Pázmány Péter Katolikus Egyetem

Információs Technológiai és Bionikai Kar

GUIDED INDIVIDUAL STUDY

Progressive Web Applications

Ghiurutan-Bura Péter

Computer Science Engineering BSc

2022

Supervisor: Naszlady Márton Bese

Contents

1 Abstract **1**

2 Introduction **2**

3 Progressive Web Applications **3**

3.1 Service workers 3

3.2 Caching strategies 4

3.3 Workbox 5

3.4 Push notifications 7

4 Summary **9**

Bibliography **11**

A Appendix **12**

Chapter 1

Abstract

Over the last few decades web applications have seen an incredible rise in popularity hand in hand with tremendous technological advances.

Generally, users hate long delays and unreliability when accessing an application. Moreover, instantly losing access to the website's content when falling offline can become cumbersome. Progressive web applications were created to provide a solution to these problems. Progressive web apps are web apps that use modern web technologies to create application experiences that don't show many of the classic limitations of the web. A progressive web application takes advantage of the latest technologies to combine the best of the two worlds: web and mobile apps. This allows Progressive Web Apps (PWA) to combine the reach (broad availability and access) of website experiences with the capability (hardware features and rich resources) of platform-specific experiences.

Throughout the guided invidiual study I was focusing on grasping both new theoretical concepts as well as implementing a PWA simultaneously. One of the first steps was to examine the core idea of progressive web apps, how service workers take the responsibility of a "middleware" between the client and the server and handle the requests.

Essentially, a service worker is able to cache resources and solve predictive prefetching which enhances the reliability of an app. By analyzing the different caching approaches I was able to implement them in a React based project.

Since user interactions are a key aspect in a web app, I shifted my attention towards the addition of push notifications and installability on both mobile and regular devices.

All in all, throughout the guided individual study I gained several technical knowledge about progressive web applications, meanwhile this period of time made room for the transformation of a starter basic application into a progressive web app that runs reliable in both online and offline environments.

Chapter 2

Introduction

Progressive web applications are web apps that are developed with certain web technologies and make use of some standard patterns. These patterns result in PWA's having the best features of both web and native applications. [1]

The web side brings out the concept of discoverability : in general it's way faster to visit and share a website than installing and loading a native app.

On the other hand, native apps are better integrated with the operating system and therefore offer a more seamless experience for the users. This opens the opportunity for offline fallbacks and push notifications. [1]

A web application, in order to be perceived as a PWA has to meet some requirements: [2]

- Discoverable: the contents are ought to be found within search engines
- Installable: should fulfill the installability requirements to become available on devices. As a normal native app it has an icon on the home screen, can be opened in a standalone window and it works offline. [3]
- Linkable: a user can share the app with a URL
- Network independent: handles well offline and poor connections
- Progressively enhanced: which means that the app works on a basic level in older browsers too, but reaches the top capabilities on the state-of-the-art browsers.
- Re-engageable: allows space for the usage of push notifications
- Responsively designed: the app should look optimal on every device width, with varying screen sizes. In my case this problem was tackled with Tailwind CSS, a design framework.
- Secure: every connection and request should be secured against malicious behaviour

Chapter 3

Progressive Web Applications

3.1. Service workers

In its core, service workers are responsible for most of the benefits introduced with progressive web apps. Service workers can be perceived as a virtual proxy between the browser and the network. [4]

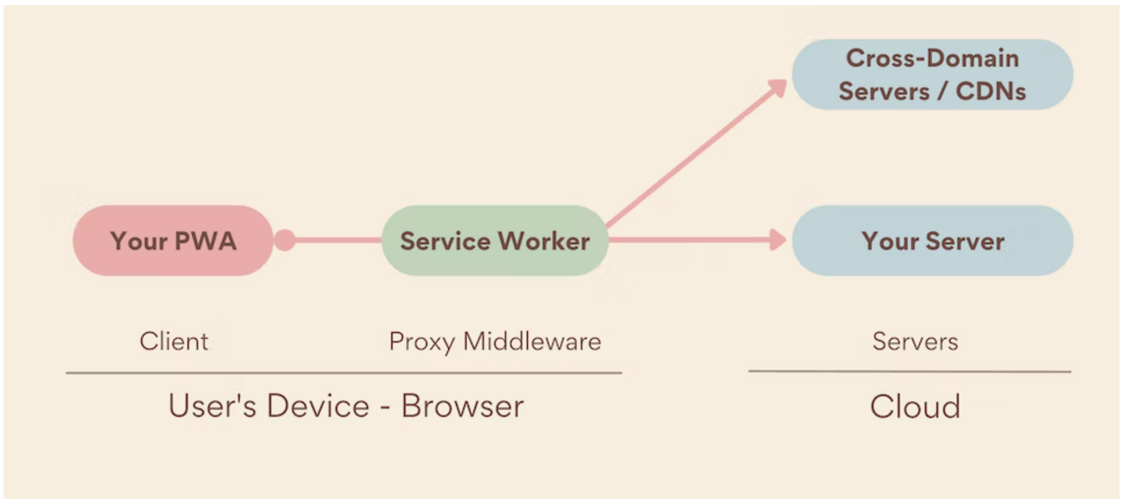

Figure 3.1: Service worker acting as a middleware between client and server. Source: [4]

When an app requests a resource covered by the service worker's scope, the service worker acts a middleware and intercepts the request. Upon defining the strategies previously, the service worker proceeds to handle the request: this could be serving the request from the cache or responding with a network resource as normally. Service workers have the capability to work entirely offline. [4]

On the other hand service workers handle interactivity with notifications, and because they run on a separate thread from the main JavaScript code they can handle heavy calculations as well. [5]

The steps required before registering the service worker, is to check the support of the browser. After passing the requirement, the service worker will register and install. [4]

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register("/serviceworker.js");
}
```

3.2. Caching strategies

Caching means to download, store, delete or update assets on the device with the use of the Cache Storage API. The service worker can then serve requests with cached resources even though there's no network connection. [6]

As previously mentioned, the service worker intercepts the network requests and acts upon it. This proxy behaviour happens in the service worker's "fetch" event. The fetch event contains the request property (url, method, name, destination) and by this information can the service worker decide how to respond. [7]

There are some predefined caching strategies that can serve different purposes depending on the use case: [7]

Cache only

Cache only strategy means that when the service worker intercepts the requests, it will only ever match the requests with resources available in the cache. Upon this logic, for the cache only pattern to work all the assets needed in cache have to be precached.

Network only

This strategy is useful for content which requires to be always up-to-date. It's hindrance is that it will not work in offline mode. All requests passed to the service worker are basically passed forward to the network without any "interception".

Cache first, falling back to network

This approach combines the previous two in a logical manner:

- The service worker intercepts the request: if the wanted resource is in the cache, serve it from there
- If the cache didn't contain the request, forward the request to the network
- After the network request is done, cache the response for the future and return the network response.

Network first, falling back to cache

Reordering the steps of the previous strategy results in network first, falling back to cache. This strategy can be useful in situations where we need the newest data, but the offline fallback behaviour should show an earlier version as well.

- Ask the network for the request first, then cache the possible response
- If the application goes offline, the cached asset can act as a fallback point

Stale-while-revalidate

Stale-while-revalidate is the most complex strategy. It also combines the first two, with eyes on prioritizing speed of access and somewhat data integrity. It can be used in cases where up-to-dateness is important, but not crucial.

- On the very first request, serve the response from the network and place the asset in the cache.
- On subsequent requests instantly serve the request from the cache, meanwhile re-requesting it from the network and if the new response and the response in the cache differ, update the cache.
- Moving forward, every request will be served from the cache with the network request that was previously fetched.

3.3. Workbox

Service workers solve hard problems and offer great utility in a PWA: cache management, strategies, prefetching. However, the actual implementation can get complex sometimes. [8]

Libraries built on abstractions make complex APIs simpler to implement. Workbox is a set of modules that simplify common service worker routing and caching. Each module available addresses a specific aspect of service worker development. Workbox aims to make using service workers as easy as possible, while allowing the flexibility to accommodate complex application requirements where needed. [9]

Good abstractions make complex APIs easier to use. That's where Workbox comes in. Workbox is a set of modules that simplify common service worker routing and caching. Each module available addresses a specific aspect of service worker development: [9]

- workbox-routing is used to match incoming requests
- workbox-strategies contain predefined caching strategies
- workbox-precaching can be used to precache assets
- workbox-expiration is for handling caches
- workbox-window is useful for service worker registration and updates

Essentially, Workbox is a tool to make the usage of service workers a lot eaasier, however it makes the implementation fully flexible and works with more complex systems too.

For a codewise implementation throughout the guided individual study, I used React.js frontend library. https://create-react-app.dev/docs/making-a-progressive-web-app/

The create-react-app setup acted as a starting point for my application. The whole idea of the project was to implement and progressively enhance some parts of a website, inspired by the university's Space system.

By using Workbox, implementing caching strategies would look like: [10]

A network first strategy was implemented for testing, using a random API sample.

```
registerRoute(
      ({ url }) => url.href.match("https://jsonplaceholder.
 typicode.com/posts"),

      new NetworkFirst({
            cacheName: "posts-cache",
      })
);
```

We are instructing the service worker that, upon receiving the request that matches with the API's url, proceed with a NetworkFirst strategy. This results in the service worker going to the network, and caching the response for possible offline behaviour.

For use cases, where the actual content changes very rarely, a Cache first, falling to network strategy is optimal. For instance the university's courses are predefined at the start of each semester, hence if we apply an estimated expiration time (when the content could be renewed) - serving it cache first is a workable strategy.

```
registerRoute(
	({ url }) => url.href.startsWith(
 "https://space.itk.ppke.hu/api/subjects"),
	new CacheFirst({
		cacheName: "courses-cache",
		plugins: [
			new ExpirationPlugin({
				maxAgeSeconds: 86400 * 30,
			}),
		],
	})
);
```

3.4. Push notifications

Push notifications serve as a great way to deliver messages and content to the users. Messages, for instance the reminder of a due data can be sent to the devices even when they're not being active. [11]

From an outer layer users, push notifications are part of enhancing interactivity: receiving timely, useful, and precise information.

The push notification feature was implemented in my project as well, the steps needed to be coded are the following: [12]

- On the React frontend, make the UI available to prompt the user for permission to send push notifications.
- Once the user prompted the permission, we can initialize the user's push subscription with the PushManager, Resulting in a PushSubscription object that contains all the required information needed to send push messages to the given user
- Additional client side logic is needed: using the service worker's "push" event we can listen to notificatins received and can display them to the browser
- The server side steps require setting up endpoints, where essentially notifications

are forwarded to the client side with the help of Web Push API

During the individual study, the results regarding push notifications have been achieved on a basic level: clients can be prompted for permissions and they are able to subscribe to PushSubscriptions. Furthermore notifications are intercepted and displayed client side when demo-ed from a browser. The development continued with the build of a basic Express.js server-side API for push notifications, however I encountered errors which require additional handling in order to work. [13]

Chapter 4

Summary

Progressive web applications proved to be an essential addition to web technologies. Recent studies showed as well that page speed has a significant impact on business metrics: optimizing the loading speed and reliability of the website is critical in enhancing user journeys. [14]

With the help of different caching strategies: cache first, network first or stale while revalidate I could optimize my codebase in such a way that it shows a reactive behaviour in every network condition.

While I focused on offline experience as well, there is a lot of room for further advancements. Providing the user with feedback regarding the shown content's age (whether it could be outdated or fetched recently) could serve as a great addition. Furthermore, pages and requests whose content change only once in a while are to be introduced in the service worker cached category.

On the other hand, my application gained only the basic push notification capabilities. Taking real use cases and extending the application in parts where pushing notifications would be benefitial could increase the interactivity of the app: for instance when a university course grade is newly inserted by an official, the user could receive a push notification on mobile from the installed app, or could receive a prompt message on its laptop.

On the whole, the guided individual study confirmed to be a perfect opportunity to research and work with a valuable technology such as progressive web applications.

Bibliography

[1] *Progressive web apps*, en. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction.

[2] *What makes an App a PWA?* en. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#what_makes_an_app_a_pwa.

[3] *Progressive web apps*, en. [Online]. Available: https://web.dev/learn/pwa/progressive-web-apps/.

[4] *Service workers*, en. [Online]. Available: https://web.dev/learn/pwa/service-workers/.

[5] *Offline service workers*, en. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_workers.

[6] *Caching*, en. [Online]. Available: https://web.dev/learn/pwa/caching/.

[7] *Caching strategies*, en. [Online]. Available: https://developer.chrome.com/docs/workbox/caching-strategies-overview/.

[8] *Service worker overview*, en. [Online]. Available: https://developer.chrome.com/docs/workbox/service-worker-overview/.

[9] *What is Workbox?* en. [Online]. Available: https://developer.chrome.com/docs/workbox/what-is-workbox.

[10] *Workbox-strategies*, en. [Online]. Available: https://developer.chrome.com/docs/workbox/modules/workbox-strategies/.

[11] *Push notifications overview*, en. [Online]. Available: https://web.dev/push-notifications-overview/.

[12] *Reengable push notifications*, en. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Re-engageable_Notifications_Push`.

[13] *Server side implementation*, en. [Online]. Available: `https://web.dev/push-notifications-server-codelab/`.

[14] S. Fourault, *How Progressive Web Apps can drive business success?* en. [Online]. Available: `https://web.dev/drive-business-success/`.

Appendix A

Appendix

The source code of the application implemented throughout the guided invidiual study can be found on this link:

https://github.com/gburapeter/react-pwa

The github branches can be apprehended as building blocks, each weeks improvements can be followed in a tree fashion.

Optionally, the readme files contain a guide on how to set up a local environment to test the application.