What we are going to cover.

- PWAs
  - Core Features
  - Building Blocks
  - Implementation

- Service Workers
  - Lifecycle
  - Network Interception
  - Caching Strategies

- Use Case
  - Service Workers in Production at Baqend
Why do(n’t) we love native apps?

Progressive Web Apps
Combine the best 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
## Advantages of PWAs

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discoverable</td>
<td>E.g. in search engines</td>
</tr>
<tr>
<td>Installable</td>
<td>Easy access from home screen</td>
</tr>
<tr>
<td>Linkable</td>
<td>Link into apps through URLs</td>
</tr>
<tr>
<td>Network independant</td>
<td>Offline mode</td>
</tr>
<tr>
<td>Progressive</td>
<td>Enhance on capable browsers</td>
</tr>
<tr>
<td>Re-engageable</td>
<td>Engage through Web Push</td>
</tr>
<tr>
<td>Responsive</td>
<td>Fit any form factor</td>
</tr>
<tr>
<td>Safe</td>
<td>HTTPS &amp; recognizable URLs</td>
</tr>
</tbody>
</table>
These apps aren’t packaged and deployed through stores, they’re just websites that took all the right vitamins.

Alex Russell, Google
Building Blocks of PWAs

PWAs are best practices and open web standards

Progressively enhance when supported

1. Manifest
2. Service Worker
Implementing **PWAs**

PWAs are **best practices** and **open web standards**

**Progressively enhance** when supported

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

```html
<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:

![Diagram showing the flow of a PWA with Web App, Website, SW.js, Cache, and Network.]
Implementing PWAs

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

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

![Diagram showing the flow of a website, SW.js, Sync, web app, and network with Push messages and Sync actions]
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 and vision of Progressive Web Apps?
Integrate payment.

Web Payment APIs

• Goal: replace traditional checkout forms
• Just ~10 LOC to implement payment
• Vendor- & Browser-Agnostic
Manage users and logins.

Credentials Management API

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

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

3. **Automatic** Sign-in afterwards
Leverage **geolocation**.

**Geofencing**

- **Notify** web app when user leaves or enters a defined area
- Requires **permission**
Build conversational interfaces.

Web Speech API

Native Speech Recognition in the Browser:

```javascript
annyang.addCommands({
  'Hello Code.talks': () => {
    console.log('Hello you.');
  }
});
```
Seemless sharing between apps.

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 **>85%** of browsers. Requires **TLS Encryption**.
Late, but all in: Microsoft

Publish PWAs to Microsoft Store

or

Bing Crawls PWAs → Convert to AppX → Microsoft Store

https://blogs.windows.com/msedgedev/2018/02/06/welcoming-progressive-web-apps-edge-windows-10/#tqIAYGJrOUcxvCWg.97
How are **Service Workers** registered?

```javascript
navigator.serviceWorker.register('/sw.js');
```
What 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
});
```
How to communicate with Service Workers?

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

// Send message to browser tab
const client = await clients.get('id');
client.postMessage(someJsonData);

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

const client = await clients.get('id');
client.postMessage(someJsonData);
```
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 Requests
- Change Responses
- Concat Responses
- Cache Responses
- Serve Cached Data
- …
Service Worker Scope

Scope determines which requests go to the Service Worker

// Default (and maximum) scope is location of Service Worker
// Gets all requests starting with '/path/
navigator.serviceWorker.register('/path/sw.js');
Service Worker Scope

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

Scope can be restricted but not widened
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
- executed when user is **online**
- **retried** when failed (exponential backoff)

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

Periodic Sync
- executed when online, according to **period options**

Use Cases
- Load updates to **social media timeline** when browser closed

Experimental
Service Worker Debugging
Cache Storage
Stores Request/Response pairs

- Supports only **HTTP**
- Only caches **GET** requests (no HEAD)

**Cache Storage**
- **Programmatically** managed
- **Persistent** and non-expiring
Caching Strategies – **Cache Only**

Gets all requests from cache or fails.
Caching Strategies – Cache, Network Fallback

 Gets requests from cache & uses network as fallback.
Caching Strategies – Network Only

Gets requests from network only.
Caching Strategies – **Network, Cache Fallback**

Gets requests from network, the cache acts as fallback (offline mode).
Caching Strategies – Cache, then Network

Gets requests from cache first and from network in background.
Major Challenge: **Cache Coherence**

All strategies either serve **outdated data** or degrade performance.
How we use Service Workers at Baqend
Problem: slow backends & networks.
Solution: Speed Kit

Service Worker rewrites & accelerates slow requests.

1. Fast
2. Less Processing
The magic: dynamic data is kept up-to-date.
Backed by 30 man-years of research.

- 7 years of research & development at the University of Hamburg
- 4 PhDs, >30 student theses, >25 research publications

Learn more.
How Speed Kit leverages **Service Workers**.

- Website with Snippet
- Speed Kit Service Worker
- Baqend Service
- 3rd Party Services
- Fast Requests
- Realtime
- Scheduled
- Existing Backend

Requests → Fast Requests → Realtime, Scheduled, Other
Use case I: optimize images.
SW sends client resolution → responsive image.

Images transcoded to WebP
Rescaled to match Screen Size
JPG and PNG Recompression
Use case II: re-route 3rd party dependencies.
Service Workers can manipulate other domains.
Use case III: handling cache coherence.

False-Positive Rate:
\[ f \approx \left( 1 - e^{-\frac{kn}{m}} \right)^k \]

Hash-Functions:
\[ k = \left\lceil \ln(2) \cdot \left( \frac{n}{m} \right) \right\rceil \]

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

**Consistency:** Δ-Atomicity, Read-Your-Writes, Monotonic Reads, Monotonic Writes, Causal Consistency

HashA(url)

poure(url)

hashB(url)

Flat(Counting Bloomfilter)

0 1 1 1 1

0 3 1 4 1
Use case IV: simple web push.
Demo:
Looking into Service Workers
Now, we have a Progressive Web App. How do we measure its performance?
A PWA can make a huge difference.
Measuring **PWA performance**.

User-perceived performance.

Speed Index  
avg. time to visibility

First Meaningful Paint  
greatest visible change

\[ \int_0^\infty 1 - VC(t) dt \]
Test your site.

www.example.com

Go

test.speed-kit.com
Wrap Up.

**PWAs**
Super cool alternative to native apps

**Service Workers**
Powerful programmable network proxy

**Use Case**
Speed Kit: Smart CDN though Service Workers
Learn more about this topic:

https://blog.baqend.com/

Rethinking Web Performance with Service Workers
30 Man-Years of Research in a 30-Minute Read

This article surveys the current state of the art in page speed optimization. It contains the gist of more than 30 man-years of research that went into Speed Kit, an easy-to-use web performance plugin to accelerate any website.
Learn more about Services Workers.

Recommended Books

- Building Progressive Web Apps
- Progressive Web Apps
- Beginning Progressive Web App Development

Blogs

- Baqend Blog: On Building a Faster Web
  - https://blog.baqend.com/
- Ilya Grigorik: Internet plumber
  - https://www.igvita.com/
- Jake Archibald: I discovered a browser bug
  - https://jakearchibald.com/

Guides & Tutorials

- Progressive Web Apps
  - A new way to deliver amazing user experiences on the web.
  - https://developers.google.com/web/progressive-web-apps/
- Progressive web apps
Catch our other talks!

14:00 - Kino 7 - Buzzing Technologies
Creating High-Performance Web Apps with WebAssembly

15:00 - Kino 6 - Architecture
Real-Time Processing Explained: A Survey of Storm, Samza, Spark & Flink