Masaryk University
Faculty of Informatics

Evaluation and usage of Google Progressive Web Apps technology

Bachelor's Thesis

Pavel Břoušek

Brno, Spring 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Pavel Břoušek

Advisor: Mgr. Juraj Michálek

Acknowledgement

I wish to express sincere thanks to my supervisor Juraj Michálek for his valuable guidance. I am also grateful to all members of the LEAF team from Y Soft Corporation a.s. for their help and support and to Evan McElravy for proofreading.

Abstract

The aim of this thesis is to analyze Progressive Web App technology and implement a YSoft SafeQ mobile terminal for Y Soft Corporation, a. s. The scope of this thesis is to analyze the technology, evaluate requirements and limitations and implement the application.

Keywords

web applications, multiplatform development, REST, Y Soft Corporation, YSoft SafeQ

Contents

1 Introduction **1**

2 Cross-platform mobile applications **3**
2.1 *Requirements* . . . 3
2.2 *Android Instant Apps* . . . 3
2.3 *Compiling single code to multiple binaries* . . . 4
2.4 *Packaging universal code with platform-specific engine* . . . 4
2.5 *Hybrid web applications* . . . 4
2.6 *Web applications* . . . 5
2.6.1 Vendor-specific web application features . . . 5
2.6.2 Progressive Web Apps . . . 6

3 Comparison of native and web applications **7**
3.1 *Generally available features* . . . 7
3.2 *Advantages of web over native* . . . 8
3.3 *Disadvantages of the web* . . . 9

4 Analysis of Progressive Web Apps **13**
4.1 *Fundamental principles* . . . 13
4.2 *Web App Manifest* . . . 13
4.3 *Service Workers* . . . 15
4.4 *Hardware access* . . . 18

5 Creation of a Progressive Web App **21**
5.1 *Prerequisites* . . . 21
5.1.1 HTTPS . . . 21
5.2 *Recommendations* . . . 22
5.2.1 HTTP/2 . . . 22
5.2.2 Gulp . . . 22
5.3 *Application shell* . . . 22
5.4 *Shims and polyfills* . . . 23
5.5 *Service Worker caching* . . . 24
5.6 *Web App Manifest* . . . 24
5.7 *Input* . . . 26
5.7.1 QR codes . . . 27
5.7.2 Bluetooth beacons . . . 28

5.7.3 NFC 29
5.8 *Quering a remote API* 29
5.9 *Security* 30
5.9.1 HTTPS 31
5.9.2 Credentials and session management 31
5.9.3 Permissions, user approval 32
5.9.4 Source code 32
5.9.5 Updates and patches 32

6 User interface **33**
6.1 *Branding* 33
6.2 *Design guidelines* 34

7 Conclusion **37**

Bibliography **39**

A Web standards and other specifications **43**
A.1 *W3C standards and notes, Living Standards* 43
A.2 *Requests For Comment (RFC)* 44
A.3 *Proprietary definitions* 44

B Can I Use support tables **45**
B.1 *High support (> 80 %)* 45
B.2 *Low support* 46

C Tools and libraries **47**

D Application screenshots **49**

List of Tables

3.1 Generally available application features *7*
3.2 Advantages of web applications *8*
3.3 Disadvantages of web applications *10*
4.1 Features available through Web App Manifest *14*
4.2 Features available through Service Workers *15*
4.3 Caching strategies available using Service Workers *16*
4.4 Widely available hardware access features *18*
4.5 Partially available features *19*

List of Figures

4.1 Service Worker communication diagram *16*
4.2 "Fastest" caching strategy *17*
4.3 "Cache first" caching strategy *17*
4.4 "Network first" caching strategy *17*

5.1 App install banners in Google Chrome *25*
5.2 Physical Web notification and discovery *28*

6.1 Example of Material Design Lite *33*
6.2 Android and iOS versions of the application UI *34*

D.1 Input screen on various devices *49*
D.2 Login screen (localized) *50*
D.3 Jobs screen *50*

1 Introduction

Mobile devices have been only increasing in number and popularity so far. There is also a wide range of applications that can be created for these devices to utilize various hardware components and exploit the fact that many people carry their mobile phone most of the time.

One of the downsides of mobile application development is the fact that it is decelerated by the number of different operating systems, which require separate development. To mitigate this difficulty, several cross-platform development options are available, including web applications.

The goal of this thesis is to analyze the Progressive Web App technology, compare it to similar existing technologies, describe its internals and develop a Progressive Web App that demonstrates the innovative features. The application should be able to access the camera to scan and read QR codes, communicate with YSoft SafeQ server and display and finish print jobs.

YSoft SafeQ printing system developed by Y Soft Corporation, a. s. offers numerous print management features. With a print management solution like this, users send documents (print jobs) to a server, which sends them to a printer after confirmation. Therefore, users are able to cancel a print job in case of a mistake.

The main reason why companies use print management solutions is that they can manage a large number of printers and users including accounting. This allows letting users print on various printers, because they can be easily billed for printing. There is also another accounting possibility. Printers can be rent by printer companies "as a service", in which case the printer company takes care of maintenance, paper and cartridges. The customer company is then billed according to the amount of printing done.

YSoft SafeQ also provides better security and privacy, because users confirm printing when they are physically located at the printer. Therefore, nobody can read or carry away their documents. It is also possible to modify some finishing options before a job is printed, e. g. force black and white printing. These are the main responsibilities of the YSoft SafeQ mobile terminal application.

The thesis is divided into four parts. The first part includes a comparison of various cross-platform mobile application development approaches and some of their inner workings.

In the second part I compare web applications and native applications in general. I list advantages and disadvantages of these two approaches based on application features.

The third part contains an analysis of the Progressive Web Apps specifics. I compare the new APIs and possibilities to those of native applications. I also mention former web-based approaches.

The last part describes the development of the application. It is divided into sections which describe implementation of individual features listed in previous chapters.

Appendices include a list of standards and definitions mentioned throughout the text, support tables of the mentioned features in web browsers, a list of tools and libraries that were used to create the application and application screenshots.

2 Cross-platform mobile applications

Mobile applications which are not required to be extremely efficient, because they do not perform complex computations and do not access the hardware heavily, can be developed as cross-platform. Cross-platform application development involves creating a single code base for the majority of application code, separating and minimizing platform-specific commands. This is beneficial in terms of reducing duplicity and development demands when targeting multiple operating systems with the same application.

2.1 Requirements

An example list of mobile application requirements can be seen on the Progressive Web Apps website [1]. Mobile applications should be able to operate with or without internet access, respond to events even when they are not running in the foreground, utilize the device's capabilities and hardware and re-engage users with push notifications.

Several methods of creating mobile applications while fulfilling the aforementioned requirements are available. Although most of the listed approaches satisfy general requirements, there are other aspects to be considered.

2.2 Android Instant Apps

Presented at the Google I/O 2016 conference [2], Android Instant Apps enhance native applications with a better ability to act similarly to web pages. Extending the idea of deep linking ("methodology for launching a native mobile application via a link" [3]), Android applications can launch an activity without downloading the whole application package. This speeds up the launching, requires less downloaded data and no installation. The main drawback of this method is its availability on a single operating system, Android.

2.3 Compiling single code to multiple binaries

Cross-platform compatibility can be achieved by compiling single source code into platform-specific binaries. This is well known from open source desktop applications and operating systems where typically the C language is used. Although this approach can be utilized in mobile application development as well, it is low level and therefore, for most applications, unnecessarily difficult.

2.4 Packaging universal code with platform-specific engine

Several cross-platform development frameworks rely on some kind of engine, virtual machine, bridge or runtime which is platform-specific, comes in a bundle with the application code and executes it. These frameworks utilize higher level programming languages than C, for example C# in Unity [4] and Xamarin [5], Java in Codename One [6], C++ in Qt [7], Javascript in Titanium [8] and NativeScript [9].

This allows easier development in comparison to C and does not impair performance excessively. On the other hand, the engine has to be bundled with the application binary (or installed alongside as a dependency) and it does not eliminate the need to compile binaries for each platform separately unless a cloud service is involved.

2.5 Hybrid web applications

For simple applications, web technologies' capabilities are often almost sufficient. So-called hybrid web apps are built upon this idea. The application itself is developed as a slightly modified web page using common HTML, CSS and Javascript. It is then bundled in an application which creates a seamless browser window to render the web page and a Javascript-to-native bridge for hardware and OS access. This is the case with Apache Cordova [10]/Adobe PhoneGap [11].

The hybrid approach combines the pros and cons of both web pages and cross-platform applications with engines mentioned previously. While allowing web developers to create native mobile applications with advanced device access using their current skills, it faces a num-

ber of issues. The web page is always rendered in the current operating system's built-in browser, which might not perform well and cannot be updated in some operating systems, so there might be performance and compatibility issues.

2.6 Web applications

Web browser vendors started to implement APIs for advanced device capability access, making it possible to create richer web applications. These allow code sharing not only across various operating systems of mobile devices, but also with desktop computers and possibly any other device featuring a web browser. Following the idea of progressive enhancement ("increasing the richness of the user experience step by step by testing for support for enhancements before applying them" [12]), at least partial compatibility can be achieved on a wide range of operating systems and devices.

2.6.1 Vendor-specific web application features

Several features crucial for web applications were independently implemented by some web browsers and operating systems in vendor-specific manners, due to the lack of a standard. For example regarding offline functionality, most web browsers (see section B.1) provide a deprecated caching API referred to as Offline Web applications (see section A.1). Using a cache manifest, which is a text file with a list of files that the browser should cache, a web application can define which of its files are needed to operate without internet access. Information about an application, such as the name or an icon, have to be defined separately and in vendor-specific ways.

An unofficial summary of web applications-related features provided by various web browsers was published in November 2016 by W3C [13]. The list of features in the summary includes bookmarking, add to homescreen functionality, standalone indication, application name and icons definition, security and privacy concerns and more. This document also includes a list of features that a web application standard should define.

2.6.2 Progressive Web Apps

"Progressive Web Apps describe a collection of technologies, design concepts, and Web APIs that work in tandem to provide an app-like experience on the mobile web" [14]. This includes various recommendations which are not unique to web applications designed for mobile devices, for example the preference of HTTPS over HTTP or responsive design. It also brings the need of new APIs required for richer user experience, for example the Web App Manifest, Service Workers or the Payment Request API (see section A.1). The Progressive Web Apps website includes a checklist [15] of features that a web application should have and mentions the APIs which can be used. New APIs and features are being standardized by teams of developers working for various browser vendors in order to replace vendor-specific APIs and tags.

3 Comparison of native and web applications

Progressive Web Apps build upon web technologies and their purpose is to align with native applications as much as possible. This chapter lists aspects of web pages and web applications in particular which are not specific to Progressive Web Apps. These can be used to determine the appropriateness of PWAs in a specific case depending on the desired features, considering the limitations of native and web applications.

Summaries and comparisons in the following sections are inspired by the Progressive Web Apps [16] website, the Web Fundamentals [17] website, the What Web Can Do Today [18] website and also by the talks given at the Progressive Web App Summit 2016 [19] held in Amsterdam.

3.1 Generally available features

General features that have been available to developers of native applications and web applications alike are listed in Table 3.1.

Feature	Native	Web
Responsive design	●	● media queries
Internationalization	●	● Intl + library
Accessibility	●	● semantic markup, WAI-ARIA
● = full support		

Table 3.1: Generally available application features

Concerning design and layout, Cascading Style Sheets (CSS) provide a powerful yet easy mechanism for manipulating the visual appearance of the content and controls based on device dimensions and possibly other attributes. New features are still being added to CSS3 that minimize the amount of required HTML markup and CSS code.

Due to the simplicity of CSS, some developers prefer CSS to the design and layout mechanisms used in native application development.

Internationalization, specifically managing translations of text, can be handled using a library or a custom implementation. The user's language and locale can be easily retrieved from the web browser. The Internationalization API, also referred to as Intl, provides built-in functions for language sensitive date and number formatting and string comparison.

In terms of semantics, web pages and applications can utilize semantic HTML5 tags such as `HEADER`, `FOOTER` etc. to mark important sections. For more advanced controls, the WAI-ARIA (Web Accessibility initiative - Accessible Rich Internet Applications Suite) attributes are used. Among other things, this allows screen readers to function properly even with custom elements, for example custom input controls.

3.2 Advantages of web over native

General advantages of web pages and web applications in particular are listed in Table 3.2.

Feature	Native	Web
Easy deployment	○	●
Easy upgrades	○	●
Discovery on the web	◐ app store website	●
Instant first content	◐ Android Instant Apps	●
● = full support, ◐ = partial support, ○ = no support		

Table 3.2: Advantages of web applications

Web pages and applications mostly consist of plain text files uploaded to a server which serves them to web browsers over HTTP. Therefore the simplest form of deployment of a web application involves only the transfer of several files to the server, usually using the

File Transfer Protocol (FTP). No complex setup, software or hardware is needed and a build process is optional. On the contrary, native application development requires specific hardware and software, the application has to be built, signed and published to an app marketplace, which users have to visit to be able to install the application.

Another great advantage of web applications is that all users are always using the latest version of the application, if not desired otherwise. It is therefore simple to fix issues of web applications for the whole user base. With native applications, a fix is applied, the application is deployed to an app marketplace and users decide whether they install the latest version and when. Therefore a native application's user may have any historically available version installed. Some native applications force their users to update whenever a new version is available, but they also have to require permanent internet access, which is a significant drawback.

In terms of discovery, web sites receive more traffic and the acquisition cost per user is significantly lower. Web pages and applications are cross-platform and they receive not only direct traffic, but also traffic from search engines, other referring web sites, social media and advertisements. Native applications are only available from an app marketplace and require a download and an installation.

Because web applications do not require an installation, they are able to provide users with first content quickly. This effectively means that a user does not need to download the whole application for reading a single article, for example, which brings time and bandwidth savings. This inconvenience of native applications was addressed by Android Instant Apps [20] which enable partial downloads of an application, but they are still only available as an "early access" by April 2017, a year after being announced at Google I/O.

3.3 Disadvantages of the web

Features which place native applications over their web counterparts are listed in Table 3.3.

Although web pages can include links to other web pages, there are limited ways of inter-web and web-to-native communication. First of all, several URI schemes are reserved for common types of information

Feature	Native	Web / PWA
Inter-app communication	●	◐ URI schemes, deeplinking
Full feature support for all targeted devices	●	◐ UA sniffing polyfills
OS integration	●	○
Native hardware access	●	◐
Advanced hardware access	●	○
Sensitive data access	●	◐ third-party login
● = full support, ◐ = partial support, ○ = no support		

Table 3.3: Disadvantages of web applications

which can be handled by a native application, these include `mailto:` for composing an email, `tel:` for phone calls, `sms:` for text messages and `geo:` for coordinates used for GPS navigation (see section A.2). Another, non-standard option is deeplinking mentioned earlier, which has to be handled by the target application. In this case the native application is registered to handle either URLs from a web site or a custom URI scheme. Some of these URI schemes are registered at IANA [21]. The main downsides of this approach include the lack of a standard, documentation and a mechanism to detect if the native application is available at all. There is also `registerProtocolHandler` method that can be used to register a web application to handle `mailto:` links, for example, although only supported in some desktop browsers (see section B.2). As of April 2017 there is a new API in development called Web Share API, which allows web applications to open a dialog with sharing options.

Native applications limit users to a single operating system (OS) and can also restrict which versions of the OS are supported. It is therefore easier to create an application that provides the same experience to all its users, by restricting the application to a range of OS versions with well-known APIs and behavior. Similar limitations may be applied to web applications as well using a technique called

User Agent (UA) sniffing. It involves reading the UA string with the browser's and OS name and version. However, this approach is not advisable because the UA information can be easily spoofed and it does not guarantee the availability of the expected APIs. Furthermore, it is not a good practice to force users to install a certain web browser for using one's application. Web applications usually apply graceful degradation or progressive enhancement principles and feature detection. They should aim to provide each user the best experience that is possible with the available APIs regardless of the browser or version.

Integration with the OS is not possible for web applications. For example, some operating systems allow applications to place widgets on the desktop, the most frequent use being weather widgets. Native applications can also change some system settings (wake lock, display brightness, alarm) or prompt the user to do so (toggle Wi-Fi, Bluetooth, NFC etc.). For web applications, at the time of writing this is only possible via third-party native applications, like Meta Widget [22] that creates desktop widgets with web pages as content. As of April 2017, there are also some new API standards in an early stage of development, such as the Wake Lock API standard.

Hardware access of web applications is limited to the APIs supported by the browser. Although the number of (standard) APIs for accessing the hardware is growing, there are still features not available or only partially available to web applications, often with little support across browser vendors. Some APIs are not (yet) standardized, like the Web Bluetooth or the Web Speech API.

Some APIs are standardized but have minimal support across browsers, such as the Proximity Sensor or the Ambient Light Sensor. Work on some standards has even been abandoned. The Web NFC API, for example, has not even had any implementation (see section A.1). Features with good support are described in other parts of this work. As of April 2017, there are also hardware features with no sign of being available to web applications at all, for example geofencing.

Native applications can ask for permissions to access sensitive personal information, such as the contact list, telephony, call history, text messages and calendar. Although work on creating related standards was started, it was abandoned. At the time of writing, there is no way for web applications to access this data other than using remote third-party services.

4 Analysis of Progressive Web Apps

Progressive Web Apps are meant to provide the users with the same or even a better experience than a platform-specific application would. The key principle is following the idea of progressive enhancement, which effectively means utilizing (standardized) APIs and features even though they do not have a 100% support across browsers, but not relying on them for the core functionality of an application. For example, despite iOS Safari not supporting Service Workers as of April 2017 (see section B.2), they can still be used to provide good offline experience for more than a half of potential users.

4.1 Fundamental principles

According to the Progressive Web Apps website, any kind of application including PWA should be "reliable, fast and engaging" [16]. This is possible thanks to some new APIs and standards, most notably Service Workers and Web App Manifest, in conjunction with common HTML5/Javascript APIs and CSS3 features. Many of the features that were made available through these new APIs used to be the aspects in which native applications out-performed web-based applications before the Progressive Web Apps came along. Eventually, these drawbacks had to be compensated using the hybrid web app approach.

Following sections list features available to Progressive Web Apps and compare them to both native applications' possibilities and former web-based approaches.

4.2 Web App Manifest

One of the most notable standards related to Progressive Web Apps is the Web App Manifest. It provides a unified mechanism for identifying a web page as an application and supplying all of the application-related information and assets, thus replacing the proprietary meta tags and configuration files. Features and options that can be set in a Web App Manifest are listed in Table 4.1.

Feature	Native	Old web	Manifest
App name, icons etc.	●	◐ proprietary	●
Add to homescreen	●	◐ proprietary	●
Fullscreen	●	◐	●
Discovery in app marketplaces	●	○	◐ new packaging (Windows)
● = full support, ◐ = partial support, ○ = no support			

Table 4.1: Features available through Web App Manifest

Web App Manifest is a simple JSON file with specified format which may include the application name, icons, splash screen image, status bar and location bar color, fullscreen mode declaration and orientation lock. Thus it replaces proprietary Apple meta tags, Android meta tags, Windows meta tags and browser configuration files (see section A.3) and also the standard meta tag `<link rel="icon">`. All of these had to be included in every page, unnecessarily increasing the amount of code. Another benefit is that the "add to homescreen" URL can be set to a different URL than that of the visited page, which is usually the desired behavior.

There is still a drawback in comparison to native applications, though. Progressive Web Apps are not (yet) included in app marketplaces, where people search for new applications. This can be mitigated by creating a new app marketplace for PWAs such as PWA rocks [23] or by packing the PWA into a native application using an automated tool that extracts all information from the Web App Manifest. This can be done for example using Windows App Studio or a tool called PWAify (see Appendix C). Progressive Web Apps might appear in app marketplaces in the future, Jacob Rossi from Microsoft has stated that he "expect[s] PWAs to be listed in the [Microsoft] Store right alongside native apps" [24].

4.3 Service Workers

Service Workers represent a breakthrough in web applications' control of networking. Application features newly achievable by introducing Service Workers are listed in Table 4.2.

Feature	Native	Old web	Service Workers
Network info	●	◐ heuristics	◐ not needed
Offline	●	◐ cache manifest	●
Unreliable network	◐	○	●
In the background	●	○	●
Performance	●	◐ E-tag Cache-control	●
Push notifications	●	◐ third-party	●
● = full support, ◐ = partial support, ○ = no support			

Table 4.2: Features available through Service Workers

Web pages used to have little control of the network, limited to failure detection if a resource could not be loaded for whatever reason. In terms of mobile applications this is not sufficient, because mobile devices often utilize costly cellular connections or unreliable Wi-Fi connections. These conditions cannot be reasonably detected and acted upon by web applications as they have no control or information about them. While there is still no standard API for distinguishing between cellular and other types of connections at the time of writing, Service Workers can be used to handle operation with no internet access, regular connection and even unreliable networks. They provide more fine-grained control than HTTP caching with HTTP headers like `E-tag` and `Cache-control` and they supersede non-standard Offline Web Applications with their cache manifest, which provides less options and has several problems [25].

"By default, requests will go from the page to the network [...] once you introduce a Service Worker, it controls pages and requests will go through it." [26]. All requests are intercepted by the Service Worker code and caching or other adjustments, for example replacing unavailable images with a placeholder image, can be done on a per-request basis, which is illustrated by Figure 4.1.

a) regular web *b) Service Worker*

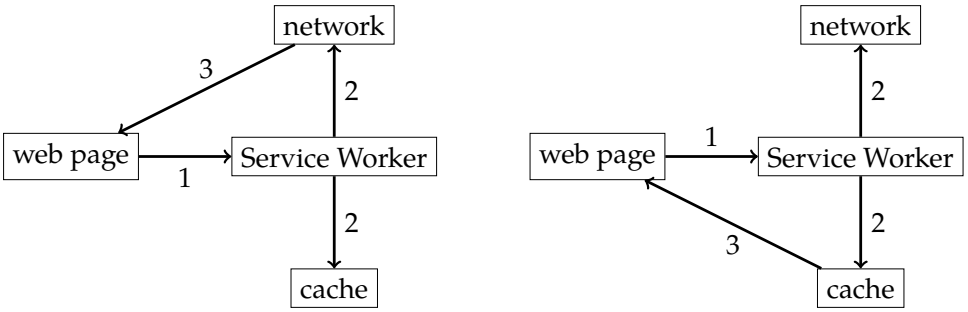
Figure 4.1: Service Worker communication diagram

Service Workers can be used to implement various caching scenarios depending on the content, network conditions etc. Common caching patterns achievable using Service Workers, or in other words algorithms for choosing between loading from the cache and the network, include fastest (Figure 4.2), cache first (Figure 4.3), network first (Figure 4.4), cache only and network only (regular behavior). Details are listed in Table 4.3.

Strategy	Advantages	To consider	Exemplary usage
Fastest	efficient, almost fresh	always network	first content
Cache first	efficient	never updated	stable data
Network first	fresh	timeout	refreshing
Cache only	efficient	never loaded	cached data
Network only		(regular behavior)	

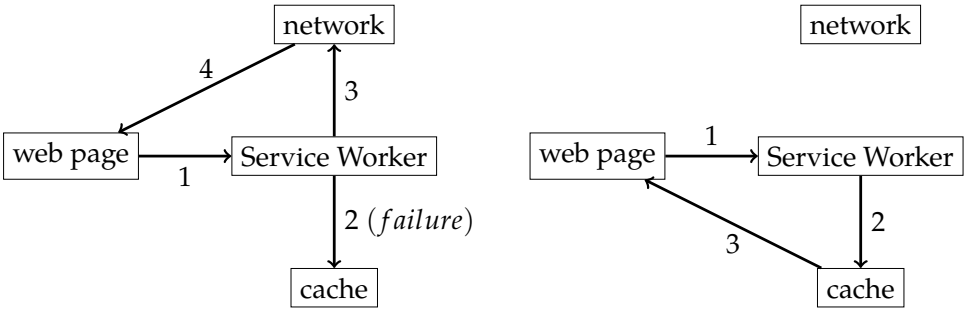
Table 4.3: Caching strategies available using Service Workers

In all of these scenarios (excluding network only), whenever a network request succeeds, the cache is updated with the new response, even though the user might have been presented with an older cached version.



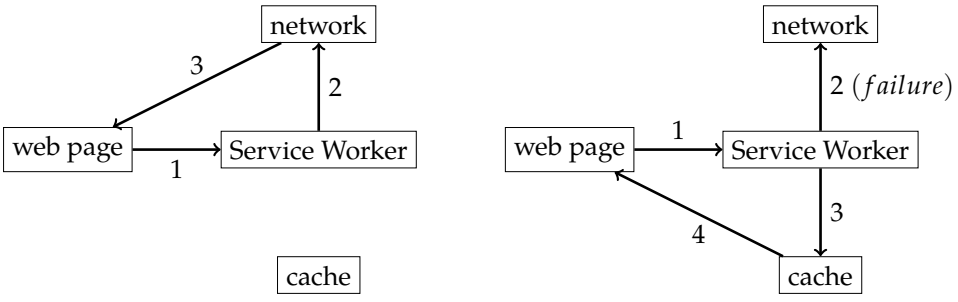
a) resource was not cached *b) resource was already cached*

Figure 4.2: "Fastest" caching strategy



a) resource was not cached *b) resource was already cached*

Figure 4.3: "Cache first" caching strategy



a) network available *b) network unavailable*

Figure 4.4: "Network first" caching strategy

4.4 Hardware access

Access to sensors and capabilities of mobile and other devices is possible through numerous APIs provided by web browsers. Many of the features listed in Table 4.4 and Table 4.5 used to be only accessible or achievable using the means of Adobe Flash, Java applets or not at all.

Feature	Native	HTML5 & CSS3 & PWA
Access to microphone*	●	● Media Capture
Access to camera*	●	● Media Capture, Stream API
Graphics	●	● CSS transform, Canvas (WebGL)
Device motion	●	● DeviceMotion
Access to clipboard	●	● Clipboard API
Files (reading)	●	● File API
Threading	●	● Web Workers
Data storage	●	● Web Storage, IndexedDB
Server-side push	●	● Server-Sent Events
Cryptography	●	● Web Cryptography API
● = full support, * = wide support in mobile browsers only		

Table 4.4: Widely available hardware access features

Features globally available according to Can I Use (see section B.1) and related APIs are listed in Table 4.4. Features and APIs that are less supported are listed in Table 4.5 (see section A.1 for API definitions).

Feature	Native	Old web	HTML5 & CSS3 & PWA
Access to gamepad	●	○	● Gamepad API
Monetization	●	◐	● Payment Request API
VR	●	○	◐ WebVR
Peer-to-peer	●	◐	● WebRTC
Media streaming	●	◐	● MediaStream Recording
Seamless sing-in	●	○	● Credential Management
Battery	●	○	● Battery Status API
Beacons	●	○	● Beacon API
Vibrations	●	○	● Vibration API

● = full support, ◐ = partial support, ○ = no support

Table 4.5: Partially available features

Some of these features have been already described in literature, for example *HTML5 and CSS3* by Castro and Hyslop and *The Modern Web* by Gasston. Considering the number of the available APIs, it is possible to build rich web applications which utilize these APIs.

5 Creation of a Progressive Web App

Since there is no exact definition of a Progressive Web App (PWA), it is usually assumed that a web application is a PWA if it meets the criteria required for the app install banner to show or if it achieves a good score for the fulfillment of these requirements checked by the Lighthouse tool (see Appendix C). Existing web applications can also implement PWA features step-by-step, for example only add Service Workers for caching.

5.1 Prerequisites

5.1.1 HTTPS

In order to create a downright PWA, a HTTPS-capable HTTP server with a valid and trusted certificate is needed, since HTTPS is a requirement for Service Workers and other APIs. Development on a local machine without internet access is possible using a local server, for example the Web Server for Chrome or XAMPP, but requires SSL setup. I chose to develop the YSoft SafeQ mobile terminal application on a web hosting that offers free SSL certificates issued by Let's Encrypt, which are trusted in most web browsers [29]. Self-signed certificates or custom certificate authorities can be used, but they have to be installed on every device which wants to use the PWA.

With a HTTPS-capable server, it is also recommended to forward HTTP traffic to HTTPS and force HTTPS using HTTP Strict Transport Security (HSTS) header. Security can be further enhanced by using Content Security Policy and some other headers. A minimal example of Apache server configuration can look like Listing 1.

```
RewriteEngine On

RewriteCond %{HTTP:X-Forwarded-Proto} =http [OR]
RewriteCond %{HTTP:X-Forwarded-Proto} =""
RewriteCond %{HTTPS} !=on
RewriteRule ^(.*)$ https://%{SERVER_NAME}/$1 [R=301,L,QSA]
```

Listing 1: Example Apache configuration

5.2 Recommendations

5.2.1 HTTP/2

In order to reduce loading times, HTTP/2 is recommended, which requires HTTPS and a capable server. HTTP/2 also impacts the recommended structure of an application. While with HTTP 1.1 it is useful to concatenate files in order to reduce the number of connections, HTTP/2 utilizes a single connection with streams, so concatenation is no longer required. Keeping a larger number of smaller files is even recommended. Caching can be done more efficiently, because a small change does not invalidate a data-heavy asset, which would need to be re-sent [30].

5.2.2 Gulp

Although a build process is not required, I used Gulp to automate deployment, introduce static code quality analysis, resource optimization (minification and compression) and some code generation, which is described later.

```
gulp.task("releaseJS", ["lintJS"], function(){
  return gulp.src(["lib/**/*.js", "src/*.js"])
      .pipe(concat("script.js"))
      .pipe(uglify())
      .pipe(gulp.dest("release"));
});
```

Listing 2: File concatenation and minification with Gulp

5.3 Application shell

It is recommended to split data and the application shell [31], in other words common HTML, CSS and Javascript files which form the user interface. In the case of this PWA, all data is loaded from a remote API. This separation allows efficient caching of the application files and optionally the data by applying proper caching strategies, described previously in section 4.3.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>YSoft SafeQ Terminal</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

Listing 3: Minimal HTML of the application shell

The application shell model is not universally suitable though, because no data is available on the very first load and it has to be fetched with another query after the whole application shell is loaded. This makes the first load of the application unnecessarily inefficient.

This effect can be eliminated for example using server-side rendering, which means generating and including the first bulk of data in the application source. Another option became available with HTTP/2 and its feature called push. It allows the inclusion of a file which was not requested by the client in the stream, effectively supplying the first bulk of data with the application files.

For development purposes, I keep separate components of the application in several files, which are concatenated during deployment. While this is counter-productive when using HTTP/2 as mentioned previously, the YSoft SafeQ mobile terminal is going to be served over HTTP 1.1.

5.4 Shims and polyfills

"A shim is a library that brings a new API to an older environment, using only the means of that environment. A polyfill is a shim for a browser API. It typically checks if a browser supports an API. If it doesn't, the polyfill installs its own implementation." [32, Chapter 30] Web applications include polyfills in order to use newest Javascript features and maintain backward compatibility. For the YSoft SafeQ mobile terminal, I included polyfills for `Promises`, `Fetch`, `classList`, `CustomEvent`, `getUserMedia` and `TEMPLATE` element (see Appendix C).

5.5 Service Worker caching

To introduce offline functionality and improve performance, I used SW Precache through Gulp to generate a Service Worker script (Listing 4).

```
gulp.task('generate-service-worker', function(callback) {
  var path = require('path');
  var swPrecache = require('sw-precache');
  var rootDir = 'release';

  swPrecache.write(`${rootDir}/service-worker.js`, {
    staticFileGlobs: [rootDir + '/**/*.{js,html,css,png,jpg,gif,svg,eot,ttf,woff}'],
    stripPrefix: rootDir
  }, callback);
});
```

Listing 4: Service Worker generation using SW Precache in Gulp

No complex rules or caching strategies are needed in this case, because all application files belong to the application shell. With this setup, API calls will be treated as network only, which is needed for POST queries to function properly. If the YSoft SafeQ server API complied with the REST rules for GET and POST methods (read operations would exclusively use GET and vice-versa) [33], it would be more appropriate to use the fastest or the network first caching strategy for GET requests.

To register the resultant script as the Service Worker, I used the exemplary `service-worker-registration.js` provided by SW Precache. A minimal example of SW registration including feature detection would look like Listing 5.

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/service-worker.js');
}
```

Listing 5: Service Worker registration

5.6 Web App Manifest

A basic Web App Manifest can be generated using a tool, I used RealFaviconGenerator. It generates all appropriate icon sizes, the

`manifest.json` file (Listing 6) and optionally all the proprietary application meta tags and the `browserconfig.xml` file mentioned earlier, for best compatibility with older devices.

```
{
    "name": "YSoft SafeQ Mobile Terminal",
    "short_name": "YSoft SafeQ",
    "icons": [
        // ...
        {
            "src": "/android-chrome-192x192.png",
            "sizes": "192x192",
            "type": "image/png"
        },
        // ...
    ],
    "theme_color": "#ffffff",
    "background_color": "#ffffff",
    "start_url": "/index.html",
    "display": "standalone"
}
```

Listing 6: manifest.json

In the application manifest, I added a `short_name` which is required for the "add to homescreen" functionality. The `orientation` can be limited to `portrait` or `landscape`. Without it the application works in both orientations thanks to being responsive. For applications that have this manifest and satisfy some other criteria, browsers may show "app install banners" which notify users about the possibility of adding the application to their home screen.

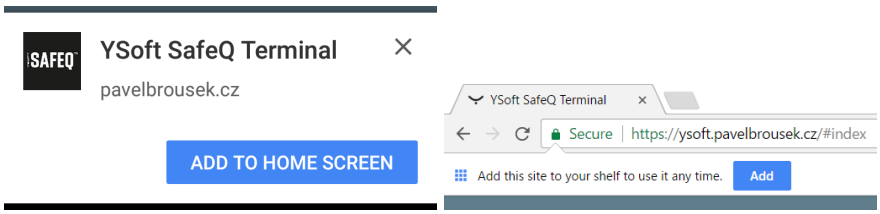


Figure 5.1: App install banners in Google Chrome

5.7 Input

The core of the input for the YSoft SafeQ mobile terminal is a terminal server endpoint URL, which is used to manipulate a printer. This URL includes the local IP address of the terminal server, an API route prefix and a printer ID, for example `https://10.0.13.134:5022/et/v1/1`.

There are several ways which the mobile terminal can use to obtain this URL. A new way of launching the mobile terminal is to append the terminal server endpoint URL to the URL of the web application. An example would be:

`https://ysofters.com/?https://10.0.13.134:5022/et/v1/1`

The address can be shorten by leaving out the protocol and minifying the IP address, port number and terminal ID by converting them to numbers with a higher base, for example 16 (hexadecimal), 32 (highest natively possible in Javascript) or 64 (letters, numbers, underscores and hyphens):

```
// base-32 compression of the IP address
var ip = "10.0.13.134";
var ipBinary = ip.split(".").map(function(a){return
 ↪  ("00000000"+parseInt(a,10).toString(2)).slice(-8);}).join("");
var ipNumeric = parseInt(ipBinary, 2);
ipNumeric.toString(32); // "5003c6"

// base-64 compression of the port number
var port = 5022;
var port64base = "";
var base64map = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-";
while (port > 0) {
  var remainder = port % base64map.length;
  port62base = base64map[remainder] + port64base;
  port = parseInt(port / base64map.length);
}
port64base; // "BOe"
```

Listing 7: IP compression in Javascript

This results in a shorter version of the URL:
`lhttps://ysofters.com?KAA2G:BOe/B`

An IPv4 address can have at most 32 bits, the port number 16 bits and up to 20 bits are reserved for the terminal ID, which makes up to 13 base64-encoded characters with 2 separators.

This extended application URL can be used by any device with a web browser, without the need of previous installation. The URL can be stored in the same ways as the sole terminal server endpoint URL and some other ones as well.

5.7.1 QR codes

YSoft SafeQ mobile terminal has to be able to accept QR codes and read their content. Multiple features can be used in a progressive-enhancement manner. In order to create an input control that allows the user to pick a file, restrict it to images only if possible and use the camera to capture a new image if possible, the code in Listing 8 suffices.

```
<input type="file" accept="image/*" capture id="qrcodeinput">
```

Listing 8: Input control for images

This input control is backward compatible with any web browser. The downside is that the picture is taken first and then parsed, so the user has to be able to take a clear picture of the QR code without knowing if it is good enough. A better approach is to use the `getUserMedia` API which grants access to the camera input as a stream, then show the data using a `video` element, capture frames on a `canvas` and parse the input simultaneously in order to get the result as fast as possible.

```
if (navigator.mediaDevices.getUserMedia) {
  var constraints = {video:{
    facingMode:  {ideal: "environment"},
    aspectRatio: {ideal: 1}
  }};
  navigator.mediaDevices.getUserMedia(constraints).then(function (stream) {
    video.srcObject = stream;
    video.onloadedmetadata = function() { video.play(); };
    setTimeout(scan, 1000);
  });
}
```

Listing 9: Accessing the camera using `getUserMedia`

It is also possible to store the extended application URL described previously in a QR code which can be read by any QR code reader.

5.7.2 Bluetooth beacons

Bluetooth beacons can hold a URL, which can be read by applications when the device they are running on is in the range of the beacon. This is used by the native YSoft SafeQ mobile terminal to automatically discover nearby printers. In this case the beacon only includes a compressed URL of the terminal server endpoint, so it can only be parsed by the native YSoft SafeQ mobile terminal. At the time of this writing, it is not directly possible for web applications to be notified about nearby beacons, neither to manually scan for them.

On the other hand, it is possible to store the extended application URL in a beacon, which allows users nearby to be notified and launch the application without having it installed, with the printer already being selected. An example of this is the Physical Web [34], which involes Eddystone format-compatible beacons that broadcast URLs. Users may be notified by the OS, e. g. Nearby Notifications in Android, or a browser, at the time of writing Google Chrome.

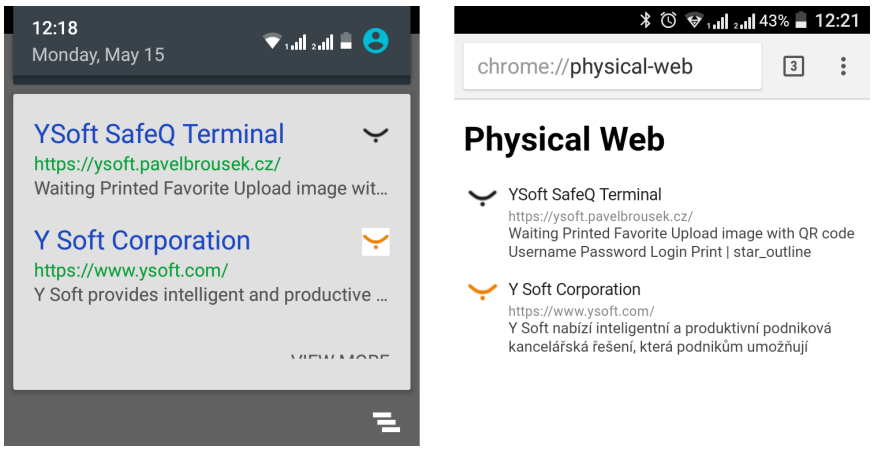


Figure 5.2: Physical Web notification and discovery

The main limitation of this approach is the 17 character limit of the URL, which is mostly occupied by the terminal server IP, port number and terminal ID. In order to use Eddystone beacons in the web-based

YSoft SafeQ mobile terminal, a URL shortener or some sort of a local DNS would have to be involved. With proper setup, the application could be discovered and used even without internet access.

5.7.3 NFC

Near field communication (NFC) tags can also contain a URL. Although web applications cannot access data in NFC tags, so it is not possible to read a terminal server endpoint URL this way, the extended application URL can be used to launch the application. NFC is supported on a notable portion of mobile phones. The user has to enable NFC in the settings, place the device near an NFC tag and confirm the URL load, similar to a QR code. In this case no application is needed, because the operating system itself handles NFC and the website loading.

5.8 Quering a remote API

Using Javascript, it is possible to communicate with a server in the background and send or retrieve data. This is referred to as AJAX, Asynchronous JavaScript and XML. For this purpose there is the `XMLHttpRequest` object and newer `Fetch` API.

By default, it is only possible to communicate with servers with the same origin (protocol and domain) as the application has. Remote servers with differing origin have to implement Cross-Origin Resource Sharing (CORS). More specifically, it is not at all possible to send HTTP requests from an HTTPS origin. This is called mixed content and it is forbidden for security reasons.[1]

In the case of the YSoft SafeQ server, this presented an obstacle, because the server uses an SSL certificate signed with a private certification authority, which is not trusted by regular web browsers unless installed manually. If the setup does not change, all users of the web application have to install the SSL certificate manually.

1. Further restrictions may be imposed using Content Security Policy.

```
POST /et/v1/4/auth HTTP/1.1
Host: 10.0.0.0
Accept: application/json
Content-Type: application/json
Origin: https://webapp.ysoft.com

HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://webapp.ysoft.com
Access-Control-Expose-Headers: Auth-SQTA-Token, YSoft-SQTA-api-version, Vendor
Content-Type: application/json
```

Listing 10: Request/response example with `Access-Control` headers

Considering CORS, the server is required to send `Access-Control` headers (Listing 10), because it has a differing origin and also answer preflight requests (Listing 11), because non-"simple" HTTP headers are being sent and JSON is being used, which is not on the safe list defined by the CORS standard (see section A.1). The YSoft SafeQ server software had to be changed to use a CORS middleware that satisfies these requirements.

```
OPTIONS /et/v1/4/auth HTTP/1.1
Host: 10.0.0.0
Accept: application/json
Content-Type: application/json
Origin: https://webapp.ysoft.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Auth-SQTA-Token, Content-Type, Accept

HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://webapp.ysoft.com
Access-Control-Expose-Headers: Auth-SQTA-Token, YSoft-SQTA-api-version, Vendor
Access-Control-Allow-Methods: GET,POST,OPTIONS
Access-Control-Allow-Headers: Auth-SQTA-Token, Content-Type, Accept, Origin
Content-Length: 0
Content-Type: text/plain
```

Listing 11: Preflight request example

5.9 Security

Security risks of web applications are relatively small due to their higher-level nature. A number of safety measures is done on a lower level then the application itself. Other possible vulnerabilities which

have to be addressed by the YSoft SafeQ server itself are not listed in this section.

5.9.1 HTTPS

Serving the application over HTTPS forces the use of HTTPS for any subsequent requests. HTTPS guarantees integrity and privacy of sent and received data, it also prevents session hijacking and some man-in-the-middle attacks. Together with Content Security Policy it prevents code injection (also known as cross-site scripting, XSS). Thanks to forward secrecy, an HTTPS communication record cannot be decoded without obtaining secret strings which are not sent over the network.

5.9.2 Credentials and session management

User credentials are either not stored at all, or saved securely using the browser's Credentials Management API. They are only sent over a secure HTTPS connection. YSoft SafeQ API session is held using short-living tokens, which change often. The use of tokens also prevents Cross-Site Request Forgery (CSRF) attacks.

A security concern, not exclusive for the web-based application, is that a user may be provided with an IP address of an attacker's fake YSoft SafeQ server, which would steal credentials used for logging in. This is addressed in the native application by only trusting a single Y Soft certification authority.

Similar measure cannot be done on the web because SSL/TLS-related actions happen at a lower level. The web application itself can only test if a server uses valid SSL/TLS including a valid certificate or not, it cannot read the certificate details. The web application can be secured by white-listing YSoft SafeQ terminal servers' IP addresses in the application source code. This would require the application to be configured by administrators of those YSoft SafeQ servers.

Another option is the use of some kind of client white-listing or client authentication, for example SSL client certificate authentication, which requires that all clients are configured before using the application.

5.9.3 Permissions, user approval

Access to most hardware features, such as the camera or the file system, has to be explicitly allowed by the user so there are no privacy concerns. Third-party trackers can only be inserted by the application author. Even access by most browser extensions, which can include malware, can be eliminated using the combination of HTTPS and Content Security Policy.

5.9.4 Source code

Web pages and applications have their source code publicly available to any user, because they are distributed as plain text and interpreted by web browsers. Although the source code can be obscured to be less readable, the security by obscurity principle, which is favored by some developers, cannot be applied.

5.9.5 Updates and patches

Web browsers are more likely to receive security updates and users are more likely to keep their browser updated, compared to many single applications. This reduces the risk of being affected by vulnerabilities found in underlying libraries for example.

Depending on the caching strategy, web applications can be updated instantly and users do not have to manually trigger the update process, so vulnerabilities in a web application itself can be fixed in a timely manner.

6 User interface

The user interface is created using Material Design Lite components. This library is suitable for PWAs for two main reason. The first being that as of April 2017 Android, which relies on Material Design, is the only mobile operating system with fully PWA-capable web browsers. The second being that this library has no external dependencies and consists of little number of files, which makes it suitable for including in the application shell. It was also developed with performance in mind and it is mobile-friendly.

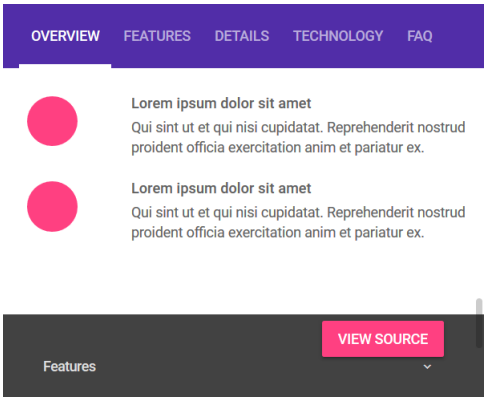


Figure 6.1: Example of Material Design Lite

As with any web application, it is possible to use any kind of user interface and it is also possible to customize it according to the operating system, so it is even possible for a single application to provide OS-specific user interface while mostly keeping a single code base.

6.1 Branding

Web applications can be easily customized, e. g. have various distributions with a varying logo. These separate versions can run on differing URLs and they can use different logo, application version or configuration. This is an advantage over the current native YSoft SafeQ mobile terminal, which would need separate build processes and separate

application records in the application markets, considerably more difficult to maintain.

Material Design Lite can be also adapted to use desired color palette. It uses the SASS CSS preprocessor and colors are parametrized as variables, so they can be adapted to match a corporate identity, for example. For the YSoft SafeQ mobile terminal, I also built custom version of Material Design Lite with colors used in the current native application.

6.2 Design guidelines

In cross-platform application development, there are several ways of coping with the user interface (UI), most notably input controls, icons and gestures, in terms of OS customization. One way is to develop the application with a unique design, which is not intuitive for any platform. A second option is to choose one of the operating systems and its design guidelines, making the application more familiar for only one platform. Third option, most difficult to achieve but most user-friendly, is to adapt the design to match the user's operating system. Native YSoft SafeQ mobile terminal combines the first and the last option, it features an almost universal design with platform-specific input controls.

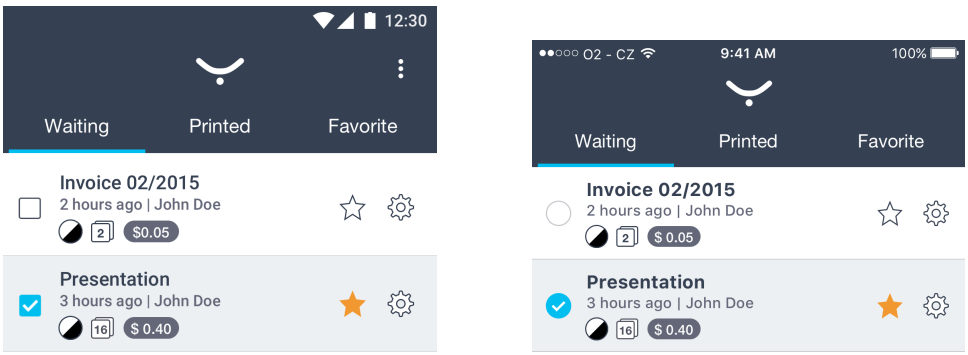


Figure 6.2: Android and iOS versions of the application UI

Native applications use input controls and elements provided by the operating system, so the per-OS customization is fairly easy. In the web application, this is more difficult to achieve because the OS

has to be detected and the UI element has to be reconstructed with the proper design. For the purpose of this thesis, I decided to keep Material Design, because Android is at the time of this writing the only fully supported operating system.

After consulting the design and the user interface with the interaction designer at Y Soft, I adjusted the design to be more in line with the Y Soft's design guidelines and recommendations aimed on application developers. The resultant appearance can be seen in Appendix D.

7 Conclusion

The objective of this thesis was to analyze the Progressive Web App technology and implement a YSoft SafeQ mobile terminal using this technology. The application had to be able to access the camera to scan QR codes, query a remote API and allow users to manage their print jobs.

In the opening chapter I listed various alternatives for cross-platform mobile application development, including some of their internals. One of them were hybrid web applications, which might get replaced by Progressive Web Apps.

In the subsequent chapter, I compared capabilities of native and web-based applications. Although some features are not available to web applications, there are numerous device capabilities that these applications can utilize.

The following chapter lists advantages of Progressive Web Apps over native applications and older web technologies. I analyzed the two most important innovations, Service Workers and Web App Manifest.

The last chapter describes the application implementation. It is divided into sections based on the APIs involved. Difficulties during development and their solutions are mentioned, too. In this chapter I also reference tools and libraries used for development.

The application can be further enhanced by customizing the design based on the operating system, implementing more features available through the YSoft SafeQ server API and by improving localization, for example by adding right-to-left languages support.

The web-based YSoft SafeQ mobile terminal was appreciated for the ease of branding, development and deployment. Because it does not need to be published in an app marketplace, it might be useful in countries where access to some app marketplaces is limited.

The deployment of this web application requires changes in YSoft SafeQ server software, namely cross-origin resource sharing (CORS) support. This requirement was submitted for approval by the security department and by product management. After the requirement is fulfilled and the application is deployed publicly, it can be used by YSoft SafeQ users to confirm their print jobs.

Bibliography

[1] Pete LePage. *Your First Progressive Web App*. Google. URL: https://developers.google.com/web/fundamentals/getting-started/codelabs/your-first-pwapp/ (visited on 11/13/2016).

[2] *Google I/O 2016 keynote*. Google, May 18, 2016. URL: https://www.youtube.com/watch?v=862r3XS2YB0&t=1h40m49s (visited on 11/13/2016).

[3] *Mobile Deeplinking project*. MobileDeepLinking.org. 2014. URL: http://mobiledeeplinking.org/ (visited on 11/13/2016).

[4] Ralph Hauwert. "The future of scripting in Unity". In: *The Unity Blog* (May 20, 2014). URL: https://blogs.unity3d.com/2014/05/20/the-future-of-scripting-in-unity/ (visited on 11/13/2016).

[5] *Xamarin Guides*. Chap. Part 1 – Understanding the Xamarin Mobile Platform. URL: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/ (visited on 11/13/2016).

[6] *Codename One*. Codename One LTD. 2012. URL: https://www.codenameone.com/ (visited on 11/13/2016).

[7] *Qt Documentation*. 5.7. The Qt Company Ltd. 2016. Chap. Deploying Qt Applications. URL: http://doc.qt.io/qt-5.7/deployment.html (visited on 11/13/2016).

[8] *Titanium*. Appcelerator Inc. 2015. URL: http://www.appcelerator.org/#titanium (visited on 11/13/2016).

[9] *Welcome to NativeScript*. Telerik AD. 2016. Chap. What is Android Runtime for NativeScript? URL: http://docs.nativescript.org/runtimes/android/overview (visited on 11/13/2016).

[10] *Apache Cordova*. The Apache Software Foundation. 2015. URL: https://cordova.apache.org/ (visited on 11/13/2016).

[11] *Adobe PhoneGap*. Adobe Systems Inc. 2016. URL: http://phonegap.com/ (visited on 11/13/2016).

[12] Christian Heilmann. "51: Graceful degradation versus progressive enhancement". In: *Opera web standards curriculum*. Feb. 3, 2009. URL: https://github.com/operasoftware/devopera-static-backup/blob/master/http/dev.opera.com/article

s/view/graceful-degradation-progressive-enhancement/index.html (visited on 11/13/2016).

[13] "Use Cases and Requirements for Installable Web Apps". Unofficial Draft. In: (Nov. 10, 2016). URL: https://w3c-webmob.github.io/installable-webapps/ (visited on 04/08/2017).

[14] Max Lynch. "What are Progressive Web Apps?" In: *The Official Ionic Blog* (May 18, 2016). URL: https://blog.ionic.io/what-is-a-progressive-web-app/ (visited on 04/08/2017).

[15] *Progressive Web App Checklist*. WebFundamentals community. Feb. 9, 2017. URL: https://developers.google.com/web/progressive-web-apps/checklist (visited on 04/08/2017).

[16] *Progressive Web Apps. A new way to deliver amazing user experiences on the web.* WebFundamentals community. URL: https://developers.google.com/web/progressive-web-apps/ (visited on 11/13/2016).

[17] *Web Fundamentals*. WebFundamentals community. Apr. 14, 2017. URL: https://developers.google.com/web/fundamentals/ (visited on 04/16/2017).

[18] Adam Bar. *What Web Can Do Today. An overview of the device integration HTML5 APIs*. Mar. 28, 2017. URL: https://whatwebcando.today/ (visited on 04/16/2017).

[19] *Progressive Web App Summit 2016 playlist*. Google Chrome Developers, Sept. 8, 2016. URL: https://www.youtube.com/playlist?list=PLNYkxOF6rcIAWWNR_Q6eLPhsyx6VvYjVb (visited on 04/16/2017).

[20] *Android Instant Apps*. Native Android apps, without the installation. Android Open Source Project. URL: https://developer.android.com/topic/instant-apps/index.html (visited on 04/22/2017).

[21] team of authors. IANA. May 19, 2017. URL: https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml (visited on 05/22/2017).

[22] Fahrbot PRI. *Meta Widget*. Aug. 31, 2016. URL: https://play.google.com/store/apps/details?id=fahrbot.apps.metawidget (visited on 04/16/2017).

[23] pwa.rocks contributors. *Progressive Web Apps*. A selection of Progressive Web Apps. Jan. 30, 2017. URL: https://pwa.rocks/ (visited on 04/16/2017).

[24] Jacob Rossi. "The Progress of Web Apps". In: (June 17, 2016). URL: https://medium.com/web-on-the-edge/progressive-web-apps-on-windows-8d8eb68d524e (visited on 04/16/2017).

[25] James D. Bloom. "Problems with Application Cache". In: (June 24, 2012). URL: http://blog.jamesdbloom.com/ProblemsWithApplicationCache.html (visited on 04/23/2017).

[26] *Instant-loading Offline-first*. Google Chrome Developers, June 20, 2016. URL: https://youtu.be/qDJAz3IIq18?list=PLNYkxOF6rcIAWWNR_Q6eLPhsyx6VvYjVb&t=281 (visited on 04/23/2017).

[27] Elizabeth Castro and Bruce Hyslop. *HTML5 and CSS3*. visual quickstart guide. Czech. 1st ed. translated from English. Brno: Computer Press, 2012. ISBN: 978-80-251-3733-8.

[28] Peter Gasston. *The Modern Web*. Multi-Device Web Development with HTML5, CSS3, and JavaScript. Czech. 1st ed. translated from English. Brno: Computer Press, 2015. ISBN: 978-80-251-4345-2.

[29] Let's Encrypt. "Certificate Compatibility". In: (Dec. 5, 2016). URL: https://letsencrypt.org/docs/certificate-compatibility/ (visited on 04/29/2017).

[30] *Instant loading with HTTP/2*. Google Chrome Developers, June 20, 2016. URL: https://www.youtube.com/watch?v=G62aCRIlONU&index=9&list=PLNYkxOF6rcIAWWNR_Q6eLPhsyx6VvYjVb (visited on 04/29/2017).

[31] Addy Osmani. "The App Shell Model". In: (Feb. 9, 2017). URL: https://developers.google.com/web/fundamentals/architecture/app-shell (visited on 04/29/2017).

[32] Axel Rauschmayer. *Speaking JavaScript*. An In-Depth Guide for Programmers. Ed. by Amy Jollymore Simon St. Laurent. 2nd ed. O'Reilly Media, 2015. URL: http://speakingjs.com/es5/index.html.

[33] RestApiTutorial.com. "Using HTTP Methods for RESTful Services". In: *REST API Tutorial* (). URL: http://www.restapitutorial.com/lessons/httpmethods.html (visited on 04/29/2017).

[34] Physical Web contributors. *Physical Web*. Walk up and use anything. URL: https://google.github.io/physical-web/ (visited on 05/13/2017).

A Web standards and other specifications

A.1 W3C standards and notes, Living Standards

Name	Link	Status	Last change
Ambient Light Sensor	w3.org/TR/ambient-light	WD	2016-08-30
Battery Status API	w3.org/TR/battery-status	CR	2016-07-07
Beacon	w3.org/TR/beacon	CR	2017-04-13
Canvas 2D Context	w3.org/TR/2dcontext	REC	2015-11-19
Clipboard API	w3.org/TR/clipboard-apis	WD	2016-12-13
Contacts API*	w3.org/TR/contacts-api	NOTE	2014-01-14
Content Security Policy 1.0	w3.org/TR/2012/CR-CSP-20121115	CR	2012-11-15
Content Security Policy 2	w3.org/TR/CSP2	REC	2016-12-15
Credential Management	w3.org/TR/credential-management-1	WD	2016-04-25
DeviceOrientation	w3.org/TR/orientation-event	CR	2016-08-18
File API (reading)	w3.org/TR/FileAPI	WD	2015-04-21
File API (writing)*	w3.org/TR/file-system-api	NOTE	2014-04-24
Fullscreen*	w3.org/TR/fullscreen	NOTE	2014-11-18
Fullscreen	fullscreen.spec.whatwg.org	LS	2017-02-24
Gamepad	w3.org/TR/gamepad	WD	2017-01-25
Geolocation API	w3.org/TR/geolocation-API	REC	2016-11-08
HTML Media Capture	w3.org/TR/html-media-capture	CR	2014-09-09
Indexed Database	w3.org/TR/IndexedDB	REC	2015-01-08
MediaStream Recording	w3.org/TR/mediastream-recording	WD	2017-04-05
Media Capture and Streams	w3.org/TR/mediacapture-streams	CR	2016-05-19
Messaging API*	w3.org/TR/messaging-api	NOTE	2014-01-14
Offline Web applications	html.spec.whatwg.org/multipage/browsers.html#offline	LS	2017-04-28
Payment Request API	w3.org/TR/payment-request	WD	2017-03-31
Proximity Sensor	w3.org/TR/proximity	WD	2016-07-19
Server-Sent Events	w3.org/TR/eventsource	REC	2015-02-03
Service Workers 1	w3.org/TR/service-workers-1	WD	2016-10-11
Vibration API	w3.org/TR/vibration	REC	2016-08-18
Wake Lock API	w3.org/TR/wake-lock	WD	2017-02-22
WebGL	khronos.org/registry/webgl/specs/1.0.3	other	2014-10-27
WebRTC	w3.org/TR/webrtc	WD	2017-03-13
WebVR	w3c.github.io/webvr/spec/latest	ED	2017-04-17
Web App Manifest	w3.org/TR/appmanifest	WD	2017-04-15
Web Bluetooth	webbluetoothcg.github.io/web-bluetooth/	DCGR	2017-04-24
Web Cryptography API	w3.org/TR/WebCryptoAPI	REC	2017-01-26
Web NFC API*	w3.org/TR/nfc	NOTE	2015-06-16
Web Share API	github.com/WICG/web-share/blob/master/docs/explainer.md	UNOFF	2016-05-30
Web Speech API	dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html	ED	2012-10-19
Web Storage	w3.org/TR/webstorage	REC	2016-04-19
Web Workers	w3.org/TR/workers	WD	2015-09-24

Updated 2017-04-26. * = discontinued, WD = Working Draft, CR = Candidate Recommendation, REC = W3C Recommendation, NOTE = Working Group Note, ED = Editor's draft, DCGR = Draft Community Group Report, LS = WHATWG Living Standard, UNOFF = unofficial

A.2 Requests For Comment (RFC)

Updated 2017-04-26.

Subject	RFC number	Link	Date
'tel' URI scheme	5341	tools.ietf.org/html/rfc5341	2008-09
'sms' URI scheme	5724	tools.ietf.org/html/rfc5724	2010-01
'geo' URI scheme	5870	tools.ietf.org/html/rfc5870	2010-06
'mailto' URI scheme	6068	tools.ietf.org/html/rfc6068	2010-10
HSTS	6797	tools.ietf.org/html/rfc6797	2012-11
HTTP/2	7540	tools.ietf.org/html/rfc7540	2015-05

A.3 Proprietary definitions

Name	Link
Add to Homescreen	developers.chrome.com/multidevice/android/installtohomescreen
Apple-Specific Meta Tag Keys	developer.apple.com/…/SafariHTMLRef/Articles/MetaTags.html
Browser configuration schema	msdn.microsoft.com/en-us/library/dn320426(v=vs.85).aspx
Pinned site metadata	msdn.microsoft.com/en-us/library/dn255024(v=vs.85).aspx

B Can I Use support tables

B.1 High support (> 80 %)

Feature	Support in %	IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mobile	Chrome/Android	Firefox/Android	IE mobile	Samsung Internet
Audio element	95.06	●	●	●	●	●	●	●	●	●	●	●	●
Canvas (basic support)	97.94	●	●	●	●	●	●	●	●	●	●	●	●
Clipboard API	85.10	◐	◐	◐	◐	◐	◐	◐	◐	◐	◐	○	◐
Content Security Policy 1.0	93.95	◐	●	●	●	●	●	●	●	●	●	◐	●
Cross-Origin Resource Sharing	95.33	●	●	●	●	●	●	●	●	●	●	●	●
CSS3 3D Transforms	94.60	◐	●	●	●	●	●	●	●	●	●	◐	●
DeviceOrientation & DeviceMotion	92.19	◐	●	●	●	○	●	●	●	●	●	●	●
File API	94.69	◐	◐	●	●	●	●	●	●	●	●	◐	●
Full Screen API	81.70	◐	◐	◐	◐	◐	●	○	◐	◐	◐	◐	◐
Gamepad API	80.43	○	●	●	●	●	●	●	○	●	●	○	●
Geolocation	95.02	●	●	●	●	●	●	●	●	●	●	●	●
HTTP/2 protocol	81.36	◐	●	●	●	◐	●	●	●	●	●	○	●
IndexedDB	93.77	◐	◐	●	●	●	●	●	●	●	●	◐	●
Inline SVG in HTML5	97.78	●	●	●	●	●	●	●	●	●	●	●	●
Internationalization API	82.81	●	●	●	●	●	●	●	●	●	○	●	●
Offline web applications	94.86	●	●	●	●	●	●	●	●	●	●	●	●
Promises	88.40	○	●	●	●	●	●	●	●	●	●	○	●
Selection API	95.06	●	●	●	●	●	●	◐	●	●	◐	●	●
Server-sent events	88.86	○	○	●	●	●	●	●	●	●	●	○	●
Session history management	94.58	●	●	●	●	●	●	●	●	●	●	●	●
Strict Transport Security	84.46	●	●	●	●	●	●	●	●	●	●	○	●
SVG fragment identifiers	82.61	●	●	●	●	◐	●	◐	●	●	●	●	◐
Video element	95.04	●	●	●	●	●	●	●	●	●	●	●	●
WAI-ARIA Accessibility features	88.60	●	●	●	●	●	●	●	●	●	●	●	●
Web Cryptography	91.00	◐	●	●	●	●	●	●	●	●	○	◐	●
Web Sockets	94.19	●	●	●	●	●	●	●	●	●	●	●	●
Web Storage - name/value pairs	95.41	●	●	●	●	●	●	●	●	●	●	●	●
Web Workers	94.22	●	●	●	●	●	●	●	●	●	●	●	●
WebGL - 3D Canvas graphics	92.08	●	●	●	●	●	●	●	●	●	●	●	●

Updated 2017-05-04 from caniuse.com. ● = supported, ◐ = partially supported, ○ = not supported

B.2 Low support

Feature	Support in %	IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mobile	Chrome for Android	Firefox for Android	IE mobile	Samsung Internet
Ambient Light API	7.53	○	●	◐	○	○	○	○	○	○	●	○	○
Battery Status API	69.16	○	○	●	●	○	●	○	●	●	●	○	●
Beacon API	75.00	○	●	●	●	○	●	○	●	●	●	○	●
Content Security Policy Level 2	76.46	○	●	◐	●	●	●	●	●	●	◐	○	●
Credential Management API	51.88	○	○	○	●	○	○	○	○	●	○	○	○
Custom protocol handling	33.93	○	○	●	●	○	●	○	○	○	○	○	○
Fetch	70.57	○	●	●	●	●	●	●	●	●	●	○	●
Filesystem & FileWriter API	55.54	○	○	○	●	○	●	○	●	●	○	○	○
getUserMedia/Stream API	72.24	○	●	●	◐	○	◐	○	◐	◐	●	○	???
HTML Media Capture	53.22	○	○	○	○	○	○	●	●	●	○	○	●
HTTP Live Streaming (HLS)	56.90	○	●	○	○	●	○	●	●	●	○	○	●
Media Source Extensions	77.93	◐	●	●	●	●	●	○	●	●	●	●	○
Payment Request API	27.91	○	◐	○	⚑	○	⚑	○	○	●	○	○	○
Permissions API	63.93	○	○	●	●	○	●	○	○	●	●	○	●
Proximity API	6.19	○	○	●	○	○	○	○	○	○	●	○	○
Push API	72.86	○	○	●	●	○	●	○	●	●	●	○	●
Service Workers	73.26	○	⚑	●	●	○	●	○	●	●	●	○	●
Speech Recognition API	58.95	○	○	⚑	◐	○	◐	○	○	◐	○	○	◐
Speech Synthesis API	74.15	○	●	●	●	●	●	●	○	●	○	○	○
Vibration API	75.79	○	○	●	●	○	●	○	●	●	●	○	●
Web App Manifest	58.57	○	○	○	●	○	●	○	●	●	○	○	●
Web Audio API	79.38	○	●	●	●	●	●	●	●	●	●	○	●
Web Bluetooth	51.69	○	○	○	●	○	●	○	○	●	○	○	○
WebRTC Peer-to-peer connections	65.19	○	●	●	●	○	●	○	●	●	●	○	●
WebVR API	0.00	○	●	⚑	⚑	○	○	○	○	⚑	○	○	○

Updated 2017-05-04 from caniuse.com. ● = supported, ◐ = partially supported, ⚑ = has to be enabled manually using a configuration flag, ○ = not supported, ??? = not stated

C Tools and libraries

Name	Link	Purpose
Apple Pay JS	developer.apple.com/reference/applepayjs	Payment Request equivalent
CSP Is Awesome	cspisawesome.com	Content Security Policy generator
Gulp.js	gulpjs.com	automation
Lighthouse	developers.google.com/web/tools/lighthouse	PWA audit
ManifeStation	webmanife.st	WAM generation from a website
Material Design Lite	getmdl.io	Material Design for web
mediaDevices-getUserMedia-polyfill	github.com/mozdevs/mediaDevices-getUserMedia-polyfill	getUserMedia polyfill
Promise Polyfill	github.com/taylorhakes/promise-polyfill	
PWAify	github.com/vladikoff/PWAify	packaging PWA for desktop
RealFaviconGenerator	realfavicongenerator.net	WAM and meta tags generator
remy's Polyfills	github.com/remy/polyfills	polyfills including classList
SASS	sass-lang.com	CSS preprocessor
Service Worker Precache	github.com/GoogleChrome/sw-precache	SW generator
Service Worker Toolbox	github.com/GoogleChrome/sw-toolbox	SW generation helpers
Web Manifest Validator	manifest-validator.appspot.com	WAM validation
Web Server for Chrome	github.com/kzahel/web-server-chrome	local HTTP server
Windows App Studio	appstudio.windows.com	packaging PWA for Windows
XAMPP	apachefriends.org	local HTTP server
window.fetch polyfill	github.com/github/fetch	

Updated 2017-04-29. PWA = Progressive Web App(s), WAM = Web App Manifest, SW = Service Worker(s)

D Application screenshots

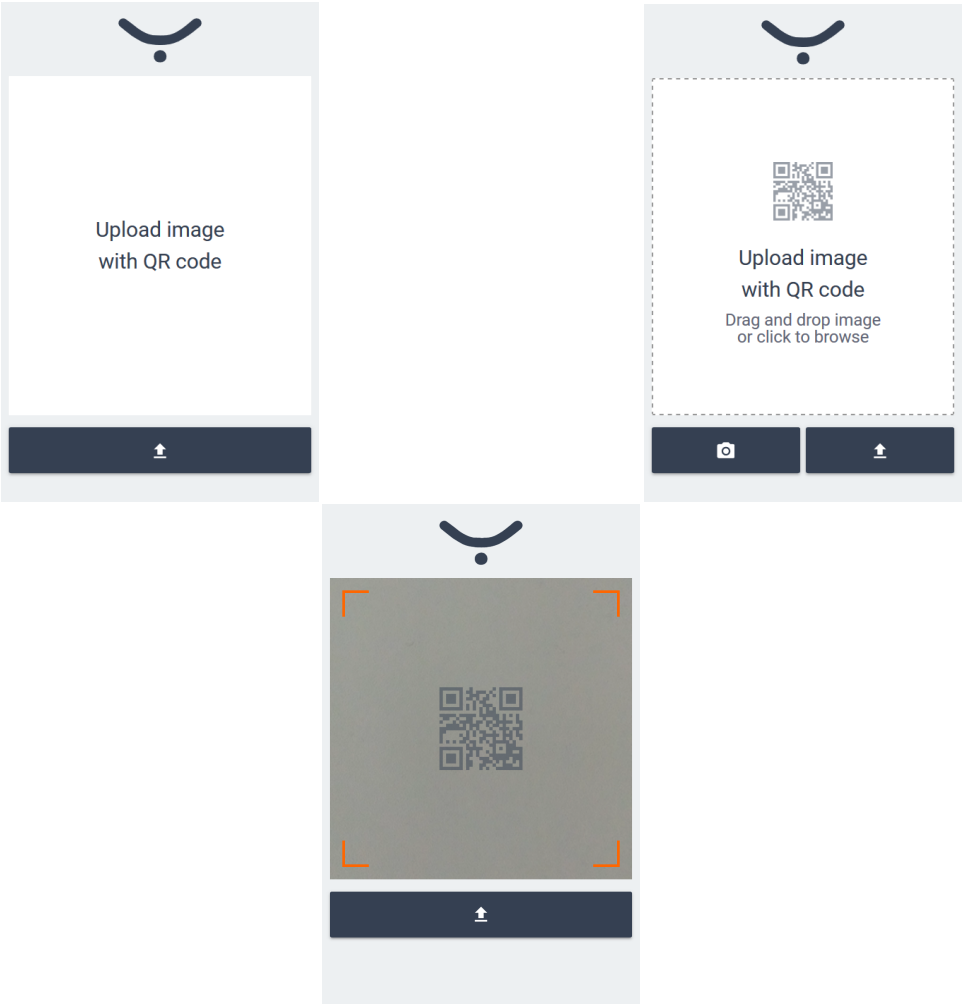


Figure D.1: Input screen on various devices

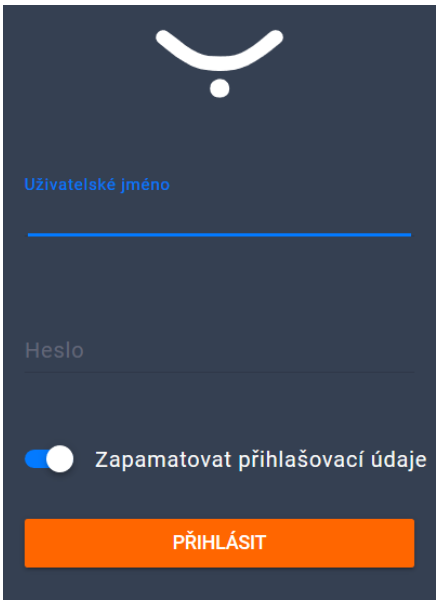


Figure D.2: Login screen (localized)

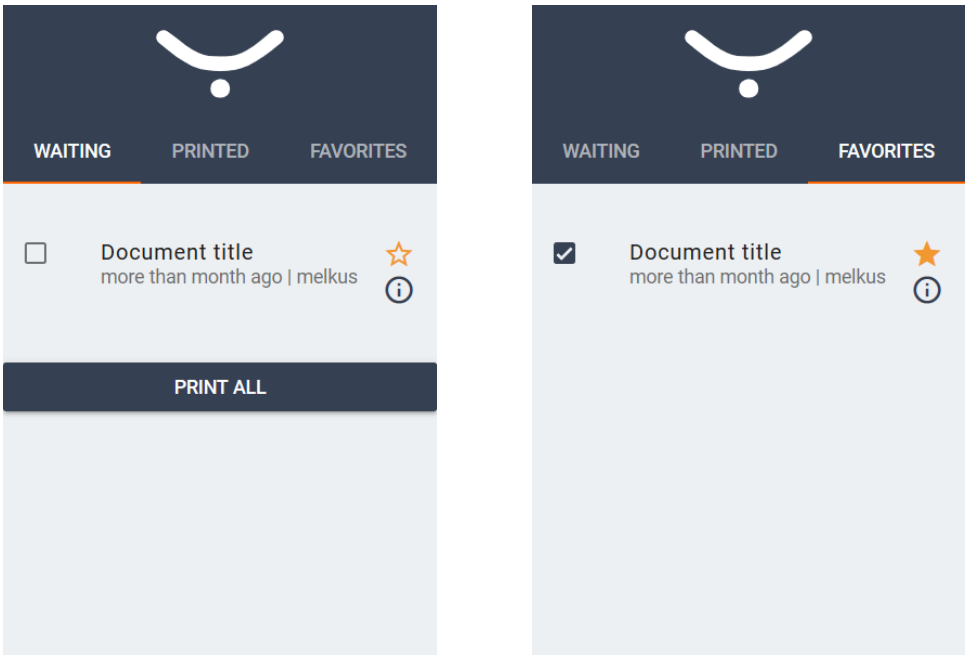


Figure D.3: Jobs screen