

Master Thesis



Czech  
Technical  
University  
in Prague

**F3**

Faculty of Electrical Engineering  
Department of Computers

## Hybrid mobile application for project planning system

**Bc. Jan Teplý**

Supervisor: Mgr. Miroslav Blaško  
May 2017

České vysoké učení technické v Praze  
Fakulta elektrotechnická  
Katedra počítačů

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jan Teplý

Studijní program: Otevřená informatika  
Obor: Softwarové inženýrství

Název tématu: Hybridní mobilní aplikace pro systém plánování projektů

## Pokyny pro vypracování:

Plantac je proprietární webová aplikace pro plánování času a nákladů projektů na platformě Java EE as vyzualizací ve frameworku ZK. Cílem projektu je prozkoumat možnosti pro vytvoření alternativního multiplatformního uživatelského rozhraní, které zpřístupní vybrané funkce systému Plantac na mobilních zařízeních i bez přístupu k internetu.

- 1) Seznamte se s webovou aplikací Plantac.
- 2) Analyzujte požadavky pro rozhraní mobilní aplikace Plantac v online a offline režimu, vytvořte technickou specifikaci.
- 3) Prozkoumejte možnosti multiplatformního vývoje aplikací s využitím JavaScriptových frameworků, JAVA EE technologií a offline režimu. Vytvořit porovnávací studii studovaných frameworků.
- 4) Navrhněte architekturu obecné aplikace pro běh v offline režimu s využitím cache-ování REST-ových služeb.
- 5) Implementujte prototyp založený na navržené architektuře.
- 6) Vysledný prototyp otestujte.

## Seznam odborné literatury:

- [1] Lanthaler, Markus, and Christian Gütl. "On using JSON-LD to create evolvable RESTful services." Proceedings of the Third International Workshop on RESTful Design. ACM, 2012.
- [2] Lanthaler, Markus. "Creating 3rd generation web APIs with hydra." Proceedings of the 22nd international conference on World Wide Web companion. International World Wide Web Conferences Steering Committee, 2013.
- [3] ReactJs framework, <https://facebook.github.io/react/>
- [4] AngularJs framework, <https://facebook.github.io/react/>
- [5] Apache Cordova -- an open-source mobile development framework, <https://cordova.apache.org/>

Vedoucí: Mgr. Miroslav Blaško, Ph.D.

Platnost zadání do konce zimního semestru 2017/2018

prof. Dr. Michal Pěchouček, MSc.  
vedoucí katedry



prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 12.9.2016



## Acknowledgements

I would like to thank Mgr. Miroslav Blaško and Ing. Jindřich Hašek for guidance in work on this thesis. And finally I would like to thank the CTU in Prague for being a very good *alma mater*.

## Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 25, 2017

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 25. května 2017

.....  
Bc. Jan Teplý

## Abstract

Plantac is the proprietary web application for project time and cost planning. Currently written on Java EE framework with ZK framework for graphical user interface. The goal of this thesis is to explore the possibility of the creation of alternative multi-platform user interface, that enables chosen functions of Plantac on mobile devices even without internet connection.

**Keywords:** web, mobile, hybrid, offline, Angular, Progressive apps, Cordova

**Supervisor:** Mgr. Miroslav Blaško

## Abstrakt

Plantac je proprietární webová aplikace pro plánování času a nákladů projektů na platformě Java EE a grafickým uživatelským rozhraním v frameworku ZK. Cílem práce je prozkoumat možnosti pro vytvoření alternativního multiplatformního uživatelského rozhraní, které zpřístupní vybrané funkce systému Plantac na mobilních zařízeních i bez přístupu k internetu.

**Klíčová slova:** web, mobil, hybridní, offline, Angular, Progressive apps, Cordova

**Překlad názvu:** Hybridní mobilní aplikace pro systém plánování projektů

# Contents

<b>1 Introduction</b>	<b>1</b>		
1.1 Goals and structure of the thesis	1		
1.1.1 Application requirements	1		
1.1.2 Mobile application frameworks comparison	2		
1.1.3 JavaScript frameworks comparison	2		
1.1.4 Offline capabilities	2		
1.1.5 Architecture	2		
1.1.6 Implementation of prototype	2		
<b>2 Plantac</b>	<b>3</b>		
2.1 Categorization of application	3		
2.2 Specification of Plantac	4		
2.2.1 Scenarios of usage	4		
2.2.2 Restrictive conditions	5		
2.2.3 Functional requirements	5		
<b>3 Mobile development platforms</b>	<b>7</b>		
3.1 Hybrid applications	7		
3.1.1 Frameworks	7		
3.1.2 Plugins	9		
3.2 Native Script, React Native	9		
3.3 Progressive web applications	9		
3.4 Comparison of Native, Hybrid and Progressive apps on mobile platform	14		
3.4.1 Native apps	14		
3.4.2 Hybrid apps	14		
3.4.3 Web apps	15		
3.4.4 Progressive web apps	15		
3.5 Summary	15		
<b>4 Java Script frontend frameworks</b>	<b>17</b>		
4.1 AngularJS	17		
4.1.1 Directives	18		
4.1.2 Custom directives	18		
4.1.3 Classes	18		
4.1.4 Services	18		
4.1.5 Filters	19		
4.2 Angular 2	19		
4.2.1 Mobile first	19		
4.2.2 Modular architecture	19		
4.2.3 New directives	19		
4.2.4 Router	20		
4.2.5 TypeScript	20		
4.2.6 Testing	21		
4.2.7 Animations	21		
4.2.8 Angular Material	21		
4.2.9 Development	21		
4.3 React	21		
4.3.1 One-Way data flow	22		
4.3.2 VirtualDOM	22		
4.3.3 JSX	22		
4.4 Benchmarks	23		
4.5 Summary	24		
<b>5 Offline Capabilities</b>	<b>25</b>		
5.1 Service Worker	26		
5.1.1 Service worker life cycle	26		
5.1.2 Events	28		
5.1.3 Fetch strategies	29		
5.1.4 Other service worker features	33		
5.1.5 Sw-precache and sw-toolbox	33		
5.1.6 Summary	34		
5.2 Problems with offline	34		
5.2.1 User authentication	34		
5.2.2 User logout	34		
5.2.3 Security of locally stored data	34		
5.2.4 Conflicts	35		
5.2.5 Too big cache	35		
5.2.6 Cleared cache	35		
<b>6 Architecture</b>	<b>37</b>		
6.1 Problems addressed by architecture	37		
6.1.1 Performance	37		
6.1.2 Independence of client	37		
6.1.3 Authentication and authorization	38		
6.1.4 Offline	38		
6.2 Proposed architecture	38		
6.2.1 Server side	39		
6.2.2 Client side	39		
6.3 Summary	40		
<b>7 Implementation of prototype</b>	<b>41</b>		
7.1 Server side	41		
7.1.1 REST API	41		
7.2 Client side prototypes	45		
7.2.1 Web application	45		
7.2.2 Hybrid application	46		
7.3 Testing of prototype	46		
<b>8 Conclusion</b>	<b>49</b>		
8.1 Future work	50		

<b>Bibliography</b>	<b>51</b>
<b>A Abbreviations</b>	<b>57</b>
<b>B Contents of CD</b>	<b>59</b>

## Figures

## Tables

2.1 Diagram of service-oriented single-page web application. . . . .	3
3.1 Apache Cordova architecture. [9]	8
3.2 Identification of shell and content in app-shell architecture. [16] . . . . .	12
4.1 Graph showing time in millisecond needed to add, remove and update list of todos in each framework.[38]	23
4.2 Comparison of speed of framework while manipulating big table. Performed by methodology of Stefan Krause [53] . . . . .	24
5.1 Simplified diagram showing states of Service worker life cycle.[2] . . . .	27
5.2 Diagram showing steps of Cache only strategy.[1] . . . . .	29
5.3 Diagram showing steps of Network only strategy.[1] . . . . .	30
5.4 Diagram showing steps of Cache, falling back to network strategy.[1]	30
5.5 Diagram showing steps of Network falling back to cache strategy.[1] . .	31
5.6 Diagram showing steps of Cache then Network strategy.[1] . . . . .	31
5.7 Diagram showing steps of Cache and Network race strategy.[1] . . . .	32
5.8 Diagram showing steps of Generic fallback strategy.[1] . . . . .	32
6.1 Diagram showing architecture of application. . . . .	38
7.1 Screenshot of web application prototype. . . . .	46
7.2 Screenshot of hybrid mobile application prototype. . . . .	47







# Chapter 1

## Introduction

Mobile devices gain enormous popularity in past years, and it became essential for web applications and services to provide access from these devices. Access from mobile devices may be easier and quicker and also possible in environments where notebook or desktops are not usable. Plantac, web based time planning and tracking application, is no exception. The application is dedicated to being used at work by managers, developers and workers. These users need an application that allows them to track time quickly and easily. The previous version of Plantac is, based on user feedback, not great at providing this. Developers of Plantac identified two possible sources of the issue. The first problem of the previous version has been the too strict model of the processes and structures of projects. The second source of the problem was identified to be the slow user interface. We then decided to switch to new technologies and concepts to create the new version of this application that will provide a quick way of time logging, fast user interface and mobile access to users. The application should also be capable of providing basic functionality without the internet connection.



### 1.1 Goals and structure of the thesis

The main goal of this thesis is to examine options for development of the multiplatform application that will provide functions of Plantac on mobile devices and without proper internet connection. The examination should lead to the definition of a category of applications where Plantac belongs and proposal of an architecture that target requirement of the application and its category.



#### 1.1.1 Application requirements

The first task of this thesis is analysis and preparation of requirements for the new version of Plantac, that will be developed under name TagIt. Requirements of the application are described in Chapter 2.

### ■ 1.1.2 Mobile application frameworks comparison

Secondly, the thesis focuses on a comparison of approaches in development for mobile devices. The main focus of Chapter 3 is possibilities of the usage of web technologies, HTML, CSS and JavaScript in the development.

### ■ 1.1.3 JavaScript frameworks comparison

JavaScript frameworks are the backbone of modern one-page web applications. The framework gives developer scalable, reusable and maintainable way to write code. In Chapter 4 thesis compares modern JavaScript framework that can be used in the development of such applications. Chosen frameworks are Angular JS, Angular2 and React.

### ■ 1.1.4 Offline capabilities

The modern, user-friendly and engaging application can gain huge benefit from running without the internet connection. In most essential way application can cache its static resources in the browser and load faster, but going offline with well written *Service Worker* application can also save data and request for future use and background synchronisation. The thesis will summarise capabilities of *Service Worker* and analyse problems that may occur in Chapter 5.

### ■ 1.1.5 Architecture

The goal of this thesis is also to design of general architecture for such applications and namely for future development of Plantac/TagIt. Architecture based on finding from the previous research will be described in Chapter 6.

### ■ 1.1.6 Implementation of prototype

Finally, analysed technologies and proposed architecture will be used in the implementation of a prototype for application TagIt.

## Chapter 2

### Plantac

Plantac is cloud-based web application used for project planning and management. The application is also used by all workers in the company to log their work time.

Plantac is currently developed with Java Enterprise Edition back-end and ZK Framework frontend. The application is particularly designed to be deployed on GlassFish Application Server with a connection to PostgreSQL database server.

The project is now in a phase of the full rewrite with modern and more suitable technologies and approaches. New version should rely on REST API for communication and JavaScript Framework on the front end of the applications. The application should also provide new UI design and functions based on feedback from the previous version of the application. New application based on Plantac is developed under work name TagIt.

### 2.1 Categorization of application

Application functionally belongs to the category of office applications and information systems that are used to help efficiently manage companies and projects. The application also has similarities with issue tracking systems.

Similar applications on the market are e.g. Togl, TrackingTime, TimeCamp, Timely or from more complex time planning and issue tracking application its Jira or Redmine.



**Figure 2.1:** Diagram of service-oriented single-page web application.

Technically, intended new version of Plantac/Tagit may belong to the

category of service-oriented single-page web applications. Schema of said type of web application can be seen on figure 2.1. Based on results of research in this thesis client application may be switched or supplemented by a dedicated mobile client.

## ■ 2.2 Specification of Plantac

This thesis mainly collects information about technologies that can be used for the new version of Plantac that should also have a new mobile application and also new fronted web client on the desktop. Great solution to fasten up development could be the usage of same technology for both interfaces.

Since developers are more familiar with web technologies than native application development, research focuses on the possibility of using HTML, CSS, and JavaScript to create the mobile application. Same JavaScript framework that will be chosen in this research will also be used for web application front-end.

Mobile and also web application could benefit from offline capabilities, so also new browser API called Service Worker is covered in this thesis.

### ■ 2.2.1 Scenarios of usage

The application will be mostly used by users divided into three roles. Those roles are employees, managers of projects and accountants.

#### ■ Employee

The employee uses the application to track his work time on an assigned task and projects. Thanks to the application employee get an overview of tasks, he should be working on and their due dates. The employee can see the amount of time he spent working on each task. Employees can use the application to communicate and share documents among themselves.

#### ■ Manager

A manager is in charge of a team of employees. His primary task is to manage resources for projects efficiently. The manager uses the application in all phases of a project. First, he creates the new project in the system, allocates needed resources and sets estimated cost of the project. In the execution phase of the project, with the use of the application, manager gains control and oversight over the completion of tasks and workload of employees. The manager can plan events and work to be done at specific times. At the end of the project manager uses the application to generate time reports and invoices. The manager can also use history of projects to analyse profits of the whole company, track budget and use it as a source for estimation and planning of future projects.

## ■ Accountant

The accountant uses the application to generate monthly work time reports for each worker and issue paychecks on the end of each month. The accountant can use the application to oversee whole company spends on running projects.

### ■ 2.2.2 Restrictive conditions

Application, especially when used on the mobile device, should be able to operate without the internet connection. The application will be dealing with personal and company private data. Because of that communication between the application and the server must be secured through SSL and HTTPs.

### ■ 2.2.3 Functional requirements

In this section, I list examples of main functional requirements for the application with assigned priority and sorted by user roles.

## ■ Employee

- Application will allow the user to show a list of assigned tasks. [High]
- Application will allow the user to log work time. [High]
- Application will allow the user to start and stop time counter independent of task. [Medium]
- Application will allow the user to sort and filter tasks and projects. [High]
- Application will allow the user to show an overview of tasks for a day. [High]
- Application will allow the user to show overview in the form of weekly calendar. [High]
- Application will allow the user to plan vacations. [Low]
- Application will allow the user to set notification alerts for a task. [Medium]
- Application will accept notifications created by the system. [High]
- Application will allow the user to show a list of news and changes in selected tasks/projects. [Medium]
- Application will allow the user to create and save the report from the meeting. [Low]
- Application will allow the user to download files saved with tasks/projects on the server. [Medium]

- Application will allow the user to add messages to discussions about tasks/projects. [Medium]

### ■ **Manager**

- Application will allow the user to show and edit details of owned projects and tasks. [Low]
- Application will allow the user to show an overview of time logs in a day/week across employees. [High]
- Application will allow the user to assign work to employees. [High]
- Application will allow the user to generated time and spending report from the history of projects. [Medium]

### ■ **Accountant**

- Application will allow the user to show reports of logged work time on tasks and projects. [High]
- Application will allow the user to show reports of logged work time by employee. [High]

## Chapter 3

# Mobile development platforms

In this chapter, I introduce possibilities and approaches of using web technologies in mobile application development and then compare them.

### 3.1 Hybrid applications

Hybrid mobile applications combine web applications with the native application. Application itself is build using HTML, JavaScript, and CSS while wrapped in native web view container and thin layer that provides communication with platform APIs.[3]

This approach opens development of mobile applications to web developers, which could be a great opportunity for teams that are focused on web apps development but also wants to make a mobile application for their product.

Hybrid mobile applications are also cross-platform. *Apache Cordova*, for example, offers good support for *Android*, *iOS*, *Windows phone* and even desktop *Windows 8-10* and *Ubuntu* and other not so widely used platforms [8]. Thanks to native wrap of the application, application can be distributed through platform specific application stores.

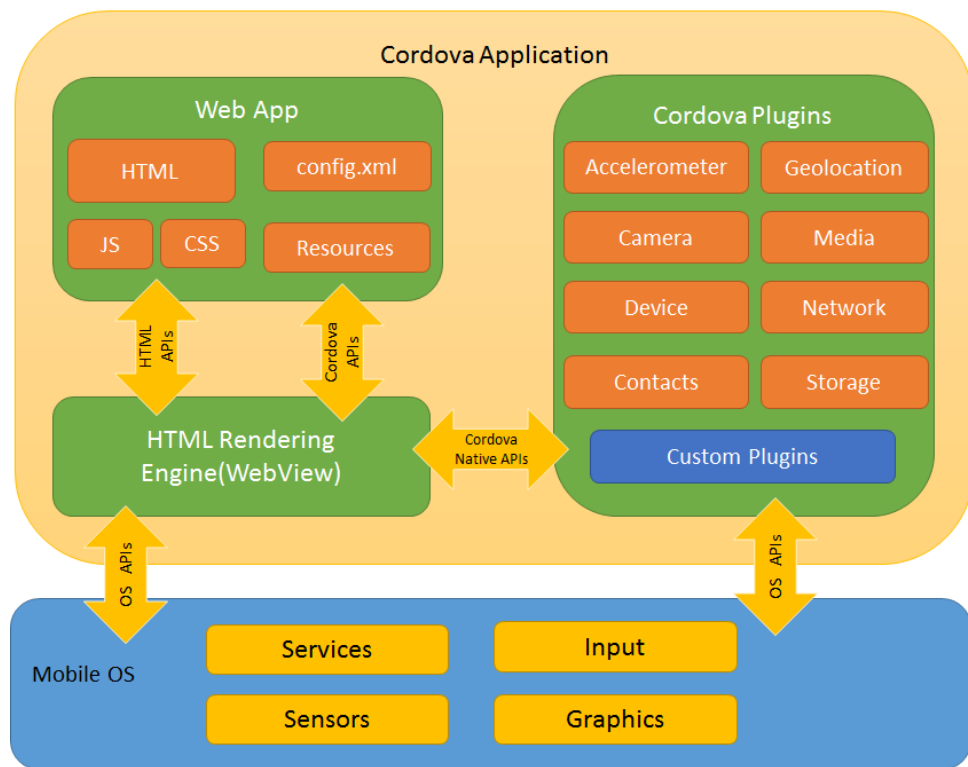
#### 3.1.1 Frameworks

Developers can use frameworks and tool prepared for hybrid application development.

#### Apache Cordova

Cordova is the main framework used for development of hybrid mobile applications. Cordova was formerly developed by Adobe as PhoneGap but then renamed and distributed under Apache Licence 2.0.[8]. Cordova framework provides the environment for applications to run in, for UI component and JavaScript framework developer have to include those or use another distribution of Cordova. For diagram of framework architecture see figure 3.1.





**Figure 3.1:** Apache Cordova architecture. [9]

### ■ Adobe PhoneGap

PhoneGap is the distribution of Apache Cordova developed by Adobe. On top of Apache Cordova as an engine of PhoneGap, PhoneGap offers a tool for developers to make development faster and easier. PhoneGap includes CLI tools, Desktop and Mobile application for creating and deploying applications during development and cloud service called PhoneGap Build that is used for building application for multiple platforms ready for distribution on app stores.[5]

### ■ Ionic

The Ionic framework also offers tools for better development with Apache Cordova. For example, Ionic Cloud offers notifications service, cloud-based builds, which is very important in development for iOS without Mac computer, since an iOS application can not be compiled on Windows, and analytics.

In my opinion, the main advantage of Ionic is the integration of Angular JS and Angular 2, in Ionic 2. Ionic also includes natively looking UI components, which will have a different style on each platform. If the developer decides for Angular 2, as I did in following chapters of this thesis, Ionic might be the choice for quick development of the natively looking application. Ionic was only for Android and iOS, but Ionic 2 will also offer support for Windows Phone 8.1.[6]

### ■ 3.1.2 Plugins

Plugins in Hybrid applications represents communication API between the native layer and the web application. Plugins are enabling the application to use resources of the platform, like camera, notifications, contacts, file system and others.

Plugins, which can be utilised with Cordova framework or frameworks based on Cordova can be found in Apache Cordova Plugins Registry. Plugins are mostly open source and developed by community or companies behind commercial distributions of Cordova.[9]

## ■ 3.2 Native Script, React Native

Another approach in making the cross-platform mobile application using JavaScript and web technologies is Native Script and React Native. These frameworks use the similar approach in the creation of mobile applications. Application logic is written with JavaScript, styles are using CSS, but instead of HTML, these frameworks uses specific XML that is then compiled to platform specific XML, which is rendered when the application is in use. That means that there is no web view involved and it might solve some inconsistency and performance issues of applications rendered in textitwebview component. [11] [12]

## ■ 3.3 Progressive web applications

Progressive web applications are the new approach to the development of web application. In core progressive web applications can use any familiar technology which makes *HTML*, *CSS* and *Java Script* web applications. But web applications adopt new technologies on top of those core and follows ten key concepts advised by Google.[14]

- **Safe** – Served via *HTTPS* to prevent snooping and ensure content hasn't been tampered with.
- **Progressive** – Work for every user, regardless of browser choice because they're built with progressive enhancement as a core tenet.
- **Responsive** – Fit any form factor: desktop, mobile, tablet, or whatever is next.
- **Connectivity** – independent – Enhanced with service workers to work offline or on low-quality networks.
- **App-like** – Feel like an app to the user with app-style interactions and navigation because they're built on the app shell model.
- **Fresh** – Always up-to-date thanks to the service worker update process.

- **Discoverable** – Are identifiable as “applications” thanks to W3C manifests and service worker registration scope allowing search engines to find them.
- **Re-engageable** – Make re-engagement easy through features like push notifications.
- **Installable** – Allow users to “keep” apps they find most useful on their home screen without the hassle of an app store.
- **Linkable** – Easily share via URL and not require complex installation.

These concepts of the progressive web application are further explained in following subsections.

### ■ Safe

Since progressive web apps may deal with sensitive data via native APIs and service worker, it’s advised to serve progressive web applications through HTTPS connection.

### ■ Progressive

Progressive enchantment is the technique used to enhance the content and capabilities of the application as much as browser or internet connection allows, but basic functionality or at least proper error message should be accessible to everyone. [15]

The core of progressive web application is new browser API *Service Worker* which is not yet available on all the browser, but the application should be still able to run even without it.

Another example of new browser API is access to the camera on the device through *getUserMedia* API. To ensure that more people can use this functionality of the application, it necessary to cover all the specific prefixes of API used by different browsers:

```
navigator.getMedia = (  
  navigator.getUserMedia ||  
  navigator.webkitGetUserMedia ||  
  navigator.mozGetUserMedia ||  
  navigator.msGetUserMedia  
);
```

And if it is still unsupported user will be prompted for it.

```
if (!navigator.getMedia) {  
  displayErrorMessage("Your browser doesn't have support for  
  the navigator.getUserMedia interface.");  
}  
else {  
  // Use Camera API
```

}

Fallbacks and polyfills should be provided where possible. The same principles go for the *CSS* and *HTML* code.

### ■ Responsive

Responsive design means that user interface is rendered in a way that fits the size of the device. The application should have a responsive design and work on every screen size. Responsive design is very critical since progressive web applications main aim to mobile devices. Responsive design can be achieved using *fluid grid* concepts, *flexible images* and *CSS3 media queries*. [15]

### ■ Connectivity independent

Service workers allow your app to work without internet connection.

A *Service Worker* a client-side proxy, written in JavaScript that puts the developer in control of the cache and how to respond to resource requests. By pre-caching essential resources, the application can eliminate dependence on the network, ensuring an instant and reliable experience for your users.

Some application still heavily depend on the online dynamic content from the servers, but even these applications can still cache UI elements and start faster and even display some cached data from the previous interaction.

In order to use *Service Worker*, developer have to create JavaScript file that will be registered as *Service Worker* and listen to at least two events, *install* and *fetch* and implement basic logic. Service worker API is further described in the following chapter.

With registered service worker, all files of the application can be saved in the browser cache and available for the user, when he comes back. Responses with data and variables of application can be saved in *localStorage* or *IndexedDB* where possible. [15]

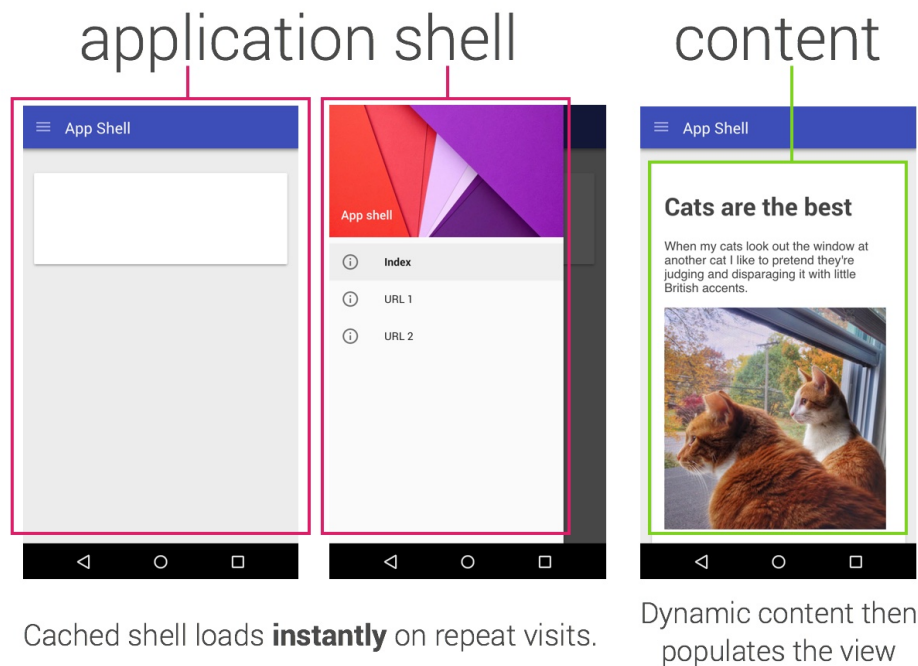
Service worker is quite new technology and is not supported by all web browsers.

### ■ App-like

Design concept called App-shell architecture is recommended to use when building Progressive web application UI. This concept separates the application into shell and content, as seen on figure 3.2.

Shell is essentially all the static parts of the application needed to show content. When is the application cached, it is necessary to cache the whole shell on the device. Cached app shell warrants that even if there is no internet connection and no cached dynamic content application will load at least familiar UI and the user will not be presented with an empty screen or default connection failure message. [16]

Content is shown within the shell. The content itself may also be cached, but it heavily depends on the type of application.



**Figure 3.2:** Identification of shell and content in app-shell architecture. [16]

### ■ Fresh

Application shell and cached content will always load from the local storage, after its cached. However, browser checks if *Service Worker* file was changed from the last visit and if so, the new version of *Service Worker* is installed for the next page load. This new service worker can cache all resources again and delete old cached data. [15]

### ■ Discoverable

Usage of Web Manifest allows the developer to provide various information about web application and additionally describe the way it should be displayed. Web Manifest also makes application installable to the home screen and openable in separate browser window. [15]

Web Manifest is a JSON file linked in the header of HTML document of the index of application. [17]

```
"name": "Progressive Web App: Plantac",
"short_name": "Plantac",
"description": "Progressive web app prototype.",
"icons": [{
  "src": "assets/icons/icon.png",
  "type": "image/png",
  "sizes": "72x72"
}, {
```

```

    "src": "assets/icons/icon-large.png",
    "type": "image/png",
    "sizes": "144x144 256x256"
  }],
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#fff",
  "theme_color": "#fff",
  "orientation": "portrait"

```

*Shortname* sets the name of application that will be displayed on the home screen. *Icons* contains array of icons with different resolutions for different devices. *Display* defines the way application is opened. *Standalone* states that application will be opened in separate window without any browser UI. Other options are *fullscreen*, *minimal-ui* and *browser*. [18]

### ■ Re-engageable

Progressive application can make use of push notifications thanks to *service workers*, *Push API* and *Notifications API* [19]. Thanks to those APIs web application can receive messages pushed from the server. These messages can be shown as notifications to the user and notify him about updates or scheduled events. That makes it easier to re-engage with the user and bring him back to the application. [15]

### ■ Installable

Any website can be saved to the home screen on Android from Chrome using "Add to Home screen" button in the menu. Unfortunately, it is not very user-friendly. [21]

However, Chrome makes it possible to prompt the user with a pop-up suggesting him to save the application to the home screen. This pop-up is not accessible by the developer, but the browser will show pop-up automatically if application fulfils three following requirements.

*Web Manifest* must be present and valid by the specification. The application must have valid and installed *Service Worker* and Application must be served over *HTTPS*. [20]

If the user confirms the dialog, the application is immediately saved to the home screen and can be comfortably accessed next time the user wants to open it. [15]

### ■ Linkable

Since progressive web applications are accessible to everyone with a browser, they can be shared directly via URL. Also content and individual pages of application should be linkable to provide shareability and possibility of reopening application at the same place. [15]

## ■ Summary

Progressive web applications are not new technology or framework, but a new approach to making web applications behave like native ones.[21] Google makes a great effort in providing tutorials and support for this method, and with more browser and platform support it might be future of mobile web. Applications can also benefit from technologies mentioned above even on the desktop, where fast and offline accessible web applications are also very useful. Especially Service Worker for caching.[15]

## ■ 3.4 Comparison of Native, Hybrid and Progressive apps on mobile platform

This section contains summary and comparison of pros and cons of possible approaches to development for mobile devices.

### ■ 3.4.1 Native apps

Native apps have wide access to all features and APIs provided by operating system and ecosystem on the mobile platform. [23] The application can use platform specific UI components, thus providing great integration and consistent user experience on the platform. Native applications can be placed in platform-specific app stores and, therefore, its installation can simply be monetized[24].

Unfortunately, code of the native application is not, in most cases, portable to another platform. In order to run the application user must have the application installed on the device. Because of this, it takes longer to start using the application, and it might be a big drawback if the user wants to use the application just once or occasionally.

### ■ 3.4.2 Hybrid apps

Application running and web view showing app written with web technologies, essentially HTML, CSS, and Java Script.[3]

Hybrid mobile applications are portable and can be released on multiple platforms. However, it depends on the accessibility of container and plug-ins for the platform. Apache Cordova supports major desktop and mobile platforms, Windows, OS X, iOS, Android, and Ubuntu. The hybrid application can be distributed via app stores. Parts of application code can be updated on start without reinstalling the whole application or update via platform specific services. [8]

Access to the operating system APIs is possible, but depends on the availability of plug-ins for the framework and also the platform, where the application should run. [10]

Since hybrid application are wrapped in the native app and its web view, it's necessary to install the application on the device. [8]

### ■ 3.4.3 Web apps

Since web apps on mobile are just UI and performance optimised web pages, they can be opened in browsers on every platform. However, they work only with the internet connection. Possibilities of web apps are limited by mobile browser APIs and performance.

### ■ 3.4.4 Progressive web apps

Progressive web applications are web applications written in an offline-first way using *Service Worker* and Manifest file to enable placement to the home screen of Android devices. optionally [29]

The progressive web application can be installed very quickly, just by accessing the website in the browser and clicking a button. Saving the application makes a bookmark on the home screen and next time is the application opened via the bookmark and in a separate window without browser bar. Since progressive web application approach also covers caching of application resources, next time user opens the application it loads instantly from the cache. An application might show just placeholders or old data, but it is much closer to the native or hybrid application feel of an app. Also, the user does not wait while looking at the white screen until the page is loaded from the internet. Progressive apps can even have an offline functionality. Progressive web apps can also use push notifications and background sync known from native applications. All communication done from service worker is SSL/TLS secured through HTTPS protocol. [15]

Compared to native and hybrid applications, progressive web apps have limited access to platform APIs. In Google Chrome on Android, it is Geolocation, Media Capture, Device orientation and Android Intent URI. Progressive web applications does not have access to device filesystem and can store data only in browser storage.[25]

In current time progressive web application works with its full potential only on Android platform. The same approach also provides above listed advantages, excluding installation, in desktop browser, but currently only in Google Chrome and Mozilla Firefox.[30] Progressive web application can be built with progressive enhancement. Therefore applications will be still useful in browsers that do not provide needed *APIs*[28]

## ■ 3.5 Summary

In my opinion, TagIt would benefit the most from the usage of progressive web applications approach. Since TagIt is primarily desktop browser application, it will benefit from most of the web applications features mentioned above. Development of good responsive design would quickly bring all of the TagIt functions to mobile devices. For this matter, I propose usage of Google Material Design, which is prepared for use on mobile devices and is also very useful in the desktop browser.



The drawback for progressive web apps is limited support on other platforms than Android. The application could still run in the browser, but without caching, offline support and installations. If developer team of TagIt decides rather to use the native app, I would suggest using the Ionic framework. Ionic offers great UI components, environment, and integration with Angular for quick development of the application.

## Chapter 4

# Java Script frontend frameworks

This chapter will cover comparison of most used modern JavaScript frameworks for front-end applications. For this comparison, I have chosen AngularJS, Angular2 and React.

### 4.1 AngularJS

AngularJS, Angular or Angular 1 is complex JavaScript Framework for front-end of web applications. AngularJS is created for making single-page web applications. Development of the framework started in 2009 by Google. A framework is accessible on the web for free under MIT license. Framework is developed in the form of open source community project, and it's now slowed down due to reallocation of most of the developers to new project, which is Angular 2, covered in next subsection.[7]

The framework includes all parts of the *Model-View-Controller* architecture pattern. *Model* is connected via two-way data-binding to HTML template, where is the data shown to user and user can manipulate it and interact with the application. Data to *View* is delivered and controlled by the *Controller*, which is connected to the template by *ng-controller directive*. *Controller* stores data and functions, which are used by the template, in *scope*, which is accessible from the template. Angular uses dirty-checking to catch changes of data in *Model* and decides if *View* should be re-rendered with new data. It's suggested keeping fewer than 2000 watchers on any page.<sup>1</sup>

Example of AngularJS code[54]:

```
dbmonControllers.controller('MainController', ['$scope', '$timeout',
function($scope, $timeout) {
    $scope.loadCount = 0;
    $scope.databases = {};

    $scope.getClassName = function(query) {
        var className = "elapsed short";
        if (query.elapsed >= 10.0) {
```

<sup>1</sup><http://stackoverflow.com/questions/9682092/how-does-data-binding-work-in-angularjs/9693933>

```
        className = "elapsed warn_long";
    } else if (query.elapsed >= 1.0) {
        className = "elapsed warn";
    }
    return "Query " + className;
};
//(...)
}
);
```

### ■ 4.1.1 Directives

*Directives* are the main part of the framework. *Directives* appear in *View* templates in form of attributes of HTML tags, classes or even tags itself. Angular provides many inbuilt *directives*, which offers easy implementation of often used functions of the application.

Main *directive* is *ng-app*, which initialize application and establish scope. *Ng-controller directive* bind *Controllers* to parts of *View*. *Ng-model* is used to two-way bind variables to *View* templates. *Ng-repeat*, for example, is used to iterate through data collections and display data in *View*, like a list or table. Angular also allows developers to create their own *directives* to further widen possibilities and functions of their applications.

### ■ 4.1.2 Custom directives

Angular allows the creation of custom *directives*, that can be used as HTML components, attributes, classes or comments. The developer is also able to limit this usage for example just to be used as an attribute only. Documentation advice using directives mainly as attributes and HTML components.

With *directive* developer can, for example, create HTML component `<person>`, with defined *View* template, attributes, style, *Controller* and *Model*. These `<person>` components can be easily reused across whole application without the need of copying or rewriting the same code. Components make templates more readable, and it also makes it easier to make changes in them.

### ■ 4.1.3 Classes

Angular also includes inbuilt *classes*, that makes it easier for *View* to react to changes in *Model*. For example, *ng-dirty* and *ng-pristine* are automatically assigned to input fields if the user made changes to them or not. *View* can react to these different states e.g. changing the colour of input.

### ■ 4.1.4 Services

If the developer wants to share some functionality among different *Controllers*, he may use *Service*. *Service* in Angular is a singleton, initialized at the moment,

when there is component depending in this *Service*. Angular also consists of many built-in services, such as *HTTP service* used for HTTP calls. *Services* are connected and delivered to *Controllers* using *Dependency Injection*.

### ■ 4.1.5 Filters

*Filters* are used to format data in *View* templates. For example, filter *date* is used to display date in desired format or filter *orderBy* is used to order data in collections. *Filters* take data and optionally other arguments and returns filtered or formatted data. *Filter* is simply a function, but specifically, bind to *View* template. *Filters* can also be chained one after another to make more than one transformation of data. Angular as well allows the developer to create custom *Filters*.

## ■ 4.2 Angular 2

Angular 2 is a new rewrite of the previous Angular 1. Development started at fall of the year 2014 and version 2.0.0 was released in September 2016. Angular 2 is not backwards compatible with Angular 1, but most concepts are very similar, and there are detailed guides how to migrate Angular 1 applications. This section covers new features of Angular 2 in comparison to AngularJS.

### ■ 4.2.1 Mobile first

Angular 2.0 focuses on the mobile development of applications and mobile web pages. Developers focus on performance problems of mobile applications first. In most cases, the mobile platform provides limited processing power, lower memory, and other limitations. [31]

All those limitations have been considered during Angular 2 development. In result, it is claimed to bring better performance also on the desktop.

### ■ 4.2.2 Modular architecture

Framework core consists only of a limited number of necessary *modules*, that way developers are able to optimise the size of the application and limit overhead of the framework in case of simple applications without the need for certain modules. Angular 2 uses module system of ECMAScript 2015 and is compatible with modern packaging tools like Webpack or SystemJS.[31]

Those modules are for example *http* or *router* module. Developers can also write custom modules for Angular. Dependency injection has also been improved.

### ■ 4.2.3 New directives

*Directives* appear in Angular in three types. *Component Directive* creates components with templates. *Attribute Directive* changes appearance or be-

haviour of element. *Structural directives* are used to modify DOM elements in rendered template. Built-in *directives* of this type are *ngIf*, renders element if conditions is fulfilled, *ngSwitch*, is template equivalent of switch known from programming languages, and *ngFor*, is used to iterate through collections of data. [33]

#### ■ 4.2.4 Router

One of the Angular modules is *Router*, that is used for work with URL and navigation through application.[34]

*Child Router* enables developers to create sub-routes, for example, the special router for user management subsection. The *child router* can be easily separated from rest of the routes and used in more places across application, or use the different base template.

*Screen Activator* is used to gain control over the process of navigation. For example, if the user has some unfinished work on the page, the application can react to it and prompt the user about it or save work automatically. The router is built as pipeline architecture and enables developers to insert functions to the pipeline. For example, preloading of next page or control of authentication and authorization during navigation to subsections with limited access.

#### ■ 4.2.5 TypeScript

Angular 2 is written in *TypeScript*. *TypeScript* is superset of Java Script developed by Microsoft. *TypeScript* implement many of functions of ES2016+, for example very useful lambda functions. Therefore, *TypeScript* have to be compiled into pure Java Script that is readable by browsers. *TypeScript* is recommended to be used in Angular 2 development, but it is not mandatory. Angular 2 can be used with pure Java Script or e.g. *Dart* programming language. [37]

Example of Angular2 code in TypeScript[54]:

```
app.AppComponent = ng.core
  .Component({
    selector: 'my-app',
    template: `
      <td *ngFor="#query of database.topFiveQueries"
        [ngClass]="getClassName(query)">

        <div class="popover left">
          <div class="popover-content">

          </div>
          <div class="arrow"></div>
        </div>
      </td>
```

```

    },
    //directives: [angular.NgFor]
  })
  .Class({
    //(...)
    getClassName: function(query) {
      var className = "elapsed short";
      if (query.elapsed >= 10.0) {
        className = "elapsed warn_long";
      } else if (query.elapsed >= 1.0) {
        className = "elapsed warn";
      }
      return "Query " + className;
    }
  });

```

#### ■ 4.2.6 Testing

Angular 2 provides support for testing with *Karma* and *Protractor*.<sup>[31]</sup>

#### ■ 4.2.7 Animations

Support for complex and high-performance animations of UI is also part of Angular 2.<sup>[31]</sup>

#### ■ 4.2.8 Angular Material

Material Design components written in Angular 2 are also in development by Angular 2 team. Material design is widely used concept of design presented by Google for Android, that then expanded to all kind of design applications.<sup>[35]</sup>

#### ■ 4.2.9 Development

In the time of finishing this thesis, Angular 2 is in version 2.4, next major version release with number 4.0.0 is planned for March 2017. This version, skipping number 3 to avoid confusion in usage with Router 4.0, will be compatible with Angular 2 with the possibility of breaking changes, but it will not be ground up rewrite as version 2.0 was. <sup>[36]</sup>

### ■ 4.3 React

React is JavaScript library dedicated to the creation of View in MVC architecture. React is developed by Facebook and community of developers under BSD license with appended patent clause and source codes are accessible on GitHub. View in React consists of components, which can recursively contain

other components, referenced by HTML tags. Main advantage of React is VirtualDOM and rendering only changes to real DOM.[13]

### ■ 4.3.1 One-Way data flow

Data is passed to View as the value of attributes in HTML tag of components. Same way can be used to pass callback functions to components. Functions are the only way to affect parent component because a change of data in child component have no effect on parent component without invocation of assigned callback function. Facebook describes this behaviour as Flux architecture. [13]

### ■ 4.3.2 VirtualDOM

An important part of React is VirtualDOM. VirtualDOM is constructed in memory, where the state of the View is rendered. This VirtualDOM is compared to DOM displayed in the browser and minimal set of changes, which are needed to be made real DOM to make it equal to VirtualDOM, is computed. Those changes are then made to DOM. In practice it means, that developer describes how the state of data should be rendered in View, rather than describing changes of View. [13]

### ■ 4.3.3 JSX

JSX is an extension of syntax for JavaScript, which enables writing of HTML directly to JavaScript. This HTML code is translated to calls of function from React library. There is also the possibility to develop in React without JSX, but the syntax of the function is not much readable in development. Therefore it is not recommended. [13]

Example of React code in JSX[54]:

```
var Query = React.createClass({
  render: function() {
    var className = "elapsed short";
    if (this.props.elapsed >= 10.0) {
      className = "elapsed warn_long";
    }
    else if (this.props.elapsed >= 1.0) {
      className = "elapsed warn";
    }

    return (
      <td className={"Query " + className}>
        {this.props.elapsed ? formatElapsed(this.props.elapsed): ''}
        <div className="popover left">
          <div className="popover-content">{this.props.query}</div>
          <div className="arrow"/>

```

```

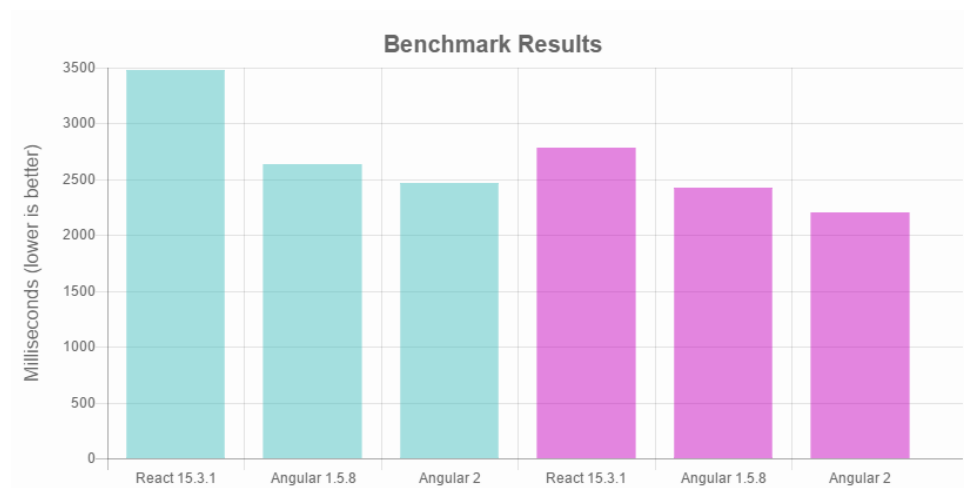
</div>
</td>
);
}
})

```

## 4.4 Benchmarks

In this section, I will present performance benchmarks comparing the speed of frameworks.

The first graph, see figure 4.1, shows time needed to perform the action on the list of 99 todos in simple ToDo applications. Angular performs much better than React in this case. However, optimisation of React had a bigger impact than optimisation of Angular 2. AngularJS stays in the middle in both cases.



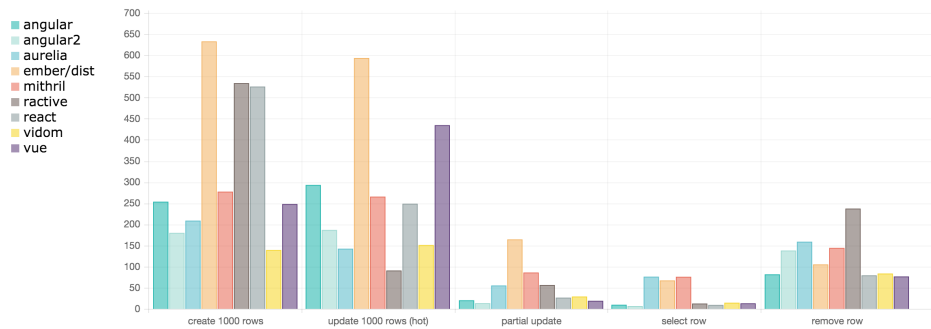
**Figure 4.1:** Graph showing time in millisecond needed to add, remove and update list of todos in each framework.[38]

The second graph, see figure 4.2, shows the comparison of a large amount of data. Methodology, prepared by Stefan Krause<sup>2</sup> compares times of creation of 1000 rows of the table, updating all of them, partial update and selection and removal of row. In this benchmark much faster then React. In updating rows React performs better or equally as Angular2. Removal of the element is faster in Angular2.

This graph can also serve as a comparison with other JavaScript frameworks.

<sup>2</sup><http://www.stefankrause.net/wp/?p=218>





**Figure 4.2:** Comparison of speed of framework while manipulating big table. Performed by methodology of Stefan Krause [53]

## 4.5 Summary

For the development of mobile and also desktop web application I decided to choose framework Angular 2.0. Angular 2.0 is a quality modern framework, addressing many problems of development with ease. According to above-mentioned benchmarks is Angular also very quick. I like the way Angular organises code into components and services, the type system of TypeScript, which makes the code more readable and safe and inbuilt function and service.

In the time of finishing this thesis, Angular 2 is already released and used by many developers. At the beginning of work, I had to consider a possibility of developing in Angular 1 and then migrate to Angular 2, which is still possible but I recommend working directly in Angular 2.

React, even if its rendering and component creation approach are easy and innovative, is in my opinion too specialise on simplifying rendering and description of *View* in MVC. In other parts of the system, the developer has in my opinion, too much freedom and have to make most of it himself or find another solution and integrate it with React framework. I like the simplicity of Angular 2 solutions and that Angular 2 offers whole *Model-View-Controller* architecture.

## Chapter 5

### Offline Capabilities

Even if mobile internet signal coverage got better with LTE technology, there are still places where internet connection is not sufficient or even absolutely unavailable. Two main examples in the Czech Republic are railway corridors and subway. Average coverage of main railway corridors in the Czech Republic is around 66 % [39] and is proposed to be 100 % first in 2021. The mobile signal in Prague is now available only in stations and in test service between stations Bořislavka and Nemocnice Motol. Full coverage of Prague subway is not proposed to any particular year, but some negotiations already took place. [40]

Internet connection might not also be unavailable due to security policies of companies. For example, if workers of the company, that is using Plantac, where hired to do their work in the compound of the company, which forbids internet access on their property, they would not be able to check their assigned task or log their work time properly. DataVision also develops another application to be used during inspections in factories. That application is now also web based, and internet connection is essential, so application is unusable during an inspection in factories with security policies forbidding internet access.

The native and hybrid application can be programmed in a way that does not require the internet connection for all operations. Since all the client logic of the application is already saved on the device, the application might require internet connection only for data fetching from and sending to the servers. This problem can be solved by for example saving of unsuccessful request in the device and retrying later the when application has the connection or manual caching of data.

Web applications need internet connection from the start. Some web application may be capable of working without the internet connection, but after the user closes the window application never loads back without internet access. To address this problem Service Worker was introduced in 2014[41].

Another possible solution for the offline desktop application made with web technologies is Electron[42]. Electron uses the mixture of Chromium and Node.js to wrap HTML, CSS, and JavaScript of the application and runs on Windows, Mac or Linux systems. It is very similar to what hybrid applications and Cordova do on mobile platforms.

## 5.1 Service Worker

*Service Worker* is an event-driven worker script<sup>1</sup> used to intercept the communication of website or application with server and resources. Via *Service Worker*, the developer can control caching of resources on a very precise level and with high control over the process.

*Service Worker* gives developer ability to create his page or application in an offline-first way or as a progressive app. That means that the web page is accessible even offline, with cached data, or faster with bad connectivity. For example, a simple web app managing users todos written without service worker will show just standard “You are offline” message by the browser when accessed without the internet connection.

However, the application is written with *Service Worker* can show cached GUI, run scripts and show the user his cached todos from the previous visit. Service worker can even register data sent to the server, like new `ToDo` or change in existing `ToDo`, and send them when the internet connection is available, or receive push notifications from the server.

Service worker is a progressive enhancement of the application or web page. That way application should still run with good internet connectivity on any device or platform it was compatible with before adding service workers. Users with compatible browsers will get better user experience with the application.

### 5.1.1 Service worker life cycle

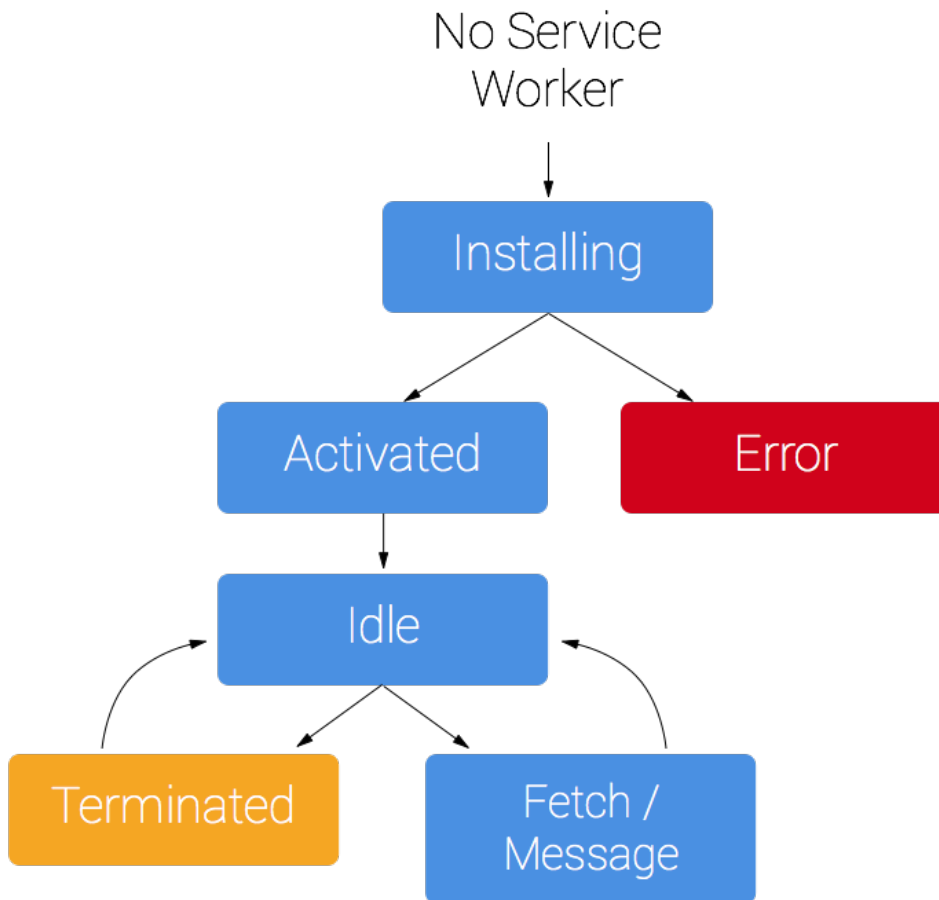
Service worker has its lifecycle separated from a web page. For simplified lifecycle diagram see 5.1 Installation of the service worker starts by registration of it in JavaScript.

```
if ('serviceWorker' in navigator) {
  // Path is relative to the origin, not project root.
  navigator.serviceWorker.register('/sw.js')
  .then(function(reg) {
    console.log('Registration succeeded.');
```

```
  })
  .catch(function(error) {
    console.error('Registration failed with ' + error);
  });
}
```

After registration browser will start installing service worker in the background. In installation stage *Service Worker* is setting up an environment, and it is the best time to cache static resources of the application or create indexedDB scheme to store cached data. If any of operations fails service, worker fails to install. An installation will be attempted again after the refresh. If the installation of the *Service Worker* succeeded and everything is prepared in the cache.

<sup>1</sup><https://developer.mozilla.org/en-US/docs/Web/API/Worker>



**Figure 5.1:** Simplified diagram showing states of Service worker life cycle.[2]

```

this.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open('v1').then(function(cache) {
      return cache.addAll([
        '/',
        '/index.html',
        '/assets/css/styles.css',
        '/assets/js/script.js',
        '/assets/icons/icon.png',
        '/assets/icons/icon-large.png',
        '/manifest.json'
      ])
    }).then(function() {
      console.log('Success');
    })
  )
});

```

```
});
```

Next step is Activation. If there is no other window with same application opened, it is time to clean previous *Service Workers* and cached static data. Otherwise, service worker waits til all client windows are closed.

Activated service worker gains control over all pages under its scope. However, the first time service worker has registered it gains control of the page after reload.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    // Try the cache.
    caches.match(event.request)
      .then(function(response) {
        // Cache, than fallback to network strategy.
        return response || fetch(event.request);
      })
  );
});
```

Activated service worker can switch to two more states. Service worker can be terminated, to save memory or proceeds to the state where it can handle events, fetches, and messages. [2]

### ■ 5.1.2 Events

Service responds to events fired during its lifecycle.

- Install

Event fired when installation starts. Service worker gains control and can prepare the environment.

- Activate

Event fired when service worker is activating. Service worker can now, for example, clean old cache.

- Message

Event fired when service worker gets the message from the application.

### ■ Functional events

- Fetch

Fired when the application makes HTTP request, navigation in scope or resource call. Here developer can intercept the communication and decide which strategy should be used to fulfil the request.

- Sync

Fired when the internet connection is established, after a failed attempt to, for example, save a todo. The application has to register for sync event and when the sync event is fired service worker do the work. For example, save new todos in the queue to the server.

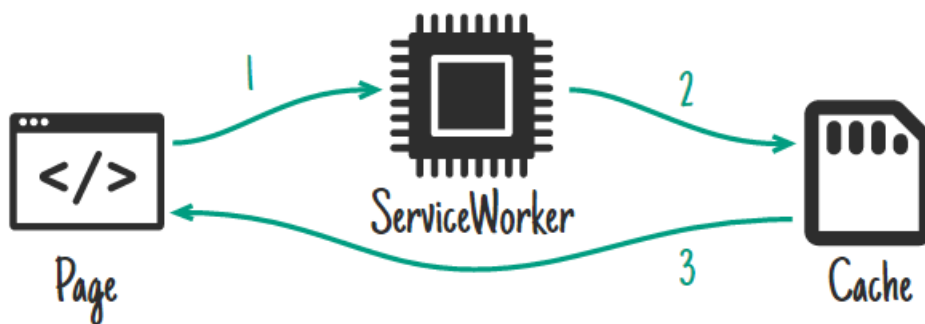
- Push

Fired when a push notification come in from notification service. Then service worker can for example update cache and show the notification to the user.

### ■ 5.1.3 Fetch strategies

There are several strategies how to react to *Fetch* event.

- Cache only



**Figure 5.2:** Diagram showing steps of Cache only strategy.[1]

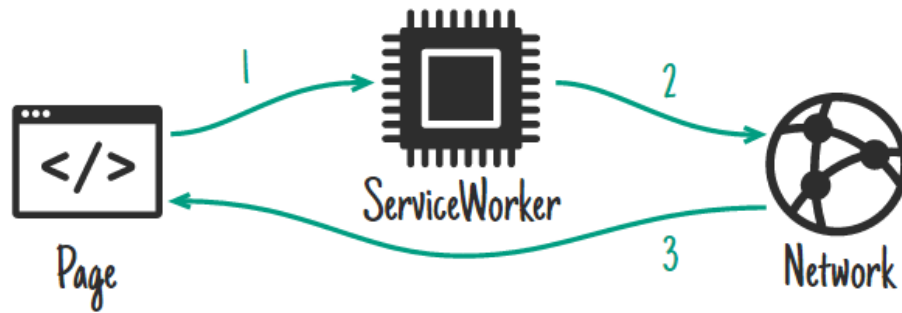
Service worker goes for response to cache only, and the response goes directly to the page, see figure 5.2. Ideal for static parts of the app which are cached in the install event handler and are determined to change only with the new version and therefore new *Service worker* installation [1].

- Network only

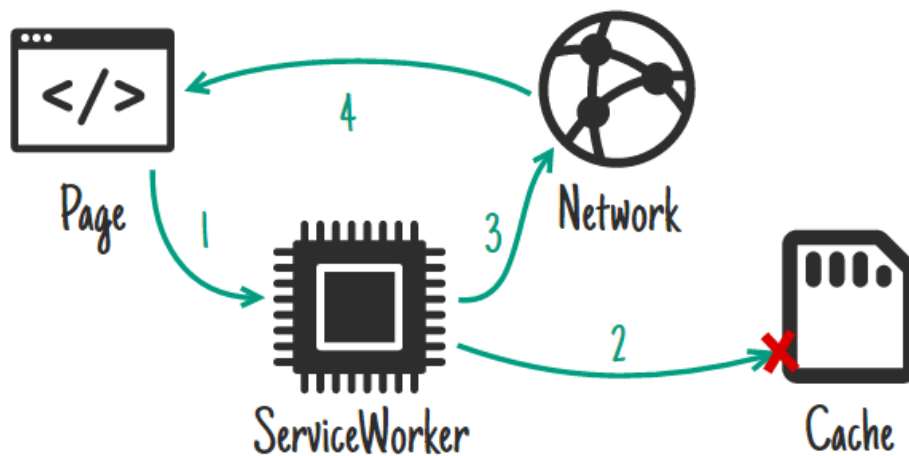
Request goes straight to network and then back to the page, see figure 5.3. This strategy is ideal for operations that can't be done offline, such as analytics pings or request with other methods than GET[1].

- Cache, falling back to network

Service worker tries to look for a response in the cache first, if there is a cached response, service worker returns the cached response to the app, else service worker tries to get the response from the network and returns the promise, see figure 5.4.



**Figure 5.3:** Diagram showing steps of Network only strategy.[1]

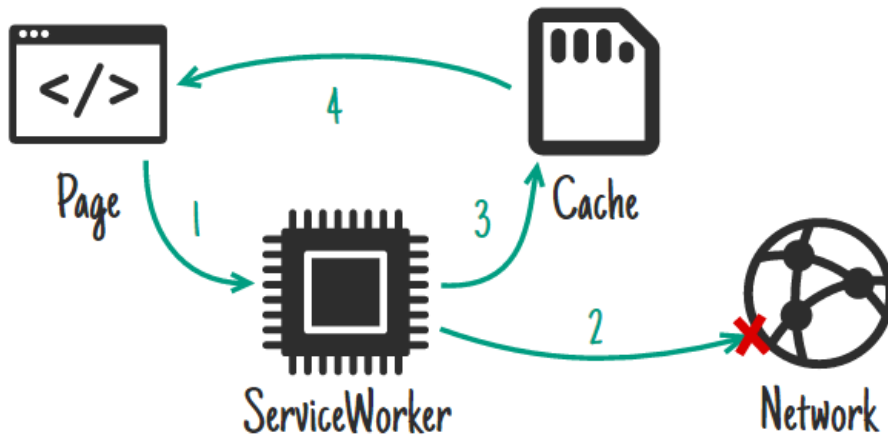


**Figure 5.4:** Diagram showing steps of Cache, falling back to network strategy.[1]

This strategy handles a majority of the request in the offline-first app. This strategy covers both cache only and network only strategies, so there is often no need to cover them separately. [1]

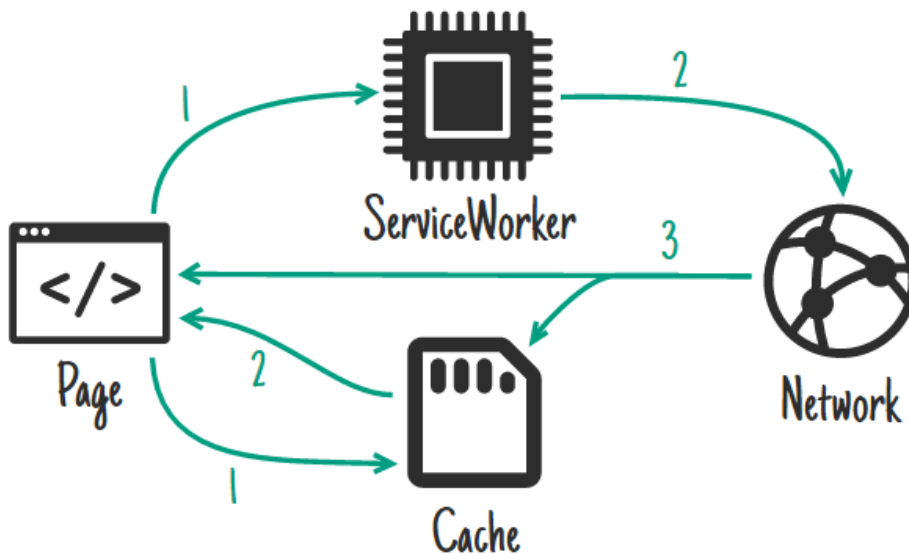
### ■ Network falling back to cache

In this strategy, request is send to the network. If this request fails, service worker returns cached response, if there is any. For image representation see Figure 5.5. This strategy gives the on-line user all the new and up-to-date data, and the offline user gets the cached version of the site. If the network request is successful, service worker should save the response to the cache for future use. This strategy is useful for frequently updated resources, such as articles, messages, news feeds, etc. This method, however, does not work properly if the user has slow or not reliable internet connection. In that case, waiting for response from the network can take a long time and it is very unpleasant user experience, see Cache then network for better strategy.[1]



**Figure 5.5:** Diagram showing steps of Network falling back to cache strategy.[1]

#### ■ Cache then network



**Figure 5.6:** Diagram showing steps of Cache then Network strategy.[1]

This strategy is ideal for the same data as the previous one. But it requires changes on the side of the application not only in the service worker. The web application has to create two requests.

One for fresh data from the network and one straight to the cache for the older cached version of data, see figure 5.6. The application will show user response from the cache, which, if it exists, probably responds first. Then, when the application gets the response from the network, the view is refreshed with new data or a new message is for example added to the list. This behaviour depends on the type of application.



Service worker in this strategy sends request to the network, then updates the cache with the response and return response to the application.[1]

#### Cache and Network race

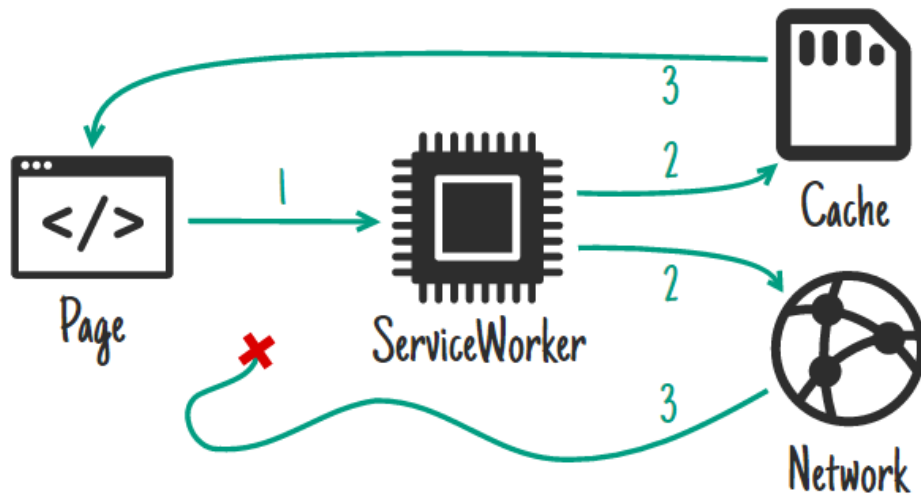


Figure 5.7: Diagram showing steps of Cache and Network race strategy.[1]

In this strategy service worker tries to get the response both the cache and the network, then serves the app the faster response, see figure 5.7. It can be used on devices where it is faster to download something than find it on the hard drive.[1]

#### Generic fallback

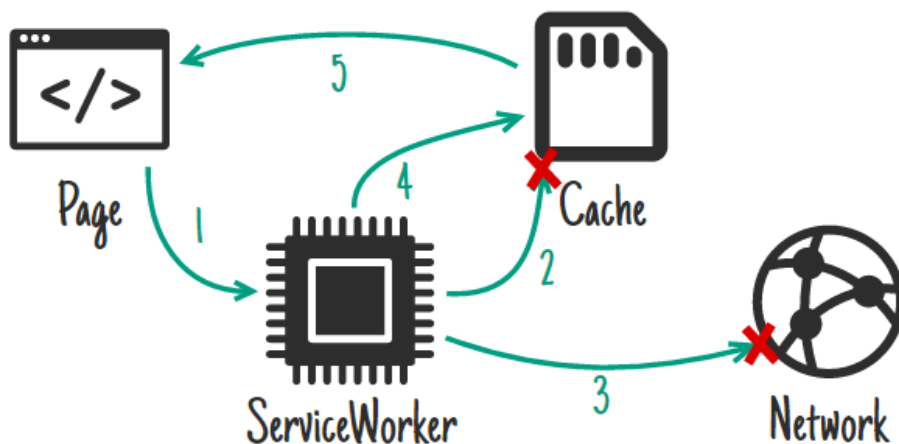


Figure 5.8: Diagram showing steps of Generic fallback strategy.[1]

If either cache or network fails to serve response, service worker may provide some generic response such as default avatar or some “No connection” error page, see figure 5.8. If the request is for example POST with a message to be send, service worker can save the request to indexedDB and send it later when the connection is available, and for now respond with code that lets page know what happened.[1]

## ■ 5.1.4 Other service worker features

### ■ Background Synchronization

Background synchronisation is a very useful feature of Service Worker. Sync events the only when there is an available internet connection, and if actions fail, it retries when there is a connection or after some time determined by exponential back-off. This is useful for non-urgent updates, for example, social media timeline or news feed. [44]

### ■ Push Notification

Service worker can also be used to handle Push Notifications and show them to the user. This feature relies on PushManager implementation in the browser, which is now available only in Firefox, Chrome, and Chrome for Android 51.[45]

## ■ 5.1.5 Sw-precache and sw-toolbox

Sw-precache<sup>2</sup> is the module for generating service worker in the build script, for example, written with *gulp*. As part of the configuration, sw-precache gets a list of URLs to static resources of the application. This list can also be in the form of patterns that are matched to files using *glob*<sup>3</sup> generates a hash of files and stores this information in generated service worker. During installation phase of service worker life cycle, files from the list are cached in *Cache Storage* of the browser. Generated service worker also includes logic that serves cached resources.

Sw-toolbox<sup>4</sup> is the library of useful functions and implemented caching strategies for simple development of custom service workers.

Sw-precache and sw-toolbox can be used together, thanks to the runtimeCaching option in the configuration of sw-precache. In runtimeCaching option developer can specify urlPatern, handler, that is a name of strategy that will be used, and optionally also options for cache, maximum of entries and name. Sw-precache then generates calls for sw-toolbox that are used to process calls for configured URLs.

---

<sup>2</sup><https://github.com/GoogleChrome/sw-precache>

<sup>3</sup><https://github.com/isaacs/node-glob>

<sup>4</sup><https://github.com/GoogleChrome/sw-toolbox>

### ■ 5.1.6 Summary

*Service Worker* is very useful new browser *API*. Developer gains full control over communication of the application and especially over caching of requests and static application resources. The application can be easily made in the offline first way thanks to service worker.

Service workers are now supported in a limited number of browsers, but their absence should not have any impact on usage of the application in standard online mode. However, application and its users will benefit from service worker in supported browsers.

With the usage of tools like *sw-precache* and *sw-toolbox* implementation of service, worker can also be very quick.

## ■ 5.2 Problems with offline

Making application run offline may bring problems with security and storage persistence.[43]

### ■ 5.2.1 User authentication

Standard authentication of user credentials can not happen without the internet connection. Credential have to be checked against database entries. However, we might create local copy of user account stored in *LocalStorage* and check credentials locally if internet connection is not available.[47] This solution might also create weak spot since a possible attacker can see how is the password hash generated and password hash is stored locally [46].

### ■ 5.2.2 User logout

If user logs out of the application, when the internet connection is unavailable or data synchronisation is not done yet, all of the unsynchronized data might be lost. The suggested solution is to prompt the user about it and let him choose if he wants to log out and delete unsynchronized data. The application could also store data in *localStorage* or *IndexedDB* even if the user is not logged in but that data might not be secure and still could be lost if the user never signs in again.

### ■ 5.2.3 Security of locally stored data

By default, data stored in browser's *localStorage* or *indexedDB* are not secured. User or admin with privileges to the machine can access *localStorage* or *indexedDB* and read, write or modify data stored there. Also, any JavaScript served from the same origin can access data in storage.[48]

### ■ 5.2.4 Conflicts

If the user works offline in collaboration with other users on same data, conflicts may occur, and the application has to address them in some way. A possible solution is, for example, informing the user about changes, which have been done to data since the user cached them, and let him review changed data if he wants to.

Another problem might be submitting same data twice. There can be situations where these duplicates can be easily detected, e.g. in Plantac creation of the same task with the same name, but there can also be situations where detection is not possible. For example in Plantac user can log time twice with same data in fields and it is correct. He just worked on the same tasks in the morning and the evening but did not specify the start and finish time while submitting the work time. However, there can be the situation where the user submits time through mobile application in offline mode, and later on the desktop, he may submit the same time again assuming the submission from application failed. But the mobile application is not yet synced due to some internet connection problems, and it submits the work time to the server when mobile gains internet connection. Unfortunately, this situation is hard to detect programmatically.

### ■ 5.2.5 Too big cache

Offline applications have to store all the data in cache, localStorage or indexedDB. Their stores have limited size. For example storage size in Chrome is limited to 6% of free disc space[55]. The developer must have this limitation in mind and cache only data needed to run the application and clean old cache entries. *Quota Management API* can be used to check the available storage space.

### ■ 5.2.6 Cleared cache

By default data saved the in the cache, localStorage or IndexedDB can be cleared by browser or user. In that case, all unsynchronized data will be lost. The application is not informed about this action by the browser. [52]

However, storage can be set to two types:

- Persistent

Persistent data is data that is intended to be kept around for a long time. This type will only be evicted if the user chooses to (for example in Firefox there is a "clear storage" button on the page info dialog for each page.)[50]

In Firefox developer can choose to save data to storage as persistent. Chrome, since 55, have its own policy and sets storage to persistent if any of following conditions is true.[51]

- The site is bookmarked (and the user has 5 or fewer bookmarks)

## 5. Offline Capabilities

---

- The site has high site engagement
- The site has been added to home screen
- The site has push notifications enabled

Persistent storage can be still cleared by the user.

- Temporary or best-effort

Temporary marked data is data that doesn't need to persist for such long time. This storage will be evicted under a least recently used policy when storage limits are reached.

## Chapter 6

### Architecture

In this section, I will cover suggestions for making the application using web technologies that can be used on mobile phones and in offline mode. First, I will describe problems addressed by architecture and afterwards describe server and client side in detail.

Architecture is intended for office applications or information systems. With the aim to deliver fast and comfort interface for users of applications.

#### 6.1 Problems addressed by architecture

Architecture is designed to address common problems and specific problems derived from feedback to the previous version of the Plantac.

##### 6.1.1 Performance

Previous version of Plantac running *Java Enterprise Edition* with *ZK Framework* had problems with performance. Users reported inconvenience in usage while they have to wait for long loading times. Proposed solution of this problem is the usage of the JavaScript framework to create one-page client side application. Application will communicate with server *REST API* over *HTTPs* protocol. Transferred data will be compressed to speed up performance further. For the *JavaScript* application I recommend usage of Angular 2 framework, but same architecture can be used with other *JavaScript* frameworks as well.

##### 6.1.2 Independence of client

Client and Server side of the application should be very loosely coupled, to provide developers with the possibility of making multiple client applications, e.g. separated desktop and mobile application or integration with other applications. Independence will be achieved by separation of client and server side code, which will in this proposed architecture communicate through *REST API* over *HTTPs*.

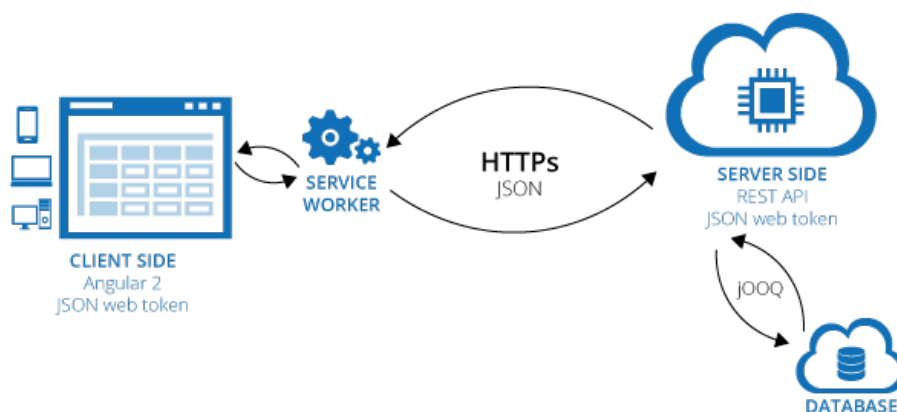
### 6.1.3 Authentication and authorization

Since communication between server and client side is stateless, server stores no session cookie or session ID. Therefore we need another way to validate authorization of that user. I propose usage of *Json Web Token* that is sent alongside every request from the client to server and stored in the memory of application or browser. *Json Web Token* token which consists of three parts, header, payload, and signature. The header contains information about token itself, type of the token itself and the algorithm that is applied for hashing. The second part, Payload, holds the claims. Claims are statements about the user and additional meta-data about the token. For example *expiration date*, *username* or *roles*. Both header and payload is encoded with *Base64*<sup>1</sup>. Signature is created by hashing Base64 encoded header and payload together with the secret that is stored on the server. Signature is used to verify that the token is authentic and was not changed along the way. These three parts are concatenated to one string separated by ".". *Json Web Token* is sent to the server with every request in authentication header of HTTP request.<sup>2</sup>

### 6.1.4 Offline

Application based on this architecture should be capable of providing client functionality without an internet connection. This will be achieved using *Service Worker*, *REST API* and following *Progressive web applications* recommendations. New proposed architecture also have the client more loosely coupled with the server than the current solution used in Plantac, where client side does work without an internet connection at all.

## 6.2 Proposed architecture



**Figure 6.1:** Diagram showing architecture of application.

<sup>1</sup><https://tools.ietf.org/html/rfc4648>

<sup>2</sup><https://jwt.io/introduction/>

### 6.2.1 Server side

Developers can choose from many frameworks and programming languages for implementation the server side. For the implementation of TagIt back-end, we have chosen the standard edition of Java with custom picked set of libraries. Primarily Jersey servlets serving REST API, jOOQ library<sup>3</sup> for data layer and Java JWT<sup>4</sup>.

The main feature of the server, in this context, is *REST API* for the client. Resources, provided in the RESTful way, can be easily cached and stored for offline use.

### 6.2.2 Client side

The user of application will interact with the client side of architecture. Client side application will communicate with server side REST API to access resources. Based on previous research I suggest client side application made with Angular 2 framework and following Progressive web applications concepts.

#### Java Script

The client side of application should be developed using modern JavaScript Framework for one-page applications, for example, as is suggested in previous sections, Angular 2.

#### Progressive Web Application

I strongly suggest usage of Progressive web application approach. Progressive web application makes great use of modern browser API such as Service Worker. Since application developed by this suggested architecture should have offline capabilities, Service Worker and caching are essential. The progressive web application also includes App-shell architecture of client application. That means that client application should be divided to *shell*, which is the static part of the application, and dynamic content accessed from the server side. This way shell can be cached in Service Worker and loaded even if the application is offline.

#### User interface

The application should work well on mobile devices, therefore it should provide special UI optimised for the mobile device or fully responsive UI. Great way to achieve this is the usage of Google Material Design or Bootstrap components<sup>5</sup>.

---

<sup>3</sup><https://www.jooq.org/>

<sup>4</sup><https://github.com/auth0/java-jwt>

<sup>5</sup><http://getbootstrap.com/>



## ■ 6.3 Summary

Application developed based on this type of architecture will benefit from statelessness of REST API by increased performance thanks to the scalability of server-side of the system. Single page application written with Angular 2 will have better performance than the currently used solution.

Con of the proposed architecture might be the usage of new technologies that are not implemented in all of the browsers or platforms. For example *Service Worker* is currently unavailable in Edge and Safari browsers [1].

# Chapter 7

## Implementation of prototype

To further investigate features of Angular 2 mentioned above, Progressive Web Apps and Hybrid mobile applications two implementations of the prototype were made.

Implemented prototype is capable of user authentication and management of tracked work time in Plantac.

### 7.1 Server side

Server side, which is used with the prototype of application, runs on Jetty server with Jersey servlets serving REST API. Data sent from and to API in the form of JSON is parsed by Jackson to Java Objects. Persistence layer of the application is provided by JOOQ library communication with PostgreSQL database.

For user authentication application uses JSON Web Tokens, that are sent alongside request.

#### 7.1.1 REST API

The client side of application communicates with the server through REST API with following resources.

##### User

*URL:* /user/auth

*Method:* POST

*Description:* Call to this resource authenticate provided credentials and return JSON Web Token or 401 response code.

*Data:*

```
Request:
{
  tenant : [string], //Name of tenant, for example company,
                    that is used as name of database scheme
  username : [string],
  password : [string]
```

```
}

```

Response:

```
{
  token : [string], //JWT token encoded in base64
}
```

## ■ Tag

Tags in Plantac(TagIt) are used to specify what kind of work time is logged and to what it belongs. Every entity (Companies, Users, Projects, Tasks, etc.) will have its unique tag.

*URL:* /tag/

*Method:* GET

*Description:* This resource provides list of tags. Tags can be filtered by *query* or *category* and paginated using *skip* and *take* parameters. By default resource returns 5 tags for each available category.

*Parameters:*

Request:

```
query: [string] //search query for tags
category: [string] //category of tags
skip: [number]
take: [number]
```

*Data:*

Response:

```
[{
  id: [string],
  name: [string],
  category: [string],
  childtags: [Tag[]] //Tags assigned to tag. For example
                    company tag assigned to project.
}]
...]
```

## ■ TimeLog

TimeLog represents logged work time.

*Base URL:* /log/

*URL:* /log/:id

*Method:* GET

*Description:* This resource return one TimeLog with provided id.

*Data:*

Response:

```
{
```

```

id: [string],
message: [string],
teamMember: [string], //Id of user this time is assigned to
date: [string], //YYYY-MM-DD date of work
start: [string], //HH:mm time of start of work
end: [string], //HH:mm time of end of work
duration: [number], //Duration of work in minutes
event: [string], //Optional type of event instead of work.
        For example vacation or illness
tags: [Tag[]], //Tags assigned to timelog.
        For example tag of project and task.
}

```

*URL:* /log/mine

*Method:* GET

*Description:* This resource return time assigned to authenticated user. Timelog can be filtered by date internal and paginated with *skip* and *take* parameters.

*Parameters:*

**Request:**

```

from: [string] //YYYY-MM-DD date
to: [string] //YYYY-MM-DD date
skip: [number]
take: [number]

```

*Data:*

**Response:**

```

[ {
  id: [string],
  message: [string],
  teamMember: [string], //Id of user this time is assigned to
  date: [string], //YYYY-MM-DD date of work
  start: [string], //HH:mm time of start of work
  end: [string], //HH:mm time of end of work
  duration: [number], //Duration of work in minutes
  event: [string], //Optional type of event instead of work.
        For example vacation or illness
  tags: [Tag[]] //Tags assigned to timelog.
        For example tag of project and task.
}
... ]

```

*URL:* /log/mine

*Method:* POST

*Description:* Creation of new TimeLog

*Data:*

```
Request:
{
  message: [string],
  date: [string], //YYYY-MM-DD date of work
  start: [string], //HH:mm time of start of work -optional
                //Either start-end or duration is needed
  end: [string], //HH:mm time of end of work -optional
  duration: [number], //Duration of work in minutes -optional
  event: [string], //Optional type of event instead of work.
                For example vacation or illness -optional
  tagids: [number] //List of ids of Tags assigned to timelog.
                For example tag of project and task.
}
```

```
Response:
//Newly created TimeLog
{
  id: [string],
  message: [string],
  teamMember: [string], //Id of user this time is assigned to
  date: [string], //YYYY-MM-DD date of work
  start: [string], //HH:mm time of start of work
  end: [string], //HH:mm time of end of work
  duration: [number], //Duration of work in minutes
  event: [string], //Optional type of event instead of work.
                For example vacation or illness
  tags: [Tag[]] //Tags assigned to timelog.
                For example tag of project and task.
}
```

*URL:* /log/mine

*Method:* PUT

*Description:* Update of new TimeLog.

*Data:*

```
Request:
{
  id: [string],
  message: [string],
  date: [string], //YYYY-MM-DD date of work
  start: [string], //HH:mm time of start of work -optional
                //Either start-end or duration is needed
  end: [string], //HH:mm time of end of work -optional
  duration: [number], //Duration of work in minutes -optional
  event: [string], //Optional type of event instead of work.
                For example vacation or illness -optional
  tagids: [number], //List of ids of Tags assigned to timelog.
```

```

    For example tag of project and task.
  }

Response:
  //Newly created TimeLog
  {
    id: [string],
    message: [string],
    teamMember: [string], //Id of user this time is assigned to
    date: [string], //YYYY-MM-DD date of work
    start: [string], //HH:mm time of start of work
    end: [string], //HH:mm time of end of work
    duration: [number], //Duration of work in minutes
    event: [string], //Optional type of event instead of work.
    For example vacation or illness
    tags: [Tag[]], //Tags assigned to timelog.
    For example tag of project and task.
  }

```

*URL:* /log/:id

*Method:* DELETE

*Description:* Deletes time log with specified id.

## 7.2 Client side prototypes

Two client side applications for research purposes were implemented. One as the web application is Angular 2 and second is the Hybrid application written in Angular 2 with Ionic 2 framework.

### 7.2.1 Web application

The web application is written in Angular 2 framework with the usage of Angular Command Line Interface. Application is using Angular Material 2<sup>1</sup> components for *User Interface*. The structure of the application is based on Angular 2 style guide.

The prototype allows users to create and edit their time logs and shows them in simple one day view and whole week view, as shown on figure 7.1. Users can also generate reports of logged time filtered by tags and team members.

Service worker was also implemented as part of the prototype to test possibilities of offline usage. /textitService Worker saves the application shell and all request for tags and time logs into the cache. Service worker also enables time logging without the internet connection by saving POST requests in *IndexedDB* and syncing when the internet connection is available.

<sup>1</sup><https://github.com/angular/material2>

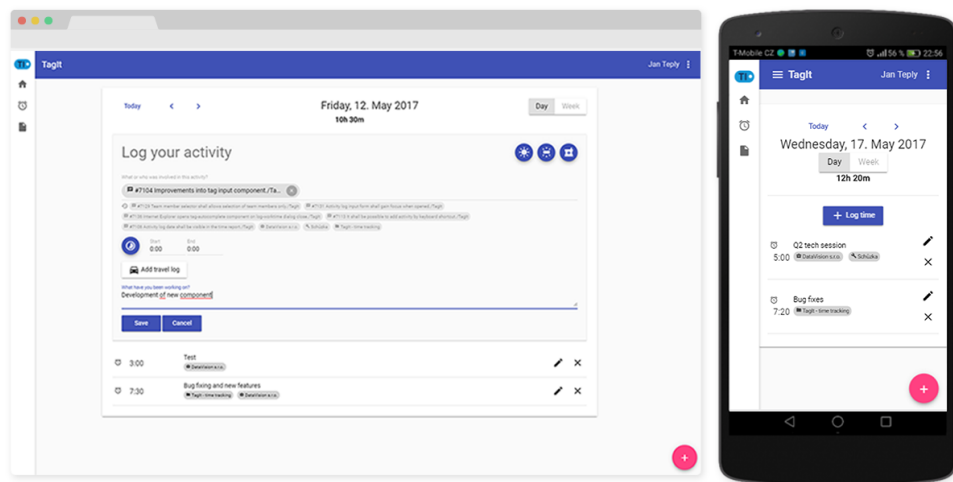


Figure 7.1: Screenshot of web application prototype.

## 7.2.2 Hybrid application

The hybrid application is developed in Ionic 2 framework. Since the application is also written in Angular 2, the application can contain the majority of the same code as the Web application. The mobile application also enables the user to log their work time and shows time logs saved for chosen date, as shown on figure 7.2.

## 7.3 Testing of prototype

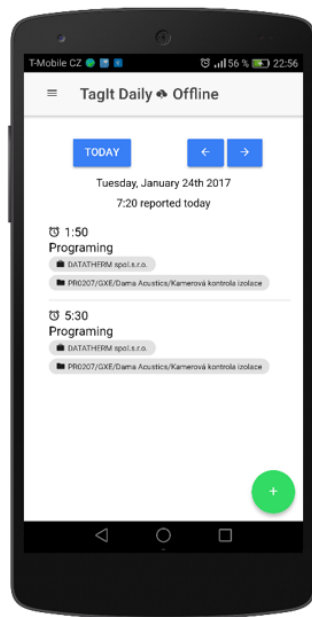
The prototypes were tested during development using manual tests following common scenarios of user actions. For example, logging in with different sets of credentials, logging time or showing reports of work time in past days. I executed These tests against prototype of back-end server. Similar tests were also run by developers of the server.

The prototype also contains set of unit test written in Jasmine<sup>2</sup> with Angular testing utilities[32] which create the test environment for Angular 2 application under test. Jasmine is behavior-driven development framework with clean and obvious syntax. Angular testing utilities are used to specify the environment in which tested component or whole application is running. For example services or other parts of the application can be mocked or replaced with stubs, so tests will not interact with production servers or services can be modified to return predefined data just for testing purposes.

Test written in Jasmine are driven by Karma test runner<sup>3</sup>. Karma is a testing tool that opens defined browser windows and runs the testing code. Karma then shows the results of the tests and is also capable of watching for changes in test code files and runs again with ever change.

<sup>2</sup><https://jasmine.github.io/>

<sup>3</sup><https://karma-runner.github.io/1.0/>



**Figure 7.2:** Screenshot of hybrid mobile application prototype.

To track test code coverage, I used Istanbul<sup>4</sup> to generate reports showing a statistic of every component in the application.

The application has prepared the environment for Protractor<sup>5</sup> tests. The Protractor is end-to-end test framework for Angular. Protractor lets developer write UI test in the way a user would interact with the application. Protractor runs these test in real browser. Protractor uses Jasmine framework for test description.

---

<sup>4</sup><https://istanbul.js.org/>

<sup>5</sup><http://www.protractortest.org/>





## Chapter 8

### Conclusion

The main goal of this thesis was to examine the option of the development of multiplatform client application and propose the architecture for the new version of Plantac time planning and logging application, that is currently called TagIt.

To choose best technologies to fulfill requirements, I compared and studied possibilities of cross-platform mobile development with emphasis on the use of web technologies, CSS, HTML and JavaScript. Based on outcomes of this comparison I suggest usage of *Progressive web applications* principles to make the application that will be accessible from mobile platforms.

After that, I made a comparison study of modern JavaScript frameworks *Angular JS*, *Angular 2* and *React JS*. I have chosen Angular 2 to main front-end framework used in further development of the application. *Angular 2* is based on my analysis best from studied frameworks mainly for its simplicity and coverage of all parts of *Model-View-Controller* architecture.

Since one of the requirements of the application is to be operational in situations without the internet connection, I examined new browser API called *Service Worker*, which is prepared for those situations.

In hand with offline capabilities of the application, several problems may appear. In section 5.2 I listed possible problems and discussed potential solutions.

Based on previous research and requirements of the application I propose architecture described in section 6 to be used in the development of the application. Architecture is also applicable to a range of similar applications with analogous requirements and functionality.

To further examine compared technologies I have developed two prototypes of the client-side application based on proposed architecture and specification of Platac/TagIt. The first prototype is a hybrid mobile application developed with the use of *Ionic 2* framework and *Angular 2* framework. The second prototype is web application developed following *Progressive web applications* principles also written in *JavaScript* and *Angular 2*. Both applications allow the user to sign in and log work time and are capable of limited functionality without the internet connection. These prototypes were tested manually and through unit test written in Jasmine.

## ■ 8.1 Future work

Future development of Plantac and TagIt will be based on proposed architecture and chosen technologies. Web application prototype will be extended to cover more requirements and deployed for usage and beta testing in Datavision company.

Next step process of development is user study, user-centered design and usability testing. This phase will ensure good user experience with the new version of the application. Part of this will also be the definition of the new concept of processes and structures of projects that will make time logging and planning easier.



## Bibliography

- [1] ARCHIBALD, Jake. *The Offline Cookbook*. In: Blog - JakeArchibald.com [online]. 2014 [cit. 2016-12-06]. Available at: <https://jakearchibald.com/2014/offline-cookbook/>
- [2] GAUNT, Matt. *Service Workers: an Introduction* In: Web, Google Developers [online]. 2016 [cit. 2016-12-20]. Available at: <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>
- [3] KORF, Mario and OKSMAN Eugene. *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options* In: Salesforce developers [online]. 2016 [cit. 2016-12-20]. Available at: [https://developer.salesforce.com/page/Native,\\_HTML5,\\_or\\_Hybrid:\\_Understanding\\_Your\\_Mobile\\_Application\\_Development\\_Options](https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options)
- [4] BRISTOWE, John. *What is a Hybrid Mobile App?* In: Telerik Developer Network [online]. 2016 [cit. 2016-12-20]. Available at: <http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/>
- [5] *PhoneGap*. [cit. 2016-12-20]. Available at: <http://phonegap.com>
- [6] *Ionic Framework*. [cit. 2016-12-20]. Available at: <http://ionicframework.com>
- [7] *AngularJS - Superheroic Java Script MVW Framework* . [cit. 2016-10-15]. Available at: <https://angularjs.org>
- [8] *Apache Cordova*. [cit. 2016-12-20]. Available at: <https://cordova.apache.org/>
- [9] *Archytectural overview of Cordova platform in Apache Cordova*. [cit. 2016-12-20]. Available at: <http://cordova.apache.org/docs/en/latest/guide/overview/index.html>
- [10] Diwakar, Sapan. *Titanium vs Phonegap vs Native application development* In: Sapan Diwakar [online]. 2012 [cit. 2016-12-20]. Available at: <http://www.sapandiwakar.in/api-research-study-iphone-and-android-applications/>

- [11] *About NativeScript Open Source Cross Platform Framework* in NativeScript. [cit. 2016-12-20]. Available at: <https://www.nativescript.org/about>
- [12] *ReactNative*. [cit. 2016-12-20]. Available at: <https://facebook.github.io/react-native/>
- [13] *React*. [cit. 2016-12-20]. Available at: <https://facebook.github.io/react/>
- [14] LEPAGE, Pete. *Your First Progressive Web App* in Google Developers. 2017 [cit. 2017-1-2]. Available at: <https://developers.google.com/web/fundamentals/getting-started/codelabs/your-first-pwapp/>
- [15] MARKOV, Danny. *Everything You Should Know About Progressive Web Apps* in Tutorialzine. 2015 [cit. 2016-12-25]. Available at: <http://tutorialzine.com/2016/09/everything-you-should-know-about-progressive-web-apps/>
- [16] OSMANI, Addy and GAUNT, Matt. *Instant Loading Web Apps with an Application Shell Architecture* in Google Developers. 2017 [cit. 2017-1-2]. Available at: <https://developers.google.com/web/updates/2015/11/app-shell>
- [17] CACERES Marcos and CHRISTIANSEN, Kenneth Rohde and LAMOURI, Mounir and KOSTIAINEN, Anssi . *Web App Manifest* in W3C. 2017 [cit. 2017-1-5]. Available at: <https://w3c.github.io/manifest/>
- [18] *Web App Manifest* in Mozilla Developer Network. 2016 [cit. 2017-1-5]. Available at: <https://developer.mozilla.org/en-US/docs/Web/Manifest>
- [19] *Push API* in Mozilla Developer Network. 2016 [cit. 2017-1-5]. Available at: [https://developer.mozilla.org/en-US/docs/Web/API/Push\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Push_API)
- [20] KINLAN, Paul. *Installable Web Apps with the Web App Manifest in Chrome for Android* in Google Developers. 2014 [cit. 2017-1-2]. Available at: [https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp\\*-manifest-in-chrome-38-for-Android](https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp*-manifest-in-chrome-38-for-Android)
- [21] LYNCH, Max. *What are Progressive Web Apps?* in The Official Ionic Blog. 2016 [cit. 2017-1-2]. Available at: <http://blog.ionic.io/what-is-a-progressive-web-app/>
- [22] WASSERMAN, Anthony *Software engineering issues for mobile application development* in Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 397-400. 2010 [cit. 2016-12-20].
- [23] *Native vs Hybrid vs Web: A comparison study* in Cabot technology. [cit. 2017-1-2]. Available at: <https://www.cabotsolutions.com/native-vs-hybrid-vs-web-comparison-study/>

- [24] VISWANATHAN, Priya. *Native Apps vs. Web Apps – What is the Better Choice?* in Lifewire. 2016 [cit. 2017-1-2]. Available at: <https://www.lifewire.com/native-apps-vs-web-apps-2373133>
- [25] ASAY, Matt. *Can We Please Stop Fighting The Native vs. Web App Wars?* in Readwrite. 2015 [cit. 2017-1-2]. Available at: <http://readwrite.com/2015/02/27/native-vs-web-apps-ceasefire/>
- [26] DASCALESCU, Dan. *Why “Progressive Web Apps vs. native” is the wrong question to ask?* in Medium. 2016 [cit. 2017-1-2]. Available at: [https://medium.com/dev-channel/why-progressive-web-apps-vs-native-is-the-wrong-question\\*/-to-ask-fb8555addcbb#.9q51w725t](https://medium.com/dev-channel/why-progressive-web-apps-vs-native-is-the-wrong-question*/-to-ask-fb8555addcbb#.9q51w725t)
- [27] WILLIS, Justin. *Service Workers: Revolution Against the Network!* in The Official Ionic Blog. 2016 [cit. 2017-1-2]. Available at: <http://blog.ionic.io/service-workers-revolution-against-the-network/>
- [28] RUSSEL, Alex. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul* in Infrequently Noted. 2015 [cit. 2017-1-2]. Available at: <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>
- [29] RUSSEL, Alex. *What, Exactly, Makes Something A Progressive Web App?* in Infrequently Noted. 2015 [cit. 2017-1-2]. Available at: <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/>
- [30] OSMANI, Andy. *Getting started with Progressive Web Apps* in AndyOsmani.com. 2015 [cit. 2017-1-2]. Available at: <https://addyosmani.com/blog/getting-started-with-progressive-web-apps/>
- [31] PANDA, Preetish *What’s New in AngularJS 2.0* in Sitepoint. 2015 [cit. 2016-10-2]. Available at: <https://www.sitepoint.com/whats-new-in-angularjs-2>
- [32] *Testing - ts - Guide* in Angular.io. [cit. 2017-6-2]. Available at: <https://angular.io/docs/ts/latest/testing/>
- [33] *Structural Directives - ts - Guide* in Angular.io. [cit. 2016-10-2]. Available at: <https://angular.io/docs/ts/latest/guide/structural-directives.html>
- [34] *Router & Navigation - ts - Guide* in Angular.io. [cit. 2016-10-2]. Available at: <https://angular.io/docs/ts/latest/guide/router.html>
- [35] *Angular Material* in Angular.io. [cit. 2016-10-2]. Available at: <https://material.angular.io/>
- [36] AVRAM, Abel. *The Next Major Version of Angular Will Be 4, Not 3* in InfoQ. 2016 [cit. 2016-10-2]. Available at: <https://www.infoq.com/news/2016/12/angular-4>

- [37] *TypeScript - JavaScript that scales*. [cit. 2016-10-2]. Available at: <https://www.typescriptlang.org/>
- [38] CZAPLICKI, Evah *Blazing Fast HTML* in Elm-lang. 2016 [cit. 2016-10-2]. Available at: <http://elm-lang.org/blog/blazing-fast-html-round-two>
- [39] *Pokrytí železničních tratí signálem mobilních sítí*. [cit. 2016-12-2]. Available at: <http://www.ctu.cz/mereni-pokryti-zeleznice>
- [40] MAREŠ *Pražané se snad konečně dočkají plného pokrytí metra mobilním signálem* in Metro Praha. 2016 [cit. 2016-12-2]. Available at: [http://metropraha.eu/prazane-se-snad-konecne-dockaji-plneho-pokryti-metra-\\*/mobilnim-signalem/](http://metropraha.eu/prazane-se-snad-konecne-dockaji-plneho-pokryti-metra-*/mobilnim-signalem/)
- [41] RUSSELL, Alex and SONG, Jungkee and ARCHIBALD, Jake and KRUISSELBRINK, Marijn. *Service Workers Nightly* in M3C. 2015 [cit. 2016-12-2]. Available at: <https://www.w3.org/TR/service-workers/>
- [42] *Electron*. [cit. 2016-10-2]. Available at: <http://electron.atom.io/>
- [43] FEYERKE, Alex. *Designing Offline-First Web Apps*. 2013 [cit. 2016-10-2]. Available at: <http://alistapart.com/article/offline-first>
- [44] ARCHIBALD, Jake. *Introducing Background Sync* on Google Developers. 2015 [cit. 2016-10-2]. Available at: <https://developers.google.com/web/updates/2015/12/background-sync>
- [45] ARCHIBALD, Jake. *Adding Push Notifications to a Web App* on Google Developers. 2016 [cit. 2016-12-22]. Available at: <https://developers.google.com/web/fundamentals/getting-started/codelabs/push-notifications/>
- [46] ABBOTT, Tom. *Where to Store your JWTs - Cookies vs HTML5 Web Storage* on Stormpath. 2016 [cit. 2016-10-2]. Available at: <https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage>
- [47] *Offline Authentication* on IBM Mobile First Developer Center. 2016 [cit. 2016-10-2]. Available at: <https://mobilefirstplatform.ibmcloud.com/tutorials/en/foundation/6.3/authentication-security/offline-authentication/>
- [48] *HTML5 Security Cheat Sheet* on Open Web Application Security Project. 2015 [cit. 2016-10-2]. Available at: [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)
- [49] *Managing HTML5 Offline Storage* on Google Chrome. [cit. 2016-10-21]. Available at: [https://developer.chrome.com/apps/offline\\_storage](https://developer.chrome.com/apps/offline_storage)

- [50] *Browser storage limits and eviction criteria* on Mozilla Developer Network. [cit. 2016-10-21]. Available at: [https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\\_API/Browser\\_storage\\_limits\\_and\\_eviction\\_criteria](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria)
- [51] WILSON, Chris. *Persistent Storage* on Google Developers. 2016 [cit. 2016-10-21]. Available at: <https://developers.google.com/web/updates/2016/06/persistent-storage>
- [52] OSMANI, Addy. *Offline Storage for Progressive Web Apps* on Medium. 2016 [cit. 2016-10-21]. Available at: <https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c#.ba4bwy4j6>
- [53] CIMPANU, Catalin. *Recent Benchmark Shows the Speed of AngularJS 2* on Softpedia. 2016 [cit. 2016-10-21]. Available at: <http://webscripts.softpedia.com/blog/recent-benchmark-shows-the-speed-of-angularjs-2-499638.shtml>
- [54] PEYROTT, Sebastián. *More Benchmarks: Virtual DOM vs Angular 1 and 2 vs Others* on Auth0 Blog. 2016 [cit. 2016-10-21]. Available at: [https://auth0.com/blog/more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js\\*/-vs-the-rest/](https://auth0.com/blog/more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js*/-vs-the-rest/)
- [55] OSMANI, Addy, COHEN, Marc. *Offline Storage for Progressive Web Apps* on Web Fundamentals. 2017 [cit. 2017-05-15]. Available at: <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa>


1

---

<sup>1</sup>Due to too long URL sequences "\*" was used to brake lines in some urls.







## Appendix A

### Abbreviations

**SPA** - Single Page Application

**UX** - User eXperience

**UI** - User Interface

**API** - Application Programming Interface

**DOM** - Document Object Model

**API** - Application Programming Interface

**JSX** - Javascript Syntax eXtension

**MVC** - Model View Controller

**IDE** - Integrated Development Environment

**NPM** - Node Package Manager

**URL** - Uniform Resource Locator

**HTTP** - Hypertext Transfer Protocol

**HTML** - HyperText Markup Language

**CSS** - Cascading Style Sheets

**REST** - Representational State Transfer

**SW** - Service Worker





## Appendix B

### Contents of CD

- **TagItIonic** - Source files of hybrid application prototype.
- **TagItWeb** – Source files of web application prototype.
- **latex-files.zip** – Thesis in latex format.
- **teplyja1-master-thesis.pdf** – Thesis in PDF format.