MORE FUN, FEWER RISKS: DEVELOPMENT OF A GAMIFIED WEB APP FOR RISK MANAGEMENT

STUDIENARBEIT

des Studienganges Informatik an der

Duale Hochschule Baden-Württemberg Karlsruhe

von

Inga Batton, Moritz Horch, Nils Krehl

Abgabedatum:

18. Mai 2020

Bearbeitungszeitraum:	TODO: XX Wochen
Matrikelnummer, Kurs:	XXX, TINF17B2
Ausbildungsfirma:	dmTECH GmbH, Karlsruhe
Betreuerin:	Ph.D., Prof. Kay Margarethe Berkling

Abstract

Erklärung

(gemäß §5(3) der "Studien- und Prüfungsordnung DHBW Technik" vom 29.09.2017)

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: "More Fun, Fewer Risks: Development of a Gamified Web App for Risk Management" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Contents

List of Figures

List of Tables

List of listings

Glossary

Item Name description

1. Introduction

context, motivation, aims, purpose, ..

Latex Cheat Sheet: Bildquelle mit Seite:Quelle normal:

Bilder normalerweise: Bild über den Seitenrand vergrößern und mittig ausrichten:

ABB. 1.1.: *Title*
[bibkey]

Fancy quotes:

> *cite*
>
> TEXT []

Tabelle:

ABB. 1.2.: *Title*
[bibkey]

Spalte1Titel	Spalte2Titel	Spalte3Titel
1	3	5
2	4	6

TAB. 1.1.: *Unterschrift*

LISTING 1.1: *Title*

```
1 print("Hello world")
```

2. Theoretical background

text...

2.1. Risk Management

Fische [**WetterAktuellWettervorhersage2019**]

2.1.1. Unterkapitel

2.2. Gamification

text.. [6]

2.2.1. Unterkapitel

2.3. Progressive Web Apps

As of today, devices such as mobile phones and personal computers come with their own app store. Microsoft offers their own Store, Google its Play Store and Apple the App Store. As a user you often find yourself worrying about an app you once saw running on another platform not being available for your platform too (e.g. Apples iOS and Googles Android). [5, p. 3] Not to mention app developers who are in a hurry to make their apps available for all known platforms.
Progressive Web Apps (PWAs) approach these concerns by trying to move away from app stores onto a platform which is available on most devices – the web browser. This means that PWAs are regular web apps at their core but can progressively leave the web browser. For

example, PWAs can be installed on the underlaying operating system and be accessed from the app switcher or the taskbar and be executed in full screen mode without the browsers interface being visible. [3, p. 26] Further characteristics of PWAs and how exactly a PWA can leave the web browser are granularly described in the following chapter.

2.3.1. Characteristics of a Progressive Web App

To transform an existing Web App into a PWA or build one from scratch one must implement different criteria's instead of including a new framework or library. [5, p. 6] The following eight characteristics represent Mozilla's (Firefox) ideas of a PWA. Other web browser manufactures such as Microsoft (Edge) or Google (Chrome) for example describe eight or ten respectively within their developer documentations. [3, p. 90][4]

1. **Progressive**: The first characteristic defines that a PWA should not exclude the user from using the core functionality but extend the user experience by embracing new features implemented by the web browser manufactures. [3, p. 100][1, p. 2]. For example, a web app to check mails should not exclude the user from checking their inbox or sending mails but could provide push notifications to inform about incoming mails. In this example the user is not excluded from using the core functionality of a mail app (checking the inbox and sending mails) and user experience is enhanced by push notifications. To avoid unexpected failures a developer should follow the *Feature Detection* principle which says an application should not blindly use a non-standardized feature before checking its existence. [3, p. 101]

2. **Network Independent**: Using a regular web app on the go can be a problem, especially in regions with little to no mobile reception or no stable Wi-Fi being around. Thus, the dynamic content of a web app does not load within a tolerable timeframe or a user is inhibited to perform actions like sending a mail. [3, p. 106] On these grounds a technology called *Service Worker* has been established and implemented by many browser manufactures. In short, a service worker is a script that is able to listen to the network traffic caused by a PWA and therefor is able to cache possible answers fetched from a server and serve them to the web app when no stable network connection is available or

do background syncing by running code even when the web app isn't in use. For more information about the service worker see chapter 2.3.3 (p. 8). [5, p. 43]

3. **Safe**: As mentioned in the previous paragraph, service workers can run code independently from the PWA. To avoid harmful service workers from running malicious code, browser manufactures expect PWAs to be served by a trusted host over a secure connection. To be more precisely, over an HTTPS connection. [5, p. 24] *HTTPS* stands for Hypertext Transfer Protocol Secure and is based on *TLS* (Transport Layer Security). Once the host has obtained a digital certificate for its domain, this certificate is being transferred to the client where it can be verified by the web browser. On success an HTTPS connection can be established and every upcoming network traffic will be encrypted. [3, pp. 112-113]

4. **Re-engageable**: A feature which native apps are using for a while now are push notifications. Push notifications are a common way to inform users about the newest events such as a new mail in the user's inbox. Thanks to the Push API that is implemented on top of the service worker, just like native apps, PWAs can keep the users engaged by sending notifications as it can be seen in figure 2.1. [1, p. 201]

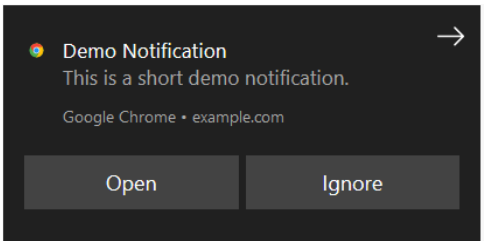

ABB. 2.1.: *Push notification to keep users engaged*

5. **Responsive**: This characteristic specifies that the PWA render its user interface corresponding to the devices used to access it. This is necessary as the available space and input method can change from device to device. The screen of a phone on average is way smaller than the screen of a notebook. Furthermore, using fingers to interact with a phone is less precise than using a mouse on a notebook. [3, pp. 115-116] In figure 2.2 for example one can see how the content and navigation arranges differently on each device. Due less space the navigation bar on the mobile version (extract on the right side)

is completely collapsed and can be accessed by clicking the so-called burger-like icon while on the desktop version the whole navigation bar is visible.

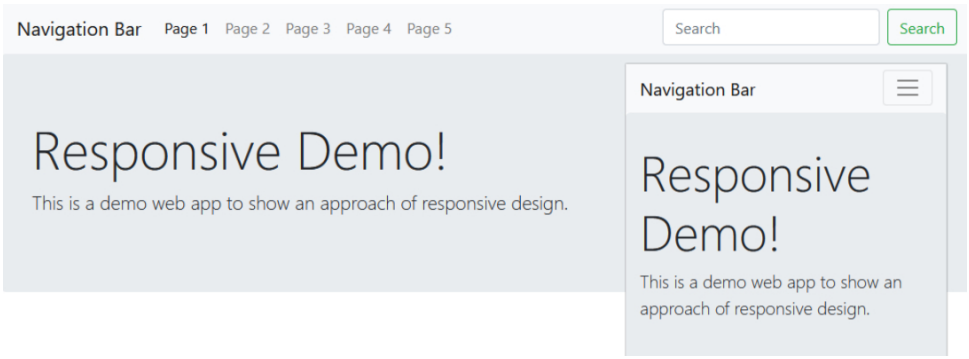

ABB. 2.2.: *Responsive design*

6. **Discoverable**: As PWAs are not a new framework or library but regular web apps at their core, there needs to be method to distinguish between a PWA and a regular web app. This is necessary for web browsers to provide additional features to PWAs such as an option to install (see next paragraph). To make a PWA discoverable a "Web Manifest" file (see chapter 2.3.2, p. 7), which contains information like the name of the PWA, needs to be provided. [3, p. 118]

7. **Installable**: To take things even further, besides offline functionality, a PWA should be installable to the user's device. In detail, a user should be able to install the PWA from within the web browser to the underlaying operating system like Android or iOS. From this point on the user can launch the PWA directly from the devices home screen like it is shown in chapter 2.3.2 (p. 7). Different browser manufactures expect different requirements to be fulfilled before they provide an option to install. Mozilla's Firefox for example expect that the PWA is network independent, safe and discoverable. [2]

8. **Linkable**: The last characteristic implies that a PWA is referable by a *URL* (Uniform Resource Locator, e.g. "www.example.com") instead of being in the need to be installed via any app store. Ideally, the URL should also point to different views of a PWA like a profile page of a specific person. Hence, the current view can be easily shared between users. As PWAs are being run by a web browser, which need an URL to access the web

app in first place, this characteristic, in its fundamentals, does not require any further attentiveness by the developer. [3, pp. 126-127]

2.3.2. Web Manifest

The web manifest is a *JSON* (JavaScript Object Notation) file. Its primary task is to make a PWA discoverable and installable (see chapter 2.3.1, p. 4) by providing descriptive information, like a short app name and paths to icons, about the PWA. The following listing shows a minimal web manifest:

LISTING 2.1: *Example Web Manifest*

```
{
  "short_name": "PWA Demo",
  "name": "Progressive Web App Demo",
  "description": "A simple Progressive Web App Demo.",
  "icons":
    [{"src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon" },
    {"src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192" },
    {"src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512" }],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#dddddd",
  "background_color": "#ffffff"
}
```

The `(short-)name` represent the name of the PWA which is used on the app switcher or home screen, depending on how much space is available. `icons` contains various file path to app icons with different sizes which are used in different scenarios like the app switcher,

home screen or the apps splash screen. For each use case the most appropriate size is chosen automatically. `start_url` defines the entry point of the PWA, `display` holds information about how the PWA will be displayed once it is installed (e.g. `standalone` for no web browser elements) and finally `theme`- and `background_color` which determine the primary color of the user interface and the background color of the splash screen respectively. [2]
Figure 2.3 shows the effects of this web manifest on an example PWA. On the left one can see the use of the icon and name field in Googles Chrome "Add to Home Screen" prompt. In the middle the short name is used due the given space. Once the PWA is launched, like on the right, the standalone display mode is used which hides all elements of the web browser.

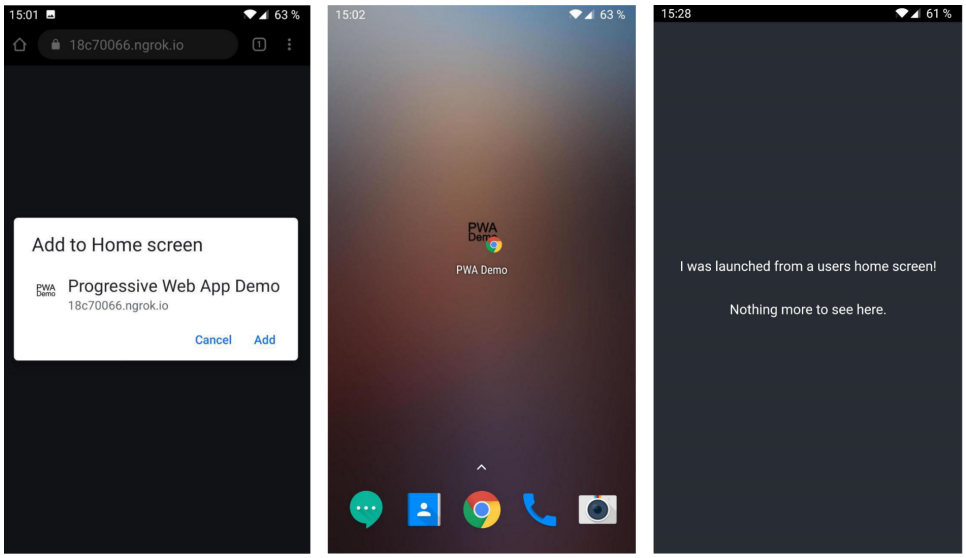

ABB. 2.3.: *Outcome of the web manifest*

2.3.3. Service Worker

A service workers, a in JavaScript written script which runs in the context of the web browser, primary purpose, as mentioned in chapter 2.3.1(p. 4), is to cache content and provide it to the PWA whenever a slow or no connection to the Internet exists. Background syncing as well as sending push notifications can also be realized with a service worker. To achieve this, a service worker has three specific traits: being *controller*, *interceptor* and a *proxy*.
After being registered by the web browser, which means that a HTTPS connection is established and it is request by the PWAs source code, it has full control to all in- and outgoing network

traffic, hence the trait of a controller. Interceptor means that the service worker can manipulate and inspect the network traffic and proxy as the service worker is able to decide if the outgoing traffic should be redirected to the requested resource or completely avoid any outgoing traffic by answering with data it stored in a cache previously. The latter is the exact reason why a service worker is indispensable for the Network Independent characteristic (chapter 2.3.2, p. 7) of a PWA.

The upcoming listing demonstrate how a service worker can manipulate and proxy outgoing network traffic. Before, lets assume that a cache called `pwa-demo` was already created by the service worker.

LISTING 2.2: *Cache First Service Worker*

```
1 self.addEventListener('fetch', event => event.respondWith(
2    caches.open('pwa-demo')
3      .then(cache => cache.match(event.request))
4      .then(response => response || fetch(event.request))
5   )
6 );
```

In the first line one can see the service worker is referencing itself and adds an event listener for all fetch-events (requesting data from outside of the web browsers context). The event listener gets passed the fetch `event` as its second argument on which it calls the `respondsWith` method. Within that method, it opens the cache called `pwa-demo` in line two to then check if the requested event matches the data stored in the cache in line three. If it does match, it then directly returns the data back to the PWA. Otherwise it executes the right part of the conditional OR expression `||` and calls the `fetch` method to get the data from the requested resource. [3, p. 60]

This strategy is called *Cache First* as the service worker will look in the cache before requesting resources outside of the browser's context. Once cached, the data will available much faster and offline but if the resource change its content, the service worker will still return the old, cached version. On the other side the *Network First* strategy exists that will always fetch data when a connection to the resource (e.g. the Internet) can be established. Thus, the user will always

receive the newest content and has a slightly older version available when being offline. On the downside, if the connection is slow the content may take a while to be fetched. Therefore, the *Cache and Network* strategy combines both, the Cache- and Network-First, strategies. To bridge the time of fetching new content, the user is presented the cached content until the result is available and then be presented by refreshing the user interface. These are just a few but popular strategies and each has their own scenarios where they work best. [1, pp. 109-111]

2.3.4. Compatibility

As with every new specification introduced it takes some time until every manufacturer has fully implemented it in their products. In this chapter a short overview of what web browsers, in terms of PWAs, are currently capable of is given. The following figures are taken of a site called www.CanIUse.com (17/10/2019) which shows to what grade a web browser version supports a specification. Red means no support, dark green fully supported and light green supported to a specific grade.
In figure **??** one can see the web browser compatibility of the web manifest for richer offline experiences like being installable.

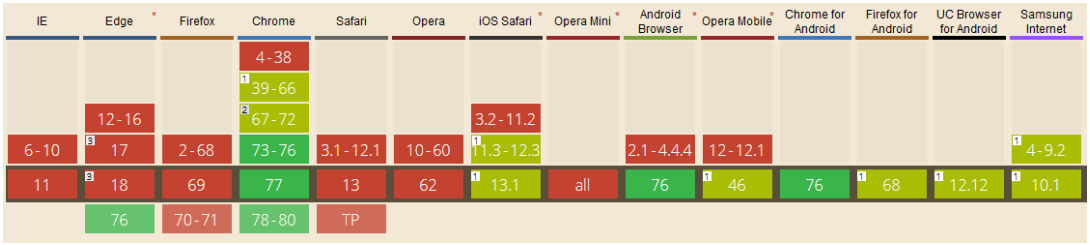

ABB. 2.4.: *Compatibility of the web manifest*

The web manifest is currently not widely supported on the desktop versions of many browsers. Currently only Googles Chrome fully supports it, in return though, most major, except Opera Mini, mobile browsers at least partly support the web manifest.
Figure 2.5 shows the web browsers support for the service worker which primary goal is the offline functionality.
The service worker, regardless of desktop or mobile platform, is currently supported by every major web browser except the Internet Explorer and Opera Mini.

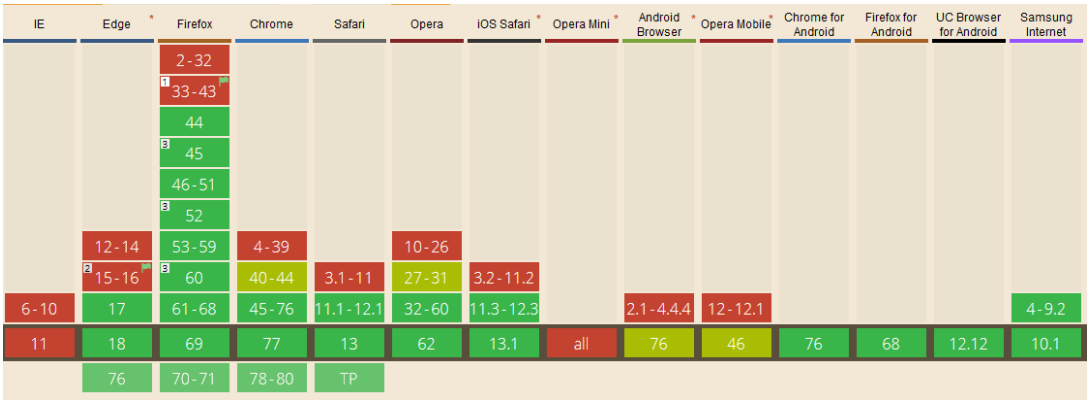

ABB. 2.5.: *Compatibility of the service worker*

If the PWA should use push notifications to inform its users about new occurrences and fulfill the re-engageable characteristic (see chapter 2.3.1, p. 4) it can use the Push API which is another specification that needs to be implemented by the web browsers manufacturer.

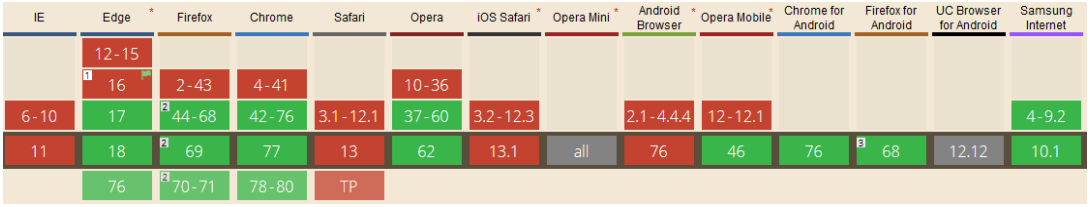

ABB. 2.6.: *Compatibility of the Push API*

As seen in figure 2.6, most manufactures have implemented this feature, Apple falls behind with their desktop and mobile web browser Safari as well as Microsoft's Internet Explorer. Thinking back upon the first characteristic of a PWA, being progressive, a non-supported feature (e.g. the web manifest in Mozilla's Firefox) won't lock the user out of the app as all these specifications should only enhance the user experience and not lock the user out from using the core functionality of a web app.

3. Domain description

text...

3.1. Survey

3.1.1. Unterkapitel

3.2. Domain Model

3.2.1. Unterkapitel

3.3. Gamification concept TBD

3.3.1. Unterkapitel

4. Software Specifications

4.1. Technologies

4.2. Requirements

4.3. Use Case Specifications

4.4. Architecture

5. Implementation

5.1. Unterkapitel -> Design, Evaluation, Methodisches, PM, ...

5.2. Unterkapitel2

6. Discussion

7. Conclusion and Outlook

List of references

[1] Majid Hajian. *Progressive Web Apps with Angular: Create Responsive, Fast and Reliable PWAs Using Angular*. OCLC: 1103217873. 2019. ISBN: 978-1-4842-4448-7 978-1-4842-4449-4. URL: http://public.eblib.com/choice/PublicFullRecord.aspx?p=5780029 (visited on 10/17/2019).

[2] *How to Make PWAs Installable*. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs (visited on 10/17/2019).

[3] Christian Liebel. *Progressive Web Apps: das Praxisbuch*. 1. Auflage. Rheinwerk Computing. Bonn: Rheinwerk Verlag, 2019. 518 pp. ISBN: 978-3-8362-6494-5.

[4] *Progressive Web Apps*. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (visited on 10/17/2019).

[5] Dennis Sheppard. *Beginning Progressive Web App Development: Creating a Native App Experience on the Web*. OCLC: 1018305973. New York: Apress, 2017. 266 pp. ISBN: 978-1-4842-3090-9 978-1-4842-3089-3.

[6] *What Is Human-Computer Interaction (HCI)?* 10.10.19. URL: https://www.interaction-design.org/literature/topics/human-computer-interaction (visited on 10/10/2019).

Appendix

A. Anhang1