Exploring progressive web applications for health care

Developing a PWA to gather patients' self assessments

Mikael Wahlström

September 1, 2017
Master's Thesis in Interaction and Design, 30 credits
Supervisor at TFE-UmU: Keni Ren
External supervisor: Håkan Petterson
Examiner: Thomas Mejtoft

Umeå University
Department of Applied Physics and Electronics
SE-901 87 UMEÅ
SWEDEN

Abstract

Many health care providers aim to become more patient-centered, and developing mobile health applications for patients might help achieve this. In the light of this, this thesis explores if the progressive web application (PWA) concept is suitable for mobile health applications. It is investigated by developing a PWA intended to be used to gather health care patients' self assessments. The work follows the double diamond design process with: a discover phase containing a literature study, interviews with experts, and partaking in a workshop; a define phase where system requirements are specified; a develop phase with lo- and mid-fi prototypes as well as usability tests with six test users; and a deliver phase where the application is implemented using Polymer 2.0 and web components. To furthermore assess the patients satisfaction of PWA, a evaluation phase is conducted where eleven test users tries it during five consecutive evenings and answers a survey at the end. The general opinions were that they thought it worked good and was easy to use, indicating that a PWA can be suitable for this purpose. Following this and discussions of findings, we suggest guidelines for how to design and implement a PWA for similar projects. However, the developed PWA was due to shortage of time not completely finished and the test users support for PWA features were rather limited, so future investigation is recommended to determine if PWAs are suitability in this context.

Contents

1 Introduction **1**
- 1.1 Problem description . . . 2
  - 1.1.1 Problem description for the case study . . . 2
- 1.2 Aim and Objective . . . 2
- 1.3 Restrictions . . . 3
- 1.4 Thesis Outline . . . 3

2 Background **4**
- 2.1 Digital health . . . 4
  - 2.1.1 eHealth . . . 4
- 2.2 County Council of Västerbotten . . . 4
  - 2.2.1 Units for eHealth and IT . . . 4
  - 2.2.2 University Hospital of Umeå . . . 5
- 2.3 Different types of mobile applications . . . 5

3 Progressive web applications **7**
- 3.1 Characteristics . . . 7
- 3.2 Requirements . . . 8
  - 3.2.1 HTTPS . . . 8
  - 3.2.2 Web app manifest . . . 8
  - 3.2.3 Service worker . . . 8
  - 3.2.4 Browser support . . . 9

4 Theoretical framework **11**
- 4.1 The double diamond design process . . . 11
- 4.2 Design aspects . . . 12
  - 4.2.1 Graphical profile for VLL . . . 12
  - 4.2.2 Design patterns . . . 13
  - 4.2.3 App shell model . . . 13
  - 4.2.4 Guidelines for filling in forms . . . 14
- 4.3 Legal aspects . . . 15

4.3.1 General Data Protection Regulation . . . 15
4.3.2 Health and medical service act . . . 16
4.3.3 Patient data act . . . 16
4.3.4 CE marking . . . 16
4.4 Implementation aspects . . . 17
4.4.1 Web components . . . 17
4.4.2 Polymer Project 2.0 . . . 17
4.4.3 Promises in JavaScript . . . 17
4.5 RESTful web services . . . 17
4.5.1 JSON web tokens . . . 18
4.5.2 Pseudonymization . . . 18
4.6 Evaluation aspects . . . 18
4.6.1 Lighthouse . . . 18
4.6.2 System usability scale . . . 19

5 Methodology 20
5.1 Discover . . . 20
5.1.1 Literature study . . . 20
5.1.2 Interviews . . . 20
5.1.3 Workshop with SKL . . . 21
5.2 Define . . . 21
5.2.1 System requirement specification . . . 22
5.3 Develop . . . 22
5.3.1 Prototypes . . . 22
5.3.2 Usability testing the mid-fi mock-up . . . 23
5.4 Deliver . . . 23
5.4.1 Abstract system overview . . . 24
5.4.2 Implementing the PWA . . . 24
5.5 Evaluation . . . 25
5.5.1 Lighthouse Audit . . . 25
5.5.2 User testing . . . 25

6 Results 27
6.1 Discover . . . 27
6.1.1 Interviews at Female clinic/NUS . . . 27
6.1.2 Interviews at the county council of Västerbotten . . . 28
6.1.3 Workshop with SKL . . . 29
6.2 Define . . . 29
6.2.1 System requirement specification . . . 30
6.3 Develop . . . 30
6.3.1 Lo-fi . . . 30
6.3.2 Mid-fi Mock-up . . . 31

6.3.3 Usability testing the mid-fi mock-up . . . 34
6.4 Deliver . . . 34
6.4.1 Changes from the mid-fi prototype . . . 34
6.4.2 Navigation flow . . . 35
6.4.3 Interface . . . 37
6.4.4 Offline notifications . . . 44
6.4.5 Example of caching HTTP requests during runtime . . . 45
6.4.6 The manifest file . . . 45
6.5 Evaluation . . . 46
6.5.1 Lighthouse audit . . . 46
6.5.2 Survey . . . 47

7 Discussion **48**
7.1 Limitations . . . 48
7.2 Findings from developing the PWA . . . 49
7.3 Evaluation of the PWA . . . 50
7.3.1 Lighthouse audit . . . 50
7.3.2 User testing . . . 51
7.3.3 Suggested evaluation for future work . . . 51
7.4 Guidelines for developing a mHealth PWA for gathering self assessments . . . 51
7.5 Suitability of using a PWA in this context . . . 53
7.6 Acknowledgements . . . 53

References **54**

Appendices **56**
.1 Questions for interviewing system developer . . . 57
.2 Questions for interviewing strategic expert (Göte) . . . 58
.3 Questions for interviewing strategic expert (Thomas) . . . 59
.4 Main questions used to create the system requirement specification . . . 60
.5 Document presented before user tests of the mid-fi prototype (in Swedish) . . . 61
.6 Lighthouse audit results . . . 62
.7 Optional answers from the survey . . . 66

List of Figures

4.1 An illustration of the double diamond design model 12
4.2 The colors and corresponding color codes specified in the graphical manual for VLL 12
4.3 A graphical representation of the *hub and spoke* design pattern 13

5.1 An illustration of the system architecture 24

6.1 A collection of some lo-fi sketches made for the application 31
6.2 Some of the mid-fi mock-up pages 33
6.3 A diagram of the navigation flow for the developed PWA 36
6.4 Screenshots of different states when the app is launched 38
6.5 Screenshots of the three views for the introduction page 39
6.6 Screenshots of different stages when a form is filled in 40
6.7 Screenshots of different result view states 41
6.8 Screenshots of different states when accessing form pages 42
6.9 Screenshots of the log in and settings pages 43
6.10 These figures show notifications in the application that tell users they are offline and need to turn on internet, using Chrome DevTools to have offline mode 44

Chapter 1

Introduction

Many health care providers strive to become more patient-centered [1, 2, 3] and may envision a future where health care can be delivered both electronically and in person [3]. One example of this is the creation of electronic health records (EHR) that are accessible for patients via the Internet [4]. The field of utilizing information and communication technologies (ICT) to support the achievement of health objectives is commonly referred to as digital or electronic health (eHealth) [5, 6]. And the field of building such solutions can be referred to as health information technology (HIT) [7]. HIT has the possibility of improving the safety, efficiency and quality of health care [8]. This by, for instance, reducing medical errors and unnecessary, wasteful or variations in care [3]; and generally relieve workload of managing information for health care providers [9]. During the 90s, hospitals and health systems generally did not give ICT high priority which gave it a slow start, but it started increasing rapidly at the beginning of the new century [8].

Mobile health (mHealth) is a sub component of eHealth and it can be described as: medical or health services supported by wireless devices, e.g. mobile phones [6]. With mHealth and mobile communication, it exists tremendous potential to improve provision of health care to patients [7, 10]. One way can be to develop mHealth applications for patients/citizens to be used as tools for treatment or diagnosis of health related matters [8, 11]. Evidently this seems to be of interest since in 2016, over 250 000 applications classified as mHealth apps were available in major app stores - with over 3 billion downloads in total [12].

In this thesis, we explore new possibilities for mHealth applications with a case study of developing such an app using state-of-the-art web technologies - namely by developing a progressive web application (PWA). PWAs can be described as special web apps that progressively utilize modern web technology to, e.g., enable: offline usage, background processing and push notifications [13]. The developed PWA is intended to be used to gather self assessments from patients for the female clinic in the University Hospital in Umeå, and the thesis work is conducted at the hospitals principal, the County Council of Västerbotten. The goals for this work were to (1) suggest guidelines for designing and implementing a mHealth PWA to gather self assessments and (2) give conclusions about how well a PWA worked for gathering self assessments in this health care context.

1.1 Problem description

Mobile health applications have the potential to substantially increase value of health care services [7, 10]. However, a lack of security, accuracy and reliability considerations when developing a mHealth app can yield negative consequences upon its users, and concerns have been raised about such applications being publicly distributed without regulations [14, 11]. Furthermore, there are also some concerns regarding the potential that insufficient design and usability of customer health application might hinder the goals of ICT potentially improving quality, safety and efficiency in health care [2].

At the County Council of Västerbotten, they consider themselves to lack knowledge and experience of developing mHealth applications for patients, and it is arguably likely that other hospital principals in Sweden might be in similar situations. And if so, the utilization of IT to improve health care services could have slow progress. Developing PWAs as tools in health care might speed up the progress of providing a more patient-centered care. New modern web technologies such as those used in progressive web apps can possibly be of great value in the progress of improving health care services. This context has although, to the authors knowledge, not been investigated.

1.1.1 Problem description for the case study

The developed PWA aims to improve the process of doing self assessments to investigate premenstrual (dysphoric) syndrome PMS/PMDS of patients. The current way the female clinic at the University Hospital in Umeå start an investigation of PMS is by giving a patient series of paper form envelops. These forms contain symptoms regarding their mental and physical health that a patient should grade on a daily basis, often during several menstrual cycles to get usable and reliable results for caregivers.

This way of conducting investigation demands a lot of work for caregivers distributing and handling papers physically. It also makes it difficult to use the data for research purposes as it is not stored digitally. Regarded to the patients, the clinic reported that with the current solution, patients complain about the tedious method, often forget to fill in daily, and find it inconvenient to keep the papers with them if they travel somewhere.

1.2 Aim and Objective

The main objective of this thesis is to explore the suitability of using a progressive web application in a health care context. This is achieved by:

1. investigating some of the concerns and needs regarding mHealth applications,
2. developing a PWA intended to be used in a real health care situation of gathering patients' self assessments,
3. evaluate how well the developed PWA support patients needs and fulfillment of the caregivers' requirements.

With this thesis work we aim to contribute with:

1. suggested guidelines for developing a mHealth PWA for self assessments,
2. how suitable a PWA seems to be in this context,

3. and, consequently, present an example of how to design and implement a progressive web application.

The outcomes of this thesis may be informative for anyone interested in building mHealth apps for patients in health care, especially those considering to use web technologies to do it. It can also be of interest for those wanting to use PWAs in other contexts.

1.3 Restrictions

As the thesis work is conducted in Sweden, the explained legal aspects regarding mHealth in this thesis originates from those that prevail in Sweden, enforced by the Swedish government directly or indirectly based on EU regulations. Furthermore, these regulations are only presented briefly and has not been reviewed thoroughly. Therefore, this thesis work might lack presentation of some legal aspects needed to be taken in to consideration when developing mHealth applications. Consequently, the developed PWA in this thesis should not be considered as an example of a publicly ready-to-release mHealth application suited for health care.

The author was not able to find a lot of scientific literature describing PWAs. This possibly depends on the fact that PWAs were only recently introduced (in 2015 [15]). Therefore, the main sources for describing it in this thesis comes from the authors usage and interpretation of developer documentation by Mozilla and Google.

Other than developing the PWA, the complete solution developed during this thesis required a database and a web service for storing, managing and transferring data for the PWA. However, the database and web service part will not be thoroughly described as the scope and focus of this work regards PWAs.

1.4 Thesis Outline

The thesis is structured into these chapters following the introduction:

Background This chapter describes digital, electronic and mobile health briefly; the organization this thesis work was conducted at; and different types of mobile applications.

Progressive web application This chapter describes some of the core aspects of the PWA concept.

Theoretical framework This chapter contains literature used to conduct this thesis work: the design process as well as legal, design, implementation and evaluation aspects.

Methodology This chapter describes the methods used during this work in the process of investigating mHealth as well as developing and evaluating the mHealth PWA.

Results This chapter presents the results from the methods used in the methodology chapter.

Discussion In this chapter findings from this work are discussed, followed by presenting the suggested guidelines along with a conclusion about the suitability of using a mHealth PWA for this context.

Chapter 2

Background

2.1 Digital health

World health organization (WHO) defines digital health as: *"The use of digital, mobile and wireless technologies to support the achievement of health objectives. Digital health describes the general use of information and communications technologies (ICT) for health and is inclusive of both mHealth and eHealth."* [5, p. VIII].

2.1.1 eHealth

The Swedish national board of health and welfare defines eHealth, translated from Swedish, as: *"using digital tools and exchanging information digitally to achieve and maintain health."* [1].

mHealth

In a report by WHO mHealth is described as: *"medical and public health practice supported by mobile devices, such as mobile phones, patient monitoring devices, personal digital assistants (PDAs), and other wireless devices."* [6, p. 6].

As for mHealth applications, they can refer to applications created for medical and health care services or for personal use [16]. Hereinafter, mHealth applications are referred to the former of these two.

2.2 County Council of Västerbotten

It lives about 260 000 people in 15 different municipalities in the county of Västerbotten in Sweden and its council is (among other things) in charge of providing health-, medical- and dental-care for them. Thereof, the council is principal of 39 health centers; three hospitals with emergency and specialized care; and 32 dental clinics [2].

2.2.1 Units for eHealth and IT

The county council of Västerbotten strives to expand HIT usage and has assigned an eHealth unit that has primary responsibility of achieving it. This unit consists of thirteen people

[1] `http://www.socialstyrelsen.se/nationellehalsa`, accessed 2017-06-26

[2] `http://www.e-magin.se/paper/5n28h32r/paper/#/paper/5n28h32r/2`, accessed 2017-04-12

that aims to strategically lead and guide the development of HIT [3].

The county council also has an IT unit which is one of the largest IT employers in its county. It consists of about 120 people with responsibilities such as operation, management, service and development of HIT solutions. [4].

2.2.2 University Hospital of Umeå

The county council is the principal of the University Hospital of Umeå (common Swedish acronym, NUS) which is one of the largest university hospital in Sweden. NUS was also elected Sweden's top university hospital in 2016 by the Swedish news magazine *Dagens medicin* [5]; a magazine that present themselves as being independent and Sweden's leading news magazine for the health care sector [6].

NUS has a female clinic contributing with the only highly specialized care within its county and the northern region of Sweden. It provides health care for females regarding gynecology and obstetric and conducts research in cooperation with Umeå University [7]. The director and professors at this clinic have for several years sought a solution to simplify the investigation of PMS.

2.3 Different types of mobile applications

When developing an application targeting mobiles, three common alternatives have traditionally been to build native, hybrid or mobile web applications; However, progressive web apps (PWAs), introduced by Google in 2015, can be considered a forth alternative to these [15]. Based upon [13, 15], the following is brief description of these:

A native application is generally coded in a device specific programming language and integrated development environment (IDE). For instance, a native iOS application is usually coded in *Swift* or *Objective-c* in the IDE XCode. While native Android application is usually coded in *Java* with the IDE Android Studio. These applications are generally installed through app stores on mobile phones and have rich access to device harware through platform specific APIs.

A hybrid application denotes an application built with web-based technologies but that can with the help of hybrid development frameworks (e.g., Apaches Cordova [8]), appear and act as a native application. This is achieved by wrapping the web-based code in a native layer/container and utilize a generic JavaScript API to access most of the native APIs. This approach enables only one code base for several platforms, although, some platform specific considerations might be needed and the JavaScript bridge for accessing platform APIs yield some performance overhead.

[3] `https://www.vll.se/Startsida/forskning-och-utveckling/e-halsa/enheten-for-e-halsa`, accessed 2017-05-01

[4] `https://www.vll.se/Startsida/jobb-och-utbildning/vara-arbetsplatser/service/informatikenheten`, accessed 2017-05-01

[5] `https://www.dagensmedicin.se/basta-sjukhusets-databas/sjukhus/norrlands-universitetssjukhus/`, accessed 2017-08-23

[6] `https://www.paperton.com/shelf/index/open_shelf/open_shelf/whitelabel/dagens-medicin`, accessed 2017-08-23

[7] `https://www.vll.se/startsida/om-landstinget/organisation-och-verksamheter/sjukhusvard/il/kvinnoklinik-umea`, accessed 2017-05-01

[8] `https://cordova.apache.org/`, accessed 2017-06-20

A mobile web application is built with web technologies, but instead of being installed on mobile devices they are hosted and served from remote web servers and displayed with web browser apps installed on these devices. Since these in general are coded with standard languages (such as with HTML, CSS and JavaScript) for browsers, a single code base can work cross-platform. With HTML5 APIs there is some support of accessing mobile hardware, e.g., camera and geolocation, but it is not at the same level as for native or hybrid applications.

A progressive web application is essentially a concept of how to build a mobile web app that is progressively enhanced with modern web technologies. PWAs are initially served from a remote web server similar to mobile web apps, but can when visited through a browser app be installed on devices as well. Another core feature they have is that they can be used regardless of network availability with the help of *service workers* that, among other things, enables caching and preloading resources. PWAs aim to bridge the gap in user experience between web and native/hybrid applications. PWAs are described more in the chapter *Progressive Web Applications* 3 beneath.

Chapter 3

Progressive web applications

This chapter describes some important aspects and technologies used with PWAs.

3.1 Characteristics

According to Google Developers, PWAs have these characteristics (quoted from [1]):

Progressive - Works for every user, regardless of browser choice because it's built with progressive enhancement as a core tenet.

Responsive - Fits any form factor: desktop, mobile, tablet, or whatever is next.

Connectivity independent - Enhanced with service workers to work offline or on low-quality networks.

App-like - Feels like an app, because the app shell model separates the application functionality from application content.

Fresh - Always up-to-date thanks to the service worker update process.

Safe - Served via HTTPS to prevent snooping and to ensure content hasn't been tampered with.

Discoverable - Is identifiable as an "application" thanks to W3C manifest and service worker registration scope, allowing search engines to find it.

Re-engageable - Makes re-engagement easy through features like push notifications.

Installable - Allows users to add apps they find most useful to their home screen without the hassle of an app store.

Linkable - Easily share the application via URL, does not require complex installation.

[1] `https://developers.google.com/web/fundamentals/getting-started/codelabs/your-first-pwapp/`, accessed 2017-06-27

3.2 Requirements

A PWA has three requirements that need to be fulfilled [13], namely that it need to:

1. be served over HTTPS,
2. have a web app manifest,
3. have at least one service worker.

Note that depending on browser support applications might not always be able to use the web app manifest or the service worker (see section 3.2.4 for a summary of current browser compatibility); However, since a PWA should *progressively enhance* the application with these technologies, the applications should still function even though a browser lacks support of them.

3.2.1 HTTPS

Hypertext Transfer Protocol Secure [2] (HTTPS) denotes a protocol where http pages are sent with an encrypting transport layer security (TLS), or formerly secure socket layer (SSL), protocol. This can improve the security of transferring web pages over a computer network (e.g., the Internet) by preventing man-in-the-middle attacks. Man-in-the-middle attacks can be achieved by someone spoofing their identity to act as an intended receiver between two communication parties. With TLS/SSL, this can be hampered by encrypting data and verifying the server identity against a third party, a certificate authority (CA). The CA receives identity information along with a public key from a server and checks this against a private key stored on its own server. If they match up, they digitally signs that they approve the identity of the server to the client.

3.2.2 Web app manifest

A web app manifest [3] is a JSON-based document for specifying metadata about a web app. It is (as of July 27th, 2017) at a working draft state by the world wide web consortium (W3C). With manifest files developers can, e.g., specify if a web app should be opened in fullscreen mode, have a custom color at the address bar and set which icons it should use when saved on devices' home screens.

3.2.3 Service worker

Service workers (SWs) can be seen as client-side proxies between a web page, browser and, if available, a network [4]. They essentially consist of JavaScript files with event-driven scripts that, due to a SW being a type of web worker [5], can be executed in the background of a web page; where in the background refers to on another thread than the main browser thread [6]. The service worker cannot directly read or manipulate the document object model (DOM) or directly call scripts imported in the page it is registered for. Instead a typical usage for it is to listen for, or sometimes rather hijack, events triggers from the page it is registered on,

[2] `https://tools.ietf.org/html/rfc2818`, accessed 2017-07-16
[3] `https://www.w3.org/TR/appmanifest/`, accessed 2017-07-16
[4] `https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API`, accessed 2017-07-18
[5] `https://www.w3schools.com/html/html5_webworkers.asp`, accessed 2017-07-18
[6] `https://developers.google.com/web/ilt/pwa/introduction-to-service-worker`, accessed 2017-07-18

i.e. from their client. In relation to this the service worker can do some predefined actions, such as e.g. caching and preloading resources and returning custom responses to requests. This service worker functionality is tightly coupled with JavaScript Promises (explained in sub section 4.4.3) to accomplish it asynchronously.

The events that a service worker (to date) can listen for are [7] [8]: install, activate, message, fetch, sync, push.

The install event occurs when a service worker is registered on a page and is typically where the core static content of an application is cached.

The activate event occurs after an install event, however only if an old version of a service worker is not currently used. If a prior version is used, the active event will be postponed until it is closed. With this event the service worker can, e.g., clean up old unused caches from an earlier service worker installation.

The message event can be used when a client wants to send a message to the service worker, which in turn can respond back with a message [9]

The fetch event is a functional event that occurs when a client requests a resource that the service worker is configured to control. This event is typically listened for to cache the resources from run-time *fetch*ed network requests.

The sync event is another functional event that helps make it possible to postpone actions until a stable network connection exists. Along with the background sync API this event can, e.g., be used to enable a user to post a network request without a network connection, close the page where the request was made, and then have it sent later when a good network connection is established again. [10].

The push event is also a functional event but, unlike fetch and sync, it is fired from a server instead of a client. It can trigger a listener at the service worker although its client is not in the foreground or even loaded at a user device [11].

3.2.4 Browser support

This subsection presents some of the, to date, browser support of the PWA requirements based on information from [12] and [13].

HTTPS support

All common browsers as of around 2012-2013 or later support the TLS protocol used for HTTPS. SSL has been deprecated, replaced by TLS, due to security benefits.

[7] `https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers`, accessed 2017-07-14

[8] `https://developers.google.com/web/fundamentals/getting-started/primers/service-workers`, accessed 2017-07-18

[9] `https://developer.mozilla.org/en-US/docs/Web/API/Client/postMessage`, accessed 2017-08-12

[10] `https://developers.google.com/web/updates/2015/12/background-sync`, accessed 2017-08-12

[11] `https://developer.mozilla.org/en-US/docs/Web/API/Push_API`, accessed 2017-08-12

[12] `https://jakearchibald.github.io/isserviceworkerready/`, accessed 2017-07-28

[13] `https://caniuse.com/`, accessed 2017-07-20

Web app manifest

Web app manifests are currently compatible with these browsers:

- Chrome: as of version 38 (October 2014)
- Opera: as of version 32 (September 2015)
- Samsung internet: as of version 4 (April 2016)

For Edge and Firefox browsers, supporting web app manifest is announced to be under development.

Service Worker Support

The following list are a compilation of which browsers that have partial or full support for service workers; However, on iOS devices there is currently no support of service workers regardless of browser. Mobile and desktop browser variants are combined as one and presented with the earliest support specified for any of them.

Service worker

Chrome: as of version 40 (January 2015)

Opera: as of version 27 (January 2015)

Firefox: as of version 44 (January 2016)

Samsung internet: as of version 4 (April 2016)

Android browser: as of version 56 (February 2017)

For Safari (which uses the WebKit browser engine) and Edge browsers, support for service workers is announced as under development.

Chapter 4

Theoretical framework

This chapter describes different literature used to conduct this thesis work. Firstly it explains the double diamond design process used during this work. Following this are sections for design, legal, implementation and evaluation aspects.

4.1 The double diamond design process

The double diamond (DD) model (see figure 4.1 [17]) derives from observing and combining several companies design processes and it consist of four phases [17]:

Discover
: During this first phase designers gather insights and explore different aspects of the challenge in a fresh way.

Define
: The second phase represents the process of taking all the findings from the discover phase and determining which are most valuable, important and feasible. The main goal is to frame the fundamental design challenge.

Develop
: The third phase denotes the period of development where solutions/concepts are created, prototyped, tested and iterated. This objective of this is to help improve the designers ideas.

Deliver
: In the forth and final phase the resulting product/service is finalized, produced and launched.

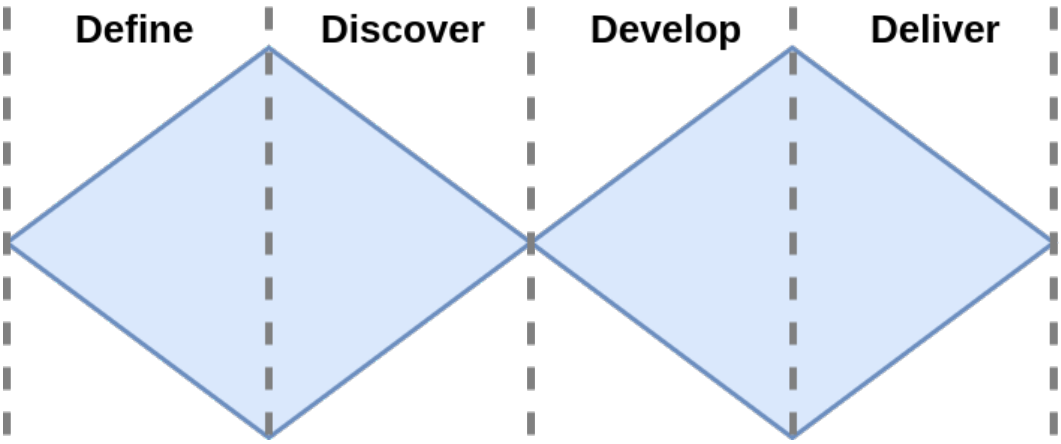


Figure 4.1: An illustration of the double diamond design model

4.2 Design aspects

This section describes design models, concepts, patterns and guidelines used during the design process of this work.

4.2.1 Graphical profile for VLL

The county council of Västerbotten has a graphical manual [1] for how to visually present themselves. Some of it contains information about padding and placement of their logo; that their primary font is *Foundry Sans*; secondary font (for long texts or when primary is not available) is *Adobe Garamond* and that they have three profile colors and five complementary colors (see these in figure 4.2).

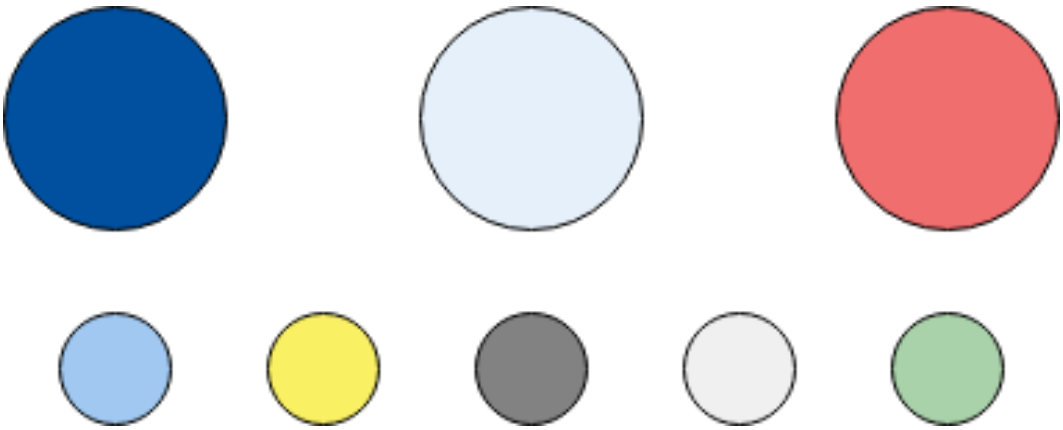


Figure 4.2: The colors and corresponding color codes specified in the graphical manual for VLL

[1] `https://www.vll.se/VLL/Filer/VLL_Grafisk_manual_juni_2017.pdf`, accessed 2017-06-03

4.2.2 Design patterns

This section contains design patterns and guidelines used to develop the application.

4.2.3 App shell model

The application/app shell [2] is an architectural model suitable for PWAs as they can enable quickly showing something to the users when an application starts. This is enabled by having a static base layout (the shell) of an interface displayed initially when an app starts. Then inside this shell, actual content can be dynamically added as soon as it has loaded. This model can be used to mimic the appearance of a native application, where the app shell typically contain things like an app header and a menu.

Wizard

The wizard pattern [18] is an information architecture pattern used to lead users, step by step, through a sequence of tasks. These are appropriate to use when the designer of the UI knows more than the user about how to complete the task successfully. This pattern can however be frustrating for a user if the task flow is not as the user wants or thinks it should be.

Hub and spoke

A common mobile device navigation pattern is the *hub and spoke* [18]. It is used when one page, typically the home screen, is the hub to navigate to other pages from (see figure 4.3 [18]). With this pattern the user can select a page to go to at the hub. When finished at that page the user comes back to the hub where another page to go to can be selected.

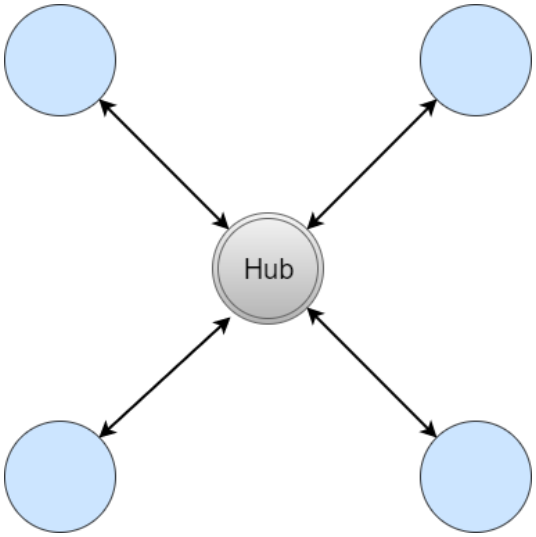


Figure 4.3: A graphical representation of the *hub and spoke* design pattern

[2] `https://developers.google.com/web/fundamentals/architecture/app-shell`, accessed 2017-08-09

Diagonal balance

Diagonal balance [18] is a page layout pattern for structuring asymmetric content with good balance/harmony. It for example mediates how pages with headers located at the top left corner of a screen can be balanced by placing an important component (e.g., a button) at the lower right location of the page.

Vertical stack

Vertical stack [18] is a useful design pattern for mobile web pages that should support different screen sizes. With this pattern, content is stacked, and if needed scrolled, vertically. Placing content vertically makes it easier to account for different screen widths. If form labels are used with the pattern, they should be placed above their inputs. Buttons can be placed side-by-side if its assured they will not expand over the visual space on the screen. The most important content should be placed within the first top 100px of the screen.

Loading indicators

Loading indicators [18] is a pattern for informing the user that content is being loaded. It implies that some kind of progress indicator should be showed either where the content is to be placed once loaded or where the user pressed to retrieve the content.

Showing loading indicators is particularly useful when the users might have bad network connections [19].

Hairlines

Hairlines [18] is a know as an aesthetic design pattern. These are one-pixel-wide lines that can be used to separate different content areas.

Streamlined Branding

The streamline branding pattern [18] denotes putting the organization's logo, colors and similar brand elements in a mobile app as it is beneficial if users can identify the source of the application. However, it also claims brand images or similar should have small sizes since mobile networks can be slow.

4.2.4 Guidelines for filling in forms

Based upon the work of Schneiderman and Plaisant [20, p.297], some guidelines for how to design online forms are that they should have:

- Meaningful titles,
- Comprehensible instructions,
- Logical grouping and sequencing of fields,
- Visually appealing layout of the form,
- Familiar field labels,
- Consistent terminology and abbreviations,
- Visible space and boundaries for data-entry fields,

- Convenient cursor movement,
- Error correction for individual characters and entire fields,
- Error prevention where possible,
- Error messages for unacceptable values,
- Marking of optional fields,
- Explanatory messages for fields,
- Completion signal to support user control.

4.3 Legal aspects

In this section some legal aspects related to mHealth applications are presented. As mentioned in the restrictions section 1.3 it is not an exhaustive review of these regulations and others might also exist that applies to mHealth applications.

4.3.1 General Data Protection Regulation

The EU general data protection regulation (GDPR) enters into force in 25th of may 2018 and is a regulation that aims to enforce better protection of personal data, with more responsibilities put on the organizations storing and processing it. This regulation generally applies to all kinds of public applications that handle personal data for individuals within EU. Some relevant points from it are that [3] [4]:

An explicit consent must be obtained in order to collect and process personal data, i.e., it can not only be assumed.

Sufficient safety measures are needed to verify that personal data is stored securely, e.g., so it is protected from destruction, loss or unauthorized access.

Documenting personal data treatment procedures is required so that it can be audited and verified in the sense that appropriate procedures have (not) been used.

Transparency in information usage , i.e., the individual has to be fairly informed of how its personal data is being used.

Rights to access and manage personal data. Individuals should, e.g., have the right to: get a free copy of their data; be able to rectify and erase their data; be allowed to restrict and object to handling/processing of their data.

[Supporting data portability] of personal data, i.e., the individual should in a safe and secure way be able to retrieve their personal data so it can be reused in other services.

[3] `http://www.datainspektionen.se/dataskyddsreformen/dataskyddsforordningen/dataskyddsdagen/`, accessed 2017-07-14

[4] `https://ico.org.uk/for-organisations/data-protection-reform/overview-of-the-gdpr/`, accessed 2017-07-14

Implementing ways to detect data breaches are required to verify the integrity of personal data. If breaches occur that may jeopardize the integrity of personal data: relevant supervisory authorities generally have to be notified within 72 hours of awareness.

A data protection officer responsible of assuring that personal data is handled correctly is needed for organizations and authorities that treats *sensitive* personal data or data that map individuals behaviour.

4.3.2 Health and medical service act

The Swedish health and medical service act contains laws for how such services should be performed. According to it, in order to fulfill good health care, health care should for example [5]:

- **Accommodate patients' need for safety and security**,
- **Be easy accessed**,
- **Build upon respect for the patients autonomy and integrity**,
- **Promote good contact between the patient and caregivers**.

4.3.3 Patient data act

The Swedish patient data act (PDA) contains regulations for how patient data should be treated within health and medical care [6]. Two of its regulations regards:

Inner confidentiality , i.e., that only the ones treating a patient should be allowed access to the patient's data. This can be addressed with restrictions and control over who gets access. Restrictions may be achieved through limiting access to the data, and control may be achieved through supervision by the means of logging who accesses data.

Enabling patients to review logs about who has accessed their data. This regulation also enforces that caregivers should be able to provide the patient with *direct* access, e.g. through the internet, to these logs.

4.3.4 CE marking

According to the Swedish Medical Product Agency [21], mobile applications used to help treat or diagnose patients in a medical or health care context are considered to be medical technical applications and therefor needs to be CE marked. These applications *usually* fit the criteria of needing a class 1 CE marking, which generally implies that a report about, e.g., the applications purpose, usage and assurance of safety need to be sent to the Agency where it is audited with current regulations, before it is released on the market.

[5] `https://patientsakerhet.socialstyrelsen.se/om-patientsakerhet/centrala-lagar-och-foreskrifter/halso-och-sjukvardslagen`, accessed 2017-08-09

[6] `http://www.datainspektionen.se/lagar-och-regler/patientdatalagen/`, accessed 2017-06-20

4.4 Implementation aspects

This section contains information about some frameworks and technologies used to implement the PWA.

4.4.1 Web components

Web components consist of four technologies (i.e., *Custom Elements*, *Shadow DOM*, *HTML imports* and *HTML Template*) that can enable user interface components to more easily be reused [7]. This is basically achieved by encapsulating code into a portable component (custom element), that can be used following an import statement of it to an HTML page [8]. Not all browsers support the technologies web components consist of and *Polyfills* for web components [9] can be to fill in the capabilities these browsers lack. With these, web components are supported on most common modern browsers, such as Firefox, Edge, IE, Safari, Opera.

4.4.2 Polymer Project 2.0

The Polymer Project 2.0, or simply Polymer 2, is essentially a JavaScript library of web components [10] developed by Google. However, it might also be described as a framework for developing PWAs and single page applications (SPAs) as it includes an app toolbox [11] with components, tools and templates to help develop these. It also has a command line interface tool, *polymer-cli*, which includes a *"build pipeline, a boilerplate generator for creating elements and apps, a linter, a development server, and a test runner"* [12].

4.4.3 Promises in JavaScript

In JavaScript, *Promises* are objects that represents the eventual success or failure of asynchronous (async) operations [13]. A promise object can use *then* and *catch* handlers for handling what to do when async methods in a promise succeeds respectively fails. So for instance, with a promise object called *foo()*, appending foo().then(*bar()*) will execute the *bar*-function as soon as the async methods in foo has completed. With a promise, what is returned from a prior async operation can be accessed as an argument in the next. The returned object is itself also a promise so it supports chaining series of then/catch handlers. If an error occurs, it can be handled by appending a *.catch()* to the promise chain.

4.5 RESTful web services

A RESTful web service [14] builds upon the REpresentationl Stateless Transfer (REST) architecture style to create services that work on the web, and they can be used to transfer resources between clients and servers. The resources are accessed with Uniform Resource

[7] `https://developer.mozilla.org/en-US/docs/Web/Web_Components`
[8] `https://www.webcomponents.org/introduction`, accessed 2017-07-17
[9] `https://www.webcomponents.org/polyfills`, accessed 2017-07-17
[10] `https://www.polymer-project.org/2.0/docs/devguide/feature-overview`, accessed 2017-08-14
[11] `https://www.polymer-project.org/2.0/toolbox/`, accessed 2017-08-14
[12] `https://www.polymer-project.org/2.0/docs/tools/polymer-cli`, accessed 2017-08-13
[13] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise`, accessed 2017-08-09
[14] `http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html`, accessed 2017-08-09

Identifiers (URIs) and manipulated using four (CRUD) operations: create, read, update and delete. For HTTP-based web services this would align with the standard HTTP methods: POST, GET, PUT, DELETE. JSON and XML formats can (for instance) be used to represent the resources. Metadata, e.g. authentication credentials, can be sent with these and for HTTP requests it can be placed in their headers.

4.5.1 JSON web tokens

A JSON web token (JWT) can be used to authenticate users over the internet [22], which consists of a header, payload and signature. In the header it is specified what kind of token it is (i.e., jwt) and what hashing algorithm it uses. In the payload it can typically be claims about who a user is (e.g., username). In the signature the information from the two others are hashed with a secret key originating from the server that generates the token. The secret key should be kept securely since as long as the secret key has not been compromised, the signature is a secure verification that the token originated from the server that signed it.

In a scenario of server-client JWT authentication: the client can receive a server generated token when it has sent valid user credentials (e.g., a username and a password) to the server. If the client want to use the token as authentication, it can be sent as *Authorization Bearer ¡token¿*, in the header of a HTTP request (given that the server it sends it to supports it) [15].

4.5.2 Pseudonymization

One way to protect a user if data storage has been compromised is to decouple data from person who owns it. This can be done with pseudonymization [16]. With this method, who the data belongs to is referenced using some kind of code key, and the mapping of a code key to an identifiable person is stored securely somewhere separate from the data. So in case someone that is unauthorized gets access to stored data they will not be able to identify who it belongs to unless they also manages to get access to the location where code keys are stored.

4.6 Evaluation aspects

4.6.1 Lighthouse

The Lighthouse tool [17] by Google can audit if a web app has the characteristics of a PWA. It automatically analyzes if baseline features for a PWA exists on a web page. To date, these PWA features are checked in the automated audit by Lighthouse:

- Registers a Service Worker
- User get prompted to Install the Web App
- Responds with a 200 when offline
- Contains some content when JavaScript is not available

[15] `https://jwt.io/introduction/`, accessed 2017-07-16

[16] `https://vardgivare.skane.se/siteassets/3.-kompetens-och-utveckling/utlamnande-av-patientdata-samrad-kvb/samrad-kvb-ordlista-2016-02-23.pdf`, accessed 2017-06-14

[17] `https://developers.google.com/web/tools/lighthouse/`, accessed 2017-07-19

- Uses HTTPS
- Redirect HTTP traffic to HTTPS
- Page load is fast enough on 3G
- Configured for a custom splash screen
- Address bar matches brand colors
- Has a ¡meta name="viewport"¿ tag with width or initial-scale
- Content is sized correctly for the viewport

Three characteristics for baseline PWA features are not automatically checked in the audit with Lighthouse and are advised to be checked manually, namely:

- Site works cross-browser
- Page transitions do not feel like they block on the network
- Each page has a URL

4.6.2 System usability scale

The system usability scale [23] (SUS) can be used to assess the usability of a system based upon answering ten statements with a five-point Likert scale. It is built upon the usability aspects: effectiveness, efficiency and satisfaction. The SUS score shows an indication of the usability of a system and it ranges from 0-100. The score 68 is thought of as the average score, so above 68 can be considered above average and below 68, below average for the usability of a system [18].

In order to determine the SUS score [23], the sum for each statements should be calculated following that a statements value is retrieved according to the following procedure: (i) for each odd ordered statement (i.e., the positive statements) the value to retrieve is the selected scale position subtracted with one; (ii) for each even ordered statements (i.e., the negative statements) the value to retrieve is five subtracted with the scale position. This sum is then multiplied with 2.5 (to make it range 0-100 instead of 0-40) to obtain the SUS score.

[18] `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`, accessed 2017-08-01

Chapter 5

Methodology

A design process following to the double diamond (DD) model was used during this thesis work. Methods used in each of its phases is presented in respective sections, i.e., the sections *Discover*, *Define*, *Develop*, *Deliver*. After these phases an evaluation of the final PWA was performed and methods for this is presented in a separate *evaluation* section.

5.1 Discover

During the discover phase we studied literature, conducted interviews and took part of a workshop to explore the context of mHealth and PWAs as well as how to define and develop the application.

5.1.1 Literature study

The literature study was generally achieved by searching for literature in both regular and academic search engines. The evaluation and design literature mostly consisted of articles and books. The progressive web application and implementation literature was mainly retrieved from documentation at developer pages by Google and Mozilla. For the legal aspect literature, the main search phrases were names of legal acts appended with "summary", "overview" or similar phrases.

5.1.2 Interviews

Several semi-structured interviews [1] were conducted to discover more about how to build the application. All these interviews were audio recorded, after an approval was given from the participants, to make it easier to focus on the interviewed rather than taking notes. These interviews were with:

A strategic expert from the eHealth unit and the areas aimed to discuss were regulations, security and hosting possibilities applicable in a health care context. The questions for this interview can be viewed in appendix .2.

[1] see e.g. `http://designresearchtechniques.com/casestudies/semi-structured-interviews/` for information about this method

Two professors at the female clinic with the intention of finding out what kind of solution they sought and to find requirements for the specification. This included question topics such as what the values, challenges, usage and expectations were for the solution. These questions are further discussed and presented in the subsection about system requirement specification 5.2.1. At the end of this meeting some sketches of lo-fi prototypes were also presented to gather their opinions on different design solutions.

Another strategic expert from the eHealth unit and during this interview authorization and data storage were the main topics. The aim was to find a solution for these that fulfilled the requirements of the application. Questions asked during this interview can be viewed in appendix .3.

A system developer from the IT Unit with the intention of gaining information about ways to implement the application and what development environment, programming languages and frameworks, the IT unit desired to make it manageable for them to support it. A few questions were predefined (see appendix .1) but most advice given about implementation were retrieved from follow up questions based on what the interviewed considered suitable and had knowledge about.

To gain further information from the clinic an unstructured interview with the director of the female clinic was also conducted. During it the progress that had been made was presented as well with a demonstration of the mid-fi prototype made in the develop phase (explained in the section 5.2 beneath). For this demonstration one of the professors from the prior interview was fetched to get both of their inputs about possible improvement.

5.1.3 Workshop with SKL

A workshop organized by the national organization for Swedish municipals and county councils (common Swedish acronym, SKL) was attended remotely. The workshop regarded finding solutions for integrating mobile applications with their national platform for health services. By accomplishing this, caregivers would, e.g., be able to use analyzing tools supported by this platform on data retrieved from custom mobile applications. During the workshop two participants also presented results from recently having conducted a pilot study of developing a native mobile application integrated with the national health service platform. The intended usage of this app was to gather information from patients through forms existing in the app.

The main purpose of attending this workshop was to untangle if it existed an easy solution for integrating the PWA with the national platform as server-side for data storage and authentication. We aimed to get an answer to this by simply listen in on what was presented and if not evident from these discussions, ask if any APIs existed to support it.

5.2 Define

During the define phase the insights and information gained during the discover phase were analyzed and contemplated in order to specify the requirements for the application.

5.2.1 System requirement specification

A system requirement specification (SRS) (influenced by [2] [3]) was produced based upon findings from the literature study, interviews and workshop in the discover phase. It was created to give a clearer picture of how the application should be like, since many alternatives arose in the discover phase. It contained what the prominent effect goals were for the application based upon what was said from interviewing professors from the female clinic. The aim was to specify the applications: data and navigation flow, choice of authentication, functionalities that shall or should be included, desired appearance, devices it should support, and what development platform that should be used. Some of the main questions asked during interviews to find this out can be seen in appendix .4; where these questions built upon examples questions from the WHO framework for monitoring and evaluating digital health interventions [5, p. 19 & 81].

5.3 Develop

During the develop phase we explored different alternatives to design the solution based on the findings from the earlier two phases.

5.3.1 Prototypes

Lo-fi

Lo-fi sketches were drawn on paper and white-board to explore design alternatives of the application, detect issues, and to generally explore what is needed for the system to function well. These sketches were also used to discuss solutions with different stakeholders. The lo-fi:s were focused on mobile devices and drawn with a realistic size, as that is helpful for discovering and solving ergonomic problems of vision and interaction [19]. The basis of them were to follow the wizard design pattern, guiding the user through series of tasks they needed to achieve. The sketches made of when answering the form also built upon having: a visually appealing layout, visual space and boundaries for data-entry fields, convenient cursor (hand) movement, and error messages; based upon the form fill-in guidelines presented in subsection 4.2.4.

Mid-fi mock-up

A mid-fi prototype was created with Axure RP 8 [4], and as a mock-up, it supported three different task flows which were based upon the three scenarios: *first time*, *forgot days*, *completed a cycle* (explained further in subsection 5.3.2 below). These scenarios were based upon supporting requirements following the results of the prior define phase.

For each scenario, the following describes the tasks that views were created for in the prototype:

First-time : logging in, reading information about the app, answering for a self assessment, seeing that they were done for the day.

[2] `https://intra.kth.se/administration/upphandling/upphandlingar-over-direktupphandlingsgransen/att-skriva-kravspecifikation-1.533512`, accessed 2017-05-16

[3] `https://www.ida.liu.se/~TDDI02/owl/diverse/exempel-kravspec.pdf`, accessed 2017-05-16

[4] visit `https://www.axure.com/` (accessed 2017-08-29) for information about it

Forgot days : logging in, selecting which forgotten day to answer for, answering with a prefilled self assessment, repeat the two prior tasks until all days are answered for, seeing that they were done for the day.

Completed a cycle : logging in, answering with a prefilled self assessment, selecting to view results, looking at the results.

5.3.2 Usability testing the mid-fi mock-up

Six females were recruited to usability test the mid-fi mock-up. The think-aloud method [5] and audio recording were used during these tests. The test procedure followed that each participant should complete three different series of tasks based upon the three scenarios:

First time visiting the application after been given a recommendation and user credentials from caregivers at the female clinic.

Forgot days to answer after they have done self assessments during a couple of weeks.

Completed a cycle and that they then were expecting to see the results of the self assessments they had made.

Before participants conducted the test they were given a document (see original Swedish version in appendix .5). This document included:

- the background of the application;
- what the application was intended for;
- a clarification that the application is tested and not them;
- a notice that audio recording would be used during the test if they approve of it;
- a statement encouraging them to think-aloud during the test;
- an explanation that they at any time could end or pause the test without needing to explain why;
- and finally, a description of the three scenarios that were going to be tested.

5.4 Deliver

This section describes the methods and procedure used to implement the PWA. but, firstly a brief explanation of the architecture for the the whole system is presented.

[5] see e.g. `https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/` for an explanation of it

5.4.1 Abstract system overview

The basic idea of how to build the solution was with a system architecture consisting of: a PWA as client app, a database for storing users and assessments, and a web service to communicate between the client and the database. A cloud hosting service would be used to distribute it over the internet: including one database server, and one HTTP based web server to host and serve the PWA and the web service. The web service would have a web (REST) API for communication with the PWA. A visual representation of this system can be viewed in figure 5.1. However, if the developed PWA is "installed", i.e. cached locally, on a device: pages would then no longer be served from the web server (unless when they have been changed). Hence, the PWA could be seen as being served locally from the user device instead.

This architecture was chosen as it, e.g., would rather easily support adding a separate admin application for caregivers in the future. With access to the database achieved by implementing an additional API in the web service.

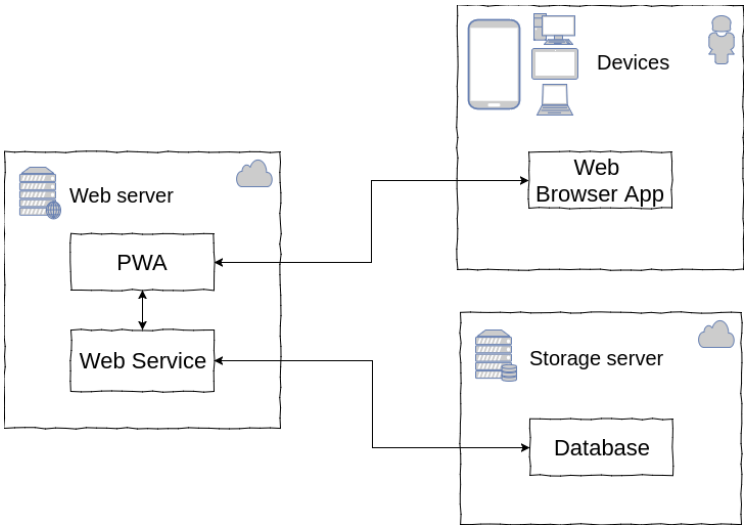


Figure 5.1: An illustration of the system architecture

5.4.2 Implementing the PWA

The PWA was developed on the operating system Windows 10 with the IDE Microsoft Visual Studio 2017 Community Edition. The programming languages used to develop it were HTML5, CSS3 and JavaScript. The Polymer Project 2.0 web component library was used to help create a single page application (SPA) and get an app layout and appearance. The initial structure of the project was generated with the *Polymer Starter Kit* through the command line interface tool, polymer-cli, from the Polymer app toolbox. The polymer-cli tool was also used to serve the application during development and to build it before deployment/publishing the app. The tool sw-precache [6] was used to enable caching features for the service worker. Features of minimizing HTML, CSS and JavaScript files, as well as generating a service worker using the sw-precache tool was accomplished by configuring the

[6] `https://github.com/GoogleChrome/sw-precache`, accessed 2017-07-14

polymer-cli build process. *Chrome DevTools* was used to debug and display the PWA on different form factors during development.

The procedure of implementing the PWA started with creating the design and layout of pages following what was defined from the earlier design process phases. After these had been created, functionality for authenticating a user with JWT tokens against the web service was implemented. Following this, functions for retrieving data from the web service were implemented (using the fetch API as the service worker would listen for fetch events later). Then the service worker functionality was added and finally the web app manifest configured.

5.5 Evaluation

The evaluation of the PWA consisted of two methods, an audit of the PWA and user tests followed by a survey that the test participants to answered.

5.5.1 Lighthouse Audit

The lighthouse audit was performed by installing the tool as an extension in the Chrome browser. The PWA, hosted online, was then visited and through an icon next to the address bar, the lighthouse tools menu was accessed were it could be selected to generate a report. After that, the lighthouse tool analyzed the web page and a report of the results was returned.

5.5.2 User testing

Eleven test participants were recruited by publicly posting a request for test users on the Facebook page for the *Swedish association for PMS*. The post basically stated that: test users were sought for a web app intended to be used to gather self assessments from patients investigating if they suffer from PMS. Furthermore, explaining that the main task during the user tests was to fill in self assessments during five consecutive evening; and that on the fifth and last day they were also going to end the current test period, view the results and lastly answer an online survey to evaluate the application. It stated that people could "like" or send a private message (PM) if they wanted to participate and that they then would receive test user credentials through a private message on Facebook. It was also explained that the application was created in collaboration with the county council of Västerbotten and the female clinic at the University Hospital in Umeå.

To setup for the tests, 40 test users were generated which had the usernames "testX", where X was a number between 1-40, and passwords consisting of six random characters. The system including the PWA was hosted online with Microsoft's Azure services so that test users could access it. At the evening on the fifth and last day of the test period: the test users were given a link to the online survey in a PM on Facebook and was also reminded to end the test period and view the results before answering the survey.

Survey

The online survey that the test users answered was created with the help of Google Forms [7]. Its questions were divided into three pages. The first contained questions about their age; what kind of device (e.g., mobile, computer), operating system (e.g., iOS, Android), and

[7] visit `https://www.google.com/forms/about/` (accessed 2017-08-29) for information about it

browser (e.g., Chrome, Safari) they primarily used to conduct the user tests; and if they added the application as an icon on their home screen.

The second page contained a system usability scale (SUS) test translated into Swedish. In the sub header of this page, the users were told to: select the answers without thinking about it to much, and that they could mark the centre point scale (three) if they felt they could not answer for that statement. The total average SUS score was calculated by combining answer values individually for each statement, following the default procedure described in subsection 5.2.1, and then divide this with the amount of answers for each statement.

The third page contained five questions that were optional to answer. These were mainly given so that user was given an opportunity to ventilate opinions if they wanted to. The questions (translated from Swedish) were:

1. Did you experience something missing from the application, if so then what?
2. Did you experience something particularly positive with the application, if so then what?
3. How did you think it felt to use the application, especially regarding the start of it?
4. Did you think the application felt quick or slow? Briefly motivate why you think so.
5. Do you consider this application to fit as web or mobile app? Briefly motivate why you think so.

Chapter 6

Results

This chapter describes the results of the methods used to design, implement and evaluate the mHealth PWA.

6.1 Discover

Other than whats presented beneath, the discover phase also resulted in what is presented in the chapters *Progressive web applications* 3 and *Theoretical framework* 4.

6.1.1 Interviews at Female clinic/NUS

Two interviews were conducted with people from the female clinic, the clients seeking the application. These mostly regards the kind of application they wanted.

Professors from the Female clinic - Marie Bixo and Thomas Bäckström

The objective of this interview was to find out the needs and wants they had for the application. They claimed that decreasing the workload for caregivers was the main effect they sought, secondly they wanted to make it easier for the patients to do the self assessments. They said they desired an application that was self-explanatory, presented the 16 negative symptoms the application should have as questions, and specified that all questions should be answered with six point (zero to five) Likert scale. Required that at least 2 menstrual cycles (with it being 20-40 days per cycle) should be possible to fill in. They claimed that patients often needed to see the results themselves in order to believe it, so they desired that the results were visualized in the application, in the patients' mobile phones; However, that these results should not be visible for the patients before a test was completed because of prior answers possible influence on subsequent answers. They preferably saw that several symptoms would be combined in one chart, but that one symptom per chart was acceptable. They wanted to be able to export data into an excel file format for research purposes. Furthermore, after being told (by the interviewer) about the possibility to establish push notification support on the web, they said that it would be very useful and a desirable feature to include. They also thought it would be beneficial if the user could select when to get a reminder, but restrict them to being at the evening.

Director of the Female Clinic - Eva Innala

The director was the intended interviewer but the professor Marie Bixo was also fetched to take part during demonstration of the mid-fi prototype that had been created before this interview.

During this interview Eva told that they for several years tried to improve this procedure of investigating PMS, e.g. with voice based phone surveys, but that it never took of due to financial constraints and economic disadvantages.

One improvement they saw for the prototype was to make it more clear that at least two menstrual cycles had to be completed to give caregivers enough data to make a reliable assessment. They also claimed they desired to see several test periods/menstrual cycles in one chart. With scrolling functionality to navigate between them. Furthermore the horizontal axis was said to aid the caregiver better if it was a digit in the range *1,2,...,n* where n is the end of test period, instead of using dates.

They were also told that a participant in usability testing of the mid-fi had a hard time understanding the question/symptom "uncontrolled", which resulted in it being rephrased to "a feeling of loosing control".

6.1.2 Interviews at the county council of Västerbotten

Three interviews were conducted with employees from the county council of Västerbotten.

eHealth strategist - Göte Lindahl

At this interview some general information about laws and security of mHealth applications were discussed. The interviewed told about the need to CE mark mobile applications used within medical care. He also informed about the EU applied general data protection regulation (GDPR) that enter into force on the 25th of may, 2018. How it specifies that personal data need to be retrievable and removable for the person it belongs to. He also briefly discussed the patient data and security acts in Sweden. For instance that the access to patient data need PDA-logged unauthorized not eligible caregivers access it. They generally identify who access personal data by logging a HSA-ID from employers' SITHS-cards.

He also mentioned issues regarding storing information on servers outside of EU. He says that for data stored outside of EU, governments might have the right to access data belonging to someone else, but that this is prohibited inside of EU. Hence they see to it that patient data is stored inside of EU.

Lastly he explain how the data inspection unit oversees that data are stored in a secure and proper manner. A representative from the data inspection unit also verifies that personal details such as social numbers are handled correctly.

System developer - Alexej Timonin

In this interview it was said that the IT unit desired any server-side part to be developed with *.NET Core* to make it easier for them to administer, manage and extract information from it. For the client (PWA) application it did not matter which frameworks that were used. He also stated that they mainly have windows environment servers so the system needed to be supported in windows in order for them to be able to host the application in the future. He also suggested using Swagger to document the API at the web service, use JWT tokens and building an authority server for authentication against the web service.

However, if there would have been a budget for it, he strongly advised using *Bank ID* e-identification services for authentication instead.

eHealth strategist - Thomas Molén

The main finding from this interview was three suggested solutions for authentication and data storage for the application, namely:

1. Store data locally on devices and let patients send or bring this result to their caregiver when they deem appropriate.
2. Store data on a separate cloud or VLL hosted server and have users authenticate to it with a username and password.
3. Integrate the application with a national platform for health care systems and utilize its storage and e-identification services (i.e., in Swedish, *nationella stöd- och behandlingsplattformen* and *Bank ID*).

The third possible solution, integrating it to a national platform, led to partaking in a workshop regarding it and the findings from this are presented in the subsection 6.1.3 below.

6.1.3 Workshop with SKL

During the workshop people from SKL claimed they currently had no support of easily connecting external applications to their national platform for health services. This was said to depend on the fact that their current (SOAP based) API was rather complicated and time consuming to understand, with rather messy documentation of it. From the pilot study, where a native app was integrated with the national platform, developers concluded that considerable extra guidance from those who built this API was needed in order to use it. Other relevant conclusions from the presentation of the pilot study were that:

- push notifications or reminders were considered important as many patients forgot to use the application (as the app had no reminder-features);
- it was hard to get patients to use/install the application, some patients claimed they were going to use the application but in practice did not;
- since not everybody have a smartphone and since those who do might not have enough space to add applications: they claimed mobile applications might not enforce equal care if they replace current wider supported solutions;
- they stated that the more features their current web interface for the national platform can satisfy, the more equal care would be.
- developers of the app stated: they and probably most app developers would like to have a REST, instead of SOAP, based API to make integration easier.

6.2 Define

This section contains the results of defining the application based on the results from the discover phase.

6.2.1 System requirement specification

The following is some of relevant key aspects coming from the system requirement specification:

The three main effects the clients (the female clinic) sought for the project were:

1. reduced workload for caregivers,
2. simplified process for conducting self assessments for patients,
3. improved possibilities to research PMS by allowing digital access to patient data.

Some requirements were that the application should:

- be intuitive/self instructing,
- support filling in answers for (at least) three menstrual cycles,
- show results on the patients phone after a test period been finished,
- contain 16 shortly phrased symptoms/questions (specified by the female clinic),
- symptoms/questions should be answered with a six point Likert scale,
- support that a female clinic researcher could download all users results to excel files,
- preferably remind/notify the user when they should answer,
- appearance should signal that county councils of Västerbotten is the owner,
- use pseudonymization for users (i.e., have unidentifiable user credentials),
- store data through a web service built with .NET Core,
- use JWT tokens as authentication against the web service.

The goal of the project was to create a PWA to investigate if, and how well, it can achieve the requirements and effects.

6.3 Develop

This section presents the results of designing the PWA by the means of prototyping and testing.

6.3.1 Lo-fi

Beneath in figure 6.1 are a collection of some early lo-fi prototypes created, they were used as aid to find suitable design alternatives and to fuel the discussion with different stakeholders.

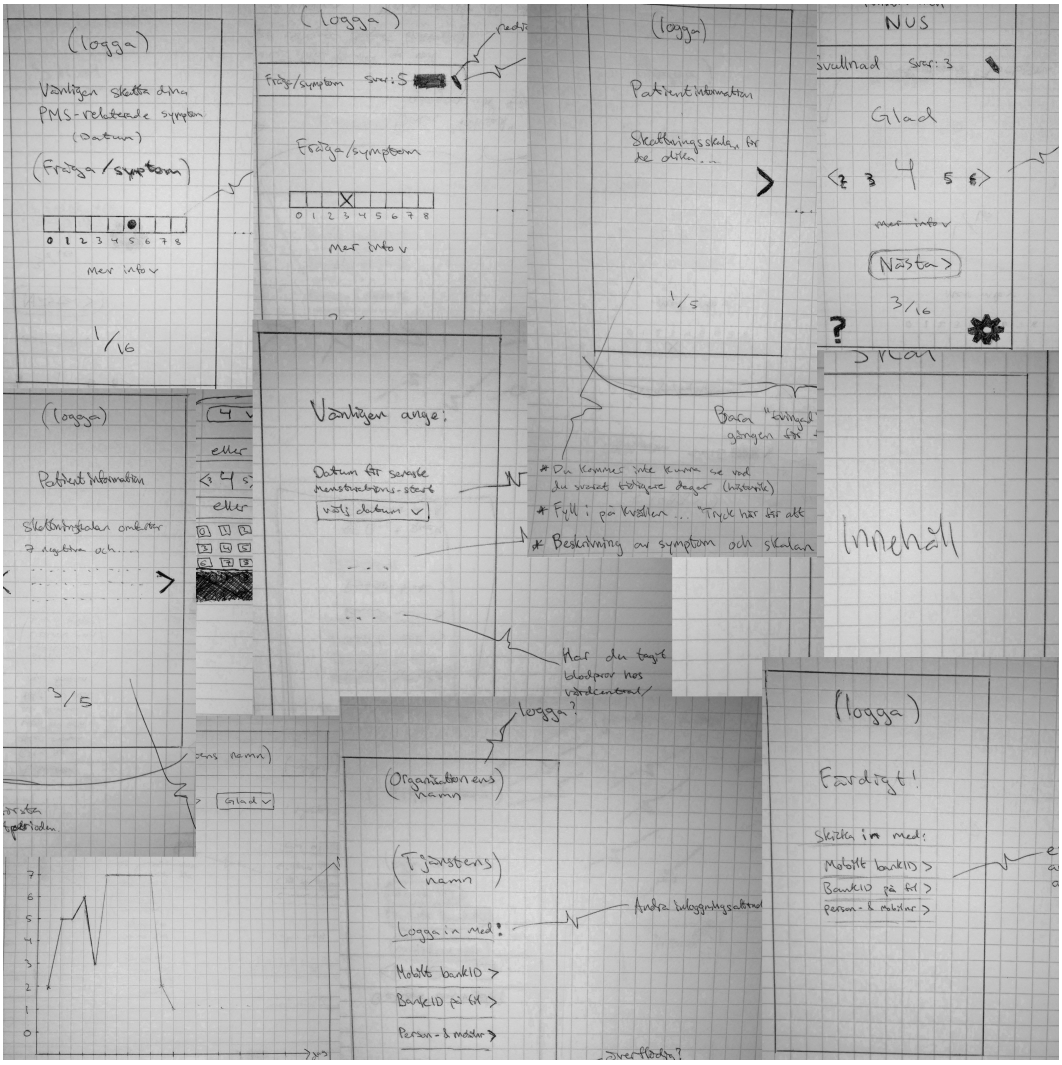


Figure 6.1: A collection of some lo-fi sketches made for the application

6.3.2 Mid-fi Mock-up

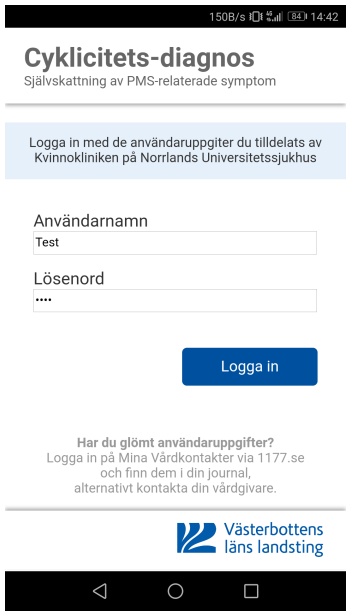
The mid-fi mock-up prototype consisted of supporting the three scenarios that the user tests consisted of. Its general navigation was based on the wizard pattern of guiding the user through each task it had to achieve. Firstly the user had to login (see figure 6.2a). After that the prototype simulated that it checked what series of tasks the user had (not) completed and, based on this, redirected the user to the page for the first task of these that user needed to fulfill. The prototype supported three of these scenarios, or series of tasks, namely that after logging in:

For the first time visiting scenario the user was shown three introduction pages (see figure 6.2b for the second of these). Then the user got to the page where they could fill in self assessments (see figure 6.2c). After sending in the answers (see figure 6.2d)

the user was shown a done page where they could select if they wanted to add push notifications and/or an icon of the application on their home screen (see figure 6.2e).

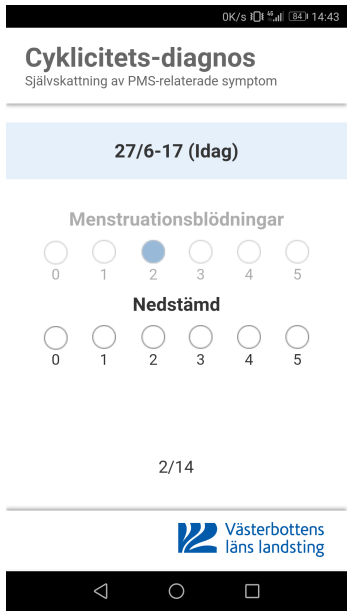
For the forgot to answer scenario the users got to a menu where they could select which of the unanswered days to answer (see figure 6.2f). After selecting one of these they came to a prefilled answering page (looking as figure 6.2d but with date adjusted in the header). Then they iterated this until all answers where finished and they got to the done page (see figure 6.2e).

For the show results scenario the users answered for a day and afterwards it was then simulated as if the prototype automatically detected that a menstrual cycle has been completed and the results should be showed, so they got to the results page (see figure 6.2g). At this page, they could only select the view results option ("Visa" in Swedish), when they did they were shown a simplified version of how the resulting graph should look like (see figure 6.2h).

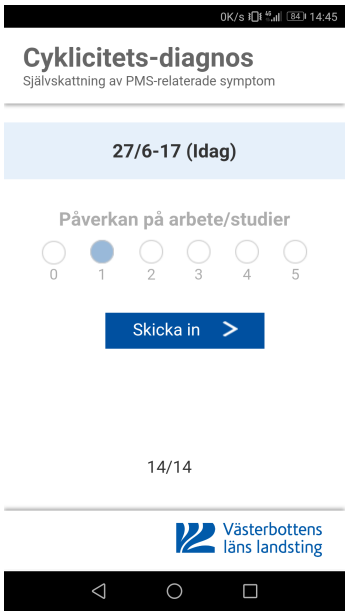


(a) The log in page

(b) One of the introduction pages

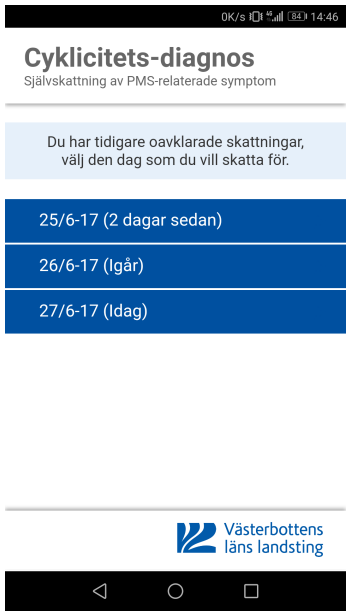


(c) The start in the answer page



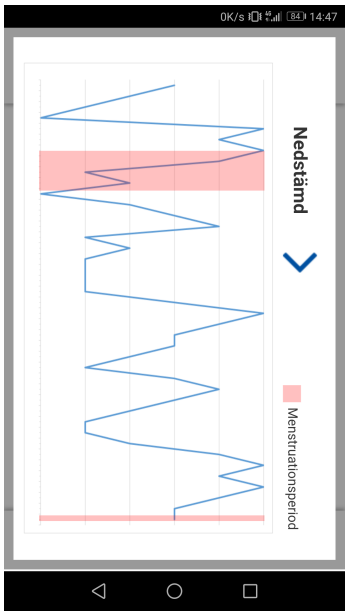
(d) The end in the answer page

(e) The done for the day page



(f) Forgot to answer page

(g) The result page



(h) Showing graph of result

Figure 6.2: Some of the mid-fi mock-up pages

6.3.3 Usability testing the mid-fi mock-up

All the six test participants could complete the tasks in each of the three scenarios. The key findings from these tests were that:

- the slow responsiveness of the interface irritated the participants;
- having no support of scrolling up and down to navigate between questions confused most participants (the users were supposed to click on the question above/beneath);
- two participants questioned how they would be able to see prior completed results after they have seen it once and closed the application;
- two participants said it was annoying to log in for each scenario, requesting a "keep me logged in" functionality;
- one participant did not fully understand the question/symptom "uncontrolled", and questioned if it meant uncontrolled in a physical or mental manner or even in the sense that others could not "control" them.

6.4 Deliver

This section describes the results from implementing the PWA.

6.4.1 Changes from the mid-fi prototype

Due to findings from the discover and define phases, the following changes were applied to the design of the PWA in relation to the mid-fi prototype:

A *hub* home page was added where all the other pages of the application could be accessed from. The selection to add it derived from: (1) the test participants wanting to see prior results in the application, which was considered easier to navigate to with a hub as home page; (2) the PWA, with its app shell model structure (see description of it in section 4.2.3), would have better usability if unauthorized static navigation content was showed at start up instead of authorized remotely retrieved content; and (3) it could help users get a better overview and hence understanding of the app on initial start, in comparison to getting to the "log in"-page.

***Remember me* functionality was added** so that users could mark if they wanted to stay logged in. The choice was based upon: (1) test participants argued it was annoying to log in each time they wanted to answer; (2) using a JSON Web Token for authentication yielded an easy way of saving credentials without clearly storing a password; For instance, by storing the token in a cookie on the browser.

***Push notification* functionality was removed** due to shortage of time (see discussion about limitations 7.1 for more information). Hence, the *done* page did not display the alternative for users to add push notifications.

***Add to homescreen* functionality was moved** from the done page to be displayed when the application initially started, with an *App Install Banner* (see the banner appear in the figure 6.4a) by the browsers which supports that technology. This derived from the pop-up banner being easily configured with the web app manifest and also because a solution for an "add-to-homescreen"-button was not found within the period of time it was developed.

The result menu page was changed to support showing several results. It also got a button for ending an ongoing period of doing self assessments since the algorithms to detect the end of menstrual cycles was not implemented (see section 7.1 for a discussion about this).

The introduction information was changed according to what the clients said about being more clear that two menstrual cycles should be filled in.

6.4.2 Navigation flow

The navigation flow (see a diagram in figure 6.3) for the PWA originates from a homepage acting as *hub* where the other pages could be accessed from. The pages accessible at the homepage view are "*form*" where self assessments can be answered, "*result*" of finished self assessment period(s), "*introduction*" containing three information panels about the test and "*settings*" containing external links to the IT unit at the county council and a link to the female clinics home page, as well as log in/out alternative.

The pages (and sub pages) for answering and seeing results are locked with a requirement to authorize themselves. The authorization is achieved by sending a JSON Web Token from the server after a user log in. So if a user is not logged in (i.e., has not got a token stored on its device) when selecting to go to the "answer" or "result" page, they get redirected to the log in page to fill in their user credentials.

At the result page, a user starts at a *menu* view where it can: select from a list of finished assessment periods (if any exists); and select if they want to start a new or end the current period of self assessments. Selecting a result to show will take them to the "*show*" view where results for that form a retrieved server-side and then visualized with the help of a *Google Chart API* web component [1].

For the page where users answers a self assessment (i.e., the *form*-page) the user flow is based upon the wizard design pattern. The user gets guided through what to do in order to complete the days that available to answer for. This is achieved by retrieving information from the server about if a user has an ongoing form, and if it has, which days that are unanswered for. The retrieved unanswered days are limited to account for if the current day and any of the two days before that is answered or not, according to the requirement given by the female clinic. To verify that a day is answered for within a valid time, the current local time for the client is checked to be between 20.00-23.59, according to the requirement given by the female clinic.

[1] `https://www.webcomponents.org/element/GoogleWebComponents/google-chart/elements/google-chart`, accessed 2017-08-10

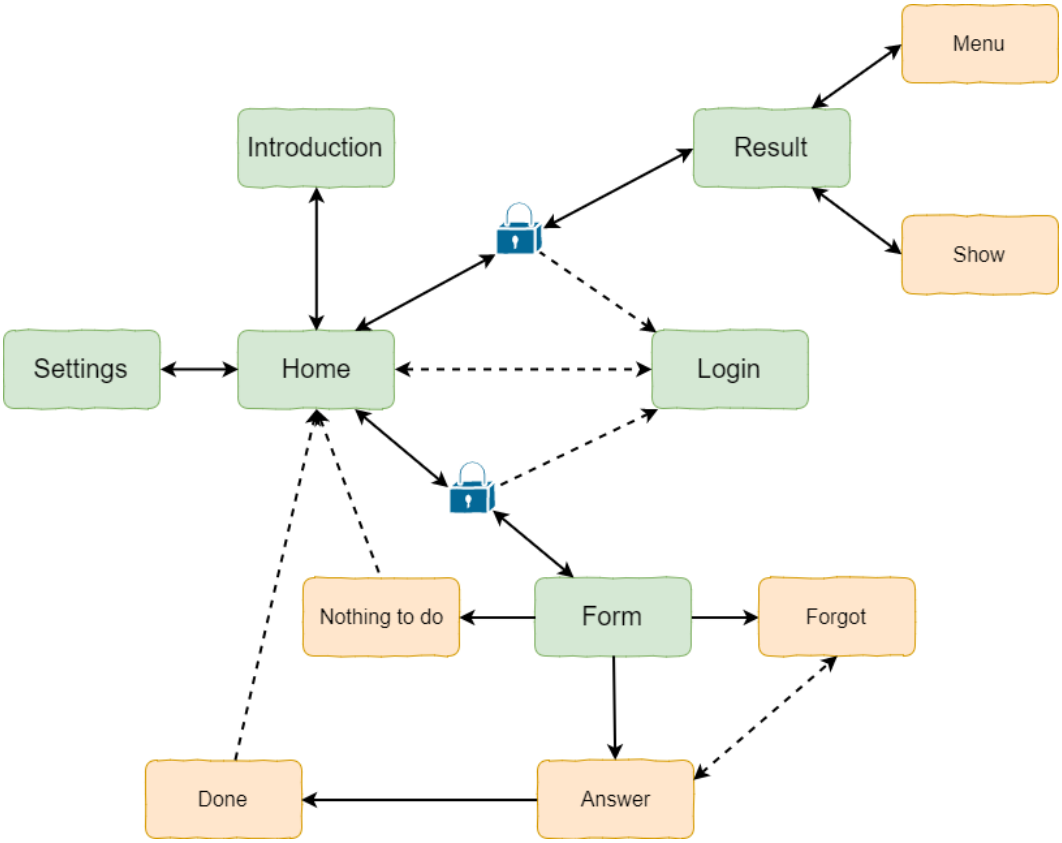


Figure 6.3: A diagram of the navigation flow for the developed PWA

Displaying the views for the form page

The routing of where to send users that chose to answer a self assessment was arguably rather tricky. Several states needed to be checked to redirect them to the right view. As guidance, the main code used to achieve this was:

```
switch (this.unansweredDates.length) {
    case 0:
        this.hasAnswered ?
            this.goTo('done')
            : this.goTo('nothing-to-do')
        }
        break;
    case 1:
        this.shouldAnswer.today ?
            this.aValidTime() ?
                this.goTo('answer')
                : this.goTo('nothing-to-do')
            : this.goTo('forgot');
        break;
    default:
```

```
            this.aValidTime ?
                this.shouldAnswered.today ?
                    this.goTo('answer')
                    : this.goTo('forgot')
                : this.goToForgotWithoutToday();
}
```

A description of this code is that:

If there are no days to answer for and the user has answered a form prior the getting there, a view telling the user that everything is done for today is showed. Otherwise, the user will be showed the view telling that there is nothing it can do right now.

If there is only one unanswered day and today is that day: then if it is a valid time go to the view where the user can answer, but if it is not a valid time go to the view showing that there is nothing the user can do right now. Otherwise if it is one day to answer and it is not today: go to the forgot view where it is shown which day that is unanswered for.

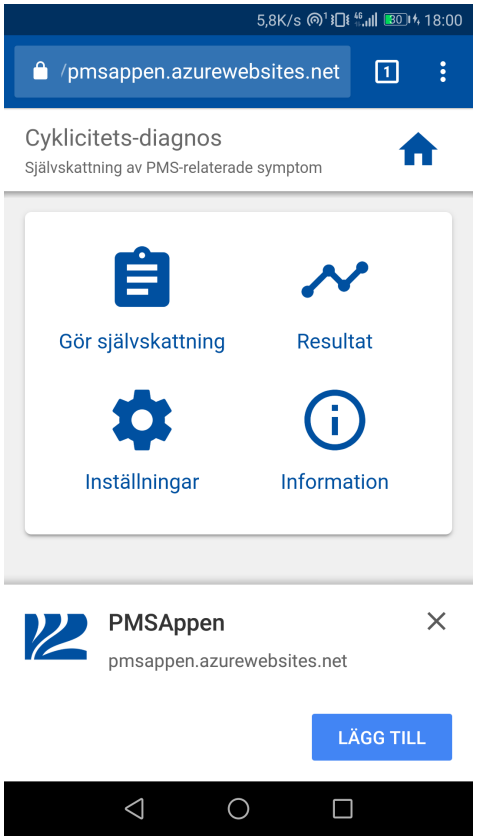
For the cases where more than one day is unanswered for and it is a valid time: then if today is one of the unanswered days go to the answer view for today, but if today is not one of those unanswered days go to the forgot view where the two prior days that are unanswered for is shown. Otherwise, if it is not a valid time to answer, go to the forgot view but exclude today as one of the days to select there.

6.4.3 Interface

The first time a user visits the app the homepage is displayed with an *app install banner* (see figure 6.4a), if supported by the browser. Here the user can select if they want to add the app in the form of an icon at its device home screen. If the app is opened through that icon a splash screen (see figure 6.4b) will be displayed when the actual page to show loads. The app will then also be viewed without a URL bar (see figure 6.4c for an illustration) and as a stand alone application (i.e., existing as an "app" separate from the browser app on the devices).

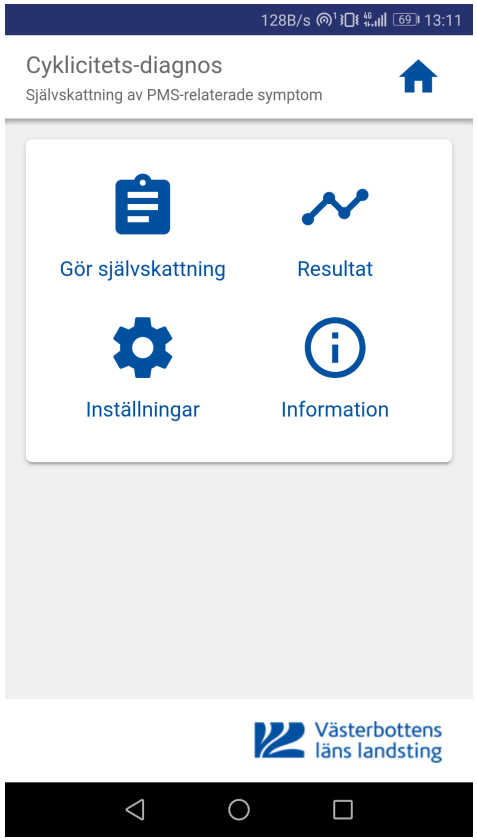
The introduction page is used to display information about the application (see figure 6.5 for the three different views the introduction page has). To be able to answer self assessments (see figure 6.6 for examples of the answer page) or view results (see figure 6.7 for examples of the result page) they need to log in which is done at the "log in"-page (see figure 6.9a). If a user has forgotten to answer prior days, it gets to a view where they can select which day to answer for (see example in figure 6.8c). When a user is done with all assessment it can do at that time, it comes to a view notifying about this after sending in an answer (see example in figure 6.8a). If a user navigates to the answer page (e.g., through the button on the home page) without having any self assessment to do, it sees a view notifying that nothing can be done at this time and when they should visit next time to be able to answer a self assessment (see example in figure 6.8b).

When the result view containing line charts of the self assessment is accessed (see figure 6.7c for an example of such a page), the user is instructed to rotate their phone into landscape mode in order to view the results if the phone is in portrait mode. The interface also consists of a basic settings page (see figure 6.9b) where the users can select that they want to log in/out, access links to external pages for support and the female clinic, and select a button to remove all data that is stored about them.



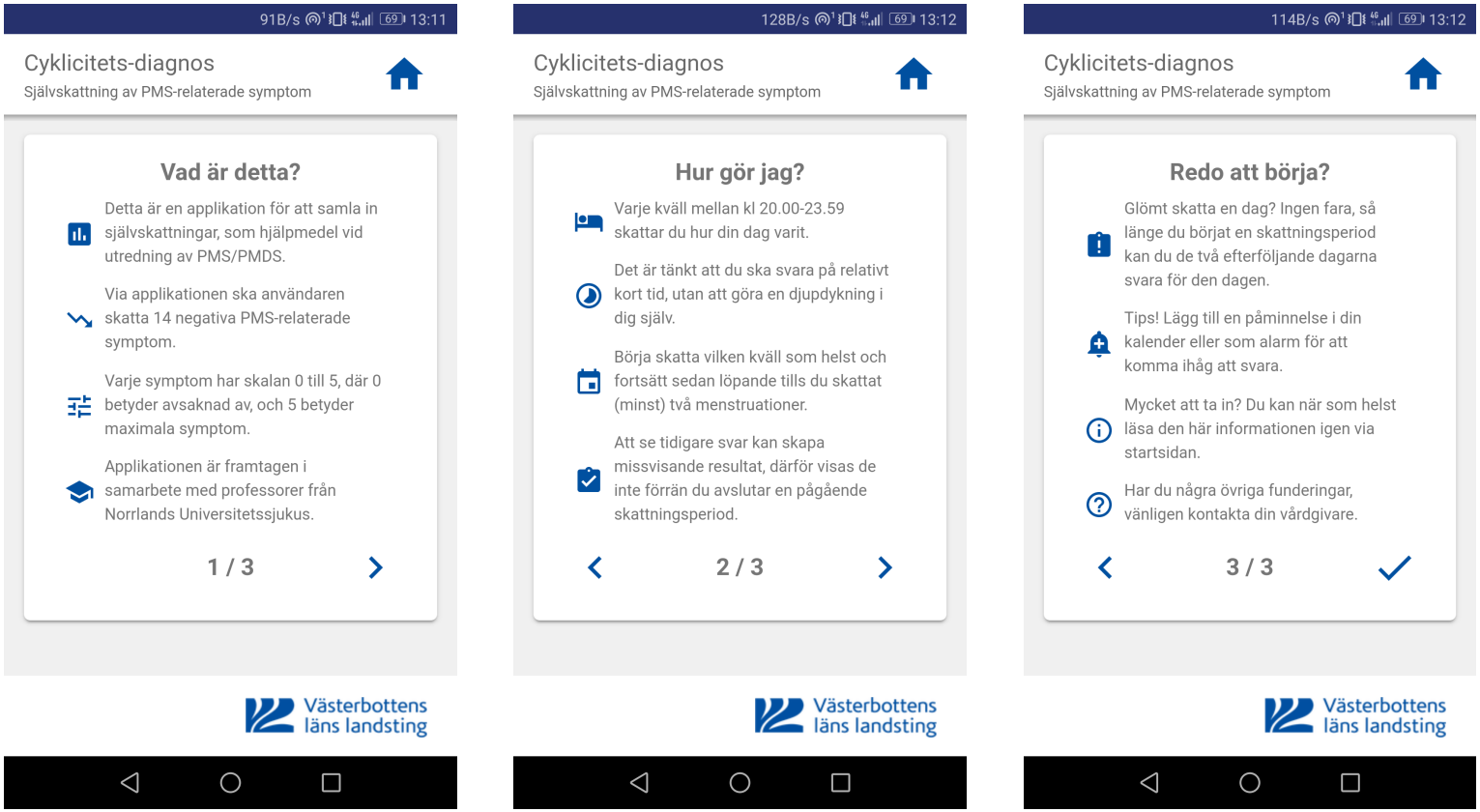
(a) Home page of the PWA showing an *App Install Banner*

(b) The splash screen for the PWA



(c) Home page of the PWA in fullscreen mode

Figure 6.4: Screenshots of different states when the app is launched



(a) The first introduction view (b) The second introduction view (c) The third and last introduction view

Figure 6.5: Screenshots of the three views for the introduction page

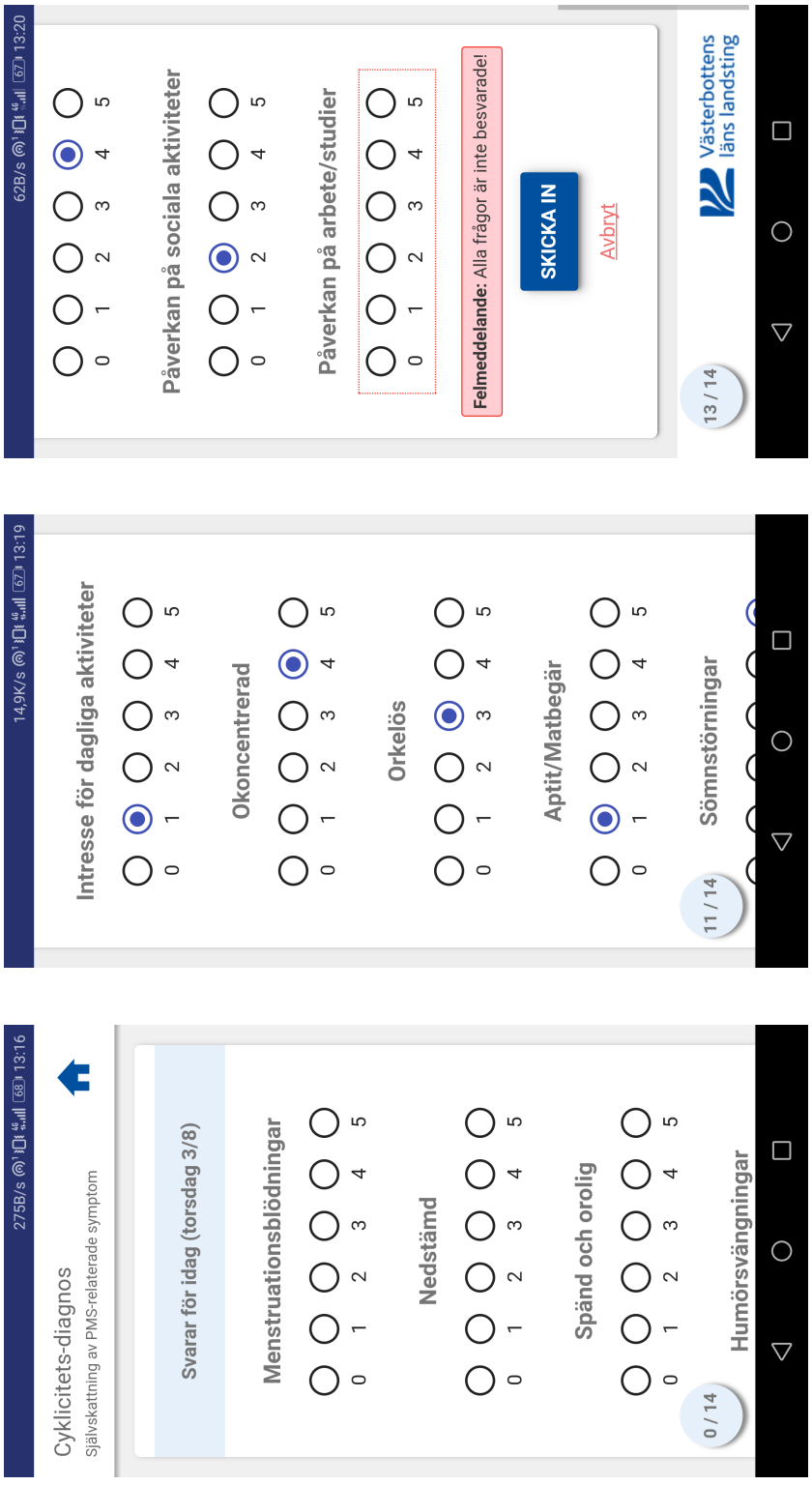
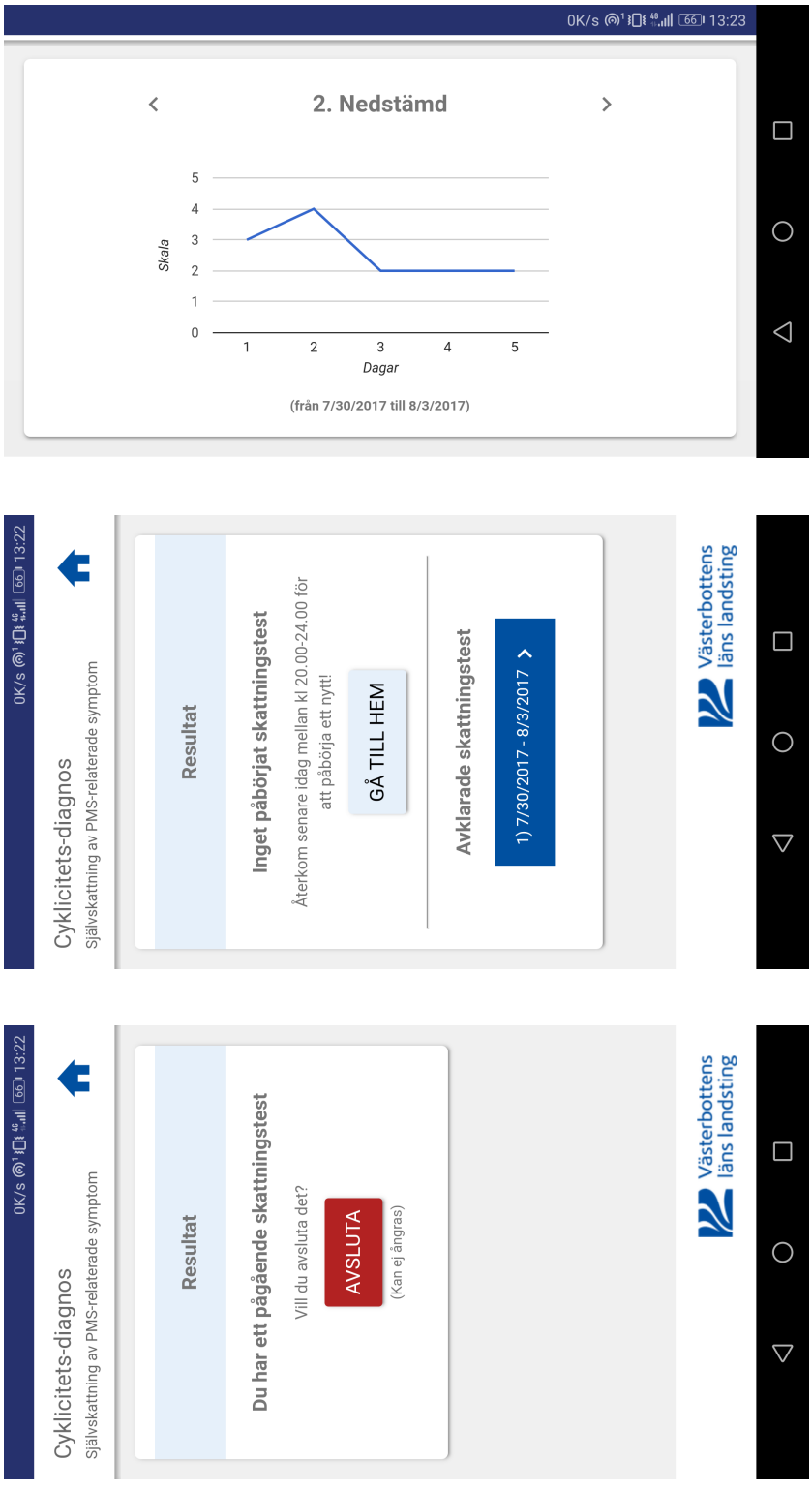


Figure 6.6: Screenshots of different stages when a form is filled in

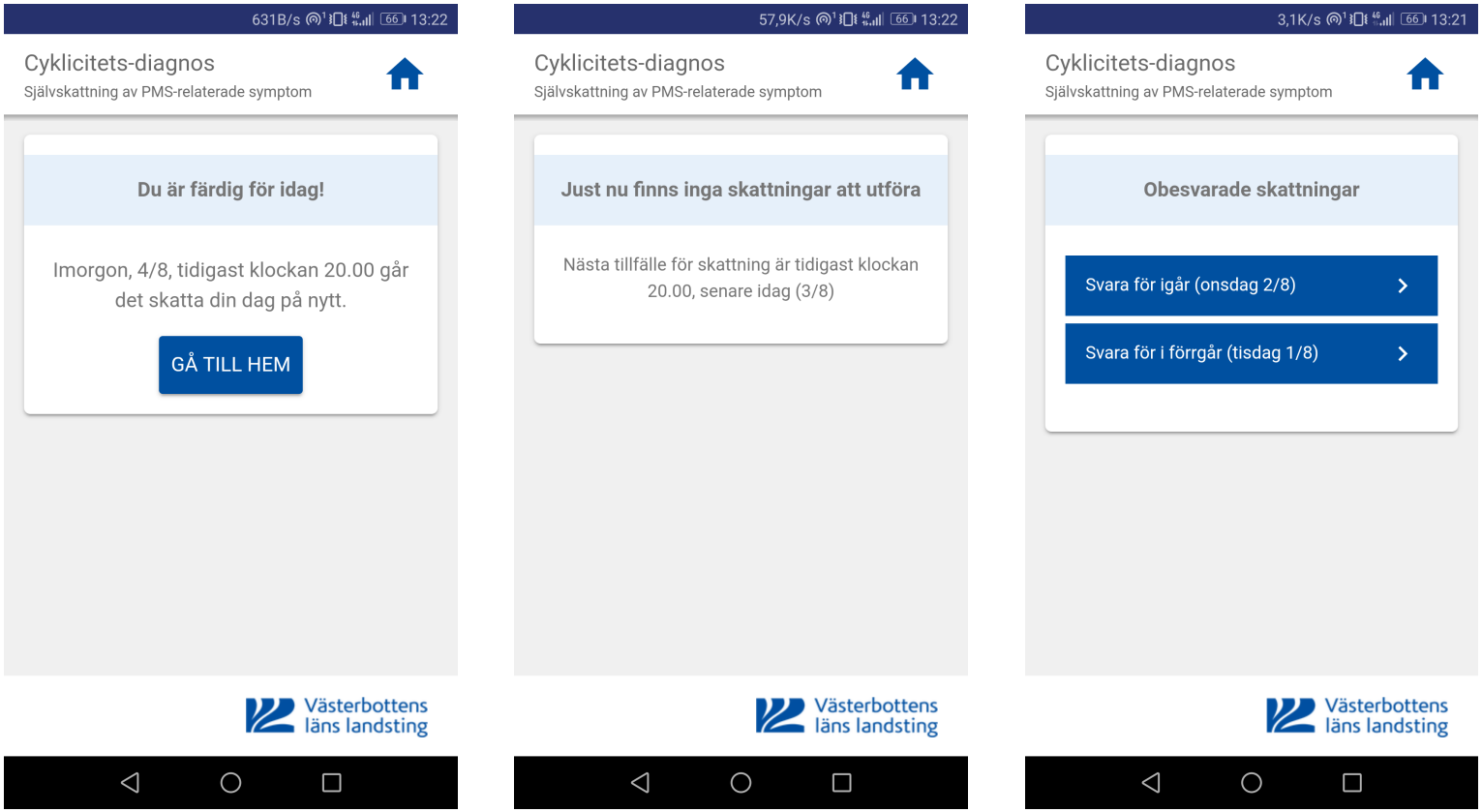


(a) Example of the results page showing one ongoing form

(b) Example of the results page with one finished form

(c) An example when a chart for one of the symptoms is presented in landscape mode

Figure 6.7: Screenshots of different result view states

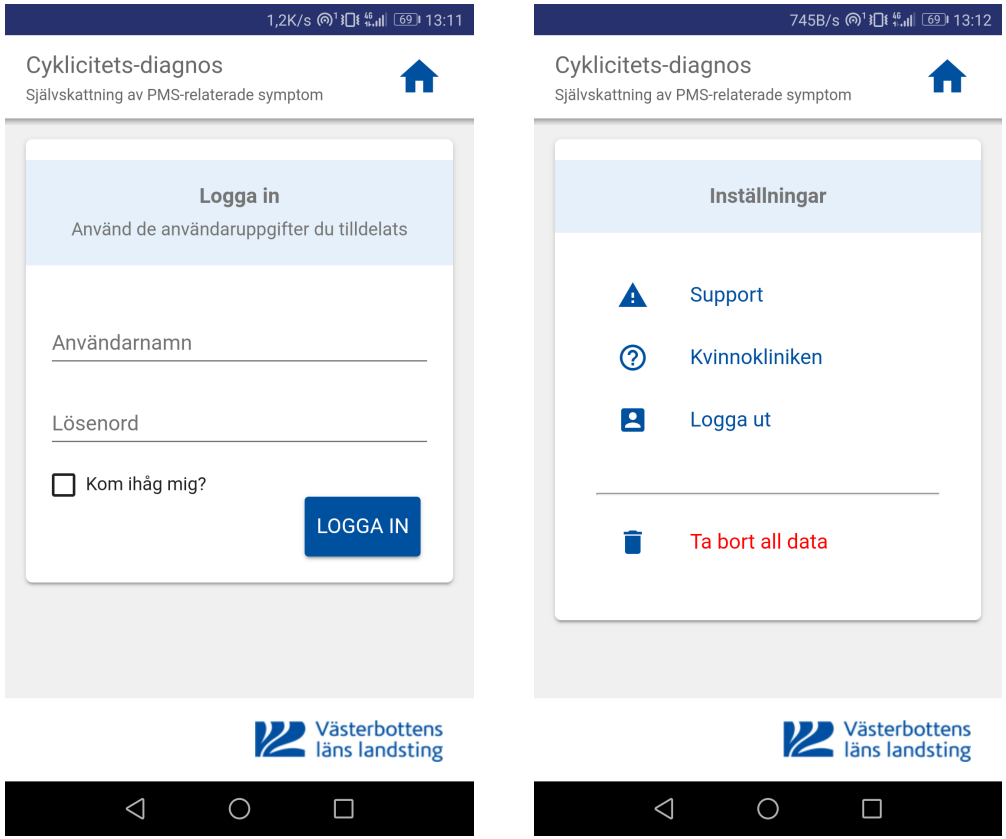


(a) Example of the page showing everything is done for this time

(b) Example of the page showing there is nothing to do for now

(c) Example of the forgotten to answer menu where two prior days are unanswered for

Figure 6.8: Screenshots of different states when accessing form pages



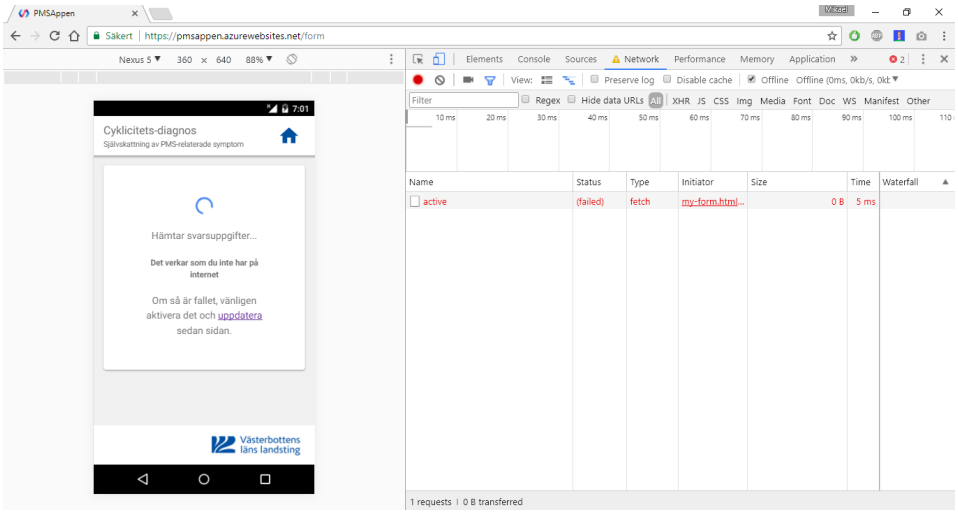
(a) The log in page.

(b) The settings page when logged in.

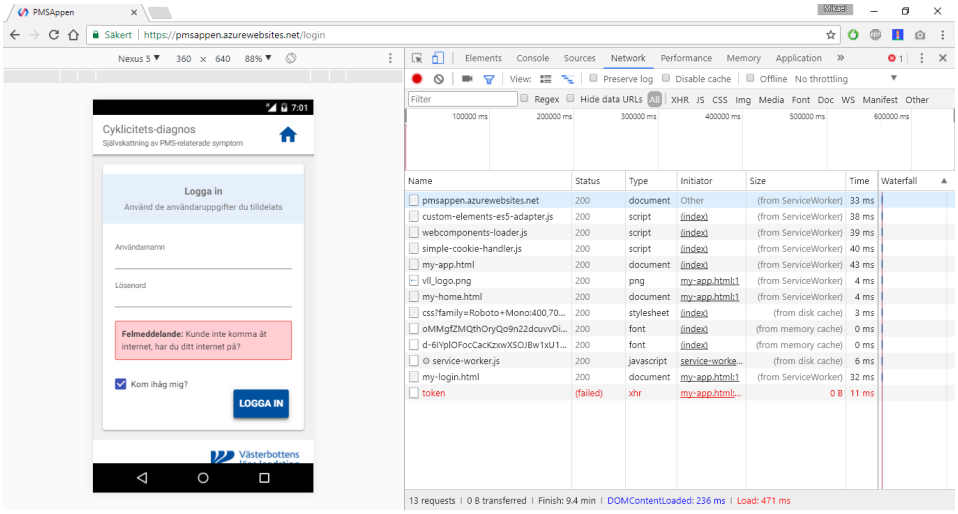
Figure 6.9: Screenshots of the log in and settings pages

6.4.4 Offline notifications

As the PWA can be interacted with while being offline (if service workers are supported) some offline notifications telling the users that they need to turn on the Internet is added. For when a page that requires retrieving information from a server is attempted to be accessed the user gets notified and can select to update the view when they turned on internet (see figure 6.10a). If a user is offline and unauthenticated when they attempt to visit a page requiring authentication they get redirected to the "*log in*"-page as usual. If they however at this page try to fill in their credentials offline they will get notified that they do not seem to have internet access and the app suggests that they check if it is on (see figure 6.10b).



(a) The offline notification showed when accessing pages that need online access



(b) The offline notification showed when attempting to log in towards the web service

Figure 6.10: These figures show notifications in the application that tell users they are offline and need to turn on internet, using Chrome DevTools to have offline mode

6.4.5 Example of caching HTTP requests during runtime

Presented beneath is two related code examples to illustrate how the runtime caching was achieved. The first code snippet shows how, with the help of fetch and promises, a JSON object with introduction data is asynchronously retrieved through the web service. This example also uses anonymous arrow functions supported in JavaScript *ECMAScript 2015/ES6* [2]

```
if (self.fetch) {
    fetch('api/introduction', {
        method: 'GET',
        headers: {
            "Content-Type": "application/json"
        }
    }).then((response) => {
        return response.json();
    }).then((json) => {
        this.introductionDTO = json;
    }).catch((err) => {
        console.log("Could not retrieve introduction, error:" + err);
    });
}
```

A service worker intercepts this request and caches its response. This functionality was set up in the configuration file for the sw-precache tool. If the request is be made again, the service worker can retrieve it from the cache storage instead of at the server over the network. Using the "fastest" handler it returns either the cached or network resource based on what is fastest retrieved, usually that is the cached version. If the network request is successful it updates the cached version. Below is a code example of this configuration:

```
runtimeCaching: [
        {
            urlPattern: "/api\/introduction/",
            handler: "fastest",
            options: {
                cache: {
                    name: 'introduction-cache',
                    maxEntries: 1
                }
            }
        },
        ...
    ]
```

6.4.6 The manifest file

With this configuration for the manifest file: a splash screen, a custom color of the header in the browser app, and icons for adding the app on the home screen of a devices are, among other things, added in browsers that support these features.

```
{
```

[2] `http://es6-features.org/`, accessed 2017-07-26

```
  "name": "PMSAppen",
  "short_name": "PMSAppen",
  "start_url": "./?homescreen=1",
  "display": "standalone",
  "theme_color": "#0050A0",
  "background_color": "#E6F0FA",
  "icons": [
    {
      "src": "images/manifest/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "images/manifest/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

6.5 Evaluation

6.5.1 Lighthouse audit

The full results from the audit with lighthouse is presented in appendix .6. The application reached an 86 out of 100 grade for the PWA audit. Nine out of eleven audits were successful, where the two failed audits were:

- **Does not redirect HTTP traffic to HTTPS**

  If you've already set up HTTPS, make sure that you redirect all HTTP traffic to HTTPS.

- **User will not be prompted to Install the Web App**

  Browsers can proactively prompt users to add your app to their homescreen, which can lead to higher engangement (Faliures: Manifest start_url is not cached by a Service Worker).

Manual checks to verify

- **Site works cross-browser**

  From manually testing the application it was proven to work successfully on newly updated/installed Chrome, Firefox, Opera and Edge browsers on a Windows 10 operating system.

- **Page transitions don't feel like the block the network**

  A subjective measurement, but the author did not sense slow transitions.

- **Each page has a URL**

With a routing web component (app-route [3]) from Polymer: pages are routed to with unique URLs and (when needed) the History API [4] is used with its pushState/replaceState functions to dynamically change the address bar to display the page it is at.

6.5.2 Survey

All of the eleven test participants used mobile phones to use the PWA. Six out of them had iPhones and used Safari Browsers, the others had an Android phone where four used the Chrome and one used the Firefox browser. Five participants were between the age 31-40 years, four between 21-30 years, one between 41-50 years, and one between 51-60 years. Two of the users (these used Chrome and Android) added the application to their homescreen.

The total average SUS score given was 87.27 based on the eleven retrieved answers, which is roughly 19 score points above what is considered average at 68.

All the answers (translated from Swedish) received for the optional questions can be viewed in appendix .7. As a brief general summary of these answers many reported that the app felt quick and was easy to use and negative remarks were mainly regarding usability issues of not showing help text for symptoms and scales. Some also argued that having this app as a native app instead might: make it easier to discover (by means of app stores); give better support of reminding users; and make users feel more comfortable. One test user also claimed that storage would be a benefit for a web compared to a native app.

[3] `https://www.webcomponents.org/element/PolymerElements/app-route`, accessed 2017-08-02
[4] `https://developer.mozilla.org/en-US/docs/Web/API/History_API`, accessed 2017-08-02

Chapter 7

Discussion

In this chapter we will first discuss limitations and relevant findings from the work process; then we suggest some guidelines for those set out to build a mHealth PWA; and finally we will conclude with the findings of how suitable a PWA is for gathering self assessments in a health care context.

7.1 Limitations

One limitation of this solution is that the legal aspects applying to mHealth applications are not considered to be investigated thoroughly enough and hence the PWA is not considered to comply with them. At the initial phase of this thesis work we believed several types of manuals for mHealth applications would exist, but found none. So to instead unravel most questions regarding this a meeting with a lawyer from the eHealth unit was scheduled, but due to vacation and sick leave that meeting never took of. Given this, it was chosen to prioritize implementing the PWA instead to be able to complete it in time for the planned user tests.

Other limitations for this solution were that some requirements the female clinic had for it was not implemented, namely:

Reminding patients when to answer. The initial idea was to get push notifications triggered locally from the users phone, with the help of a service worker, but no such solution was found. Setting up push notifications triggered remotely from a server with the Notification API [1], Push API [2] and a service worker should be possible, but we did not have enough time to investigate and try this out.

Exporting data to excel. The intention of this was to enable researchers from the clinic to access patient data. As the data is stored on a database server, an API for researchers to retrieve data can be developed. The intended solution was not to create a user with access to all patients data within the PWA for this, but instead create another application used within the hospitals intranet for it. Mainly due to that it arguably exposes less threats of compromising the patients' data.

Algorithms for detecting when a menstrual cycle ended. Storing data for each menstrual cycle could arguably be a good way of structuring data to e.g. automatically

[1] `https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API`, accessed 2017-08-09
[2] `https://developer.mozilla.org/en-US/docs/Web/API/Push_API`, accessed 2017-08-09

detect when to end simplify presentation of it. The complexity of building such an algorithm did however increase. Mainly due to the fact that some females can have ovulation bleeding, and might fill that in as menstrual bleeding. Creating an algorithm covering this issue was considered too time consuming to manage during this thesis work.

7.2 Findings from developing the PWA

Mentioned in limitations, legal aspects were not considered to be sufficiently addressed in the final PWA due to it being more complex than what was initially expected. However, some guidelines were although discovered at a later phase in the thesis work that we believe would have made it easier to comply with regulations during design of the interface and system in whole. Given a new similar project we suggest combining some of these to produce a legal specification to help us during development of a PWA. Three examples of such guidelines are:

- A checklist [3] (in Swedish) for how to process personal data on mobile devices, made by the Swedish Data Protection Authority (DPA), in Swedish *Datainspektionen*).
- EU's code of conduct for mHealth privacy [4], facilitated by the European Commission. It is, to date, waiting for approval and comments by the Article 29 Data Protection Working Party and they consider it crucial to wait for their opinion before it is applied in practice.
- A checklist [5] (in Swedish) for supporting access logs following the patient data act (PDA), also made by the Swedish DPA.

During the phase of creating the lo- and mid-fi prototypes an idea arose to present the data of "menstrual bleeding" on top of other symptoms in the visualization of results. The clients at the female clinic liked this idea since knowing about the menstrual period is essential to seeing patterns of PMS. In the deliver phase this overlapped data was intended to be represented as vertical bars (see an example in figure 6.2h) with an addition of having the scale value corresponding to the amount of opacity. However we did not have enough time to implement this during the delivery phase and since it was not something the clients suggested or required we do not view it as an limitation. Neither is it presented as a suggestion as it was not tested in the final PWA. However, for future work in this specific scenario of PMS self assessments, it might be interesting to further investigate this.

The *polymer starter kit* boilerplate used in this project is considered very helpful. It gave a starting point with, e.g., an app structured to support the app shell model, an example manifest file, added polyfills for supporting web components in incompatible browsers. Building the app with the polymer-cli tool also enabled, e.g., minimizing javascript, html and css files in the build process.

Using the sw-precache tool along with Polymer to get offline support with service workers was considered rather simple as its was mostly configuration and not coding. With this in mind it was although rather time-consuming to code functionality in the application to

[3] `http://www.datainspektionen.se/lagar-och-regler/personuppgiftslagen/sakerhet-enligt-personuppgiftslagen/mobila-enheter/`, accessed 2017-08-24

[4] `https://ec.europa.eu/digital-single-market/en/privacy-code-conduct-mobile-health-apps`, accessed 2017-08-24

[5] `http://www.datainspektionen.se/lagar-och-regler/patientdatalagen/systematisk-logguppfoljning/`, accessed 2017-08-24

handle different scenarios when a user's network connection becomes unavailable. Many bugs regarding this might certainly exists and is thought to mainly derive from transferring data remotely and syncing the state of the PWA according to what is stored at the database. If the user choose to use several devices (e.g., phone and tablet) it makes it even more difficult to keep them all synced. Different scenarios of losing network connection is not tested thoroughly for this application and that is considered needed before this PWA is ready to be released.

Some findings for beneficial aspects of PWAs in this context might be: cost effectiveness from only needing to develop one solution working on many platforms and devices as they are accessed through browsers. Accessibility and storage benefits from not needing to install an app (only cache pages) to use it. We also considered it felt smooth only having to implement a single solution and get cross-browser support and hence be more or less platform independent. Furthermore, although it was not the focus, but as a responsive design was used the solution was more or less device independent as well given that the devices have a common web browser app. I recon it would have taken a lot more time to develop this kind of range and support with the same features, using other solutions for developing mobile applications.

One issue we experienced during implementation of the PWA was a lack of documentation. This might be due to the fact that a lot of newly developed web technology was used which not always were that extensively covered in the documentation. For instance, since Polymer 2.0 was released as late as in May 2017 [6] guidance for how to use it outside of their own homepage was rather limited. In hindsight rather simple solutions became unnecessary time consuming.

We consider the biggest contradiction to develop PWAs for this usage is the, to date, limited browser support of technologies used in PWAs, especially for the service worker API. Implementing PWA features naturally takes time and generally costs money, so if they are not supported in devices/browsers used by a substantial amount of users, it will for these arguably been more or less for nothing. Building on the findings from the workshop that equal care is important, PWAs arguably neither enforces it as some get more benefits than others. However, support for service workers, the core technology of PWAs, are under development for the major browsers that currently do not support it, namely the Safari and Edge browsers [7]. So seemingly in a near future, many PWA features will be supported in modern browsers from the most common browser providers. Furthermore, given that PWAs use progressive enhancement, if they are implemented correctly, they should still work for the browsers that lack support. So it does not hinder using With that being said, we do not think PWAs should replace current solutions but instead coexists with them until PWAs have more extensive support.

7.3 Evaluation of the PWA

In this section the results of the evaluations are discussed followed by a suggested method for evaluating similar future work.

7.3.1 Lighthouse audit

In the lighthouse audit, the PWA failed regarding redirecting HTTP to HTTPS. HTTPS was supported for the application but configuration for redirecting it was neglected due to

[6] `https://www.polymer-project.org/blog/2017-05-15-time-for-two`, 2017-08-01
[7] `https://jakearchibald.github.io/isserviceworkerready/`, accessed 2017-08-14

shortage of time setting it up.

The other failure of the audit regarded not showing the app install banner prompts for users. However this can be seen as a false negative depending on that the *start_url* had a redirect in the manifest configuration making it undetected as a cached asset [8]. Also, evident within figure 6.4a, the prompt is shown by the application. The manual checks were also confirmed during the user tests where different browsers worked and users claimed the application felt quick.

7.3.2 User testing

Based on the assumption that all test users that participated answered the survey: all test users could use the application which yields an 100 percent success rate for the PWA. The SUS score given based on the survey answers can be considered rather good since 87.27 is a bit over the average of 68. In this sense, the PWA seemed to work well for gathering self assessments.

Knowing what PWA features test users supported were rather unclear since information about browser versions were not collected. It is although clear that six out of the eleven participants used an iPhone with the Safari browser, so these had no support for service workers or the web app manifest. And the user with the Firefox browser would not have support for the web app manifest as it is to date not supported by Firefox. However, since two Android users with the Chrome Browser added the app to their devices' homescreen it showed at least those supported the web app manifest. Due to this it is difficult to make an assumption of how great of an impact the PWA features had for the usability.

The fact that support for PWA features is limited presents some

7.3.3 Suggested evaluation for future work

WHO has produced a guide for research and assessment of digital health interventions [5]. It, e.g., suggests evaluation and monitoring methods to use in different phases of developing them. It also contains a checklist for reporting and assessing mHealth interventions, namely the mHealth evidence reporting and assessment (mERA) checklist (see [24] to read more about it). We consider this guide very useful but sensed it would require too much time from developing the PWA to obey it, considering the time limitation of this thesis work. For similar future work, with a greater time span, we do however advice using it as it can, e.g., help produce measurable and comparable evidence to assess suitability and success of mHealth interventions.

7.4 Guidelines for developing a mHealth PWA for gathering self assessments

The following ten guidelines are suggested for those who set out to develop a PWA for gathering self assessments in a health care context. Note that some of these are not limited to PWAs: they might be suitable for other applications as well.

Use boilerplates for PWAs to kick-start implementation. Developing the PWA during this work was helped a lot with the Polymer Starter Kit template generated with

[8] `https://developers.google.com/web/tools/lighthouse/audits/cache-contains-start_url`, 2017-07-20

the polymer-cli tool. Using this or similar starter kits for PWAs is advised to get "much for free" and it might also help quickly getting an understanding of PWAs.

Use web component libraries to help get the look and feel of a native app. Using the Polymer web components library (which e.g. contains material design UI elements) [9]) in this project made it rather easy to create a native app appearance. This library, or similar solutions (e.g. Ionic [10]) is suggested. However, note that support for old browser version may not always be possible with these.

Utilize tools that help you create service workers. The sw-precache tool was used during this thesis and it does much of the hard work of enabling caching features with service workers. Using this or similar tools (e.g., workboxjs [11]) will most likely save you a lot of time.

Store self assessments locally to simplify making the PWA work offline. Unless there are some prominent reasons why you want to collect data to a server (e.g., for research or to enable real-time access for caregivers) we suggest you aim to store self assessments locally first if offline support is a main concern. For a series of self assessments, they could then instead be sent all at once when a self assessment period is finished. Although note that with this approach, using several different devices to answer would not easily be implemented, but doing self assessments without needing a network connection would more easily be implemented.

Produce a legal specification from legal checklists and guidelines to help comply with regulations. Combining different checklist and guidelines to produce a legal specification for the app might likely make it less overwhelming and time consuming to comply with regulations, compared to interpreting descriptions of regulations. Furthermore, contacting lawyers or relevant authorities with knowledge of mobile health application regulations might likely help a lot as well. We experienced that tackling legal aspects on your own, as a developer, was very complicated as they are split up on many regulations.

Pay for established authorization services if there is a budget for it. As it is not easy to verify a patients identity over the internet by yourself. If there is a budget for it, using a trusted external electronic-identification service to verify identification and authenticate users could simplify this a lot.

Show results on the users device - but not until they completed a test period. During interviews with the female clinic and during user tests, it was argued that being able to see results directly in the phone was positive. However, the professors at the female clinic claimed seeing prior could influence trailing self assessments, so we suggest deferring visualization until a test period is completed.

Add descriptions to scales and ambiguous questions when users are answering. During user tests they reported it hard to remember what the scales stood for, adding a description of the scales when they are answering could help the users with this. Also following what was said by test participants, if the questions are not instructive enough on their own adding a help text box, possibly containing examples, might help describing the questions. Be aware that these should probably be

[9] `https://elements.polymer-project.org/`, accessed 2017-08-13
[10] `https://ionicframework.com/`, accessed 2017-08-13
[11] `https://workboxjs.org/`, accessed 2017-08-13

Add reminders/push notifications if possible. Given that both during user testing and the workshop it was mentioned that the patients wanted reminders, having that functionality would likely make users satisfied. Due to time limitations, how to accomplish this with PWAs is however not investigated enough in this thesis.

Have "Remember me"-functionality for logging in. During the usability tests of the mid-fi prototype test participants stated it was annoying to log in each time. Remembering credentials to enable automatic authentication is therefor suggested. Although be aware that this might yield undesired exposure of data as anyone with access to a device can see data without needing to authorize themselves somehow.

7.5 Suitability of using a PWA in this context

Given what was managed to be completed and evaluated, we consider that this work gives an indication that a PWA could be suitable for gathering patients' self assessment. Two reasons for this is considered to be that: (1) all test users could use it to successfully fill in self assessments and (2) test users seemed satisfied as they gave the PWA a rather high SUS score and positive feedback.

However, since (1) legal aspects for mHealth applications were not thoroughly treated and (2) all requirements from the caregivers were not fulfilled in this work; we suggest further investigation to prove that PWAs are suitable for this context.

7.6 Acknowledgements

I would like to thank my supervisor, Keni Ren, from Umeå University for her solid support, guidance and reviews throughout my project. Another thanks goes to my external supervisor, Håkan Petterson, at the county council of Västerbotten for the time dedicated to discuss the project and set up meetings. I would also like to thank Alexej Timonin for all the help and advice given which accelerated the server-side implementation of the system remarkably, as well as for setting up hosting of the system. Finally I would like to thank Jan Nyhlen and Hanna Pålsson for reviewing my thesis and giving great suggestions for improvements.

References

[1] Socialstyrelsen. En mer tillgänglig och patientcentrerad vård: Sammanfattning och analys av landstingens och regionernas handlingsplaner - delrapport, 2016. `https://www.socialstyrelsen.se/Lists/Artikelkatalog/Attachments/20115/2016-3-22.pdf` accessed 2017-05-18.

[2] Teresa Zayas-Cabán and Brian E Dixon. Considerations for the design of safe and effective consumer health it applications in the home. *Quality and Safety in Health Care*, 19(Suppl 3):i61–i67, 2010.

[3] Tommy G Thompson and David J Brailer. The decade of health information technology: delivering consumer-centric and information-rich health care. *Washington, DC: US Department of Health and Human Services*, 2004.

[4] Benjamin Abaidoo and Benjamin Teye Larweh. Consumer health informatics: the application of ict in improving patient-provider partnership for a better health care. *Online journal of public health informatics*, 6(2), 2014.

[5] World Health Organization et al. Monitoring and evaluating digital health interventions: a practical guide to conducting research and assessment, 2016.

[6] Misha Kay, Jonathan Santos, and Marina Takane. mhealth: New horizons for health through mobile technologies. *World Health Organization*, 3:66–71, 2011.

[7] Joaquin A Blaya, Hamish SF Fraser, and Brian Holt. E-health technologies show promise in developing countries. *Health Affairs*, 29(2):244–251, 2010.

[8] Bruno MC Silva, Joel JPC Rodrigues, Isabel de la Torre Díez, Miguel López-Coronado, and Kashif Saleem. Mobile-health: A review of current state in 2015. *Journal of biomedical informatics*, 56:265–272, 2015.

[9] Peter R Orszag. Evidence on the costs and benefits of health information technology. In *testimony before Congress*, volume 24, 2008.

[10] Maged N Kamel Boulos, Steve Wheeler, Carlos Tavares, and Ray Jones. How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from ecaalyx. *Biomedical engineering online*, 10(1):24, 2011.

[11] Maged N Kamel Boulos, Ann C Brewer, Chante Karimkhani, David B Buller, and Robert P Dellavalle. Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Online journal of public health informatics*, 5(3):229, 2014.

[12] research2guidance. mhealth app developer economics study 2016 trend: Development compared to last year, 2016. `https://research2guidance.com/r2g/r2g-mHealth-App-Developer-Economics-2016.pdf`.

[13] Ivano Malavolta. Beyond native apps: Web technologies to the rescue! (keynote). In *Proceedings of the 1st International Workshop on Mobile Development*, Mobile! 2016, pages 1–2, New York, NY, USA, 2016. ACM.

[14] Borja Martínez-Pérez, Isabel De La Torre-Díez, and Miguel López-Coronado. Privacy and security in mobile health apps: a review and recommendations. *Journal of medical systems*, 39(1):1, 2015.

[15] Ivano Malavolta, Giuseppe Procaccianti, Paul Noorland, and Petar Vukmirović. Assessing the impact of service workers on the energy efficiency of progressive web apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pages 35–45, Piscataway, NJ, USA, 2017. IEEE Press.

[16] Y Tony Yang and Ross D Silverman. Mobile health applications: the patchwork of legal and liability issues suggests strategies to improve oversight. *Health Affairs*, 33(2):222–227, 2014.

[17] Design Council. Eleven lessons: Managing design in eleven global companies-desk research report. *Design Council - A study of the design process*, 2007.

[18] Jenifer Tidwell. *Designing interfaces: Patterns for effective interaction design.* " O'Reilly Media, Inc.", 2010.

[19] Claire Rowland, Elizabeth Goodman, Martin Charlier, Ann Light, and Alfred Lui. *Designing connected products: UX for the consumer Internet of Things.* " O'Reilly Media, Inc.", 2015.

[20] Catherine Plaisant Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* 4th Edition. Addison Wesley, 4 edition, 2004.

[21] Läkemedelsverket. Medicinska appar ska följa regelverkets krav för att vara säkra. *Läkemedelsverket*, 2015.

[22] Michael B. Jones, John Bradley, and Nat Sakimura. Json web token (jwt), 2015. `https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html`.

[23] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194), 1996.

[24] Smisha Agarwal, Amnesty E LeFevre, Jaime Lee, Kelly L'Engle, Garrett Mehl, Chaitali Sinha, and Alain Labrique. Guidelines for reporting of health interventions using mobile phones: mobile health (mhealth) evidence reporting and assessment (mera) checklist. *Bmj*, 352:i1174, 2016.

Appendices

.1 Questions for interviewing system developer

1. What do you consider is the IT units experience of developing mobile applications?
2. What developing environments do the IT unit primarily use?
3. Which programming languages do you consider the IT unit have good knowledge of?
4. Does the IT unit follow any particular frameworks for design, code management or documentation?
5. Do you have any proposal for how to integrate external services within health care, possibly against your primary electronic health record system?
6. Do you have a view on how to support interoperability between health care services?
7. Do you have any suggestions on how to store data and host a web server for a progressive web app intended for health care services?

.2 Questions for interviewing strategic expert (Göte)

1. Do you know of any regulations applying to mobile health applications?
2. Do you have any advice regarding privacy for patients?
3. Do you have any suggestions for how to store information safely?
4. Do you have any suggestions for how to securely transport data over the web?
5. What do you think about hosting the application using cloud services?
6. Do you have anything else you would like to add that you think i might have missed?

.3 Questions for interviewing strategic expert (Thomas)

1. What are the greatest values an application like this can achieve for your department?
2. What do you consider the main benefits of using an application like this one are for your health care patients?
3. Do you see any possibility of integrating the application with existing health care systems?
4. Do you see any suitable alternatives for storing data for this application?
5. Do you see any suitable alternatives for user identification/authentication for this application?

.4 Main questions used to create the system requirement specification

SRS Questions

Benefits & values

1. What is the prominent benefit/effect you see with this project?
2. How can this project increase value for your organization, within health care/female clinic?
3. How do you think this project can increase value for your patients?
4. What is most prioritised part of the application according to you?
    a. e.g., gathering, visualizing or analyzing information?

Challenges & problems

5. Can you mention some of the common challenges you face with current solution?
6. Do you know of any problems or contradictions against realizing this product?
7. Have any patients or personnel expressed dissatisfaction?

Usage & management

8. Can you describe characteristics for users, i.e. patients and personnel?
9. Can you estimate how many use the current (paper-based) solution today?
10. Can you estimate how many that might use the new application
    a. yearly?
    b. at the same time?
11. How will/should caregivers use the data from patients?
12. Can you estimate the time/patient personnel spend with current solution?

Requirements & expectations

13. What are your primary expectations of this project?
    a. What do you generally want it to improve?
    b. What do you want to avoid from current solution?
14. Could you mention what you want you require for the application regarding:
    a. Usability/User experience,
    b. Functionalities/Features,
    c. Customizability/adaptability,
    d. Security/privacy,
    e. Accessibility,
    f. Or other aspects?
15. What do you consider a successful result of this project?

.5 Document presented before user tests of the mid-fi prototype (in Swedish)

Information om användartestet

Den prototyp du kommer användartesta är tänkt att användas för självskattning av symptom för utredning av PMS/PMDS inom vården. Den bygger på en “cyklicitets-diagnos” framtagen av kvinnokliniken på NUS. Jag vill tydligt poängtera att det inte är ni som testas utan det är prototypen, i avseendet att förbättra applikationen.

Ifall ni tillåter spelas ljud in, som enbart jag kommer ta del av, för att jag i efterhand ska kunna gå igenom vad som sagts utan att behöva ta anteckningar under testet. När ni testar applikationen uppmuntras ni till att “tänka-högt”, en metod för att lättare märka brister hos prototypen när de dyker upp under testet (det kan vara svårt att minnas nämna dem i efterhand). Ni får när som helst ställa frågor om det något ni undrar över samt ta en paus/avbryta testet utan att behöva förklara varför.

Användartestet är uppdelat i 3 olika scenarion vilka beskrivs nedanför. För att genomföra det första scenariot ska man fylla i symptom, i det fallet behöver ni *inte* svara sanningsenligt utan kan hitta på uppgifterna. Det är dock önskvärt att ni ändå läser och reflekterar över frågorna för att kunna påpeka ifall någon/några av dem är otydligt. I de två övriga scenarierna har ifyllning av symptom hoppats över, d.v.s. ni kommer direkt kunna “Skicka in” fast skattning ej blivit ifylld.

Scenario 1 - Första gången

Du har blivit rekommenderad att använda den av vårdpersonal på kvinnokliniken, som även gett dig användaruppgifter för att logga in, vilket i detta fall är:
Användarnamn: *test*
Lösenord: *test*
Och det är första gången du startar applikationen.

Scenario 2 - Glömt dagar

Du har gjort testet i några veckor men har haft fullt upp de senaste två dagarna så du har inte hunnit eller kommit ihåg att fylla i skattningar under två föregående dagarna.

Scenario 3 - En cykel avklarad

Du är precis på väg att fylla i en skattning som meddelar att en menstruationscykel är avklarad. Du förväntar dig då att få se resultatet av skattningarna du gjort.

.6 Lighthouse audit results

Results for: https://pmsappen.azurewebsites.net/
Aug 3, 2017, 4:28 PM GMT+2 • Runtime settings

82	83	100	85
Progressive Web App	Performance	Accessibility	Best Practices

Progressive Web App

82

These audits validate the aspects of a Progressive Web App, as specified by the baseline PWA Checklist.

2 failed audits

- ✗ Does not redirect HTTP traffic to HTTPS
  If you've already set up HTTPS, make sure that you redirect all HTTP traffic to HTTPS. Learn more.
- ✗ User will not be prompted to Install the Web App
  Browsers can proactively prompt users to add your app to their homescreen, which can lead to higher engagement. Learn more.
  Failures: Manifest start_url is not cached by a Service Worker.

9 Passed Audits

- ✓ Registers a Service Worker
  The service worker is the technology that enables your app to use many Progressive Web App features, such as offline, add to homescreen, and push notifications. Learn more.
- ✓ Responds with a 200 when offline
  If you're building a Progressive Web App, consider using a service worker so that your app can work offline. Learn more.
- ✓ Contains some content when JavaScript is not available
  Your app should display some content when JavaScript is disabled, even if it's just a warning to the user that JavaScript is required to use the app. Learn more.
- ✓ Uses HTTPS
  All sites should be protected with HTTPS, even ones that don't handle sensitive data. HTTPS prevents intruders from tampering with or passively listening in on the communications between your app and your users, and is a prerequisite for HTTP/2 and many new web platform APIs. Learn more.
- ✓ Page load is fast enough on 3G
  A fast page load over a 3G network ensures a good mobile user experience. Learn more.
- ✓ Configured for a custom splash screen
  A default splash screen will be constructed for your app, but satisfying these requirements guarantee a high-quality splash screen that transitions the user from tapping the home screen icon to your app's first paint
- ✓ Address bar matches brand colors
  The browser address bar can be themed to match your site. Learn more.

- Has a `<meta name="viewport">` tag with `width` or `initial-scale` ✓
  Add a viewport meta tag to optimize your app for mobile screens. Learn more.
- Content is sized correctly for the viewport ✓
  If the width of your app's content doesn't match the width of the viewport, your app might not be optimized for mobile screens. Learn more.

Manual checks to verify

These audits are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

- Site works cross-browser
  To reach the most number of users, sites should work across every major browser. Learn more.
- Page transitions don't feel like they block on the network
  Transitions should feel snappy as you tap around, even on a slow network, a key to perceived performance. Learn more.
- Each page has a URL
  Ensure individual pages are deep linkable via the URLs and that URLs are unique for the purpose of shareability on social media. Learn more.

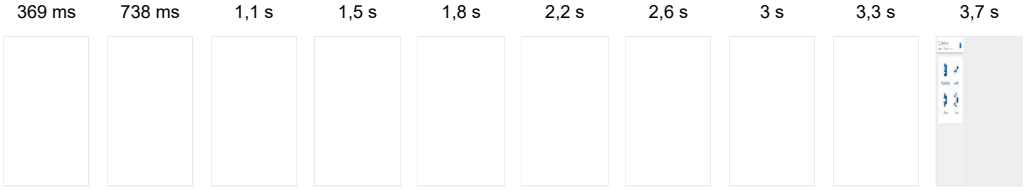
Performance

These encapsulate your app's performance.

83

Metrics

These metrics encapsulate your app's performance across a number of dimensions.



- First meaningful paint — 3 390 ms
  First meaningful paint measures when the primary content of a page is visible. Learn more.
- First Interactive (beta) — 3 390 ms
  The first point at which necessary scripts of the page have loaded and the CPU is idle enough to handle most user input.
- Consistently Interactive (beta) — 3 390 ms
  The point at which most network resources have finished loading and the CPU is idle for a prolonged period.
- Perceptual Speed Index: 3 482 (target: < 1 250) — 74
  Speed Index shows how quickly the contents of a page are visibly populated. Learn more.
- Estimated Input Latency: 16 ms (target: < 50 ms) — 100

The score above is an estimate of how long your app takes to respond to user input, in milliseconds. There is a 90% probability that a user encounters this amount of latency, or less. 10% of the time a user can expect additional latency. If your score is higher than Lighthouse's target score, users may perceive your app as laggy. Learn more.

Opportunities

These are opportunities to speed up your application by optimizing the following resources.

Reduce render-blocking stylesheets — 980 ms

Link elements are blocking the first paint of your page. Consider inlining critical links and deferring non-critical ones. Learn more.

View Details

URL	Size (KB)	Delayed Paint By (ms)
/src/my-app.html	75,88 KB	979 ms

Reduce render-blocking scripts — 650 ms

Script elements are blocking the first paint of your page. Consider inlining critical scripts and deferring non-critical ones. Learn more.

View Details

URL	Size (KB)	Delayed Paint By (ms)
…webcomponentsjs/custom-eler	1,17 KB	573 ms
…webcomponentsjs/webcompor	1,91 KB	588 ms
…js/simple-cookie-handler.js	0,73 KB	653 ms

Diagnostics

More information about the performance of your application.

Critical Request Chains: 5

The Critical Request Chains below show you what resources are required for first render of this page. Impr page load by reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources. Learn more.

Longest chain: **3 638,8ms** over **1** requests, totalling **10,22 KB**

View critical network waterfall:

Initial Navigation

- / (pmsappen.azurewebsites.net)
  - …webcomponentsjs/custom-elements-es5-adapter.js (pmsappen.azurewebsites.net) **- 572,9ms**
  - …webcomponentsjs/webcomponents-loader.js (pmsappen.azurewebsites.net) **- 584,8ms, 1,91**
  - …js/simple-cookie-handler.js (pmsappen.azurewebsites.net) **- 571,7ms, 0,73 KB**
  - /src/my-app.html (pmsappen.azurewebsites.net)

.7 Optional answers from the survey

1. Did you experience something missing from the application, if so then what?

   More articulate questions would be good.

   Nothing i could think of.

   I am thinking that it might be good to be able to fill in if you have a cold or similar? A cold affects both the mood and stamina which gives another result on the assessment. But that might not make a difference.

   To get a reminder on the phone would have simplified it. I also wish you could get the result curves on the same table in order to compare them with each other.

   Perhaps it is more vivid how the results should be interpreted when you filled in during some months.

   Help text in connection to the actual questions. PMS can include some confusion, e.g. that I suddenly forget what the grades stand for.

   Perhaps an info box about what the scale 1-5 stands for if they are not unmarked with purpose.

2. Did you experience something particularly positive with the application. if so then what?

   Easy to use, it did not take long to fill in each evening. Good that you get to take part of the results after you finished all days.

   Easy to use.

   Easy to add to the home screen. Very easy to use.

   I use the app *PTracker* since a couple of years. It is good since I can choose by myself to add symptoms both mental and physical.

   That it is noticed that there is a need and benefit with it!

   Easy, felt like the filling in the information was quickly done. Think that makes it a lot easier to get it done. Appealing and professional appearance.

   A good way to clearly see patterns.

3. How did you think was to use the application, especially regarding the start of it?

   No problems.

   Did not experience any troubles.

   Easy to get started. Easy to understand.

   Had a problem to log in since my password had a big (i) and I thought it was a small (l). Otherwise it was smooth.

   None after the bug was fixed.

   Very easy!

   Went quick to login, and a clear front page. Could not ”get lost”

4. Did you think the application felt quick or slow? Briefly motivate why you think so.

   Quick.

   Quick.

Quick, both for filling in questions and to browse around.

Pretty slow, especially when questions where answered when it often froze with a grey circle around the circle i tried to press. For those occasions i had to press 4-5 times before the answer was selected.

Neither

Rather quick, but the target area for filling in the form perhaps could have been a little bigger? It was often only a grey circle around the button I wanted to mark. I then had to stop and select more accurately.

Quick.

5. Do you consider this application to fit as web or mobile app? Briefly motivate why you think so.

   Surely as both. Perhaps a bit smoother to have an app so that you easier can remember to fill in and can save the details so you do not have to log in each day.

   If the idea is that the public is gonna use the app, then I believe a mobile application would be better. This because that people searches the appstore if they are looking for something more useful, such as health apps like this. Should it only be online it might perhaps be difficult to find it.

   Storage aspect, that is an advantage with a web app. A mobile app takes up space in the phone, that could make it so that people do not download it.

   Either or. But perhaps people are a bit more comfortable with apps. Although with good information I think a shortcut from homepage to web might work.

   Mobile.