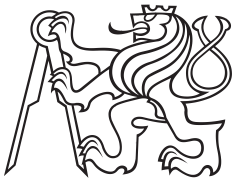
Master Thesis



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
December 2016

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 technologii 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 odtestujte.

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

L.S.

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 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, December 21, 2016

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

V Praze, 21. prosince 2016

.....
Bc. Jan Teplý

Abstract

Plantac is proprietary web application for project time and spendings planing. Curently writen on Java EE framework with ZK framework for vizualization. Goal of this thesis is to explore possibility of creation of alternative multi-platform user interface, that enables choosen function 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 as vyzualizací ve 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

Figures

Tables

Chapter 1

Introduction

Mobile devices gain huge popularity in past years and it became essential for web applications and services to provide access from these devices. Plantac, currently web based time planing and tracking application, is no exception.

Access from mobile devices may be easier and quicker and also possible in environments where notebook or desktops are not usable.

1.1 Mobile application frameworks comparison

This thesis focuses on comparison of frameworks and approaches in development for mobile devices, especially with usage of web technologies, HTML, CSS and Java Script. Hybrid mobile applications and progressive web applications allows developer to use those technologies in mobile development.

1.2 JavaScript frameworks comparison

JavaScript frameworks are backbone of moder one page web applications. Framework enables to developer with scalable, reusable and maintainable code. Thesis compares modern Java Script framework that will be used in development of such applications. Chosen frameworks are Angular JS, Angular2 and React.

1.3 Offline capabilities

Modern, user friendly and engaging application can gain huge benefit from running without internet connection. In most basic way application can cache its static resources in browser and load faster, but going offline with well written *Service Worker* application can also save data and request for future use and backgroud synchronization.

1.4 Architecture

In this thesis I propose general architecture for such applications and namely for future development of Plantac.

Chapter 2

Plantac

Plantac is cloud based web application used for project planing and management. Application is also used by all worker in company to log their work time.

Plantac is developed with Java Enterprise Edition backend end ZK Framework frontend. Application is specifically developed to be deployed on GlassFish Application Server with connection to PostgreSQL database server.

Project is now in phase where all off the application will be rewriten with modern and more suitable technologies and approaches. Mainly using REST API for communication and Java Script Framework on the frontend of the applications. Application should also provide new UI design and functions based on feedback from previous version of application. New application based on Plactac is developed under work name TagIt.

2.1 Specification of Plantac

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

Since developers are more familiar with web technologies than native application development, research focuses on possibility of using HTML, CSS and Java Script to create mobile application. Same Java Script framework that will be chosen in this research will be also 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.

Chapter 3

Mobile application

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 combines web applications with native application. Application itself is build using HTML, Java Script 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 great opportunity for team that are focused on web apps development, but also wants to make a mobile application for their product.

Hybrid mobile applications are also crossplatform. *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 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 environment for applications to run in, for UI component and Java Script framework developer have to include those or use another distribution of Cordova.

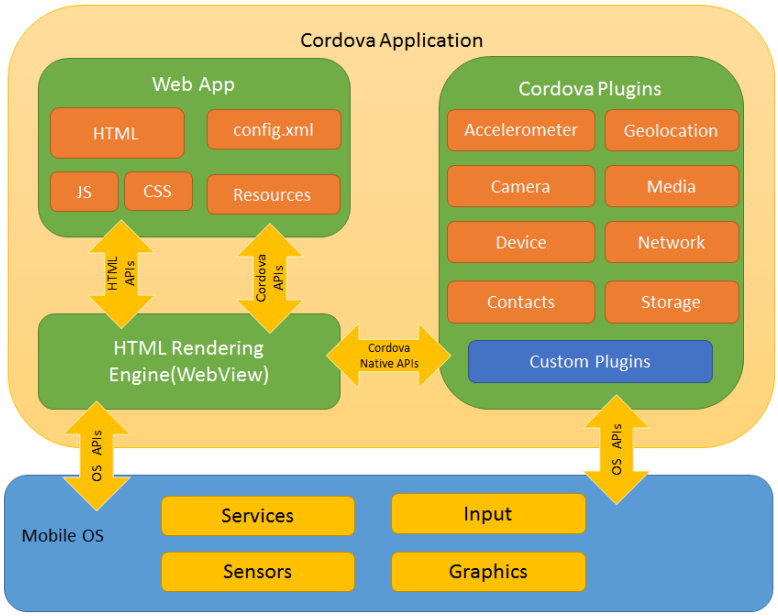


Figure 3.1: Apache Cordova architecture. [9]

Adobe PhoneGap

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

Ionic

Ionic framework also offers tool 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 iOS application can not be compiled on Windows, and analytics.

Main feature of Ionic is integration of Angular JS and Angular 2, in Ionic 2. Ionic also includes native looking UI components, which will look different on each platform. If developer desides for Angular 2, as I did in following chapters of this work, Ionic might be the choice for quick development of native 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 native layer and web application. Enabling application to use resources of platform,

like camera, notifications, contacts, file system and others.

Plugins, that can be used with Cordova framewrok 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 NativeScript, React Native

Another aproach in making cross-platform mobile application using Java Script and web technologies is NativeScript and React Native. These frameworks uses similar approach in creation of mobile applications. Application logic is written with Java Script, styles are using CSS but instead of HTML, these frameworks uses specific XML that is than compiled to platform specific XML, which is rendered when application is in use. This means that there is now web view involved and it might solve some inconsistency and performance issues of applications rendered in webview. [11] [12]

3.3 Progressive web applications

Progressive web application are new approach to 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 10 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 progressive web application are further explained in following subsections.

Safe

Since progressive web apps may, and probably will deal with sensitive data via native APIs and service worker, its advised to serve progressive web applications through HTTP connection.

Progressive

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

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

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

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

And if it is still unsupported user will be prompted about 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

Application should have responsive design and work on every screen size. This is very critical since progressive web application heavily aim to mobile devices. Responsive design can be achieved using *fluid grid* concepts, *flexible images* and *CSS3 media queries*. [15]

Connectivity independent

Service workers allows your app to work even without internet connection.

A *Service Worker* is like a client-side proxy, written in Java Script and puts developer in control of the cache and how to respond to resource requests. By pre-caching key resources application can eliminate the 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 previous interaction.

In order to use *Service Worker*, developer have to create Java Script file that will be registered as *Service Worker* and listen to at least these two events and implement basic logic. Service worker are further described in following chapter.

With registered service worker all files of this application will be saved in browser cache and available for the user, when he comes back. Responses with data and variables of application should be saved in *localStorage* or *IndexDB* 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 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 its necessary to cache whole shell on the device. This warrants that even if there is no internet connection and no cached content application will load at least familiar UI and user is not presented with empty screen or default connection failure message. [16]

Content is shown within the shell. Content itself may be also cached but it heavily depend on the type of application.

Fresh

Application shell and cached content will always load from the local storage, after its cached. However, browser checks for changes in *Service Worker* file and for the next page load new version of service worker is installed. This

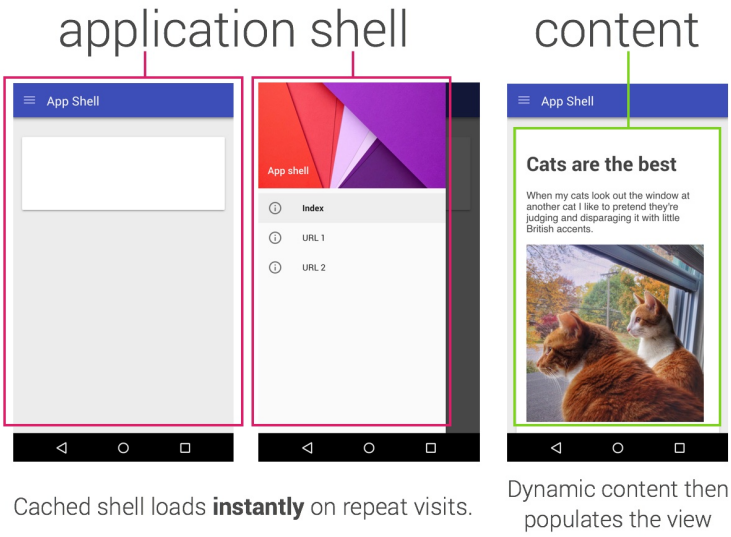


Figure 3.2: Identification of shell and conent in app-shell architecture. [16]

new service worker can cache again all resources and delete old cached data. [15]

Discoverable

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

Web Manifest is a JSON file linked in header of HTML document of 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*, *standalone*, *minimal-iu* 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. This messages can be shown as notifications to the user and notify him about updates or scheduled events. This 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 menu. Unfortunately its not much user-friendly. [21]

However, Chrome makes it possible to prompt user with a pop-up suggesting him to save application to home screen. This pop-us in not accessible by developer, but it will be show automatically if application fulfills these three requirements.

Web Manifest must be present and valid, Application must have valid and installed *Service Worker* and Application must be served over *HTTPS*. [20]

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

Linkable

Since progressive web application are accessible for everyone with a browser they can be shared simply 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]

Conclusion

Progressive web applications are not new technology or framework, but a new approach of making web applications more like native.[21] Google makes a great effort in providing tutorials and support for this approach and with more browser and platform support it might be future of mobile web. Applications can also benefit from above mentioned technologies even on 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 compassion of pros and cons of possible aproaches of development for mobile devices.

3.4.1 Native apps

Native apps have very good access to all features and APIs provided by operating system and ecosystem on mobile platform. [23] 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 monetized [24]

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

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

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

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

3.4.3 Web apps

Since web apps on mobile are just UI and performance optimized web pages, they can be opened in browsers on every platform. However, they works only with internet connection. Main advantage is that web app for mobile are fast to create.

Possibilities of web apps are quite limited by mobile browser APIs and performance.

3.4.4 Progressive web apps

Web application whiten in offline-first way using service worker and manifest to enable placement to home screen of android devices. [29]

Progressive web application can be installed very quickly, just by accessing website in browser and clicking a button. This simply makes a bookmark on the home screen and next time is the application open via bookmark and in 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. This might be just placeholders or old data but its much closer to the native or hybrid application feel of an app. Also user does not wait looking at white screen until the page is loaded from the internet. Progressive apps can even have a 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/TSL secured through HTTPS protocol. [15]

Compared to native and hybrid applications, progressive web apps have limited accses 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 file system and can store data only in browser storage.[25]

In current time progressive web application works with its whole 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 build with progressive enhancement, therefore applications will be still useful in browsers that does not provide needed *APIs*[28]

3.5 Conclusion

In my opinion, Plantac or future TagIt would benefit the most from usage of progressive web applications approach. Since Plantac is primarily desktop browser application it will benefit from most of progressive web applications features. Development of good responsive design would easily bring all of Plantac 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 desktop browser.

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

Chapter 4

Java Script frontend frameworks

This chapter will cover comparison of most used modern Java Script 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 Java Script Framework for front-end of web applications. AngularJS is created for making single-page web applications. Development of framework was started in 2009 by Google. Framework is accessible on web for free under MIT licence. Framework is developed in manner of open source community project, and it's now slowed down due to reallocation of most of developers to new project of rewrite, which is Angular 2 cover in next subsection.[7]

Framework covers all part of *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 application. Data to *View* is delivered and controller by the *Controller*, which is connected to the template by *ng-controller directive*. *Controller* stores data and functions, which are used by 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 in new It's suggested keeping fewer than 2000 watchers on any page.[1]

Example of AngularJS code[53]:

```
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) {
```

[1] 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 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 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 creation of custom *directives*, that can be used as html components, attributes, classes or comments. Developer is also able to limit this usage for example just to be used as 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 all of the application without need of copying or rewriting the same code. This makes 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 field based on if user made changes to them or not. *View* can react to these different states by i.e. changing colour of input.

4.1.4 Services

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

moment, when there is component depending in this *Service*. Angular also consists of number of built in services, such as *HTTP service* used for HTTP cals. *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 developer to create custom Filters.

4.2 Angular 2

Angular 2 is new rewrite of the previous Angular 1. Development started at fall of year 2014 and version 2.0.0 was released in September 2016. Angular 2 is not backward 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 AngulaJS.

4.2.1 Mobile first

Angular 2.0 focuses on mobile development of applications and mobile web pages. Developers focus on focus on performance problems of mobile applications first. In most cases, 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 limited number of necessary *modules*, that way developers are enabled to optimized size of the application and limit overhead of the framework in case of simple applications without need of certain modules. Angular 2 uses module system of ES2015 and is compatible with modern packaging tools like Webpack or SystemJS.[31]

Those modules are for example *http* or *router* module, may not be necessary for simple application. Developers can also write custom modules for Angular. Dependency injection have been also 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. Buil-in *directives* of this type are *ngIf*, renders element if conditions is fulfilled, *ngSwitch*, is template equivalent of switch known from programing languages, and *ngFor*, is used to iterate through collections of data. [32]

4.2.4 Router

One of Angular modules is *Router*, that is used for work with URL and navigation through application. New router provides interesting functions. [33]

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

Screen Activator is used to gain control over process of navigation. For example, if user have some unfinished work on the page, application can react to it and prompt user about it or save work automatically. 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 by used with pure Java Script or i.e. *Dart* programing language. [36]

Example of Angular2 code in TypeScript[53]:

```
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 also provides support for testing with *Karma* and *Protractor*.[31]

4.2.7 Animations

Support for complex and high-performance animations of UI.[31]

4.2.8 Angular Material

Material Design components written in Angular 2 are also in development by Angular 2 team. Material desing is widely used concept of design presented by Google for Android, that then expanded to all kind of desing applications.[34]

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 possibility of braking changes, but it will not be ground up rewrite as version 2.0 was. [35]

4.3 React

React is Java Script library dedicated to creation of View in MVC architecture. React is developed by Facebook and community of developers under BSD licence 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 value of attributes in html tag of components. Same way can be used to pass callback functions to components. Function are the only way to affect parent component, because 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

Important part of React is VirtualDOM. VirtualDOM is constructs in memory, where state of the View is rendered. This VirtualDOM is compared to DOM displayed in the browser and minimal set of changes is needed to make to real DOM to make it equal to VirtualDOM. 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 extension of syntax for Java Script, that enables writing of HTML directly to Java Script. This html code is translated to calls of function from React library. There is also possibility to develop in React without JSX, but the syntax of function is not much readable in development, therefore it is not recommended. [13]

Example of React code in JSX[53]:

```
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 speed of frameworks.

First graph, see figure 4.1, shows time needed to perform action over list of 99 todos in simple todo applications. Angular 2 performs much better than react in this case. However, optimalization of react have bigger impact than optimization of Angular 2. AngularJS stays in the middle in both cases.

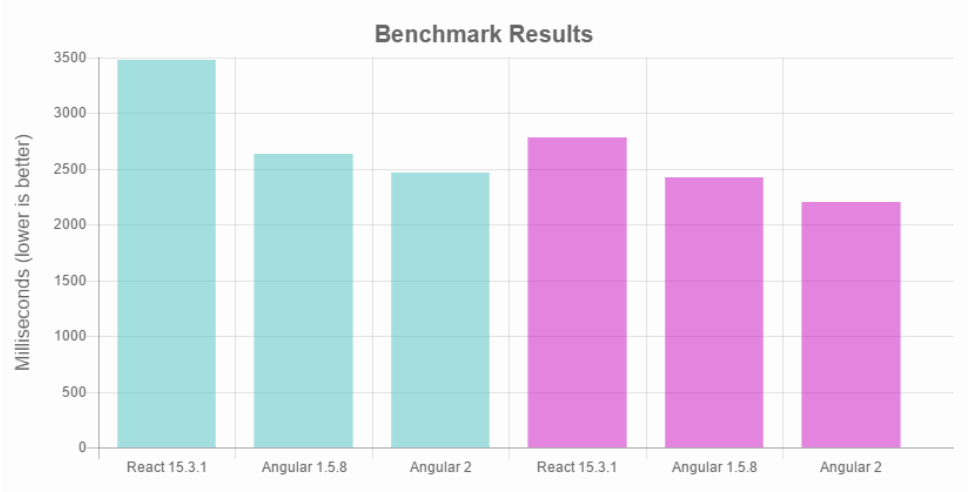


Figure 4.1: Graph showing time in millisecond needed to add, remove and update list of todos in each framework.[37]

Second graphs, see figure 4.2, shows comparison on large amount of data. Methodology, prepared by Stefan Krause[2] compares times of creation of 1000 rows of 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 element is faster in Angular2.

This graph can also server as comparison with other JavaSript frameworks.

4.5 Conclusion

For development of mobile and also desktop web application I decided to choose framework Angular 2.0. Angular 2.0 is 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 organizes code into components and services, type system of TypeScript, which makes code more readable and safe and in-build function and service.

[2] `http://www.stefankrause.net/wp/?p=218`

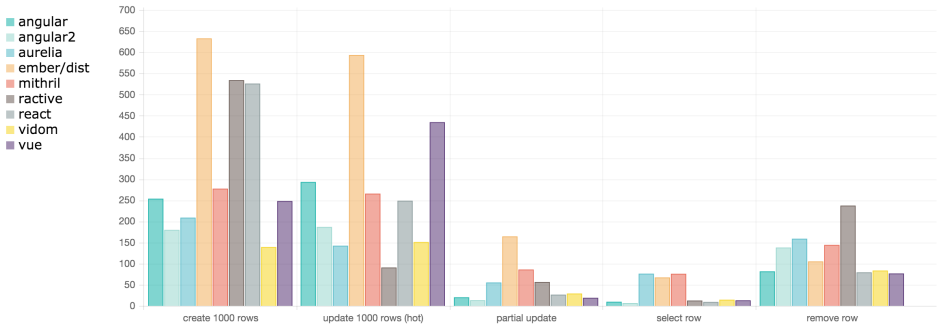


Figure 4.2: Comparison of speed of framework while manipulating big table. Performed by methodology of Stefan Krause [52]

In time of finishing this thesis Angular 2 is already released and used by many developers. In the beginning of work I had to consider possibility of developing in Angular 2 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 is very interesting and innovative, is in my opinion too specialize on simplifying rendering and description of *View* in MVC. In other parts of system developer have 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 complexity 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 good enough 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 % [38] and is proposed to be 100 % first in 2021. Mobile signal in Prague metro 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. [39]

Internet connection might not be also unavailable due to security politics of companies. For example if workers of company, that is using Plantac, where hired to do their work in compound of company, that 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 inspection in factories with security politics forbidding internet access.

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

Web applications need internet connection from the start. Some web application may be capable of working without internet connection, but after user closes the window application never loads back without internet access. To address this problem Service Worker was introduced in 2014[40].

Another possible solution for offline desktop application made with web technologies is Electron[41]. Electron uses mixture of Chromium and Node.js to wrap HTML, CSS and Java Script of the application and run it 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[1] used to intercept communication of website or application with server and resources. Via service worker developer can control caching of resources on very precise level and with high control over the process.

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

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

Service worker is progressive enhancement of the application or webpage. 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 diagam see 5.1 Installation of the service worker starts by registration of if in Java Script.

```
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 enviroment and it is 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. Installation will be attempted again after refresh. If installation succeeded everything is prepared in cache.

[1] https://developer.mozilla.org/en-US/docs/Web/API/Worker

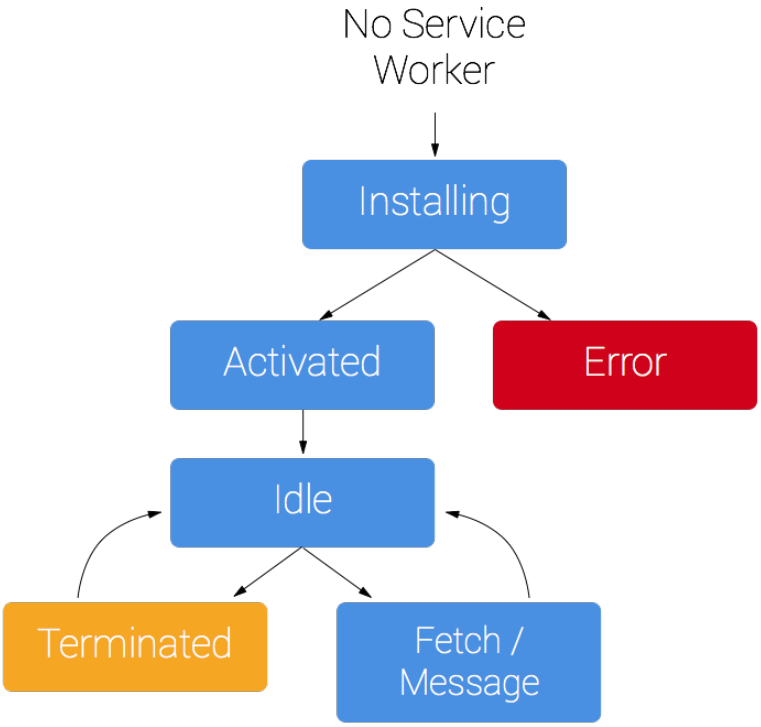


Figure 5.1: Simplified diagram showing states of Service worker life cycle.[2]

```
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 Actiovation. If there is no other window opened its time to clean previous service workers and also cache. 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 is 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 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 environment.

- Activate

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

- Message

  Event fired when service worker get message from the application.

Functional events

- Fetch

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

- Sync

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

- Push

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

5.1.3 Fetch strategies

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

Cache only

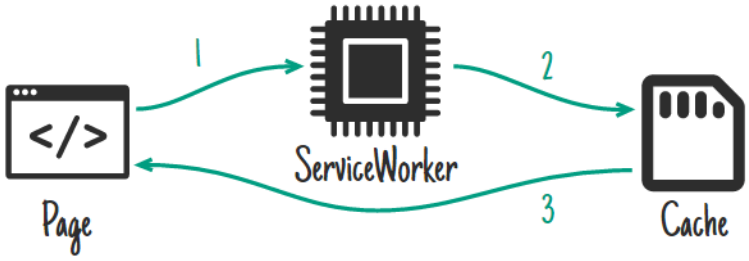


Figure 5.2: Diagram showing steps of Cache only strategy.[1]

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

Network only

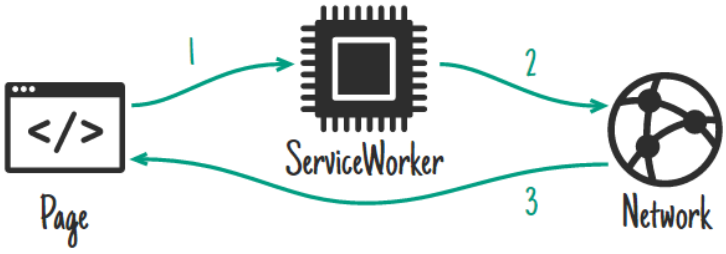


Figure 5.3: Diagram showing steps of Network only strategy.[1]

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

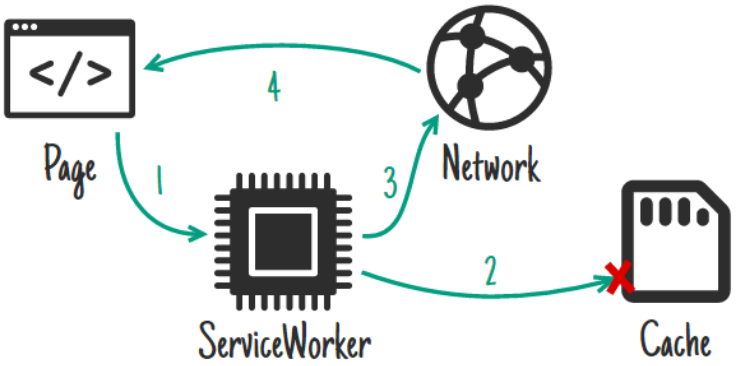


Figure 5.4: Diagram showing steps of Cache, falling back to network strategy.[1]

Cache, falling back to network

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

This strategy handles majority of request in 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

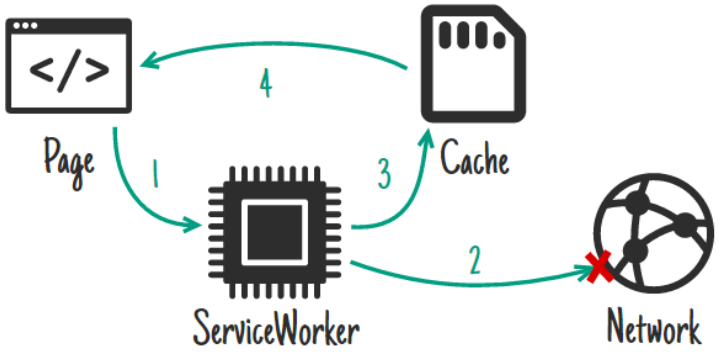


Figure 5.5: Diagram showing steps of Network falling back to cache strategy.[1]

In this strategy request is send to the network. If this request fails, service worker return cached response, if there is any. For image representation see figure 5.5. This strategy gives on-line user all the new and up-to-date data and offline user gets 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 user have slow or not reliable internet connection. In that case waiting for response from the network can take ages and it very unpleasant user experience. See Cache then network for better strategy.[1]

Cache then network

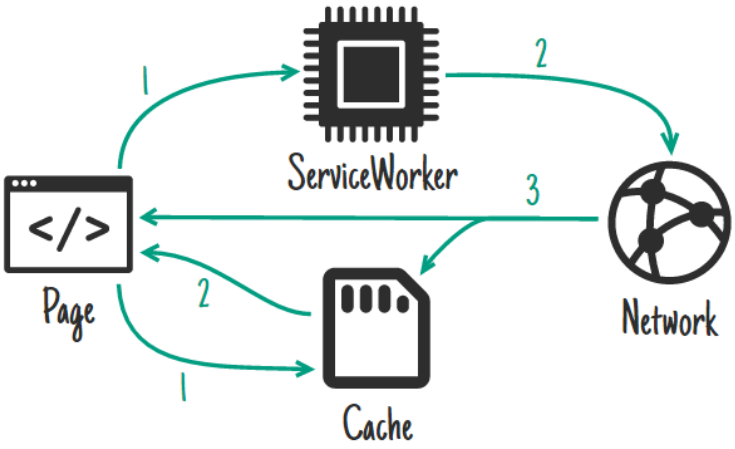


Figure 5.6: Diagram showing steps of Cache then Network strategy.[1]

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. Web application have to create two requests.

One for fresh data from the network and one straight to the cache for older cached version of data, see figure 5.6. Then application will show user response from cache, which, if exists, probably responds first. Then, when application gets response from the network, view is refreshed with new data or 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

In this strategy service worker tries to get 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 its faster to download something than find it on the hard drive.[1]

Generic fallback

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

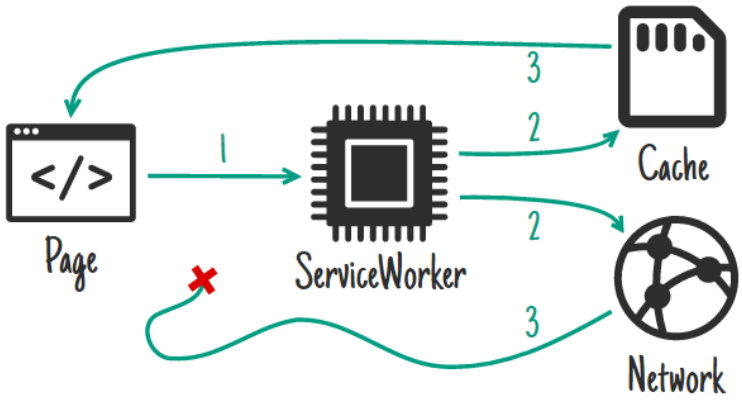


Figure 5.7: Diagram showing steps of Cache and Network race strategy.[1]

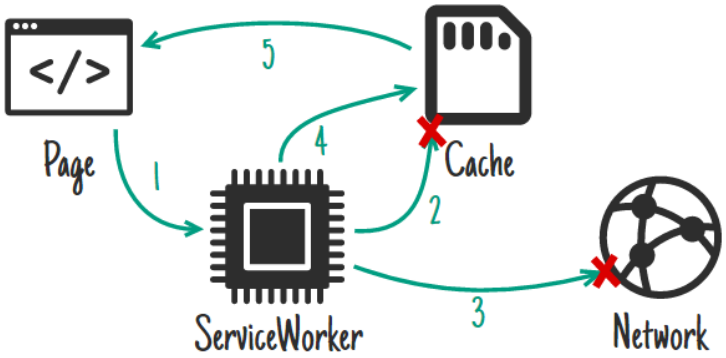


Figure 5.8: Diagram showing steps of Generic fallback strategy.[1]

be send, service worker can save the request to indexedDB and sent 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 synchronization is very useful feature of Service Worker. Sync events are fired only when there is available internet connection, and if actions fails, it retries when there is an connection or after some time determined by exponential back-off. Even if browser tab with the page is closed. Its useful for non-urgent updates, for example, social media timeline or news feed. [43]

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.[44]

5.1.5 Sw-precache and sw-toolbox

Sw-precache[2] is module for generating service worker in build script, for example written with *gulp*. As part of configuration sw-precache gets list of urls to static resources of application, this list can also be in form of patterns that are mached to files using *glob*[3], 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 browser. Generated service worker also includes logic that serves cached resources.

Sw-toolbox[4] is 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 runtimeCaching option in configuration of sw-precache. In runtimeCaching option developer can specify urlPathern, handler, that is a name of strategy that will be used, and optionaly 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.

5.1.6 Conclusion

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

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

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

5.2 Problems with offline

Making application run offline may bring problems with security and storage presistance.[42]

5.2.1 User authentication

Standard authentication of user credentials can not happen without internet connection. Credential have to be checked against database entries. However,

[2] https://github.com/GoogleChrome/sw-precache
[3] https://github.com/isaacs/node-glob
[4] https://github.com/GoogleChrome/sw-toolbox

we might create local copy of user account stored in LocalStorage and check credentials locally if internet connection is not available.[46] This solution might also create weak spot since possible attacker can see how is the password hash generated and password hash is stored locally [45].

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 unsynchronised data might be lost. Solution is prompt the user about it and let him choose if he really wants to logout and delete unsynchronised data. Application could also store data in localStorage or IndexedDB even if user is not loged in but that data might not be secure and still could be lost if user never sing in again.

5.2.3 Security of locally stored data

By default, data stored in browser localStorage or indexedDB are not secured. User or admin with privilages to the machine can access localStorage or indexedDB and read, write or modify data stored there. Also any all Java Script served from same origin can access data in storage.[47]

5.2.4 Conflicts

If user work offline in collaboration with other user on same data, conflicts may occur and application have to address them in some way. Possible solution is for example informing user about changes, that have been done to data since he 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, i. e. in Plantac creation of same task with 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 task in the morning and in the evening but did not specified the start and finish time while submitting the worktime. However, there can be situation where user submits time through mobile application in offline mode, and later on the desktop he may summit the same time again assuming the submission from application failed. But actually mobile application is not yet synced due to some internet connection problems and it submits the worktime to server, when mobile gains internet connection. Unfortunately this situation is hard to detect programatically.

5.2.5 Too big cache

Offline applications have to store all the data in cache, localStorage or indexedDB. There stores have limited size. For example localStorage in Chrome is limited to 5 MB[48]. Developer must have this limitations in mind and cache only data needed to run the application and clean old cache entries.

5.2.6 Cleared cache

By default data saved in cache, localStorage or IndexedDB can be cleared by browser or user. In that case all unsynchronised data will lost. Application can is not informed about this action by browser. [51]

However, storage can be set to two types:

- Persistent

  This is data that is intended to be kept around for a long time. This 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.)[49]

  In Firefox developer can chose to save data to storage ass persistent. Chrome, since 55, have its own policy and sets storage to persistent if any of following conditions is true.[50]

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

- Temporary or best-effort

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

Chapter 6

Architecture

In this section I will cover general suggestions for making application using web technologies that can be used on mobile and in offline mode.

6.1 Server side

For server side developer can choose from many frameworks. Main feature of server, in this context, should be RESTful *API* for client. Resources, provided in RESTful way, can be easily cached and stored for offline use.

6.2 Client side

6.2.1 Java Script

Client side of application should be developed using modern Java Script Framework for one page applications, for example, as is suggested in previous sections, Angular 2.

6.2.2 Progressive Web Application

I strongly suggest usage of Progressive web application approach. Progressive web application make 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 is essential. Progressive web application also includes App-shell architecture of client application. This means that client application should be divided to *shell*, which is static part of application, and dynamic content accessed from the server side. This way shell can be cached in Service Worker and loaded even if application is offline.

6.2.3 User interface

Application should work well on mobile devices, therefore it should provide special UI optimized for mobile device or fully responsive UI. Great way to

achieve this is usage of Google Material Design or Bootstrap components[1].

[1] http://getbootstrap.com/

Chapter 7

Implementation of prototype

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

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

7.1 Server side

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

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

7.1.1 REST API

Client side of application communicates with server through REST API with following resources.

User

URL: /user/auth

Method: POST

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

Data:

```
Request:
    {
    tenant : [string], //Name of tetant, 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,...) will have its unique tag.

URL: /tag/

Method: GET

Decsription: 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

Decsription: 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
Decsription: This resource return time assigned to authenticated user. Timelog can be filted 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
Decsription: 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
Decsription: 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
Decsription: Deletes time log with specified id.

7.2 Client Side

Two client side application was developed for research purposes One is written as web application is Angular 2 and second is Hybrid application written in Angular 2 with Ionic 2 framework.

7.2.1 Web application

Web application is written in Angular 2 with Angular2-seed[1] as base project. Application is also using Angular Material 2[2] component for *User Interface*.

Service worker was also implemented as part of prototype to test possibilities of offline usage.

7.2.2 Hybrid application

Hybrid application is developed in Ionic 2 framework. Since application is also written in Angular2, application can contain most of the same code as Web application.

[1] https://github.com/mgechev/angular-seed
[2] https://github.com/angular/material2

Chapter 8

Conclusion

Aim of this work was to compare possibilities of frameworks for cross platform mobile development with use of web technologies, comparison of JavaScript frameworks and investigation of possibilities and problems of offline applications. These comparison are listed above. During writing of this comparison I learned a lot about these technologies and I plan to work with them in the future. From comparison I have chosen Progressive web application and Angular 2 as best candidate for future development of Plantac(TagIt).

This work also contain proposal of general architecture that should be used in development of offline-first mobile and web applications. I have also implemented simple prototypes of Plantac(TagIt) mobile a web applications to prove the architecture usable.

I will continue work on Plantac(Tagit) in the future with use of there acquired knowledge from this thesis.

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`

[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.`

[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 Plaform 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`

[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`

[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#.9q5lw725t`

[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] *Structural Directives - ts - Guide* in Angular.io. [cit. 2016-10-2]. Available at: `https://angular.io/docs/ts/latest/guide/structural-directives.html`

[33] *Router & Navigation - ts - Guide* in Angular.io. [cit. 2016-10-2]. Available at: `https://angular.io/docs/ts/latest/guide/router.html`

[34] *Angular Material* in Angular.io. [cit. 2016-10-2]. Available at: `https://material.angular.io/`

[35] 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`

[36] *TypeScript - JavaScript that scales.*[cit. 2016-10-2]. Available at: https://www.typescriptlang.org/

[37] 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

[38] *Pokrytí železničních tratí signálem mobilních sítí.*[cit. 2016-12-2]. Available at: http://www.ctu.cz/mereni-pokryti-zeleznice

[39] 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/

[40] 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/

[41] *Electron.*[cit. 2016-10-2]. Available at: http://electron.atom.io/

[42] FEYERKE, Alex. *Designing Offline-First Web Apps.* 2013 [cit. 2016-10-2]. Available at: http://alistapart.com/article/offline-first

[43] 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

[44] 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/

[45] 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

[46] *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/

[47] *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

[48] *Managing HTML5 Offline Storage* on Google Chrome. [cit. 2016-10-21]. Available at: https://developer.chrome.com/apps/offline_storage

[49] *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`

[50] WILSON, Chris. *Persistent Storage* on Google Developers. 2016 [cit. 2016-10-21]. Available at: `https://developers.google.com/web/updates/2016/06/persistent-storage`

[51] 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`

[52] 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`

[53] PEYROTT, Sebastián. *More Benchmarks: Virtual DOM vs Angular 1 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/`[1]

[1] Due to too long url sequences "*/" was used to brake lines in some urls.