Is a Progressive Web App an Alternative for Native App Development?

A prototype-based usability evaluation of a health insurance app

Julian Diekmann[1], Mathias Eggert[2]

Abstract: The existence of several mobile operating systems, such as Android and iOS, is a challenge for developers because the individual platforms are not compatible with each other and require separate app developments. For this reason, cross-platform approaches have become popular but lack in cloning the native behavior of the different operating systems. Out of the plenty cross-platform approaches, the progressive web app (PWA) approach is perceived as promising but needs further investigation. Therefore, the paper at hand aims at investigating whether PWAs are a suitable alternative for native apps by developing a PWA clone of an existing app. Two surveys are conducted in which potential users test and evaluate the PWA prototype with regard to its usability. The survey results indicate that PWAs have great potential, but cannot be treated as a general alternative to native apps. For guiding developers when and how to use PWAs, four design guidelines for the development of PWA-based apps are derived based on the results.

Keywords: Progressive Web App, PWA, Cross-platform, Evaluation, Mobile web, Usability

1 Introduction

In recent years, the popularity of mobile applications for smartphones, also known as apps, has increased enormously, which has also led to a steady growth in the development of such applications [AL18]. Moreover apps are a driver for digitalization. One of the biggest app development challenges is the handling of multiple smartphone platforms. Knowledge about one operating system cannot be transferred to another and apps that need to run on different platforms must be developed separately [JMK13]. So-called cross-platform approaches address this challenge by allowing the developer to implement an app that runs on multiple platforms [HHM12]. The investigation of cross-platform approaches is part of current research [BG19, BG18, RM19]. In this context, a relatively new concept, which is called Progressive Web App (PWA), has emerged, which so far has been experiencing a lack of scientific investigation [MBG18]. PWAs run in a web browser like a classic web app and can be installed directly on the device. They use web technologies to provide a user experience that is similar to a native app [BMG17].

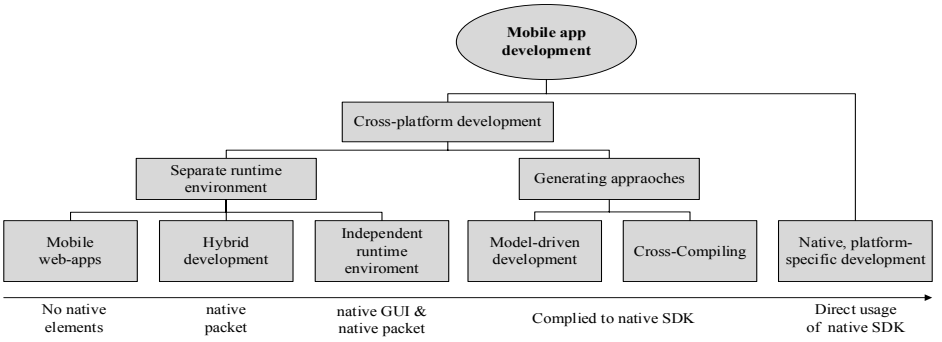
Research in the area of cross-platform apps focus either on technical insights into app development [e.g., BG18, XX13] or its usability [e.g., BG19, RM19]. So far, solely

[1] Fachhochschule Aachen, Eupener Str. 70, 52066 Aachen, julian.diekmann@alumni.fh-aachen.de.
[2] Fachhochschule Aachen, Eupener Str. 70, 52066 Aachen, eggert@fh-aachen.de.

[MBG18] and [BMG17] investigate PWAs. Both focus on concepts or prototypes and do not contribute to the evaluation and comparison of a PWA with its native counterparts. The paper at hand follows the call of Biørn-Hansen et al. [BMG17] to continue the development of technical artefacts in the field of PWA in order to evaluate and compare them with established development approaches. We contribute to this call by answering the following research question: *Is a PWA a suitable alternative for native apps in terms of usability?* The study at hand provides insights into a particular case at a health insurance company, which had to decide whether to support an existing app in its native version or to invest in a new cross-platform app, developed as a PWA. The paper contributes to the body of knowledge in two ways: empirically through the evaluation of a PWA prototype as well as a methodologically through the development of design guidelines.

2 Related work

Different manufacturers, such as Apple and Google rely on different platforms that are incompatible with each other [HKM15] and require a separate code base. In contrast to native development, cross-platform approaches enable a platform independent code. Besides native app development, Heitkötter et al. [HM12] divide cross platform approaches in generating approaches and the ones having an independent runtime environment. Fig. 1 provides an overview about current and popular approaches.



Adapted from Heitkötter et al. [HM12].

Fig. 1: App development approaches

Native app development is the traditional approach to develop a mobile application [MBG18]. Apps are developed for a specific platform using the respective software development kit (SDK) and its frameworks. By using platform-specific application programming interfaces, developers can access device features, such as camera, GPS and push notifications [HHM13]. Native apps provide a good user experience with efficient code that ensures fast performance and full access to data and hardware [XX13]. However, the developers are limited to the manufacturers' specifications [HM12]. Cross-platform approaches address these shortcomings [HHM13] by combining different frameworks or approaches and programming paradigms, aiming at code reuse [BG19]. *Cross platform*

approaches are divided into generating approaches that use a common, cross-platform code base that is transformed into purely native apps [HM12] and separate runtime environments that interpret platform independent code for the respective operating system and serves as a bridge between the app and the platform [MBG18].

A *PWA* belongs to cross platform approaches and aims at adding functionality to Web Apps, so that they feel, look and behave like any other installed app [MBG18]. The term PWA was firstly mentioned in a blog entry by Alex Russell [RB15]. Together with Frances Berriman, he lists features that this new technology should fulfil, for example the installability on the smartphone and the ability to be used offline. Meanwhile, the Google Web Fundamentals Group took over the concept [BMG17]. The design of a PWA is based on three pillars to create a user experience that is similar to a native app. First, it should be able to use functions that were previously solely available in installed apps. Second, they should work reliably - even with poor or no network connection. Third, a PWA must be installable [RL20]. These three pillars describe PWA's contribution to the unification of web apps and native apps. Any website that meets a certain set of criteria can be implemented as a PWA and thus profit from features like being able to be installed and distributed without app marketplaces [BMG17]. Furthermore, PWAs are easy to implement compared to other approaches, because standard web development skills are required. This is also the reason why there is an immense community of developers and a huge number of tutorials and profound tool support are available [RM19].

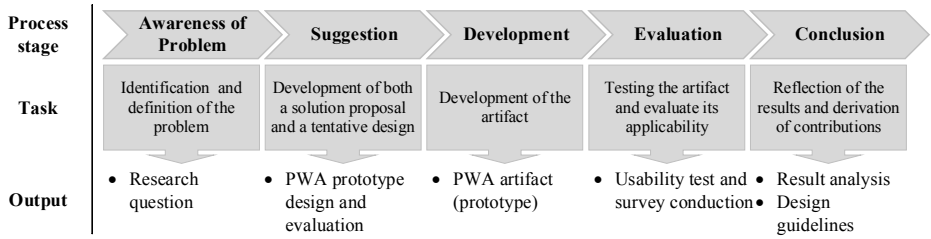
The number of works on PWA is very limited. For the first time, Biørn-Hansen et al. [BMG17] try to raise awareness of PWAs in the academic community. For this purpose, they develop three artefacts that are compared with each other and based on different approaches: Hybrid, Interpreted and PWA. Majchrzak et al. [MBG18] aim at answering the question whether PWAs are the approach to choose for cross-platform development by studying the literature. However, they do not provide a definitive answer to this question. The most recent work of Rieger and Majchrzak [RM19] also deals with PWAs. They propose a framework for evaluating cross-platform development approaches and test their criteria on PWAs, React Native, PhoneGap and native apps. Regarding the usability of web apps, Nielsen [Ni10] describes five attributes that make usability measurable: Learnability, Efficiency, Memorability, Errors, and Satisfaction. For the research work at hand, we focus on evaluating errors of and the satisfaction with a PWA.

3 Research design

3.1 Research process

The present work uses a Design Science Research approach [GH13, HC10] to investigate whether PWAs are a suitable alternative for native apps in terms of usability. We apply prototyping as a qualitative design-oriented method to develop an artefact, which we then evaluate by a survey among app familiar and non-familiar users in order to receive insights into perceptions and differences of a PWA prototype and its native counterpart. The individual process steps are performed once and without iteration as suggested by

[VKP04] because we could not ask the respondents more than once. In total, the research work comprises five phases, which are depicted in Fig. 2.



Adapted from Vaishnavi et al. [VKP04].

Fig. 2: Research process

3.2 Awareness of problem and suggestion

To identify the problem, we analyzed the status quo of app development processes within a large German insurance group by searching in the company Wiki for the terms "app development" and "app strategy", which brought us to an already existing app, which belongs to the health insurance division. We discussed the current development challenges with two experts, who were involved in the app launch. The interviews revealed the redundant native development for two platforms (Android and iOS) as well as a large development effort. So far, no developer within the insurance group applied cross-platform approaches and the interviewees question its usefulness in terms of usability, which leads to the proposal of cloning the existing native health insurance app and use it as a basis for evaluating a cross-platform prototype. By conducting a literature review, we ensured that no existing research about PWA usability exists and that PWA is one of the most advanced cross-platform approaches (cp. Section 2).

3.3 Development

The development phase began with the analysis of the existing native app, which served as a template for the PWA prototype clone. For this purpose, the app was installed on a test device and viewed from the user's perspective. We navigated through the app step by step and analyzed three elements: Structure, functions, UI.

In the next step, we decided the choice of technologies to be used, as there are several options available for development. A PWA describes the features that a web app must have, but does not specify any frameworks or technologies. For this reason, we applied web development technologies that are common within the investigated health insurance company (e.g. the JavaScript Framework Vue.js and UI libraries, such as Vuetify and Hooper). To optimize the software selection for the prototype, we discussed it with experienced developers of the insurance company. After the analysis of the template app and the definition of the technologies to be used, the PWA prototype was developed.

3.4 Evaluation and conclusion

We evaluated the developed prototype in a two-step procedure: Testing the prototype (step 1) and answering a questionnaire (step 2). The aim of the first step is to make the user familiar with the implemented functions of the PWA prototype app. After an introductory text explaining the survey, the respondents are guided through a tutorial on how to install the PWA on their smartphones. Afterwards, we asked whether the installation was successful. If the question was answered with "No", the user is asked to provide a description of the problem. Finally, we requested demographic data and the frequency of use of self-installed apps in order to reveal details about the normal app usage behavior. In case of successful installation, the test participant had to perform six tasks in the PWA, which we provide in Table 1. The tasks guided the participant's procedure step by step. After each completion, the respondent must return to the home page as a starting point.

No.	Task
1	Navigate to a subpage with a camera function. Take a photo and then delete it.
2	Navigate to a subpage with GPS service. After activating the location permission, tap a button that locates the smartphone and displays the location on Google Maps.
3	Navigate to a subpage that provides contact options. Tap on "Write e-mail".
4	Swipe through a carousel slider to find a named tile, tap it, and open a subpage. On the bottom of the page, he should tap on a button that opens a PDF via an external link.
5	Swipe through a carousel slider to tap a tile that names another app. On the bottom of the opened subpage, tap on a button that opens the app in the app store in use.
6	Swipe through a carousel slider to find another named tile and tap it. On the bottom of the opened subpage, the user should tap the "Call Directly" button, which opens the phone app.

Tab. 1: PWA prototype user tasks

After completing the tasks, the participants answered a questionnaire. We decided to create an own questionnaire since the existing questionnaires for evaluating user experience do not completely fulfil the requirements of the current case (e.g., questions regarding the test scenarios and navigation are not existent). However, some existing items, such as stimulation and efficiency, are applied for the questionnaire at hand. The measurement is conducted by applying a 5-point Likert scale. Possible answers are: 'I agree', 'I rather agree', 'neither', 'I rather disagree' and 'I disagree'. We further integrated two optional free text fields, in which the respondents may report problems during the test and other comments.

We conducted the survey online by using LamaPoll. All tasks and questionnaires were pre-tested by two test persons. We collected the study data from two independent respondent groups: Respondent group 1 is not familiar with the original prototype. Respondent group 2 knows the native counterpart of the PWA prototype. We invited employees of the health insurance company by sending an e-mail containing a link to the survey. In addition, we invited potential users outside the company in order to obtain a wider range of participants. All questions for both respondent groups are provided in Tab. 2. The demographics of both respondent groups are summarized in Tab. 3.

Questionnaire for users, unfamiliar with the native app (respondent group 1)	Questionnaire for users, familiar with the native app (respondent group 2)
1. I was able to solve the tasks of the test scenarios without any problems.	1. I did not realize that the app was actually a web page.
2. I was able to navigate through the app without errors.	2. The PWA is visually almost indistinguishable from the native app.
3. I never got stuck in the app at any time.	3. The PWA behaved as expected.
4. The response time of the app was satisfactory.	4. The response time of the app was satisfying.
5. The animations (swiping the tiles, waiting time circle) of the app ran smoothly.	5. I was able to solve my tasks as well as in the native app.
6. While using the app, no unexpected behaviour occurred (display errors, app crash).	6. While using the app, no unexpected behaviour occurred (errors in display, crash of the app).
7. The app is user friendly.	7. The functionality of the PWA is almost the same as in the native app.
8. I did not notice that the app is actually a web page.	8. I am satisfied with the current functionality of the PWA.
9. It was fun to use the app.	9. I would not prefer the native app over the PWA app with the same functionality.
10. In summary, I was satisfied with the experience of using the app.	10. In summary, I was satisfied with the experience of using the app.

Tab. 2: PWA usability questionnaire

	Respondent group 1 (unfamiliar users)						Respondent group 2 (familiar users)					
Gender:	m	f					m	f				
	41	12					8	2				
Age:	<18	18-26	27-36	47-56	>= 57	n/a	<18	18-26	27-36	47-56	>=57	n/a
	16	6	8	17	5	1	0	0	3	4	3	0
Platform:	iOS	Android	Other				iOS	Android	Other			
	16	35	2				6	4	0			
Browser:	Chrome	Firefox	Safari	Other			Chrome	Firefox	Safari	Other		
	26	6	16	4			2	1	6	1		
App usage:	often	rather often	rather seldom	seldom			often	rather often	rather seldom	seldom		
	21	18	8	6	*n=53*		6	1	2	1	*n=10*	

Tab. 3: Respondent demographics

4 Results

4.1 PWA prototype

Structure, Functions and UI

After setting up the site structure, the functions of the app are examined in more detail. Due to confidentiality reasons, we cannot provide screenshots. Instead, the page structure of the original health insurance app is depicted in Fig. 3. The app provides functions that we classify into three function groups. The first group contains information and services based on internet links. These sites redirect a user to external pages via buttons, such as apps and homepages of partner companies or PDF files containing health insurance advice.

In addition, the phone's keypad can be started and the e-mail app can be opened. These services are easy to implement in a PWA.

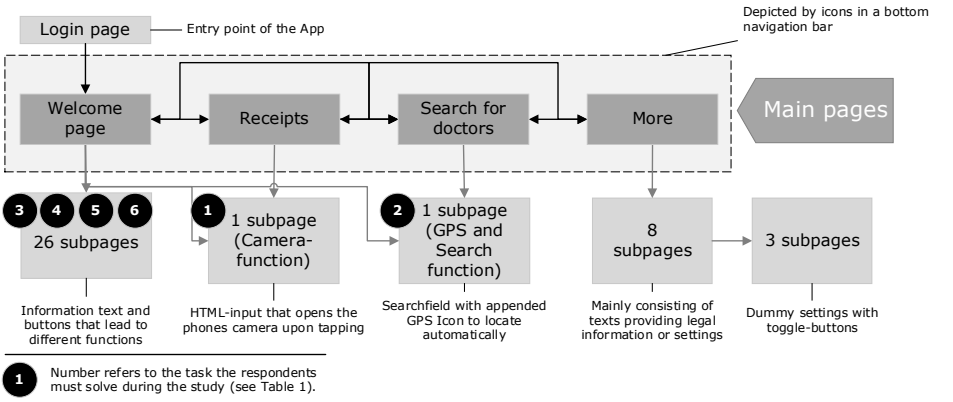


Fig. 3: Page structure of the original health insurance app

The second group includes services based on foreign programming interfaces. The app receives data via server requests and prepares them or sends data to a server, e.g., a service for waiting time information. We exclude this type of functions, as the integration of an API would most probably have no usability effect. The third group comprises services that access device features. The native app offers the possibility to take pictures of documents via the smartphone's camera and to access the current location via GPS when searching for a doctor. These functions are mandatory, as they are among the services that provide the greatest customer value. Subsequently, we analyze the usability of the app. The app uses an independent design and does not appear in the native design of the respective platform. Accordingly, the apps on iOS and Android hardly differ. In addition to a navigation bar at the bottom of the page, the app uses UI elements typical for mobile devices, such as large buttons and tiles. The navigation on the pages works via swipe gestures. Two subpages use a simple animation that can be recreated with CSS.

Technology selection

The company applies the JavaScript framework *Vue.js* for the front-end development of Web Apps. At the time of development, Vue.js is in version 2.6.10. The *command line interface (CLI)* requires Node.js and enables rapid prototyping through pre-built configurations and the independent installation of desired tools. Projects created with the CLI use the JavaScript Module Bundler Webpack, which can be extended with plugins. One of the core plugins offered by the CLI is the *cli-plugin-pwa*. The plugin is based on Workbox, a set of libraries that aims at making a PWA easy to create. Vue.js components have their own HTML, CSS and JavaScript sections. By combining HTML template, JavaScript logic and styling in one file, the component becomes more coherent and maintainable. Vue.js organizes an app into a tree of nested components the component in

the tree structure that represents the page itself is called view. For navigation between the different views, we use the *Vue router*.

Furthermore, we apply *Vuetify* version 2.1.0 as the basis for developing the individual UI elements. Vuetify is a UI library for Vue.js and builds up on Google's material design, which makes the UI elements conform to the Android UI design guidelines. On the start page, the app contains links to subpages on different topics, grouped by tiles. The tiles for a theme are lined up side by side and can be moved using swipe gestures. This so-called Carousel Slider is not included in the Vuetify component library. In order to implement it, we apply the *Hooper* component. Hooper is optimized for Vue.js and provides a customizable Carousel Slider for mobile use, which enables the replication of the existing health app. We utilize IntelliJ IDEA Ultimate as development environment, which supports Vue.js with code completion for Vue-specific elements. Finally, we need to provide the prototype for the survey participants. Therefore, we apply *Google Firebase*, which enables the hosting of web apps and can be integrated into the project via the package manager of Node.js.

Camera access can be realized in two ways. The camera is started without leaving the PWA by applying the WebRTC API. However, this approach is unsuitable, due to a bug in iOS version 13.3. Although the bug is fixed in iOS 13.4, it is not expected that all respondents have that iOS version. Therefore, we apply the second variant and use the device's own camera app to create the photos.

The base-project was created within minutes by the CLI and offered a fully usable PWA. Afterwards, the page structure was built using *Vue router* followed by the UI. The final PWA consists of 44 pages which were realized through only 20 view-files, thanks to the component-based approach of Vue.js. Ultimately, the functions needed to be implemented. The app offers links to partner apps that can be downloaded through the according app marketplace. To differentiate between an Android phone and iOS device, the User Agent’s header is checked. For access to the user’s current location during the doctor search, the browser’s *Geolocation API* is used. The camera access revealed to be the hardest function to implement, mostly due to the aforementioned bug. The camera function was implemented by using a html-input-tag that only accepts images. Upon tapping on it the input field would open the devices camera and display the result afterwards inside the PWA. This functionality differed the most from the native app. Without the bug and with more experience in web-development the gap between the apps in this function could have been mitigated. Apart from that, the development process revealed to be easily executed.

4.2 PWA usability

After the completion of all tasks (cp. Tab. 1), a respondent receives depending on its group (familiar or non-familiar users) a questionnaire (cp. Tab. 2) and is asked to answer the ten questions. In respondent group one, we received 50 completed questionnaires (three are

incomplete due to app installation problems). The comparative survey among familiar users had 21 visitors, out of which 16 took part in the survey and 10 completed it.

The results of the survey with users, who do not know the original app are summarized in Tab. 4. None of the items receives less than 72% positive ratings (I agree / I rather agree), which indicates a predominantly satisfaction. The first statement "I was able to solve the tasks of the test scenarios without any problems" represents the success of the participants in completing the tasks. The feedback on this statement is decisive for the validity of the further usability evaluations. 40 participants had no problems with the tasks and three respondents answered the statement with "I rather agree". Solely the rating of the reaction time (item 4), which is rated as completely satisfactory by 44 participants, receives a higher satisfaction. The statement "It was fun using the app" receives the lowest agreement ratings. 18 of 50 chose "I agree" and "I rather agree". One user explains his rating as follows: "The topic of the app is just boring, therefore not the full rating at 'Fun'".

No	Question	I agree	I rather agree	neither	I rather disagree	I disagree
1	I was able to solve the tasks of the test scenarios without any problems.	40	3	1	3	3
2	I was able to navigate through the app without errors.	28	12	1	2	7
3	I never got stuck in the app at any time.	25	12	0	4	9
4	The response time of the app was satisfactory.	44	4	2	0	0
5	The animations (…) of the app ran smoothly.	37	9	1	2	1
6	While using the app, no unexpected behaviour (…) occurred.	37	5	0	2	6
7	The app is user friendly.	22	19	3	6	0
8	I did not notice that the app is actually a web page.	31	8	4	3	4
9	It was fun to use the app.	18	18	10	3	1
10	In summary, I was satisfied with the experience of using the app.	26	15	5	3	1

Tab. 4: Descriptive survey results (unfamiliar users)

The statement "I never got stuck in the app at any time" gets the most disagreement ratings with 9 for "I don't agree" and 4 for "I rather disagree". The statement "I was able to navigate through the app without any errors" receives a similar rating. One participant writes: "Backward navigation was sometimes difficult. The lower [navigation] bar disappears on the individual subpages. It would be practical if this bar would not only appear on the start page, but also on all other pages". Furthermore, the statement "The app is user-friendly" receives 22 agreements. Notably, this is the only statement where none of the respondents voted "I disagree". Together with the 19 for "I tend to agree", the item achieves an overall rate of 82% in positive feedback, which expresses a high level of user satisfaction with the PWA's usability. In total, 82% of the participant agree or rather agree to the final satisfactory statement (10). In addition to the quantitative analyses, we deeply analyze the most frequent statements made in the free text fields. After removing statements regarding UI design decisions, which we cannot influence due to the reproduction of an existing app, we receive 24 responses to noticed problems.

13 participants (5 Android users and 8 iOS users) complain about missing navigation arrows. In some cases, the subpages of the app do not have buttons that link to the parent page. Navigation back to the main pages is only possible on Android using the back button of the operating system, on iOS it is possible to navigate back to the main pages using a swipe gesture in Safari. One Android user writes: "Navigation is only possible via the back button on the smartphone; there should be a link to the start page on the individual pages of the app". An iOS user writes: "I didn't immediately know how to get back to the start page or previous page from a subpage. I would still provide a navigation in the sense of a back button on the pages. Older users (like me) are not used to navigating by gesture/swiping only". Three participants even mention that after navigating to the subpages, it was necessary to restart the app on their iOS device.

The second most common problem mentioned by seven participants is the unintentionally navigation to the login page. One participant wrote: "Swiping down on the start page leads (surprisingly) back to the password entry". Two other participants wrote that they accidentally come to the login page by navigating back. The reason for this behavior is the non-existent client-side storage of the successful login. This state is a problem that appears due to the prototype environment. Four participants stated that they were unable to perform the last step of the first task, in which taken photo should be deleted. However, an unexpected behavior occurred during the deletion attempt. One participant writes: "Image cannot be deleted and instead the slide log for uploading an image is displayed again". So far, we cannot explain the reasons for this behavior.

No	Question	I agree	I rather agree	neither	I rather disagree	I disagree
1	I did not realize that the app was actually a web page.	4	4	0	0	2
2	The PWA is visually almost indistinguishable from the native app.	6	2	1	0	1
3	The PWA behaved as expected.	4	3	0	2	1
4	The response time of the app was satisfying.	4	6	0	0	0
5	I was able to solve my tasks as well as in the native app.	4	2	0	3	1
6	While using the app, no unexpected behavior occurred.	5	0	0	2	3
7	The functionality of the PWA is almost the same as in the native app.	1	7	0	2	0
8	I am satisfied with the current functionality of the PWA.	2	5	0	3	0
9	I would not prefer the native app over the PWA app with the same functionality.	3	0	3	0	4
10	In summary, I was satisfied with the experience of using the app.	3	5	0	2	0

Tab. 5. Descriptive survey results (familiar users)

In addition to insights into PWA usability within unfamiliar users, we also evaluated the prototype with users, who are familiar with the original native app (respondent group 2). Tab. 5 provides the descriptive evaluation results of the Likert-Items. Eight respondents agree or rather agree to the statement "I didn't notice that the app is actually a web page". Additionally, eight participants could not distinguish between the PWA optically from the native app (statement 2). The evaluation of the reaction time of the PWA turned out to be exclusively positive. Regarding the solving of tasks and its comparison to the native app, we receive inconsistent results. Six Participants agree or rather agree to statement 5, while four respondents rather disagree or disagree to it. One respondent wrote: "To submit a document, you have to make too many clicks to create a photo. In the [original] app you can open the camera directly with one click". The reason is the decision to realize the camera function via an input field. Opening the device's camera app provides additional steps. The statement "No unexpected behavior occurred while using the app" receives the most negative tendency. Again, one participant reports that he/she returns to the login page when the page refreshes. Another user could not access the site despite several attempts to adjust the settings.

The comparative survey as well as the survey with unfamiliar users reveal navigation concerns. 3 of the 10 participants wrote that they were not able to get back to the main menu from subpages. All three participants are iPhone users, which underlines the assumption that this problem occurs particularly on iOS devices. Despite the differences in the evaluation of the individual statements, 80% of the app familiar users give a positive tendency regarding their summarized satisfaction with the usage experience of the PWA.

4.3 PWA design guidelines

Both, the development process and the evaluation do not clearly answer the question whether PWAs are a suitable alternative to native apps. The survey results indicate that the users are satisfied with the usability of the PWA. In Addition, the prototype development phase also indicates that a PWA outperforms the necessary development time. However, after analyzing the evaluation results and the development process, we also identified some shortcomings of this approach, which lead to four design guidelines for further PWA development projects.

In the beginning of the PWA development phase, we first checked whether all functions of the native app (e.g. camera integration, GPS, carousel slider) are available for the PWA clone. This leads to the first guideline: *Consider basic feature support*. Before deciding to develop a PWA, the question of which functions the app should offer, must be clarified. Starting with the definition of the required functions, it must be checked whether they can be implemented with web technologies at the time of development. We recommend using the feature list on https://whatwebcando.today. If the required functions can be implemented with web technologies, the next question is about the supporting platforms. Android and iOS support features to varying degrees. As for our prototype the camera and GPS interfaces are required, we checked, whether these functions are available for both

iOS and Android platform. Thus, *considering platform support of the features* is the second design guideline.

When all required features can be implemented with web technologies and are supported by both platforms, the last thing to check is the guaranteed feature support of the operating systems version. Older versions do not support newly implemented features, so they may not be available to all users. In our case, we struggled with the camera API, which was not available for a desired iOS version. For this reason, it must be determined which operating system is the minimum requirement for the end user, which leads to the third guideline: *Consider the version of feature support*. Three iOS users that participate in the study reported that they were not able to get back to the main menu from subpages. We believe that this occurs due to the habit of iOS users to have a backward navigation icon instead of the back button as Android users have. Alternatively, developers may implement a navigation bar, which is always visible. This avoids the risk of dead ends in the PWA. This leads to the final guideline: *Use explicit navigation elements*.

5 Conclusion and outlook

The paper at hand presents the results of a comparative study to evaluate the applicability of PWA. We developed a PWA prototype that represents a clone of an already existing native app in terms of "look & feel" and access to device features. Afterwards, we evaluated the app. In total, 60 people attended in the survey and completed it. Regarding the research question of this article, the results of the development phase and the prototype evaluation indicate that PWAs are not in any environment an alternative to native apps. If for example the desired native app function is not available for PWAs, the developer is forced to individually code for all necessary platforms. In order to give orientation for relevant factors that must be considered in advance, we proposed four design guidelines to support the decision making process.

The research contribution of this paper is twofold. First, we shed light on the usability of a real PWA prototype and disclose PWA development procedures. Second, the derived design principles foster fruitful discussions in the community and invite PWA experts to suggest extensions. Practitioners receive an argumentative basis for cross platform development and may reduce costs through the prevention of redundant development.

The expressive power of the results is limited. The sample size of 60 survey participants is quite small and has a gender bias. We did not completely apply a standard questionnaire for usability, since we would like to gain insights into the perception of task completion. However, repeating this study with an additional usability questionnaire could provide additional insights into the usability of PWA. For future research, we suggest investigating whether the support of PWAs on iOS expands so that iOS users are not discriminated. It is also meaningful to conduct additional usability studies with a more representative set of participants. Furthermore, the industry demands an investigation of economic advantages of PWA development projects compared to native developments.

Literaturverzeichnis

[AL18] Ahmad, A. et al.: An Empirical Study of Investigating Mobile Applications Development Challenges. In IEEE Access, 2018, 6; S. 17711–17728.

[BG18] Biørn-Hansen, A.; Ghinea, G.: Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. In (Bui, T. Hrsg.): Proceedings of the 51st Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences, 2018.

[BG19] Biørn-Hansen, A. et al.: An Empirical Study of Cross-Platform Mobile Development in Industry. In Wireless Communications and Mobile Computing, 2019, 2019; S. 1–12.

[BMG17] Biørn-Hansen, A.; Majchrzak, T. A.; Grønli, T.-M.: Progressive Web Apps: The Possible Web-native Unifier for Mobile Development: Proceedings of the 13th International Conference on Web Information Systems and Technologies. SCITEPRESS - Science and Technology Publications, 2017; S. 344–351.

[GH13] Gregor, S.; Hevner, A. R.: Positioning and Presenting Design Science Research for Maximum Impact. In MIS Quarterly, 2013, 37; S. 337–355.

[HC10] Hevner, A.; Chatterjee, S.: Design Research in Information Systems. Theory and Practice. Springer Science+Business Media LLC, Boston, MA, 2010.

[HHM12] Heitkötter, H.; Hanschke, S.; Majchrzak, T. A.: Comparing Cross-platform Development Approaches for Mobile Applications: Proceedings of the 8th International Conference on Web Information Systems and Technologies, 2012; S. 299–311.

[HHM13] Heitkötter, H.; Hanschke, S.; Majchrzak, T. A.: Evaluating Cross-Platform Development Approaches for Mobile Applications. In (Cordeiro, J.; Krempels, K.-H. Hrsg.): Web Information Systems and Technologies. 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers. Springer Berlin Heidelberg, Berlin/Heidelberg, 2013; S. 120–138.

[HKM15] Heitkötter, H.; Kuchen, H.; Majchrzak, T. A.: Extending a model-driven cross-platform development approach for business apps. In Science of Computer Programming, 2015, 97; S. 31–36.

[HM12] Heitkötter, H. et al.: Business Apps: Grundlagen und Status quo. Förderkreis der Angewandten Informatik an der Westfälischen Wilhelms-Universität Münster e.V., Münster, 2012.

[JMK13] Joorabchi, M. E.; Mesbah, A.; Kruchten, P.: Real Challenges in Mobile App Development: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013; S. 15–24.

[MBG18] Majchrzak, T. A.; Biørn-Hansen, A.; Grønli, T.-M.: Progressive Web Apps: the Definite Approach to Cross-Platform Development? In (Bui, T. Hrsg.): Proceedings of the 51st Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences, 2018.

[Ni10] Nielsen, J.: Usability engineering. Kaufmann, Amsterdam, 2010.

[RB15] Russell, A.; Berriman, F.: Progressive Web Apps: Escaping Tabs Without Losing Our Soul – Infrequently Noted. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/, 05.01.2020.

[RL20] Richard, S.; LePage, P.: What are Progressive Web Apps? https://web.dev/what-are-pwas/, 24.4.2020.

[RM19] Rieger, C.; Majchrzak, T. A.: Towards the definitive evaluation framework for cross-platform app development approaches. In Journal of Systems and Software, 2019, 153; S. 175–199.

[VKP04] Vaishnavi, V.; Kuechler, W.; Petter, S.: Design Science Research in Information Systems. http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf, 18.05.2020.

[XX13] Xanthopoulos, S.; Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In (Diamantaras, K. et al. Hrsg.): Proceedings of the 6th Balkan Conference in Informatics on - BCI '13. ACM Press, New York, New York, USA, 2013; S. 213.