Linköping University | Department of Computer and Information Science

Master thesis, 30 ECTS | Datateknik

2018 | LIU-IDA/LITH-EX-A--18/037--SE

Progressive Web Applications and Code Complexity

– An analysis of the added complexity of making a web application progressive

---

Progressiva webbapplikationer och kodkomplexitet

Fabian Johannsen

Supervisor : Sahand Sadjadee
Examiner : Erik Berglund

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Fabian Johannsen

Abstract

Web applications have a common code base across multiple platforms, but have previously lacked some core features compared to native applications. However, recent web technology advancements have, in terms of functionality and user experience, reduced the gap between the two development approaches. Applications that leverages these technology advancements are dubbed progressive web applications, or PWA. This thesis explores the concepts of PWA and how it, in terms of code complexity, affects an Angular web application. The results show that implementing considered PWA features does not excessively increase the size of the application and that the overall added complexity is low. The complexity of PWA lies in all the new technology concepts, which are probably unfamiliar to most developers. To reduce this complexity, automated PWA tooling shows great promise, and using Angular PWA tooling when building Angular applications seems to minimize this complexity.

Acknowledgments

I would like to thank my examiner **Erik Berglund** for his valuable guidance throughout this thesis. I would also like to thank **Martin Kaldma** at Exsitec for always being supportive and genuinely interested in my work. A very special thanks goes to **Fanny Lindmark** because of many, many reasons.

Linköping, June 2018
Fabian Johannsen

Contents

List of Figures

List of Tables

1 Introduction

Developing platform independent applications can be expensive for any company. Having to develop applications for each specific platform is often a too resource-demanding and cost-inefficient task. Each platform may support different programming languages, all with their own set of propriety APIs, making the development of the same application, with regards to functionality and design, vastly different depending on which platform it is developed[12] [4]. Still, many companies need to support both desktop and mobile, and perhaps other platforms, for their applications.

Web applications, i.e. applications that are accessed via a web browser, can be used on every platform that supports browsers. This means that the application can be built once in the languages of the web, i.e. HTML, JavaScript and CSS, and still maintain a high accessability throughout every platform. The problem has previously been that web applications lack some core features compared to platform specific applications. Some of these features includes offline capabilities, hardware-access and inferior user experience across different browsers [4][3][32]. This in turn has in some cases made the development of a platform specific application more of a favourable choice, even though the cost is significantly higher.

Progressive Web Applications, PWA, is a novel technology concept with the intent of overcoming some of the above mentioned problems. Progressive web applications are leveraging the latest web technology advancements and it serves to make web applications as good as native applications by including some of the features web applications previously have lacked [21] [20]. Progressive web applications are supposed to be reliable, fast and engaging and include features such as offline support, web push notifications and deliver an app-like user experience.

Making a web application progressive can potentially increase the overall complexity of the application. Studies have shown that complex code negatively affects the maintainability[1] of the application, it makes the code more error-prone[17] and it generally makes the software less reliable [18]. Therefore, it is crucial to be able to measure the software complexity. An accurate use of software complexity metrics can then be used in cost projection, manpower allocation and program evaluation[15].

This thesis explores the concept of Progressive Web Applications and to what extent the software complexity is affected when considering implementing PWA features. An Angular application is developed with a set of PWA features that are specifically useful for web based business applications. The application is evaluated based on a set of software complexity metrics, which will result in a quantifiable complexity evaluation of the implemented features.

1.1 Aim

The underlying purpose for this thesis is first of all to explore the possibilities of PWA in web based business applications. The thesis also aims to answer if the developers at Exsitec (see chapter 2) should embrace the concepts of Progressive Web Applications, or if the technology is at an too novel state and would just add unnecessary complexity to their applications.

1.2 Research questions

The research questions for this thesis are stated as follows:

1. How does the implementation of PWA features in the Angular framework affect the complexity of an application?
2. How do you develop a web application with PWA features so that the code complexity of the application remains low?

1.3 Delimitations

Due to time constraints, the applications built for this thesis can not be considered production ready. The technologies used are cutting edge, meaning that some implementation approaches used in this thesis might not be valid in a not so distant future.

2 Background

This thesis was carried out at the company Exsitec and the division Digitalisation. Exsitec is an IT consultancy firm working mainly with ERP systems and business applications. The division Digitalisation is developing tailor made, web based business applications for a variety of different companies and they are focusing of delivering highly efficient and user friendly applications. In order to stay competitive on the market, Digitalisation is always investigating new technologies which may be beneficial for their applications.

Digitalisation is developing applications in the front-end framework Angular. They are using the language Typescript, which is a superset of JavaScript, because that is currently the main language of Angular. The front-end code is communicating with a back-end API mostly written in .NET C#.

3 Theory

This chapter contains the theoretical framework for this thesis. This chapter starts with an overview of mobile applications and mobile application development. It includes the characteristics of mobile applications and common ways to develop them. The chapter then continues with an overview of web applications and how to leverage the latest web technology advancements, i.e. PWA. An introduction to the Angular framework and tools used to build PWAs are presented. The chapter ends with a presentation of common complexity metrics and an evaluation of them.

3.1 Mobile Applications

A software application is a computer program which serves to perform a set of tasks or activities. Common issues for application development in general includes performance, reliability and availability and this applies for all applications independent of the platform they are developed for. A mobile application is just like a traditional software application, but with some additional technical obstacle to overcome.

Wasserman has in his research[35] listed some requirements which makes mobile applications different from conventional software applications. Wasserman argues that the following requirements are more commonly found in mobile applications:

- Potential interaction with other applications - A mobile device can use multiple different applications from multiple different sources, which makes the possibility of interactions between them greater.
- Sensor handling - A modern mobile device has several different sensors such as camera, microphone, accelerometer, touch screen, GPS. All of these could be accessible by a mobile application.
- Families of hardware and software platforms - There are several different mobile platforms and the applications may have to be custom built for a specific platform. A spe-

cific platform can have several different versions, and a developer needs to be aware of potential compatibility issues when developing for a specific platform.

- Security - The mobile platform is considered more vulnerable due the extensive downloading of applications from different sources.
- User interfaces - A mobile application cannot be tied to one user interface design since mobile devices come in all different shapes and sizes.
- Complexity of testing - A mobile application is more difficult to test since the platform for developing the application is not the same as where the application is going to be used. The application also needs to be tested in different environments, e.g. different network environments, and on different devices.
- Power consumption - A mobile device is dependent on its battery in order to work. A mobile application developer has to consider optimizing for maximum battery life.

3.1.1 Mobile Application Development

There are essentially three different ways one can develop a mobile application. A mobile application could be developed on the web platform as described in section 3.2. It could also be a native developed application or it could be a hybrid application[14]. These two latter techniques will be discussed below as well as a comparison between all three of them.

Native Application Development

When one thinks of mobile applications it is probably the native application that first comes to ones mind. A native application is downloaded, usually from Googles Play Store or Apples App Store, and installed on the mobile device. A native application is platform dependent, meaning that one has to use platform specific programming languages and APIs in order to develop an application. For example, if an Android application were to be developed one has to use Java (or Kotlin) and if an iOS application were to be developed one has to use Objective C or Swift. Because of platform specific APIs and tools, native applications can be developed with rich user experiences, heavy advanced graphics and with high performance [21].

Hybrid Application Development

As the name implies, a hybrid application leverages the single code-base which the web technology brings and is at the same time still able to access the native APIs. This is done by using a hybrid development framework, such as Apache Cordova, which provides a native wrapper for containing the web-based code and a JavaScript API that bridges all the web-based code to the corresponding platform API[21].

Web vs Native and Hybrid Application Development

The three mobile application development types have different strengths and weaknesses. Charland et al. wrote an article [4] about the comparison between native mobile application development and mobile web application development. They argue that web applications are cheaper to develop due to its compatibility on several platforms, but it is a performance and user experience trade-off compared to native development. Native code is usually faster because it is compiled, and not interpreted as web code is, and the access to proprietary APIs makes drawing of e.g. UIs faster which gives a more pleasant user experience.

Charland et al. also talk about the lack of hardware access for web applications. Since web applications are accessed through a web browser they cannot access the mobiles low level

hardware APIs, making native development more appropriate if one wants to access e.g. senors, the camera or the microphone.

Performance is an important attribute which increases the overall user experience. Charland et al. argue that latency and load times are important aspects of mobile application performance and that the web technology stack has some catching up to do compared to the native stack.

A hybrid application leverages some of the advantages of a web application, and maybe the most prominent one is the fact that it uses web code which can be used on many platforms. A hybrid application can use platform-specific APIs by the use of a JavaScript bridge provided by the hybrid application framework, and the application is limited to what that bridge is capable of translating.

3.2 Web Applications

A web application differs from conventional applications by being stored on a remote server, accessed on-demand and interpreted through a web browser. A web browser can interpret HTML, CSS and JavaScript which are the main languages for developing web applications. Every device that is capable of installing a web browser has the possibility to use a web application since it runs within the browser. A web application is therefore platform independent and the code written in the languages stated above only has to be written once in order to work on all the platforms[24]. A mobile web application is just a web application that is optimized to work on the mobile platform as well as any other platform [4].

3.2.1 Client-Server Model

A web application follows the client-server model which describes the relationship between a provider of a service (a server) and a service requester (clients). For a web application, the communication between a client and a server is through the Hypertext Transfer Protocol, or also referred to as HTTP. HTTP is an application-level protocol which resides on top of the TCP/IP communication protocol and serves to exchange or transfer hypertext.

The request-response messaging pattern in a web application can be seen in figure 3.1. It starts with a client (1) requesting a resource by sending an HTTP request (2). After a while the request reaches the server (3) and depending on the request the server might need to query a database (4) to return or post some data. When complete, the server sends an HTTP response (5) about the status of the executed request. Depending on the request, the server might send some arbitrary data retrieved from the database.

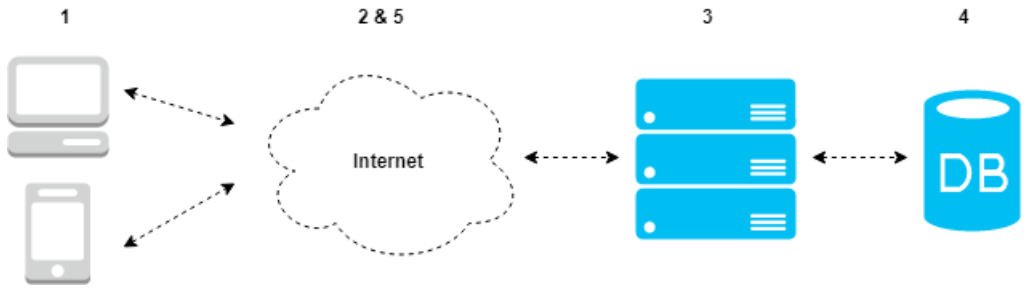

Figure 3.1: Request-response messaging pattern

3.3 Progressive Web Applications

There is no clear definition of what a progressive web application really is[33]. Though, progressive web applications have some fundamental quality attributes which are necessary for a web application to be called progressive. According to Google, they should be fast, reliable and engaging[27]. Each one of these quality attributes comes with a set of new techniques and guidelines of how to leverage the latest web technology advancements.

3.3.1 Considered PWA Features

Google has compiled a list of features which they believe are baseline requirements for a progressive web application[26]. These features include:

- The application is served over HTTPS - For security reasons, a PWA should be served over the more secure HTTP protocol
- The application is responsive - The design of the application should be user-friendly across platforms
- The application loads while offline - Regardless of the network state, the application should always be able to display content
- Metadata provided for Add to Home screen - The metadata should be provided in the web application manifest
- First load fast even on 3G - The application should load fast enough to maintain high user experience
- Site works cross-browser - The application should work accordingly on every major browser
- Page transitions do not feel like they block on the network - View transitions should not feel snappy or choppy
- Each page has a URL - Individual pages should be deep linkable via the URLs.

The above mentioned features are required in order for a web application to be considered progressive. There are more, non-required, features which a progressive web application may adapt. Some of these features includes[26][33]:

- Push Notifications - Re-engageable notifications as described in section 3.3.3
- Background Sync - Synchronize data while offline
- Payment Request - The capability of web payment using the Web Payment Request API
- Seamless sign-in process across devices - The capability of using the Credential Management API for sign-in processes.

3.3.2 Service Worker

A reliable web application works independent of the network state. Since a web application follows the Client-Server model, as explained in 3.2.1, the reliability of a web application has previously solely been dependent of the current network state in order to work. This means that if the network connection is lost no application data can be retrieved nor saved during the that time.

At the heart of every progressive web application is the service worker. A service worker serves to provide an offline experience for the users of a web application. A service worker is a client-side script that runs separated from the web application and on a separated JavaScript thread. It enables developers to programmatically cache and preload data so that the application can be loaded from the browser cache if the network connection is lost. One can think of a service worker as a dedicated, client-side network proxy that can intercepts network request sent to and from a web application[20]. A visual representation of a service worker can be seen in figure 3.2.

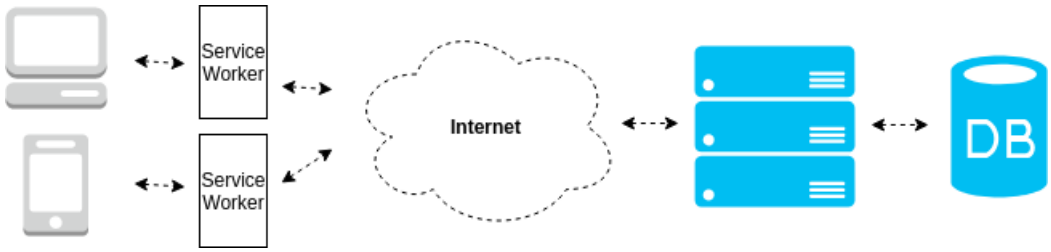

Figure 3.2: Visual representation of a service worker

Offline Support

A service worker has mainly six different events; Install, Activate, Message, Fetch, Sync and Push[34]. The service worker event that is used for supporting an offline experience is primarily the fetch event and the install event. In order for the fetch event to be triggered, the service worker needs to be installed and activated, which is explained below.

When a user first visits the web application, the service worker script, alongside the application, is downloaded from the server. When the download is complete, the application registers the service worker and the service worker install event is triggered. The Service worker registration, and the additional check for service worker support in the browser, can be seen in listing 3.1.

```
1 // App.js - Register the serivce worker
2 if ('serviceWorker' in navigator) {
3   window.addEventListener('load', function() {
4     navigator.serviceWorker.register('/service-worker.js');
5   });
6 } else {
7     console.log('Service Workers not supported by this browser');
8 }
```

Listing 3.1: Register Service Worker

If service workers are supported by the browser and the registration was successful, then the install event is fired in the service worker file. During this event, it is common to cache

assets that are necessary in order for the application to work offline. The install event, and the process of caching assets, can be seen in listing 3.2.

```
// service-worker.js - Install the service worker
self.addEventListener('install', function(event) {
   event.waitUntil(
   // Cache scripts, assets, etc.
    caches.open('cacheKey').then(function(cache) {
      return cache.addAll([
        '/app.js',
        '/styles.css',
        '/index.html'
      ]);
    })
  );
});
```

Listing 3.2: Install event

At some point in time the service worker will, probably, be needing an update. This is when the activate event of the service worker is used. Once the newer version of the service worker is active on the application server, the newer version will be downloaded and installed on e.g. page load or page refresh. The install event will be triggered and then it will enter a waiting state. This is because the old service worker is still in control over the application. Once all of the tabs and windows of the application are closed, the old service worker is terminated. Upon reopening the application, the new service worker enters the activate event. This whole process is due to the rather complex life cycle of the service worker. Imagine if the user has two tabs with the application open and one tab makes a page reload, i.e. downloading a new service worker. If that service worker were to be installed directly, then there would be two different version of the app running in two different tabs, which is most of the time an undesirable attribute. The activate event makes sure that the application versions are consistent for the user. When this event fires it is also common to manage the cache, e.g. deleting old caches. The activate event with cache management can be seen in code snippet 3.3.

```
self.addEventListener('activate', function(event) {
  var cacheWhitelist = ['newCacheKey'];
  event.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (cacheWhitelist.indexOf(key) === -1) {
          return caches.delete(key);
        }
      }));
    })
  );
});
```

Listing 3.3: Activate event

The service worker fetch event is triggered when a resource, controlled by the service worker, is fetched. A controlled resource is in this case the assets that were cached in the install event. A fetch event will intercept the network request when application resources are requested and respond with the proper cached resource (or in some cases go to the server and fetch fresh

data). In the listing 3.4 a fetch event is triggered due to a resource request by the application and the corresponding cached request is returned from the event. If there are no cached resources that corresponds to the request, the request is forwarded to the server. How a request is handled in the fetch event is up to the developer to decide. There are several different strategies that can be implemented, e.g. a network first strategy where the request is forwarded to the application server and if it fails (possibly because the user is offline), then fetch the resource from the cache. This process is what makes the application work offline.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response
        if (response) {
          return response;
        }
        // No cache hit - make a network request
        return fetch(event.request);
      }
    )
  );
});
```

Listing 3.4: Fetch event

3.3.3 Web Push Notifications

Push notifications is a technology that makes an application more re-engagable. As the name implies, a push notification is an arbitrary message that is pushed from a remote server to a users device. The Web Push API and the Web Notifications API [28], together with the service worker, has made it possible to send push notifications to web applications.

In order for an application server to send push notifications to a web application, the server needs to make an API call to a push service. A push service is responsible to validate and deliver a push message to a specific browser. Common push services includes FCM (Firebase Cloud Messaging), ASPNS (Apple Push Notification Service), MPNS (Microsoft Push Notification Service) and BBPS (BlackBerry Push Service)[19].

In order for a client, i.e. a browser, to receive a push message it has to subscribe to push notifications. This is done by PushManager interface of the Push API. With this interface it is possible to call the subscribe function which returns a PushSubscription object that was created by the push service. A PushSubcription object includes authentication keys, a push service endpoint, push subscription expiration time and more. This object is unique to the specific service worker that is in control of the application that called the subscribe function. In order for the application server to keep track of all the PushSubscription objects, it is common to store the object on the application database. It is also possible to unsubscribe from push notification using the PushManager interface. The push service is notified that a user wants to unsubscribe from push notifications and invalidates the PushSubscription object so that no push messages can be sent. The push notification subscription and unsubscription flow can be seen in figure 3.3.

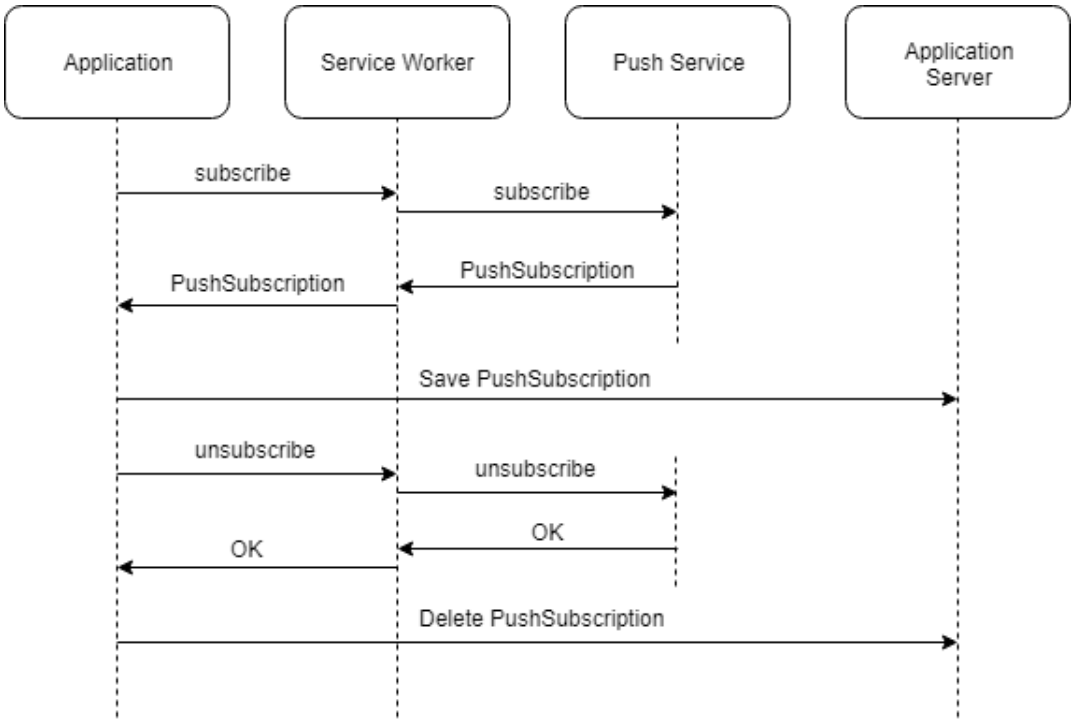

Figure 3.3: Subscription of web push notification

When a user has a valid subscription and the application server has stored the PushSubscription object, it is possible to send push notifications. The application server can use the information in the PushSubcription object, such as the push service endpoint, authentication keys and so on, to deliver a push message payload to the push service. The push service then forward the payload to the corresponding service worker on the application. When this happens, the push event on the service worker is triggered. During the push event the push payload is handled and will result in a pop-up displaying the message to the user. The process of sending web push notifications can be seen in figure 3.4.

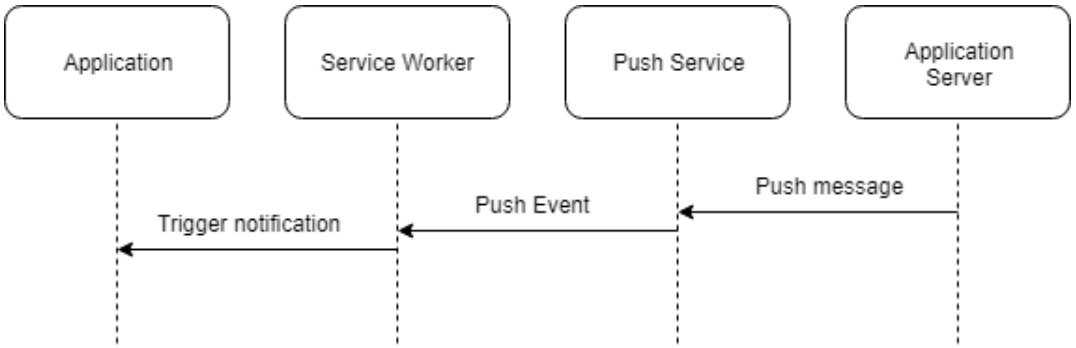

Figure 3.4: Sending web push notification

3.3.4 Web Application Manifest

The web application manifest serves to provide information about how the application should appear to the user in areas where they would expect to see the application. Progressive web applications should be platform independent, meaning that the application could be used on a desktop as well as on a mobile. To some users, a mobile application should

be accessible through the device's home screen as a clickable icon, and this is an example of what the web application manifest provides meta-data for.

Including a web application manifest file in the root project of the application can make the application more "native-like". By providing relevant meta-data about e.g. a splash screen, application theme color and launch icons, the web application will be more familiar to the mobile application users. Once the user navigates to the application (via the browser), the browser will ask the user if he or she wants to install the application on the mobile device home screen. When installation is complete the application will appear on the home screen with a clickable launch icon. Depending on the provided information in the manifest file, the clickable launch icon could open a browser and display the application, or it could open in standalone-mode which hides the browser UI and it will then look almost identical to an native application. A web application manifest is presented in listing 3.5.

```
//manifest.json
{
  "short_name": "PWA",
  "name": "Progressive Web Application",
  "icons": [
    {
      "src": "launcher-icon.png",
      "type": "image/png",
      "sizes": "48x48"
    }
  ],
  "start_url": "index.html?launcher=true",
  "theme_color": "#CA005D",
  "background_color": "#FFFFFF",
  "display": "standalone",
  "orientation": "portrait",
}
```

Listing 3.5: Manifest File

3.4 The Angular Framework

Angular[1] is a web application framework for building platform independent applications. It is developed with the intention of being a framework that ease the process of building fast web applications with high performance and high availability. Angular was first released in 2010[2] under the name AngularJS. It has since then been under constant development, and in 2015[3] an alpha version of the new Angular framework, with the name Angular 2, was released. As of today, several improvements on Angular 2 have been made and a sixth version of the framework has been released.

3.4.1 Architecture

The Angular framework is mainly written in Typescript, which is also the language used for building applications with the framework. Typescript is a superset of JavaScript, which provides optional static typing. Typescript compiles to plain JavaScript, which means that the application written in the language will work on any browser.

[1] https://angular.io/
[2] https://github.com/angular/angular.js/releases?after=v0.9.5
[3] https://github.com/angular/angular/releases?after=2.0.0-alpha.21

An Angular application consists of a set of modules and components. A module could be seen as a container for code that is dedicated to an application domain. An Angular application consists of a root module and a set of feature modules. As the name implies, the root module is the module that is bootstrapped when the application launches. The children of a root module, i.e. feature modules, are a collection of application related functionality that are imported in the root module.

A module consists of a set of components. As the root module, an application also has a root component. The root component serves to connect a component hierarchy with the page DOM (Document Object Model). A component controls some part of the screen, also called a view, and it can for example be a list of items, a set of buttons or a form of some sort. Each component consists of a template, i.e. HTML code that defines the view, which is associated with data and logic.

3.5 Service Worker and PWA Tooling

Writing your own service worker can be a time-consuming task since it consists of relatively complex parts. The technology is also quite new, which means that there are probably many new API's for the developer who tries to implement a service worker in their application. Fortunately, there exists some automatic tooling for generating service workers when developing an application in the Angular framework. Two of them are discussed below.

3.5.1 Angular Service Worker

The Angular service worker module is Angular's own module for generating and configuring a service worker. It was first released with the version 5.0.0 of the Angular framework and the intent of the module was to simplify the process of generating a service worker specially designed for the framework. The service worker provided by this module is optimized for applications that are used with slow or unreliable network connection, while at the same time be able to serve the latest content of the application.

The Angular service worker module generates a service worker at build time, i.e. when the application is compiled. The Angular service worker loads a manifest file from the application server, which describes the resources to cache and includes hashes of every file's content. The hashes are useful because when the application is updated, the hashes are also updated, which means that the service worker knows when to download new content. This mechanism ensures that application can work offline, whilst fresh content is always used by the user when network is available.

The manifest file on the application server is generated from a configuration file called "ngsw-config.json". A developer can use this file to specify what to cache, when to cache it and for how long it should stay in the cache. The configuration file is divided in two groups of caching policies; asset group and data group. The asset group specifies the resources that are part of the app version that update along with the app, e.g. resources that are loaded from the page's origin. As a developer, you can specify different caching policies for each asset in the asset group. They can either be prefetched, i.e. assets are prefetched and cached at once, or they can be lazy cached, i.e. the assets are cached first when requested. The data group specifies caching policies for data requests that are not versioned along with the application, e.g. API requests. A developer can specify two different caching strategies for data group requests; performance and freshness. A request which is configured with the performance strategy first serves the response from the cache. If it does not exists in the cache, it will proceed with a network request. A request configured with the freshness strategy will do

the exact opposite, i.e. first make a network request and if it fails, it will proceed to make a response from the cache.

3.5.2 Workbox

Workbox is a framework independent PWA library which facilitates the development of a service worker. Workbox is a lot like the Angular service worker module and the generation and configuration of a service worker has been made easy. The Workbox library comes with a set of handy API's which facilitates the caching of different assets.

Workbox is, at the time being, somewhat more customizable than the Angular service worker module because of the many API's provided by the library. The service worker is also much more extendable. With Workbox you can either chose to generate a complete service worker specified by a configuration file, or you can develop our own service worker using Workbox APIs. With the Workbox library you can specify what files that should be precached and and they are maintained efficiently if these files were to be changed. Precaching files makes the application work offline and the Workbox library helps to keep these files up to date when online. As with the service worker module provided by Angular, you can specify runtime caching with the Workbox library. Runtime caching is caching of request that are not versioned along with the application, e.g. API request. You can specify what routes the Workbox service worker should be "listening" on and how to handle the response of the request.

3.5.3 Push Notification Library

The use of libraries could also be effective when working with web push notifications. The libraries mentioned above have support for receiving push notifications to the service worker, but some additional functionality has to be developed on the application server in order to send push notifications.

One web push notification library, which is the library that will be used in this thesis, is the "web-push-libs"[4]. It consists of a set of helper functions to send push notifications from an application server. The library simplifies the use of the Web Push Protocol and takes care of all the necessary encryption for sending data along with the push notification. Web-push-libs supports most of the major application server languages such as PHP, C, NodeJS and JAVA.

3.6 Software Complexity & Software Complexity Metrics

Software complexity could be seen as the relationship between a program and a programmer working on some programming task. It is a measure of how resource-demanding a system is while interacting with a piece of software to perform a given task. An interacting system could be a computer, then the complexity is a measurement of execution time or how much storage that is required to perform the task. An interacting system could also be a programmer, then the complexity could be a measurement of how hard the software is to code, to test, to modify or to maintain[15].

Measuring software complexity could give the persons involved with the software a hint of how much resources it will take to develop and maintain software projects. It will give an indicator of how great the workload of programming will be and how much it will cost to develop. Studies have shown that software complexity significantly affect the software maintenance cost[1] and that software complexity has a strong relationship to program errors[17].

[4] https://github.com/web-push-libs

The following of this section is a presentation of popular software complexity metrics, which are used to quantify the complexity of a program.

3.6.1 McCabe's Cyclomatic Complexity

McCabe's Cyclomatic Complexity metric[22] is one of the most frequently used software complexity measurement[36]. McCabe's cyclomatic complexity is derived from a flowgraph and uses graph theory to compute the program complexity. The cyclomatic complexity is a metric that relates to the number of paths through a program. The cyclomatic complexity, as described by McCabe, is defined in equation 3.1.

$$v = e - n + 2p \tag{3.1}$$

Where v is the cyclomatic complexity, e the number of edges, n the number of nodes and p is the number of connected components.

McCabe further demonstrate that if p is 1, i.e. the number of connected components is equal to 1, then a more simplified version of the cyclomatic complexity could be derived[23]. The simplified version is seen in equation 3.2.

$$v = \text{number of decision statements} + 1 \tag{3.2}$$

A simple example of a flow graph in MaCabe's cyclomatic complexity is shown in 3.5. The graph shows that there are eight edges, $e = 8$, seven nodes, $n = 7$, and $p = 1$, which means that the cyclomatic complexity is $8 - 7 + 2 = 3$.

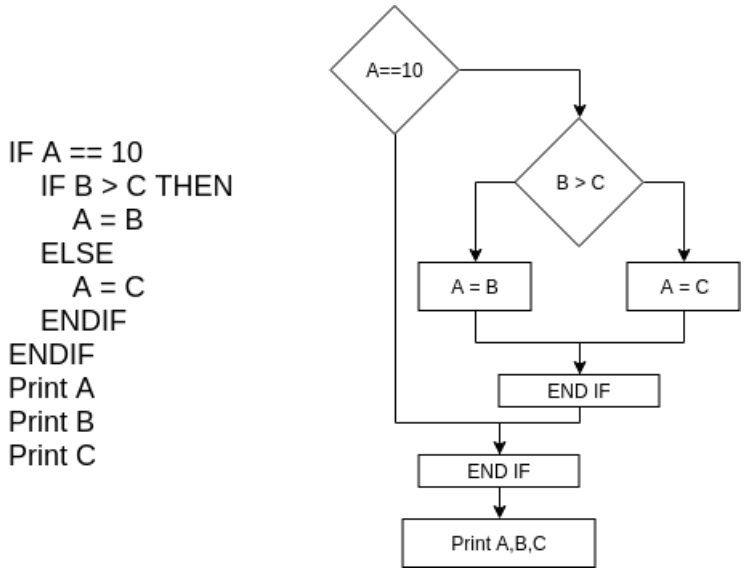

Figure 3.5: Cyclomatic Complexity

3.6.2 Halstead Complextiy Measure

Another metric for computing program complexity is the Halstead Complexity measure[8]. Halstead measure program program properties by the following notation:

$$n_1 = \textit{Number of distinct operators}$$
$$n_2 = \textit{Number of distinct operands}$$
$$N_1 = \textit{Total number of operators}$$
$$N_2 = \textit{Total number of operands}$$

Following this notation, several measurements could be derived. The calculation of the program volume, V, can be seen in equation 3.3 and the program difficulty, D, can be seen in equation 3.4.

$$V = (N_1 + N_2)\log_2(n_1 + n_2) \tag{3.3}$$

$$D = \frac{n_1}{2} * \frac{N_2}{n_2} \tag{3.4}$$

Combining equation 3.3 and equation 3.4 results in the Halstead effort metric, E, which is seen in equation 3.5.

$$E = V * D \tag{3.5}$$

As an example, a simple function can be seen in snippet 3.6. A calculation of unique and total number of operators and operands of the function can be seen below.

- $n_1 = 6$
- $n_2 = 7$
- $N_1 = 8$
- $N_2 = 12$

These numbers will result in the Halstead volume, difficulty and effort and are calculated as follows:

- $V = (8 + 12)log_2(6 + 7) = 74.01$
- $D = (6/2) * (12/7) = 5.14$
- $E = 74.01 * 5.14 = 380.41$

```
function fn(a, b, c) {
    var avg = 0;
    avg = (a + b + c) / 3;
    return avg;
}

// # of unique operators: [function, var, =, + , /, return]
// # of unique operands: [fn, a, b, c, avg, 0, 3]
```

Listing 3.6: Halstead Example Code

3.6.3 Lines of code

Logical Lines Of Code (LLOC) and Source Lines Of Code (SLOC) are two software size measurements. Both of these measurements are widely used in software planning and evaluation[16]. There are some ambiguity when defining these two metrics and the definition may vary. One common definition of SLOC is the count of lines excluding comments[25]. LLOC on the other hand measures the number of executable statements, which strongly depends on what programming language the software is developed with.

3.6.4 Chidamber & Kemerer

Chidamber and Kemerer developed a metric suite especially designed for object-oriented programming[6]. The metric suite consists of a set of different metrics, including the following:

WMC - Weighted Methods Per lass

Consider a class, C, with a set of methods, M, then

$$WMC = \sum_{i=1}^{n} c_i \tag{3.6}$$

Where c_i is the complexity of the methods and n is the number of methods in C.

DIT - Depth of Inheritance Tree

The depth of inheritance of the class is the DIT metric. The DIT metric regards to the maximum length from a node to the root of a tree, and the deeper the tree constitutes, the greater the design complexity is.

NOC - Number Of Children

The NOC metric refers to the number of direct subclasses of a class in the class hierarchy.

CBO - Coupling Between Objects classes

The metric CBO for a class is the number of other classes to which a class is coupled. A class is coupled to another class if it uses methods or instance variables of another class.

RFC - Response For a Class

The definition of the RFC metric is $RFC = |RS|$, where RS is the response set for the class. The response set could be defined as $RS = \{M\} \cup_{all\, i} \{R_i\}$, where $\{R_i\}$ is the set of methods called by method i and $\{M\}$ is the set of all methods in the class.

LCOM - Lack of Cohesion in Methods

Chidamber and Kemerer defines LCOM as similarity of methods, and that the LCOM value is a count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. LCOM is a metric which relates to an object's attributes and it is tightly coupled to the instance variables and methods of a class.

3.7 Software Complexity Metrics Evaluation

In a study performed by Weyuker[36], the author evaluates and compare some well-known complexity metrics. The comparison includes, among others, McCabe's cyclomatic number and Halstead's programming effort. Weyuker states that assessing complexity measures is often a difficult task because it is not always clear what the measure is suppose to be measuring. Software complexity characteristics frequently include the difficulty of implementing, testing, understanding, modifying, or maintaining a program, and Weyuker's contribution with the study is to formalize the properties for software evaluation with regards to complexity.

Weyuker compares McCabe's cyclomatic complexity and Halstead's programming effort by first stating some desirable complexity measurement properties. Weyuker argues that the produced properties are properties that a syntactic complexity measure should fulfill. Some of these properties includes that a measurement should not rank all prgrams as equally complex, a measure should not be too coarse, a measure should not be to fine and and assign to every program a unique complexity, and more. With these properties defined, Weyuker concludes that one wekness of cyclomatic complexity is that it rates too many programs as equally complex and that it is not sensitive enough to distinguish programs differences with regards to complexity. Weyuker continuous by concluding that Halstead's effort measure fails to fulfill one fundamentally important property, i.e. that the components of a program should not be more complex than the program itself. Weyuker argues that the Halstead's effort measure's usefulness is questionable as a complexity metric because it fails to fulfill this important property. Weyuker also argues that both cyclomatic complexity and the effort measure fails to differentiate between nested and sequential loops, but that the effort measure may have better responsiveness to the interaction among program units (e.g. responsive to statement order), than the cyclomatic complexity.

In a continuation of Weyuker's study, Cherniavsky and Smith [5] further explored the properties (as defined by Weyuker) that a software complexity metric should fulfill. In their study they present a complexity metric which fulfills all of Weyuker's proposed properties, but which has no practical utility in measuring the complexity of a program. The authors conclude that fulfilling Weyukers' properties is a necessity for a good complexity measure and that more properties needs to be defined in order to properly evaluate a complexity metric. They end with concluding that the search for an ideal complexity metric is to no avail since it probably does not exists, and that almost any existing complexity measure, including McCabe's cyclomatic complexity and Halstead's effort measure, is good enough to be used in software development.

Basili et al. conducted an empirical validation[2] of the metric suite defined by Chidamber and Kemerer[6] with regards to their ability to identify fault-prone classes. The study aimed to demonstrate the usefulness of the proposed metric suite in practice. The authors argue that a measure may be correct in a theoretical perspective, but lack real world use, and that a measure that is not 100 % correct in theory may still have a good enough approximation as a metric in practice. The study was carried out on data from eight medium-sized information management systems all with identical requirements and all written in an object-oriented

language. The author's findings of the study was that five out of six of the Chidamber and Kemerer's metric suite was useful to predict class fault-proneness during the design phases of the development life-cycle. These five metrics included WMC, DIT, RFC, NOC and CBO. The LCOM metric was shown to be insignificant with regards to fault-proneness. Interestingly enough, both CBO and WMC seem to be more significant as a metric for UI (User Interface) classes.

4 Method

This chapter includes the methodology used to carry out this thesis. First, the PWA features that were going to be implemented were explored. Then, the identified PWA features were implemented and specific PWA tooling were chosen. Lastly, software complexity metrics were chosen to analyze the added complexity to the application after the implementation state.

4.1 PWA Features

The PWA features that were implemented and demonstrated in the application were first identified. The minimum requirement was that the application should follow the base line requirements for progressive web applications (see section 3.3.1). The chosen features are suppose to be beneficial for web based business applications, and the pre-study of this thesis was to identify what PWA features that would improve an existing web based business application.

Two existing web based business applications provided by the company were examined. The examination were exploratory, and the application were navigated throughout their different views. Notes were taken about the applications' functionalities, as well as lack of functionalities, and their overall user experience. The exploratory study, together with the notes, and the baseline requirements laid the foundation for what PWA features that were going to be implemented.

4.2 PWA Implementation

The implementation of the application in this thesis was simple in the terms of functionality. The application was developed with the intent of demonstrating useful PWA features for web based business applications.

The front-end of the application was developed with the Angular framework (see section 3.4), and the language used was Typescript. The ngrx libraries[1], which is built upon ReactiveX[2], was used to manage the application state. The back-end was built in ASP .NET Core and the language used was C#. The application stored data in a SQL database and used the object-relational mapper, Entity Framework[3], for the communication between the application server and the database.

4.2.1 PWA implementation approach

As described in section 3.5, there exists some automatic tooling for building applications with PWA features. In order of answering the second research question of this thesis, i.e. how to build a PWA so that the code complexity remains low, three different service worker implementation approaches were chosen and then analyzed. The three different approaches were:

- The Angular framework's own service worker module
- The Workbox.js service worker library
- No thirds-party library used and manually creating and configuring a service worker

The identified PWA features that were dependent on a service worker, e.g. offline support, web push notifications and more, were implemented using the aforementioned implementation approaches. The features that were not dependent on a service worker were all implemented in the same way. For the implementation of push notification-related features on the application server, the library "web-push-libs"[4] was used. Since the application server were written in C#, the corresponding C# push notification library was used.

4.3 Complexity Metrics

After the identification and implementation of the PWA features, the overall added complexity was analyzed. The chosen metrics for the complexity evaluation were based on a software complexity literature study (see section 3.7).

The complexity metrics chosen to analyze the application were:

- Source Lines Of Code (SLOC)
- The maximum cyclomatic complexity of a module (CC max)
- The Halstead Effort
- Files added
- Files Modified

McCabe's cyclomatic complexity, Lines of code and the Halstead's complexity metric were chosen because they have been widely used and evaluated in several studies and have been

[1] https://github.com/ngrx
[2] https://github.com/ReactiveX
[3] https://docs.microsoft.com/en-us/ef/
[4] https://github.com/web-push-libs

proven effective for measuring complexity[36][7][13]. Keeping track of how many files that are added, as well as how many files that are modified, is an interesting metric because it has been shown that more added and modified files will increase the complexity of an application. In studies conducted by Hassan et al.[10][11], they conclude that more complex code changes to a file will result in a higher chance that the file will contain faults. Since fault-proneness is a non-desirable attribute for any kind of software project, these two metrics are included.

4.3.1 Static Code Analysis Tools

To calculate the metrics presented in section 4.3, a number of different static code analysis tools were used. For calculations on the front-end code, Plato[5] were used to calculate the Halstead Effort, SLOC[6] were used to calculate the SLOC and JSHint[7] were used to calculate the cyclomatic complexity. For calculations on the back-end, the Visual Studio Static Code Analyzer[8] was used.

An effort was made to find scientific literature that either supported the use of these tools, or proposed another set of tools for this given context. No such findings were made, and the motivation for using these specific set of tools are based on the number of downloads and project commendations.

[5]https://github.com/es-analysis/plato
[6]https://github.com/flosse/sloc
[7]https://github.com/jshint/jshint
[8]https://docs.microsoft.com/sv-se/visualstudio/code-quality/code-analysis-for-managed-code-overview

5 Results

This chapter presents the progressive web application built for this thesis and it's core features. It also presents the results obtained when analyzing the complexity of the code that made the web application progressive.

5.1 PWA Features

The PWA features that were chosen to be implemented, i.e. features that were deemed beneficial for web based business applications, are the following:

Table 5.1: Implemented PWA features

Feature	Description
F1	The application should be able to work offline (with some reduced functionality)
F2	The application should be able to sync failed POST request sent when offline
F3	The application should notify users when an offline state has been reached
F4	All GET requests to the application server (except precached assets) should follow a network-first principle to ensure data freshness
F5	The user should always use the latest version of the application, and if not, the user should be prompted to download it
F6	The user should be able to subscribe and unsubscribe to push notifications
F7	The notifications sent from the application server should consist of arbitrary body data, a title, an image and a badge image for mobile phones
F8	The application should be installable on mobile phones and it should notify the user that it is installable
F9	On installation, a launch icon should appear on home screen for mobile phone users
F10	The application should look and feel like a native application for mobile phone users

One thing to note is that feature F2, i.e background sync, was not possible to implement when using the Angular service worker module. Feature F5, i.e. notifying users of available updates, was not possible to implement using the Workbox service worker.

5.2 Implementation

This section includes the application developed for this thesis. In figure 5.1 an add-to-home-screen prompt for mobile users can be seen. When the the user accepts to add the application to its home screen, the application is installed and an icon appears. The home screen icon can be seen in fig 5.2.

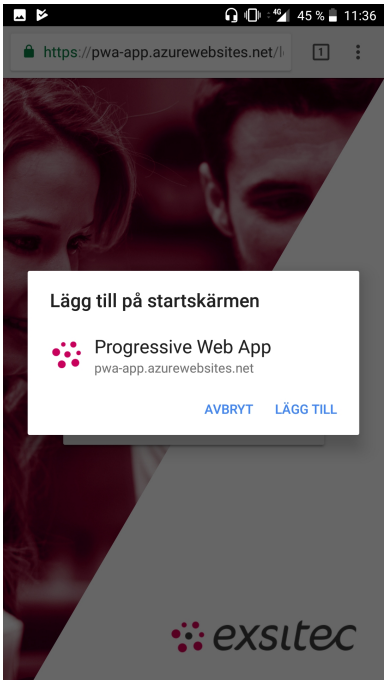

Figure 5.1: Add to home screen prompt

Figure 5.2: Icon on home screen after install

When the user click the icon on the mobile home screen. A splash screen will appear as the application is loading. The splash screen can be seen in figure 5.3. When the application is loaded, a sign-in prompt is displayed to the user. This, and the standalone mode (i.e. no web browser user interface) of the application can be seen in figure 5.4.

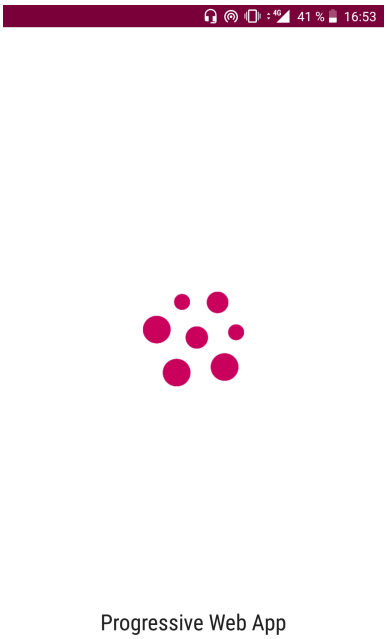

Figure 5.3: Splash screen

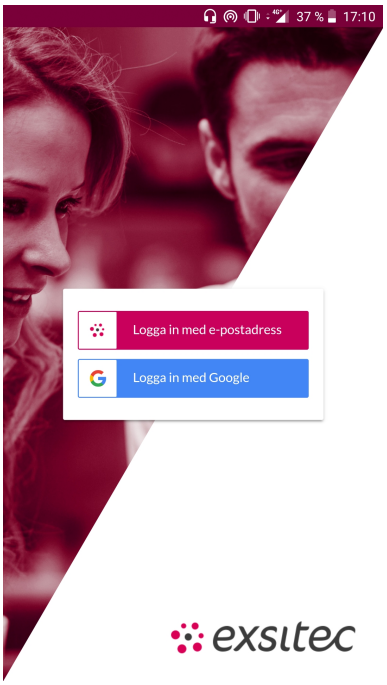

Figure 5.4: Standalone mode for mobile users

The push notification permission prompt is seen in figure 5.5. If the user does not accept, nothing will happen, but if the user accepts it will be able to receive push notifications from the application server.

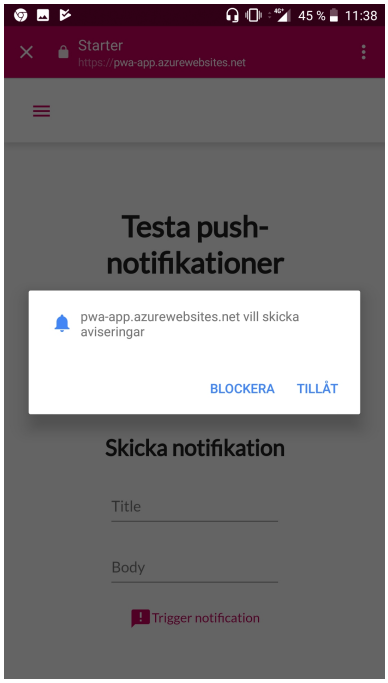

Figure 5.5: Push notification permission prompt

A received notification on the mobile platform can be seen in figure 5.6. A received notification on the desktop platform (using Ubuntu OS) can be seen in figure 5.6.

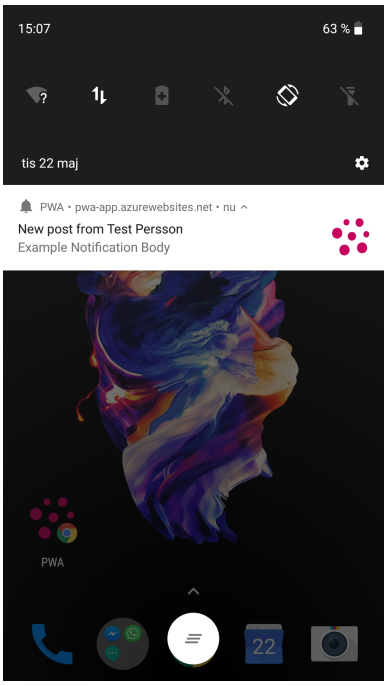

Figure 5.6: Notification for mobile users

Figure 5.7: Notification for desktop users

When a new version of the application is available, a prompt is shown informing the user that a new version can be downloaded. This can be seen at the bottom of figure 5.8. A user is notified when the connection to the network is lost. This can be seen in figure 5.9.

Figure 5.8: New version is available

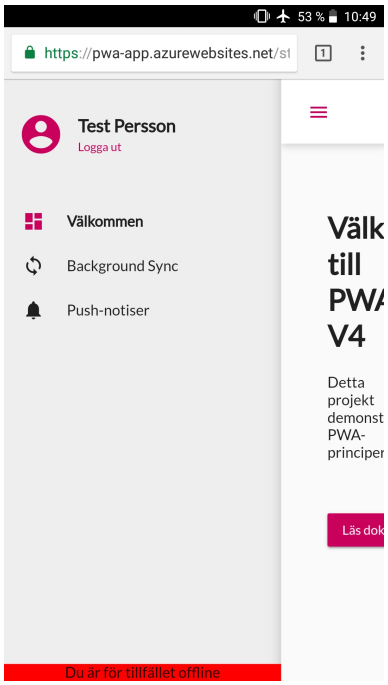

Figure 5.9: Offline state reached

5.3 Complexity Analysis

This section includes the complexity analysis of the application.

5.3.1 Service Worker Creation & Registration

Down below is the complexity analysis of the creation and registration of a service worker in the application. The service worker creation and registration are done manually without any tools, with the Angular framework's service worker module and with the workbox library. All three different approaches were analyzed with the given complexity metrics and the result is shown below.

Manual service worker

The results of creating and registering a manually build service worker can be seen in table 5.2. The file "sw.js" is the actual service worker script, the "inject-precache-url.js" is a script for completing and moving the service worker to the appropriate build folder and the main.component.ts file is modified to handle the registration of the service worker in the actual application.

Files Added	**SLOC**	**CC max**	**Halstead Effort**
sw.js	119	6	240835
inject-precache-url.js	23	2	8016
Files modified	**SLOC**	**CC max**	**Halstead Effort**
main.ts	20	4	2896

Table 5.2: Creation and registration of a manual service worker

Angular service worker

The results of creating and registering an Angular service worker can be seen in table 5.3. The file "ngsw-config.json" is a JSON structured configuration file and the calculation of the cyclomatic complexity and the Halstead complexity metric is not applicable. The file "sw-fix.js" is a necessary bug-fix script and the "app.module.ts" is modified to handle the registration of the service worker in the application.

Files Added	**SLOC**	**CC max**	**Halstead Effort**
ngsw-config.json	54	Not applicable	Not applicable
sw-fix.js	48	5	11095
Files modified	**SLOC**	**CC max**	**Halstead Effort**
app.module.ts	2	1	242

Table 5.3: Creation and registration of the Angular service worker module

Workbox service worker

The result of creating and registering a Workbox service worker can be seen in table 5.4. The file sw.js is the service worker script, the sw-build.js is a script for completing and moving the service worker to the appropriate build folder and the main.component.ts file is modified to handle the registration of the service worker in the actual application.

Files Added	**SLOC**	**CC max**	**Halstead Effort**
sw.js	55	2	18488
sw-build.js	17	1	2392
Files modified	**SLOC**	**CC max**	**Halstead Effort**
main.ts	20	4	2896

Table 5.4: Creation and registration using the Workbox library

5.3.2 Feature F6-F7 - Push notifications

For feature F6-F7, i.e. the features regarding web push notifications, the implementation approach was the same for the manual service worker and the Workbox service worker. The implementation somewhat differed when using the Angular service worker, therefore this section is divided into two parts.

Manual and Workbox service worker

The complexity analysis result can be seen in table 5.5. The "subscription.component.ts" file handles the permission and subscription of push notifications on the front end of the application. The file "PushNotificationController" on the back end is responsible for responding to requests made from the application and the "PushNotificationService" handles the logic of the request.

Files Added	**SLOC**	**CC max**	**Halstead Effort**
subscription.component.ts	49	2	28293
Files modified	**SLOC**	**CC max**	**Halstead Effort**
PushNotificationController.cs	35	1	5679
PushNotificationService.cs	74	1	29324

Table 5.5: Complexity analysis of adding push notifications using a manual/Workbox service worker

Angular service worker

The complexity analysis results when using the Angular service worker can be seen in 5.6. The same files were added and modified as when using the manual or the Workbox service worker.

Files Added	**SLOC**	**CC max**	**Halstead Effort**
subscription.component.ts	29	2	8987
Files modified	**SLOC**	**CC max**	**Halstead Effort**
PushNotificationController.cs	35	1	5679
PushNotificationService.cs	74	1	29324

Table 5.6: Complexity analysis of adding push notifications using the Angular service worker

5.3.3 Feature F8-F10 - Installable, home screen icon and native-like

For feature F8-F10, i.e. the application should be installable, it should have a home screen icon and it should be native-like, and the main technology for this is the web application manifest. One requirement for these features is that a service worker is registered, and it does not matter how it is registered (it does not matter which of the three service worker implementation one uses). Therefore, all three implementation approaches used the same web application manifest. The results can be seen in table 5.7. The manifest file was added

and a reference to it was added in the "index.html" file. Neither the cyclomatic complexity nor the Halstead effort was applicable for these features.

Files Added	**SLOC**
manifest.json	108
Files modified	**SLOC**
index.html	1

Table 5.7: Web application manifest regarding feature F8-F10

6 Discussion

This chapter contains a general discussion about the results and the method used. It does also contain a critical analysis of the sources used and a discussion about this work in a wider context.

6.1 Results

When examining the results of the complexity analysis there is one interesting conclusion to be drawn. Implementing features so that a web application becomes progressive does not excessively increase the size nor the complexity of the application. These conclusion can be drawn because the SLOC count, the maximum cyclomatic complexity of a module and the Halstead effort can all be considered low. The SLOC count does not exceed 250-350 for any type of implementation. This is only a fraction of the total number of SLOC for the whole project, which is well above 5000. Therefore, the number of source lines of code is low for making an web application progressive.

The maximum cyclomatic complexity of a module was mostly around 1-2, but the highest was 6, which was found in the manual service worker. This is no suprise because as mentioned in section 3.3.2, the service worker intercepts all network request in the fetch event. Each intercepted request needs to be checked to determine which type of request it is. Each request check will add to the number of cyclomatic complexity, which is probably the reason why it is the highest in the manual service worker. McCabe suggests that a reasonable upper limit to cyclomatic complexity is 10 [22], and most of the modules were clearly lower than that. Therefore, the maximum number of cyclomatic complexity of a module is low when implementing PWA features.

The Halstead effort differed depending on what module that was changed or added. The Halstead effort seems to correlate with SLOC, i.e. the higher SLOC count, the higher Halstead effort. The highest effort count was again the manual service worker, which was significantly higher than any other analyzed module. Halstead suggests that the time required to develop a module could be calculated as $T = E/18$. This would mean that the manual service worker would require $T = 240835/18 \approx 13380 seconds$, which is $\approx 3.7h$. The manual

service worker definitely took longer time to develop, mainly because there were a lot of new API's which was unfamiliar to the developer. However, if the same service worker were to be developed again the time measurement could potentially be a good approximation for how long it would take. An interesting conclusion is that the Halstead effort maybe more relevant as a complexity metrics when the developers are more familiar with the details of the new technology.

Another thing to notice about the results is that there were few added and modified files in order to make the application progressive. Much of the implemented features were controlled by two main files; the service worker file and the manifest file. Other files were mostly an extension to build the service worker, such as "build-sw.js" or "inject-precache-url.js", or necessary logic to control the data flow, such as "PushNotificationController.cs" or "PushNotificationService.cs". One conclusion is that adding and registering a service worker and a manifest file will result in the possibility of including a lot of PWA features, and that this process is not too demanding when it comes to adding and modifying existing files.

To summarize, adding features that are considered progressive for Angular web applications does not excessively increase the overall complexity of the application. The first research question of this thesis, i.e. "how does the implementation of PWA features in Angular affect the code complexity of an application?", is therefore answered. However, the metrics used to evaluate the added complexity does not fully consider the cognitive complexity and that code complexity is a highly individual measurement for every developer. Code complexity is a difficult topic, and the metrics used in this thesis are one way of quantifying it.

To reason about the second research question of this thesis, i.e. "How do you develop a web application with PWA features so that the code complexity of the application remains low?", the author would argue that the Angular service worker module is superior. This is mainly because the implementation is optimized for the Angular framework and because the steps of getting started are minimal. The process of generating a service worker are highly automated and configuring it is easily done with the "ngsw-config.json" config file. The Angular service worker module also provides the developer of handy API's, which eases the process of for example subscribing and unsubscribing from push notifications. However, at the time of this thesis, syncing failed request to the application server is not yet implemented and the module has some bugs (hence the "sw-fix.js" bug-fix file). It is also not as extendable as writing your own service worker or using the Workbox library, but bug-fixes and more features will probably be added to the module in a not so distant future. The author would also argue that adding push notification library on the application server is also a way of minimizing the added complexity. As with the manual service worker, developing something from scratch could be a highly complex task, and the push notification library significantly reduce this complexity.

6.2 Method

This sections provides a discussion about the method used in this thesis. It treats the implementation of the progressive web applications, the complexity analysis of them and some general discussion about this work in a wider context.

6.2.1 Implementation

As briefly mentioned in section 3.3, the term "progressive web application" is at the time being somewhat ambiguous. It does not exists a clear definition of a progressive web application really is and what technology concepts that needs to be included in order for a web application to be considered progressive. The implications that this causes for this thesis is that the

features implemented might, for some, not be sufficient in order to be called a progressive web application. However, some automated tools, such as Lighthouse[1], have been used to ensure that the application is in fact a progressive web application.

In this thesis, three different implementation approaches of to build a progressive web application were embraced. When developing almost any type of software, the code written is highly dependent on the mind of the developer. The same problem could have a multitude of different solutions, which means that the complexity of a solution may vary depending of who comes up with it. This could potentially be a possible threat to the replicability of this thesis. This is especially true for the manual service worker implementation. The manual service worker were developed from scratch and had to satisfy some functional requirements. Due to time constraints, it was not possible to create a full, production-ready service worker. A lot of error handling and corner cases handling has to be implemented in order for it to be production-ready. This means that it could be difficult to replicate the exact same service worker as developed in this thesis.

Another possible threat to replicability and reliability is the fact that the technologies used in this thesis are cutting edge, and sometimes bleeding edge. As an example of this, one API had a major update which invalidated some of the work done during the time of development. Another example is that the Angular service worker had a severe bug, which broke some features of the application. Therefore, an additional file (the "sw-fix.js") was needed and since it was a necessary fix, the file's complexity was also analyzed. This means that it's possible that some APIs, or manual bug-fixes, used in this thesis might be outdated in a not so distant future.

As mentioned in result section 5.2, feature F2, i.e background sync, was not possible to implement when using the Angular service worker module. Feature F5, i.e. notifying users of available updates, was not possible to implement using the Workbox service worker. These two features were not possible to implement because at the time of this thesis, either the library did not support the feature, or the feature was incompatible with other frameworks used in this thesis. If, or when, they will be compatible, a complexity analysis might yield a different result.

6.2.2 Complexity analysis

In this thesis there were three different complexity metrics used to quantify the added complexity when making a web application progressive. Each of these metrics and their applicability to the stated research questions of this thesis will be discussed.

SLOC

As mentioned in section 3.6.3, SLOC, or source lines of code, is a widely used metric for determining the size and the complexity of a software project. It can give a good approximation of how much time it will take to solve a given problem, and how difficult it will be to solve it. The SLOC metric may give a decent indication of the overall added size and complexity, but the metric also has some downsides which has been proven true in this thesis.

SLOC is sensitive to the way a developer likes to format his' or her's code. It is also sensitive to a constantly changing language, which JavaScript is. SLOC does not take the actual functionality of the code into account, which means that a code block can have the same functionality but vastly different SLOC count. An example of this is shown in listing 6.1. Both

[1] https://developers.google.com/web/tools/lighthouse/

the functions, i.e "fetchAsyncA" and "fetchAsyncB", have the same functionality, i.e asyncronously fetch data from some origin. The function "fetchAsyncA" has a SLOC count of 1 and "fetchAsyncB" has a SLOC count of 4, which is 400% more, but some would probably argue that "fetchAsyncB" has a lower complexity because the readability is higher.

```
1 const fetchAsyncA = async () => await (await fetch(URL)).json();
2
3 const fetchAsyncB = async () =>  {
4   const response = await fetch(URL);
5   const json = await response.json();
6   return json;
7 }
```

Listing 6.1: Functionality and SLOC

However, as for this thesis, the development of the applications was done by a single developer, which means that the format and syntax used to develop the applications were equivalent. This makes the comparison, in terms of SLOC, of the three different development approaches more valid.

McCabe's Cyclomatic Complexity

The cyclomatic complexity as presented by McCabe counts the number of linearly independent paths through a programs source code. As mentioned in section 3.6.1, the cyclomatic complexity is often used to determine the complexity of a program. The cyclomatic complexity is at it's core a simple metric that can yield a good approximation of the complexity of a block of code, but as for SLOC, the cyclomatic complexity is also sensitive to the syntax of the code. This means that also this metric is quite heavily dependent of what language being used and what programming paradigm that is embraced by the developer.

In this thesis, the cyclomatic complexity was used determine the independent paths through code that makes a web application progressive. As stated by researchers before, the cyclomatic complexity is far from a perfect complexity metric [31] [30], but for this thesis, the metric gave a coarse approximation of the added complexity. When looking at the results of this thesis, no function or method had a unacceptably high cyclomatic complexity, which is perhaps the most vital information in the results with the regards to the metric.

Halstead Effort

Halstead identified measurable properties of source code that could be used to determine the complexity of the program. In this thesis the Halstead effort were used to get an approximation of complexity of the added code. As with the cyclomatic complexity, researchers have found that the mterics proposed by Halstead have some shortcomings [29][9].

A limitation of the Halstead metrics found in this thesis is that there does not seem to be a clear definition of the operators and operands for JavaScript or TypeScript. When studying this topic, it was found that there exists a lot of different opinions of how to count operators and operands. For example, some sources said that curly brackets "{}" should be counted as an operator. The Halstead effort is calculated by multiplying the difficulty, see equation 3.4, and the volume, see 3.3. This means that an increasing number of operators would result in a more complex program. The curly brackets acts as an structure element, which in many cases increases the readability of a code block. Higher readability should be the a result of a less complex code, and counting curly brackets as an operator will increase the complexity, which is somewhat contradictory.

Since a third-party program was used to calculate the Halstead metrics, an unanimous way of calculating the operators and operands were carried out. However, it was only unanimous for JavaScript/TypeScript, and another program were used to calculate the Halstead metric for C# code. This means that it might not be valid comparison between the Halstead effort on the front-end and the Halstead effort on the back-end.

Static Analysis Tools

There were several different tools used the statically analyze the code and to derive a result of the aforementioned metrics. An effort was made to find scientific literature that supported the use of these tools in the given context, but no such findings were made. A possible threat to reliability is that if one were to analyze the code with the same metrics, but with different tools, another result may be expected.

6.2.3 Source Criticism

The concept of progressive web applications is relatively new. Therefore, the number of well-cited, scientific sources are few. Most of the theory regarding the specific technologies are taken from Mozilla Web Developer Documentations[2], which is documentation about web development created by an open community of developers. These source is, in the opinion of the author, a credible source when it comes to web development. A lot of theory about the technologies are also taken from the company Google. One thing to be cautious about is that a lot in this thesis have some connection to Google. The development of the Angular framework is led by Google, the browser Chrome (which was primarly used in this thesis) is from Google, the push notification service is provided by Google and so on. It's important to be aware of this and efforts have been made to find and use other, similar resources from different sources.

6.3 The work in a wider context

There are not many clear ethical aspects to consider when developing a progressive web application. This is also true for this thesis. One may argue that in a society where stress and mental illnesses are a nationwide problem, one should not be developing applications where you are always exposed to information (such in the case of web push notifications). With offline support, one will always be able to use one's application, which may be deemed stressful. One of the "slogans" for progressive web applications is that they are engaging, meaning that they are developed with the intent of be used over a longer period of time. In a society where time spent in front of a screen is increasing, this might be an issue.

Corporate applications have in this thesis been examined in order to get a better understanding of the applications the company is developing. Efforts have been made to not disclose any sensitive corporate information in this thesis.

[2]https://developer.mozilla.org/en-US/

7 Conclusion

This thesis has explored the concept of progressive web applications and the technologies used to leverage the latest advancements in the web development area. The underlying purpose of this thesis was twofold, to investigate the usefullness of considered PWA features in web based business applications and to examine the added complexity of making a web application progressive. The result of this thesis was to ultimately answer the question whether or not the company should embrace the concepts of PWA, or if it just adds another layer of unnecessary complexity to their applications.

To achieve the aim of this thesis and to reach a conclusion to the aforementioned question of issue, two research questions were stated to define the scope of this thesis. The research questions were: "How does the implementation of PWA features in Angular affect the complexity of an application?" and "How do you develop a web application with PWA features so that the code complexity of the application remains low?". The intent of these questions was that if an answer were to be derived, then the underlying purpose were to be accomplished.

To answer the stated research questions and to explore progressive web applications, an proof-of-concept application were developed with a set of PWA features. The features were meant to be beneficial for the applications the company is developing today, i.e. web based business applications, and some existing applications provided by the company were examined. When the requirements of the proof-of-concept application were determined and the exploration of PWA tools and libraries were done, the development process began. When the application were deemed complete, the process of analyzing the application with a set of complexity metric was done.

The implementation and later on the complexity analysis of the implementation resulted in some general conclusions. Developing applications using the web platform sees a great future. What has previously been the downsides of web applications, e.g. no offline support and overall lesser user experience compared to native applications, are now possible to mitigate, making the gap between native and web smaller in many aspects. The overall added complexity of making a web application progressive is low. It does not excessively increase the size of the application and the programming effort is kept on a reasonable level. How-

ever, many concepts regarding progressive web applications are cutting edge, which means that their are unfamiliar to many developers. The most complex process regarding PWAs is the rather steep learning curve, which is hard to quantify. To minimize the added complexity one should embrace the automated tools for developing PWA features. When developing applications with the Angular framework one should opt in using the automated PWA tools provided by the framework.

7.1 Future Work

The are numerous of interesting continuations of this thesis. It would be interesting to investigate the cognitive complexity, i.e. how developers actually comprehends the code for making progressive web applications. This thesis only treated static code analysis, resulting in a coarse approximation of code complexity. Researching the experienced complexity could for example be done with different code workshops or code surveys. The results of such a research could later on be compared with the complexity analysis results of this thesis. Another interesting research would be a comparison between a progressive web application, a native application and a hybrid application. It would be interesting to see if users can differentiate between them in terms of user experience and performance. Another possible comparison between them is to measure the phone's battery consumption, CPU and memory usage and load times. The result will tell if either one of the implementation techniques is superior to the other with regards to performance.

Bibliography

[1] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. "Software Complexity and Maintenance Costs". In: *Commun. ACM* 36.11 (Nov. 1993), pp. 81–94. ISSN: 0001-0782. DOI: 10.1145/163359.163375. URL: http://doi.acm.org/10.1145/163359.163375.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on Software Engineering* 22.10 (Oct. 1996), pp. 751–761. ISSN: 0098-5589. DOI: 10.1109/32.544352.

[3] Kathleen Buettner and Anna M. Simmons. "Mobile Web and Native Apps: How One Team Found a Happy Medium". In: *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. Ed. by Aaron Marcus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 549–554. ISBN: 978-3-642-21675-6.

[4] Andre Charland and Brian Leroux. "Mobile Application Development: Web vs. Native". In: *Commun. ACM* 54.5 (May 2011), pp. 49–53. ISSN: 0001-0782. DOI: 10.1145/1941487.1941504. URL: http://doi.acm.org/10.1145/1941487.1941504.

[5] J. C. Cherniavsky and C. H. Smith. "On Weyuker's axioms for software complexity measures". In: *IEEE Transactions on Software Engineering* 17.6 (June 1991), pp. 636–638. ISSN: 0098-5589. DOI: 10.1109/32.87287.

[6] S. R. Chidamber and C. F. Kemerer. "A metrics suite for object oriented design". In: *IEEE Transactions on Software Engineering* 20.6 (June 1994), pp. 476–493. ISSN: 0098-5589. DOI: 10.1109/32.295895.

[7] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics". In: *IEEE Transactions on Software Engineering* SE-5.2 (Mar. 1979), pp. 96–104. ISSN: 0098-5589. DOI: 10.1109/TSE.1979.234165.

[8] Maurice H Halstead. *Elements of software science*. Elsevier computer science library : operational programming systems series. New York, NY: North-Holland, 1977.

[9] Peter G. Hamer and Gillian D. Frewin. "M.H. Halstead's Software Science - a Critical Examination". In: *Proceedings of the 6th International Conference on Software Engineering*. ICSE '82. Tokyo, Japan: IEEE Computer Society Press, 1982, pp. 197–206. URL: http://dl.acm.org/citation.cfm?id=800254.807762.

[10] A. E. Hassan and R. C. Holt. "The chaos of software development". In: *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings.* Sept. 2003, pp. 84–94. DOI: 10.1109/IWPSE.2003.1231214.

[11] Ahmed E. Hassan and Richard C. Holt. "Studying the Chaos of Code Development". In: *Proceedings of the 10th Working Conference on Reverse Engineering*. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 123–. ISBN: 0-7695-2027-8. URL: http://dl.acm.org/citation.cfm?id=950792.951383.

[12] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. "Evaluating Cross-Platform Development Approaches for Mobile Applications". In: *Web Information Systems and Technologies*. Ed. by José Cordeiro and Karl-Heinz Krempels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 120–138. ISBN: 978-3-642-36608-6.

[13] I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability". In: *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*. Sept. 2007, pp. 30–39. DOI: 10.1109/QUATIC.2007.8.

[14] M. E. Joorabchi, A. Mesbah, and P. Kruchten. "Real Challenges in Mobile App Development". In: *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. Oct. 2013, pp. 15–24. DOI: 10.1109/ESEM.2013.9.

[15] Joseph P. Kearney, Robert L. Sedlmeyer, William B. Thompson, Michael A. Gray, and Michael A. Adler. "Software Complexity Measurement". In: *Commun. ACM* 29.11 (Nov. 1986), pp. 1044–1050. ISSN: 0001-0782. DOI: 10.1145/7538.7540. URL: http://doi.acm.org.e.bibl.liu.se/10.1145/7538.7540.

[16] Chris F. Kemerer. "Reliability of Function Points Measurement: A Field Experiment". In: *Commun. ACM* 36.2 (Feb. 1993), pp. 85–97. ISSN: 0001-0782. DOI: 10.1145/151220.151230. URL: http://doi.acm.org/10.1145/151220.151230.

[17] T. M. Khoshgoftaar and J. C. Munson. "Predicting software development errors using software complexity metrics". In: *IEEE Journal on Selected Areas in Communications* 8.2 (Feb. 1990), pp. 253–261. ISSN: 0733-8716. DOI: 10.1109/49.46879.

[18] K. S. Lew, T. S. Dillon, and K. E. Forward. "Software complexity and its impact on software reliability". In: *IEEE Transactions on Software Engineering* 14.11 (Nov. 1988), pp. 1645–1655. ISSN: 0098-5589. DOI: 10.1109/32.9052.

[19] N. Li, Y. Du, and G. Chen. "Survey of Cloud Messaging Push Notification Service". In: *2013 International Conference on Information Science and Cloud Computing Companion*. Dec. 2013, pp. 273–279. DOI: 10.1109/ISCC-C.2013.132.

[20] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovic. "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps". In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. May 2017, pp. 35–45. DOI: 10.1109/MOBILESoft.2017.7.

[21] Ivano Malavolta. "Beyond Native Apps: Web Technologies to the Rescue! (Keynote)". In: *Proceedings of the 1st International Workshop on Mobile Development*. Mobile! 2016. Amsterdam, Netherlands: ACM, 2016, pp. 1–2. ISBN: 978-1-4503-4643-6. DOI: 10.1145/3001854.3001863. URL: http://doi.acm.org.e.bibl.liu.se/10.1145/3001854.3001863.

[22] T. J. McCabe. "A Complexity Measure". In: *IEEE Transactions on Software Engineering* SE-2.4 (Dec. 1976), pp. 308–320. ISSN: 0098-5589. DOI: 10.1109/TSE.1976.233837.

[23] Thomas J. McCabe and Charles W. Butler. "Design Complexity Measurement and Testing". In: *Commun. ACM* 32.12 (Dec. 1989), pp. 1415–1425. ISSN: 0001-0782. DOI: 10.1145/76380.76382. URL: http://doi.acm.org/10.1145/76380.76382.

[24] Emilia Mendes, Nile Mosley, and Steve Counsell. "The Need for Web Engineering: An Introduction". In: *Web Engineering*. Ed. by Emilia Mendes and Nile Mosley. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–27. ISBN: 978-3-540-28218-1. DOI: 10.1007/3-540-28218-1_1. URL: https://doi.org/10.1007/3-540-28218-1_1.

[25] Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry W. Boehm. "A SLOC Counting Standard". In: 2007.

[26] *Progressive Web App Checklist*.

[27] *Progressive Web Apps | Web | Google Developers*. URL: https://developers.google.com/web/progressive-web-apps/ (visited on 04/04/2018).

[28] *Push API*. URL: https://www.w3.org/TR/push-api/ (visited on 03/28/2018).

[29] V. Y. Shen, S. D. Conte, and H. E. Dunsmore. "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support". In: *IEEE Transactions on Software Engineering* SE-9.2 (Mar. 1983), pp. 155–165. ISSN: 0098-5589. DOI: 10.1109/TSE.1983.236460.

[30] M. Shepperd and D.C. Ince. "A critique of three metrics". In: *Journal of Systems and Software* 26.3 (1994), pp. 197–210. ISSN: 0164-1212. DOI: https://doi.org/10.1016/0164-1212(94)90011-6. URL: http://www.sciencedirect.com/science/article/pii/0164121294900116.

[31] Martin Shepperd. "A Critique of Cyclomatic Complexity As a Software Metric". In: *Softw. Eng. J.* 3.2 (Mar. 1988), pp. 30–36. ISSN: 0268-6961. DOI: 10.1049/sej.1988.0003. URL: http://dx.doi.org/10.1049/sej.1988.0003.

[32] D. Sin, E. Lawson, and K. Kannoorpatti. "Mobile Web Apps - The Non-programmer's Alternative to Native Applications". In: *2012 5th International Conference on Human System Interactions*. June 2012, pp. 8–15. DOI: 10.1109/HSI.2012.11.

[33] Thomas Steiner. "What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser". In: *WWW '18 Companion* (Apr. 2018). DOI: 10.1145/3184558.3188742. URL: https://doi.org/10.1145/3184558.3188742.

[34] *Using Service Workers - Web APIs | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (visited on 03/28/2018).

[35] Anthony I. Wasserman. "Software Engineering Issues for Mobile Application Development". In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. FoSER '10. Santa Fe, New Mexico, USA: ACM, 2010, pp. 397–400. ISBN: 978-1-4503-0427-6. DOI: 10.1145/1882362.1882443. URL: http://doi.acm.org.e.bibl.liu.se/10.1145/1882362.1882443.

[36] E. J. Weyuker. "Evaluating software complexity measures". In: *IEEE Transactions on Software Engineering* 14.9 (Sept. 1988), pp. 1357–1365. ISSN: 0098-5589. DOI: 10.1109/32.6178.