change**
- **Fully Loaded**

<table>
<thead>
<tr>
<th>Speed</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.54x Faster</td>
<td>1470ms</td>
</tr>
<tr>
<td>2.34x Faster</td>
<td>1406ms</td>
</tr>
<tr>
<td>53.33x Faster</td>
<td>960ms</td>
</tr>
<tr>
<td>3.27x Faster</td>
<td>1610ms</td>
</tr>
<tr>
<td>0.79x Faster</td>
<td>1580ms</td>
</tr>
<tr>
<td>1.8x Faster</td>
<td>2.3s</td>
</tr>
</tbody>
</table>

- No visual changes have occurred.
Measuring Web Performance

How can we measure user-perceived performance?

Even asynchronous JS has completed
The Speed Index

- Visual Completeness (VC)

\[ \int_{0}^{\infty} \]
The First Meaningful Paint

VC
Visual Completeness

Moment of biggest layout change

Time

[Graph showing the concept of First Meaningful Paint with time on the x-axis and VC (Visual Completeness) on the y-axis.]
Does it work for **Your Site**?

www.example.com

Want to double your free tier?

Send a mail with WPMEETUP to support@baqend.com

**test.speed-Kit.com**
Wrap Up

PWA
Super cool alternative to native apps

Service Worker
Powerful programmable network proxy

Speed Kit
Combines Service Worker and cache coherence
Good Resources

- https://developers.google.com/web/fundamentals/performance/?hl=en
- https://hpbn.co/

Performance Tools

- https://developers.google.com/speed/pagespeed/
- https://test.speed-kit.com
- https://www.baqend.com/
- http://www.webpagetest.org/
We are hiring.

- Frontend Developers
- Mobile Developers
- Java Developers
- Web Performance Engineers

Contact us.

Erik Witt · ew@baqend.com · www.baqend.com/jobs.html