

**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačních technologií**



**Diplomová práce**

**Front-end design webových aplikací**

**Bc. Tomáš Novák**

**© 2020 ČZU v Praze**

**!!!**

**Místo tohoto textu vložte PŘEDNÍ stranu zadání práce,  
které si můžete vyexportovat do PDF v IS.CZU.cz,  
pokud již máte schválené zadání i děkanem PEF.**

**!!!**

**!!!**

**Místo tohoto textu vložte ZADNÍ stranu zadání práce,  
které si můžete vyexportovat do PDF v IS.CZU.cz,  
pokud již máte schválené zadání i děkanem PEF.**

**V případě, že Vaše zadání je na více než 2 strany, vložte i  
další strany.**

**!!!**

### **Čestné prohlášení**

Prohlašuji, že svou diplomovou práci "Front-end design webových aplikací" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.



V Praze dne 30.11.2020

---

### **Poděkování**

Rád bych touto cestou poděkoval Ing. Petru Bendovi, Ph.D. za cenné podněty při vedení této práce.

Dále bych rád poděkoval všem lidem, kteří mě ve studiu podporovali. Zvláště pak MUDr. Juditě Ďurovské, Ph.D. za to, že výrazně přispěla k tomu, že jsem se mohl úspěšně vrátit ke studiu, které jsem měl v letech 2016 až 2019 přerušené ze zdravotních důvodů.

# Front-end design webových aplikací

## Abstrakt

Předmětem diplomové práce je Front-end design webových aplikací, kde se autor zaměřuje na analýzu nejpoužívanějších JavaScript a UI frameworků. Poznatky vzešlé z analýzy jsou následně uplatněny na ukázkové aplikaci z oblasti zdravotnictví, která nese pracovní název „HealthCare-App“. Práce je rozdělena na dvě části.

V první části jsou popsána teoretická východiska. Jedná se tedy především o popis nejdůležitějších technologií a postupů týkajících se architektury moderních JavaScript frameworků. Následně jsou popsány nejdůležitější aspekty responzivního designu webových aplikací a možnosti usnadnění návrhu responzivních aplikací pomocí moderních UI frameworků. Na závěr této části je popsána vícekriteriální analýza variant, jejíž metodický postup je využit při analýze frameworků v druhé části práce.

Druhá část je věnována vlastní práci. Zde jsou na začátku popsány funkční a nefunkční požadavky ukázkové aplikace HealthCare-App. Následně je provedena vícekriteriální analýza variant s cílem nalézt jak u JavaScript, tak i u UI frameworků kompromisní variantu – tj. nejvhodnější frameworky pro implementaci této ukázkové aplikace. Vícekriteriální analýza je provedena pomocí metody váženého součtu. Výsledkem této analýzy je, že jako JavaScript framework je vhodné použít Angular. A jako UI framework pak Twitter Bootstrap.

Analýzou získané poznatky jsou následně prakticky uplatněny v ukázkové aplikaci HealthCare-App, která je následně otestována na různých typech koncových zařízení.

**Klíčová slova:** telemedicína, front-end, Mobile First, design, JavaScript, HTML, CSS, SVG, angular, React, Ember, Vue, Knockout, PrimeNG, Bootstrap, Foundation, Skeleton, statistika

# Web applications front-end design

## Abstract

The subject of the master thesis is web application front-end design, where the author focuses on the analysis of the most used JavaScript and UI frameworks. The findings of the analysis are then applied to a sample application in the field of healthcare, which has the working title "HealthCare-App". The work is divided into two parts.

The first part describes the theoretical background. It is therefore primarily a description of the most important technologies and procedures related to the architecture of modern JavaScript frameworks. Subsequently, the most important aspects of responsive design of web applications and the possibilities of facilitating the design of responsive applications using modern UI frameworks are described. At the end of this part, a multi-criteria decision analysis of variants is described, the methodological procedure of which is used in the analysis of frameworks in the second part of the work.

The second part is focused on the own work. There are described the functional and non-functional requirements of the sample HealthCare-App application. Subsequently, a multi-criteria decision analysis of variants is performed in order to find a compromise variant for both JavaScript and UI frameworks - the most suitable frameworks for the implementation of this sample application. Multi-criteria decision analysis is performed using the weighted sum method. The result of this analysis is that it is recommended to use Angular as a JavaScript framework. And as a UI framework it is recommended to use Twitter Bootstrap.

The knowledge gained through the analysis is then practically applied in the sample application HealthCare-App, which is then tested on various types of devices.

**Keywords:** telemedicine, front-end, Mobile First, design, JavaScript, HTML, CSS, SVG, Angular, React, Ember, Vue, Knockout, PrimeNG, Bootstrap, Foundation, Skeleton, statistics

## Obsah

<b>1 Úvod.....</b>	<b>11</b>
<b>2 Cíl práce a metodika.....</b>	<b>14</b>
2.1 Cíl práce.....	14
2.2 Metodika.....	14
<b>3 Teoretická východiska.....</b>	<b>15</b>
3.1 Popis nejpoužívanějších technologií.....	15
3.1.1 Front-end technologie.....	15
3.1.1.1 HTML a CSS.....	15
3.1.1.2 CSS preprocesory.....	16
3.1.1.3 JavaScript.....	16
3.1.2 Front-end JavaScript frameworky.....	17
3.1.2.1 Angular.....	17
3.1.2.2 React.js.....	18
3.1.2.3 Vue.js.....	18
3.1.2.4 jQuery.....	19
3.1.2.5 PrimeNG.....	19
3.1.3 Middleware a webové služby.....	20
3.1.3.1 JSON.....	20
3.1.3.2 XML-RPC.....	20
3.1.3.3 SOAP.....	21
3.1.3.4 REST.....	21
3.1.4 Back-end technologie.....	22
3.2 Responzivní design.....	23
3.2.1 CSS a responzivní design.....	23
3.2.1.1 Mediální dotazy (Media Queries).....	23
3.2.1.2 Plovoucí struktury (Fluid Grids).....	25
3.2.1.3 Škálovatelné obrázky (Scalable Images).....	28
3.2.2 Princip Mobile First.....	30
3.2.3 UI frameworky.....	33
3.3 Model vícekritériální analýzy variant.....	33
3.3.1 Saatyho metoda.....	35
3.3.2 Analýza metodou váženého součtu.....	36
<b>4 Vlastní práce.....</b>	<b>38</b>
4.1 Analýza požadavků.....	38



4.1.1 Funkční požadavky.....	38
4.1.2 Nefunkční požadavky.....	39
4.2 Porovnání JavaScript frameworků.....	40
4.2.1 Fáze č. 1 - Výběr podle aktivity vývojářské komunity.....	41
4.2.2 Fáze č. 2 – Zhodnocení kvality a kompatibility zbylých frameworků.....	45
4.2.2.1 Angular.....	47
4.2.2.2 React.....	50
4.2.2.3 Ember.....	56
4.2.2.4 Vyhodnocení variant a nalezení nejlepší kompromisní varianty.....	58
4.3 Porovnání UI frameworků.....	61
4.3.1 Twitter Bootstrap.....	63
4.3.2 ZURB Foundation.....	67
4.3.3 Skeleton.....	69
4.3.4 Vyhodnocení variant a nalezení nejlepší kompromisní varianty.....	71
4.4 Implementace ukázkové aplikace.....	73
4.4.1 Angular a jeho konfigurace.....	73
4.4.2 Způsob využití CSS.....	73
4.5 Implementace jednotlivých komponent aplikace.....	74
4.5.1 Komponenta Login.....	74
4.5.2 Komponenta Overview.....	76
4.5.3 Komponenta Questionare.....	79
4.5.4 Komponenta Statistics.....	82
4.5.5 Komponenta Medication.....	87
4.5.6 Komponenta Profile.....	91
4.6 Testování.....	94
4.6.1 Manuální testování pro různá koncová zařízení.....	94
4.6.2 Unit testy.....	94
<b>5 Výsledky a diskuse.....</b>	<b>96</b>
5.1 Výsledky analýzy JavaScript frameworků.....	96
5.2 Výsledky analýzy UI frameworků.....	98
<b>6 Závěr.....</b>	<b>101</b>
<b>7 Seznam použitých zdrojů.....</b>	<b>103</b>
<b>8 Přílohy.....</b>	<b>105</b>
8.1 Příloha č. 1.....	105
8.2 Příloha č. 2.....	106
8.3 Příloha č. 3.....	107
8.4 Příloha č. 4.....	108
8.5 Příloha č. 5.....	109

## Seznam obrázků

Obrázek 1: Ukázka využití datového atributu data-src pro načtení obrázků.....	29
Obrázek 2: Ukázka načtení obrázku z data-src pomocí JavaScriptu.....	29
Obrázek 3: Kriteriační matice.....	33
Obrázek 4: Saatyho matice.....	35
Obrázek 5: Frameworky – aktivita za poslední měsíc.....	41
Obrázek 6: Frameworky – aktivita za poslední rok.....	42
Obrázek 7: Ukázka implementace služby (service) v Angularu.....	47
Obrázek 8: Ukázka nadřazené (parent) komponenty a volání její sub-komponenty.....	48
Obrázek 9: Obrázek: Ukázka implementace sub-komponenty.....	48
Obrázek 10: Ukázka implementace služby (service) v Reactu.....	50
Obrázek 11: Ukázka implementace nadřazené (parent) komponenty v Reactu.....	51
Obrázek 12: Ukázka implementace sub-komponenty v Reactu.....	52
Obrázek 13: Mřížka a zlomové body v Bootstrapu.....	60
Obrázek 14: Podpora Bootstrapu na mobilních prohlížečích.....	61
Obrázek 15: Podpora Bootstrapu na desktopových prohlížečích.....	61
Obrázek 16: Podpora Foundation v jednotlivých webových prohlížečích.....	64
Obrázek 17: Zlomové body ve Skeleton.....	65
Obrázek 18: Komponenta Login v mobilní verzi.....	70
Obrázek 19: Komponenta Login v desktopové verzi.....	71
Obrázek 20: Komponenta Overview v mobilní verzi.....	72
Obrázek 21: Komponenta Overview v desktopové verzi.....	73
Obrázek 22: Hlavní menu v mobilní verzi.....	74
Obrázek 23: Komponenta Questionare v mobilní verzi.....	76
Obrázek 24: Komponenta Questionare v desktopové verzi.....	77
Obrázek 25: Komponenta Statistics v mobilní verzi – spojnicový graf.....	79
Obrázek 26: Komponenta Statistics v desktopové verzi – spojnicový graf.....	80
Obrázek 27: Komponenta Statistics v mobilní verzi – pavučinový teorém.....	80
Obrázek 28: Komponenta Statistics v desktopové verzi – pavučinový teorém.....	81
Obrázek 29: Zdrojový kód unit testů pro komponentu Login.....	89
Obrázek 30: Výsledek spuštěných unit testů pro komponentu Login.....	89
Obrázek 31: Celkový užitek variant – JavaScript frameworky.....	92
Obrázek 32: Celkový užitek variant – UI frameworky.....	94

## Seznam tabulek

Tabulka 1: Funkční požadavky.....	38
Tabulka 2: Kritéria pro vícekritériální analýzu JavaScript frameworků.....	46
Tabulka 3: Ohodnocení variant podle jejich kritérií.....	58
Tabulka 4: Stanovení vah kritérií dle Saatyho metody.....	59
Tabulka 5: Stanovení ideální a bazální varianty.....	59
Tabulka 6: Kriteriační matice R.....	60
Tabulka 7: Celkový užitek variant a jejich pořadí.....	60
Tabulka 8: Kritéria pro vícekritériální analýzu UI frameworků.....	61
Tabulka 9: Ohodnocení variant podle jejich kritérií.....	71
Tabulka 10: Stanovení vah kritérií dle Saatyho metody.....	71
Tabulka 11: Stanovení ideální a bazální varianty.....	72

Tabulka 12: Kriteriační matice R.....	72
Tabulka 13: Celkový užitek variant a jejich pořadí.....	72

# 1. Úvod

V dnešní době by se dalo říci, že internet téměř hýbe světem. Jedná se o relativně mladou (zhruba 30 let starou) počítačovou síť, která se však v průběhu těchto 30 let velmi rychle rozvíjela. Tím pochopitelně i významně rostl počet lidí, kteří jsou připojeni k internetu a využívají stále více k činnostem kvůli kterým museli ještě před pár lety obcházet různé instituce (úřady, lékaře atd.) či místo využívání webových aplikací vyplňovat různé papírové formuláře apod.

Zároveň v posledních letech výrazně roste se i počet uživatelů, kteří si webové aplikace prohlíží z jiných zařízení, než v desktopu. A sice z tabletů a především z chytrých mobilních telefonů. Pro takové zařízení je potřeba Front-end webových aplikací optimalizovat a zpravidla i zjednodušit oproti desktopové verzi. A to logického důvodu, neboť mobil má obvykle menší displej s orientací na výšku a disponuje i menším výkonem, než klasický počítač. Koncept optimalizace Front-endu webových aplikací pro různá koncová zařízení se označuje jako responzivní design. Weby s podporou responzivního designu lze snadněji navrhovat s využitím nějakých UI frameworků určených pro tyto účely – např. Twitter Bootstrap či ZURB Foundation.

Taktéž roste komplexnost webových aplikací a je velmi žádoucí webové aplikace stavět podle konceptu tzv. Single Page Application (SPA) <sup>1</sup>, kdy k překreslení určité části kódu v DOMu webové prezentace není potřeba v prohlížeči znovu načíst celý web, ale jen zavolat HTTP požadavek na tu konkrétní část webu. K tomu slouží především moderní JavaScriptové frameworky, jakými jsou například Angular, React a několik dalších.

Důležitou součástí této diplomové práce bude porovnání různých UI a JavaScriptových frameworků a následný výběr těch nejvhodnějších pro implementaci ukázkové responzivní webové aplikace.

Jednou z oblastí, kam internet (a s ním i responzivní webové aplikace) v poslední době proniká, je zdravotnictví. Zdravotnictví v rámci těchto zmiňovaných zhruba 30 let dlouho fungovalo spíše v „papírové“ podobě, neboť zdravotnictví je jako obor specifické tím, že provést zde jakoukoliv inovační změnu obvykle trvá delší dobu. Nicméně i tak se stále více do popředí dostává moderní koncept zdravotnictví, který se někdy nazývá jako „telemedicína“. Vzhledem k tomu, že se autor této práce o zdravotnictví dlouhodobě zajímá, tak v rámci analýzy a následné syntézy front-end designu webových aplikací budou poznatky vzešlé z analýzy aplikovány na ukázkové responzivní webové aplikaci právě z oblasti telemedicíny.

---

<sup>1</sup> <https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/>

Ukázková responzivní webová aplikace ponese pracovní název „HealthCare-App“. Aplikace by měla sloužit pro usnadnění komunikace mezi lékařem a jeho pacienty, kteří se u daného lékaře léčí dlouhodobě s nějakým chronickým onemocněním. Základní myšlenkou aplikace je fakt, že pokud se pacient dlouhodobě léčí s nějakým chronickým onemocněním, tak zpravidla mnohokrát navštíví svého ošetřujícího lékaře. A také má daný pacient obvykle (především na začátku léčby) mnoho otázek. Otázky, které dostává ošetřující lékař, se zpravidla velmi často opakují a ve velké většině otázky od pacientů obvykle nebývají nijak extrémně závažné z hlediska klinického stavu pacientů. Zásadním problémem je fakt, že lékař takových otázek dostane každý den mnoho a tyto e-mailové otázky nejsou nijak strukturované - například podle závažnosti klinického stavu daného pacienta. To znamená, že není ve časové kapacitě ošetřujícího lékaře (a často ani jeho sestry) všechny dotazy od pacientů za den projít a tím se snadno může stát, že se k ošetřujícímu lékaři nedostane nějaký závažný dotaz od pacienty, který je vhodné urychleně začít řešit – a to dříve, než na další domluvené osobní kontrole mezi pacientem a jeho lékařem.

Typickým příkladem diagnózy, kdy se může měnit pacientův stav mezi 2 kontrolami (a pacient má potřebu to řešit se svým lékařem) je lymfská borelióza v chronickém stádiu. Lymfská borelióza je infekce bakteriálního původu přenášena na člověka nejčastěji klíštětem. Pokud se tato nemoc zachytí brzy, když je ještě v akutním stádiu, tak na léčbu zpravidla stačí 21 dní antibiotik po kterých je pacient bez větších komplikací vyléčen.

Pokud se nemoc u pacienta včas nerozpozná, když je ještě v akutním stádiu, tak může přejít do chronického stadia. A tady je již léčba podstatně delší a komplikovanější.<sup>2</sup>

Již nestačí pouhá 1 antibiotika na 21 dní, ale nasazuje se obvykle dlouhodobý komplexní protokol, kdy pacient postupně vystřídá všechno možné – antibiotika (obvykle v kombinaci), imunoglobuliny atd. Velmi často se stává, že se pacientův stav po určité době od nasazení této léčby začne kolísat tak, jak pacient reaguje na léčbu, neboť dochází k tzv. Jarisch-Herxheimerově reakci<sup>3</sup>. To znamená, že pokud má daný pacient správně nastavenou léčbu, tak by jeho klinický stav sice měl mít zlepšující se trend, ale dočasně se může i zhoršovat a následně zase zlepšovat.

A přesně v takové situaci má pacient trpící chronickou boreliózou či nějakou jinou chronickou nemocí obvykle potřebu kontaktovat svého lékaře s dotazem, jestli je vše v

---

<sup>2</sup> HOROWITZ, Richard I. How Can I Get Better?: An Action Plan for Treating Resistant Lyme & Chronic Disease. 1. vydání. New York: St. Martin's Press, 2017.

<sup>3</sup> Jarischova-Herxheimerova reakce [online]. Dostupné z: [https://www.wikiskripta.eu/w/Jarischova-Herxheimerova\\_reakce](https://www.wikiskripta.eu/w/Jarischova-Herxheimerova_reakce)

pořádku, když jeho klinický stav takto kolísá. Kromě dotazů týkajících se Jarisch-Herxheimerových-reakcí naopak může mít lékař i zájem zeptat se pacienta typicky třeba po 2 týdnech od nasazení léčby, jak se mu daří a jestli dané léky zabírají. Opět se lékaři hodí mít tuto konverzaci nějak systematicky kategorizovanou.

A přesně to je hlavní myšlenka této aplikace - možnost systematické komunikace mezi lékařem a pacientem podle moderního konceptu, který se ve zdravotnictví někdy nazývá jako “telemedicína”.

Ukázková aplikace bude navržena jako responzivní webová Single-Page aplikace, která bude implementována s využitím nějakého moderního JavaScript a UI frameworku. Před samotnou implementací bude provedena srovnávací analýza nejpoužívanějších JavaScript a UI frameworků, jejíž výsledek bude výběr toho nejvhodnějšího řešení pro daný Use-Case aplikace.

Z hlediska funkčních požadavků bude aplikace navržena tak, že každý pacient bude mít svůj uživatelský účet, pod kterým bude s aplikací pracovat. Po přihlášení bude pacient pravidelně vyplňovat dotazník týkající se jeho aktuálních symptomů, který bude v grafu přehledně vyhodnocován a porovnáván s předchozím vyplněním symptomů ze strany pacienta. Data pro tyto grafy budou získávána tak, že jednotlivé odpovědi jsou obodovány podle toho, jak pacient odpoví. Tyto body se následně sečtou a součet zanesou do grafu k danému dotazníku.

## 2. Cíl práce a metodika

### 2.1. Cíl práce

Cílem této diplomové práce je provedení analýzy a následné syntézy technologií a frameworků pro tvorbu Front-end designu u moderních webových aplikací. Jedná se především o technologie a frameworky postavené nad CSS a JavaScriptem.

Dále bude provedena analýza, jaká specifika obnáší zobrazení responzivních webových aplikací na různých zařízeních, jež mají různou velikost displeje (typicky desktop, tablet, mobil).

Analyzované poznatky budou prakticky využity ve webové aplikaci z oblasti zdravotnictví, která pro účely této práce ponese název „HealthCare-App“.

### 2.2. Metodika

Z pohledu metody budou především srovnávány jednotlivé UI a JavaScript frameworky určené pro tvorbu moderních Front-end aplikací.

Konkrétně bude u JavaScript frameworků analyzováno a následně srovnáváno:

- Aktivita vývojářské komunity u jednotlivých JavaScript frameworků za poslední měsíc a za poslední rok
- Nedůležitější části architektury JavaScript frameworků podle následujících kritérií:
  - Velikost frameworku
  - Architektura a API
  - Nástroje
  - Komponenty

Architektura JavaScript frameworků bude posouzena v návaznosti na požadavky ukázkové webové aplikace HealthCare-App, která bude implementována ve fázi konceptu. Posouzena bude na základě výsledků provedené vícekritériální analýzy variant – konkrétně metodou váženého součtu.

Jednotlivé UI frameworky budou taktéž analyzovány a následně srovnávány podle vícekritériální analýzy variant – metodou váženého součtu. Sledována budou tato kritéria:

- Velikost frameworku
- Mřížka a zlomové body
- Responzivní ovládací prvky
- Customizace
- Podpora webových prohlížečů

Správnost zobrazení aplikace na různých zařízeních (desktop, tablet, mobil) bude ověřena u ukázkové aplikace v rámci uživatelského testování. Dále budou pro vybranou funkcionalitu napsány unit testy.

### **3. Teoretická východiska**

#### **3.1. Popis nepoužívanějších technologií**

##### **3.1.1. Front-end technologie**

###### **3.1.1.1. HTML a CSS**

HTML je jazyk definovaný konsorciem W3C. A je určený pro definici struktury webové stránky. HTML konkrétně umožňuje:

- Publikovat online dokumenty s nadpisy, texty, tabulkami, seznamy, fotogaleriemi atd.
- Načíst online informace kliknutím na hypertextové odkazy.
- Navrhnout formuláře určené pro provádění transakcí se vzdálenými službami, pro hledání informací, zadávání rezervací, objednávání produktů atd.
- Zahrnout tabulky, videa, zvukové nahrávky a další aplikace přímo do těchto online dokumentů.

Struktura webové stránky se v HTML píše pomocí tzv. HTML tagů – např. `<h1>`, `<p>`, `<table>`, `<ul>`, `<img>` atd.

CSS je jazyk definovaný také konsorciem W3C. A slouží k definování vizuální podoby dané webové stránky, jejíž struktura se nadefinuje v HTML. U jednotlivých HTML elementů můžeme pomocí CSS definovat barvy, layout, fonty atd. To umožňuje přizpůsobit webovou stránku různým typům zařízení (pomocí návrhu responzivního designu). CSS je nezávislý na HTML a lze jej použít v jakémkoliv značkovacím jazykem založeným na XML. Výhoda oddělení struktury webu (HTML) od jeho grafické podoby (CSS) spočívá ve snadnější údržbě webu, sdílení stylů mezi různými stránkami a schopnost přizpůsobit stránky různým prostředím.<sup>4</sup>

---

<sup>4</sup> HTML & CSS [online]. Dostupné z: <https://www.w3.org/standards/webdesign/htmlcss.html>



### 3.1.1.2. CSS preprocessory

CSS preprocessory v podstatě napravují staré neduhy čistého CSS, které v podstatě neumožňuje ve stylech definovat proměnné, funkce či zanořování selektorů do stromové struktury. A přesně tyto nedostatky řeší CSS preprocessory, které následně v nich napsané styly konvertují do standardního CSS tak, aby mu rozuměly i webové prohlížeče.

„**Proměnné** výhodně použijete na výčet barev nebo šířek bloků. **Výrazy a vestavěné funkce** vám pak pomohou s jejich použitím - nebudete již muset sčítat okraje a šířky, napíšete prostě jen `margin-left: 20px + $aside-width` a je to. Podobně na barvu textu se může hodit `color: darken(@cocacola-red, 10%);`.

Určitě jste také již kódovali styl pro nějakou zvláštní část stránky a po chvíli jste zjistili, že vám deset pravidel pod sebou začíná např. `.footer`. V takovém případě jistě oceníte **vnořování**. **Objektové vlastnosti** vám umožňují deklarovat si kousky kódu a pak je opakovaně používat na mnoha místech a dokonce i dědit od nich. Bystřejší už vědí, že přesně toto budou chtít použít na rozložené střípky CSS3. Použití pak bude jednoduché: `.rounded-corners(10px);`“<sup>5</sup>

Používají se především 3 různé preprocessory a to, LESS, SASS (nebo SCSS<sup>6</sup>) a Stylus. V ukázkové aplikaci této práce bude použit SCSS, neboť se v současné době jedná v podstatě o standard na poli CSS procesorů.

### 3.1.1.3. JavaScript

JavaScript jako programovací jazyk prošel v posledních pár letech velmi výrazným rozvojem. A lze s určitou nadsázkou říci, že JavaScript, který se používal pro vývoj aplikací cca. Před 10 lety je vlastně úplně jiný JavaScript, než ten, jež se pro vývoj moderních Front-end aplikací používá dnes. Tento trend je patrný zejména od roku 2012, kdy Google vydal první verzi populárního frameworku AngularJS. Taktéž v této samé době došlo k bouřlivému rozvoji chytrých telefonů a tabletů, které zásadním způsobem proměnily způsob, jakým se dnes vyvíjí Front-end webových aplikací.

---

<sup>5</sup> CSS preprocessory: méně psaný, vyšší efektivita [online]. Dostupné z: <https://www.zdrojak.cz/clanky/css-preprocessory-mene-psani-vyssi-efektivita/>

<sup>6</sup> SCSS je modernější verze SASS

### 3.1.2. Front-end JavaScript frameworky

#### 3.1.2.1. Angular

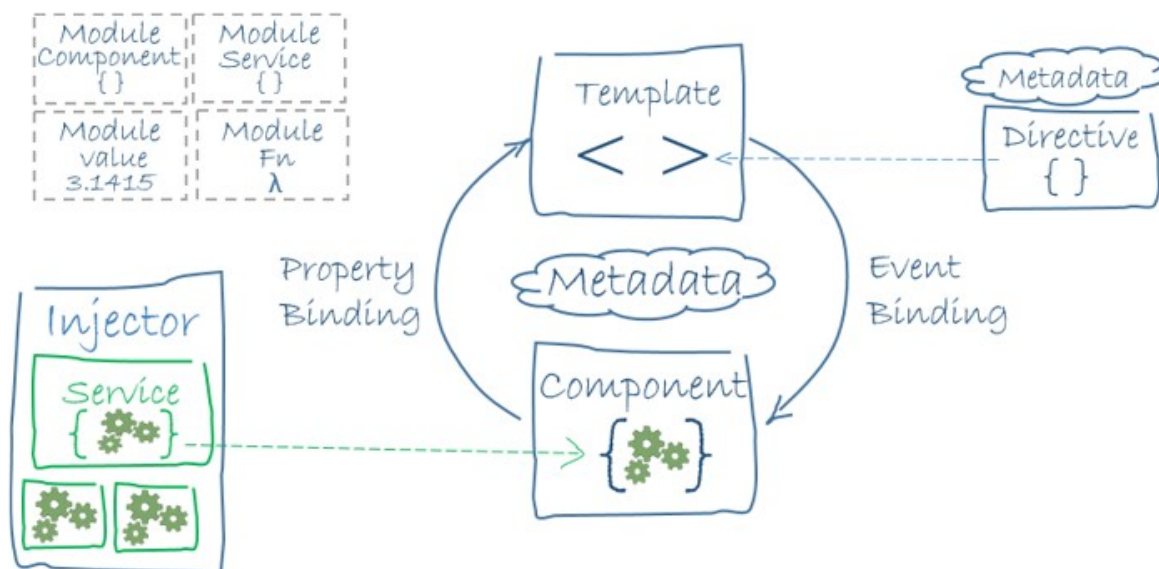
„Angular je frontendový framework pro tvorbu webových aplikací. Staví na komponentové architektuře se službami a používá jazyk TypeScript (TS) namísto čistého JavaScriptu.“<sup>7</sup>

„Angular je zcela postaven na tzv. komponentách. Základem každé komponenty je programová třída (`class`) a tím pádem staví na základech objektově orientovaného programování (dále jen OOP). K této třídě pak Angular naváže vlastní HTML šablonu a dokonce i CSS pomocí tzv. direktiv. Tímto způsobem můžeme pak rozdělit webovou stránku na samostatné celky a poskládat ji právě pomocí principů OOP. Např. menu bude komponenta, komentáře na stránce budou komponenta a vlastně i celá stránka bude komponenta skládající se s dílčích celků. A stejně jako v objektovém programování mají tyto komponenty vlastní izolovanou funkcionalitu a snaží se být ideálně znovupoužitelné. To tedy máme vykreslování a funkcionalitu na stránce. Co nám ale ještě chybí, je samotná logika aplikace. Pro tu se v Angularu používají tzv. služby. Ty slouží např. pro komunikaci a distribuci dat mezi klientskou aplikací v Angularu a serverovým API. Jednotlivé komponenty pak mohou využívat tyto služby pro svoji potřebu a získají je pomocí tzv. Dependency Injection (DI), kterou tento framework také obsahuje.

Na závěr je dobré říct, že komponenty i služby se pak dále dají logicky rozdělit do tzv. Modulů.“<sup>7</sup>

---

<sup>7</sup> Lekce 1 – Úvod do frameworku Angular [online]. Dostupné z: <https://www.itnetwork.cz/javascript/angular/zaklady/uvod-do-angular-frameworku>



Obrázek: Architektura Angularu <sup>7</sup>

### 3.1.2.2. React.js

„React je JS open-source knihovna od společnosti Facebook pro tvorbu uživatelského rozhraní (UI). Na rozdíl od různých kompletních frameworků (např. Angular) se tedy soustředí na jednu specifickou oblast a pokud se na ní podíváme z hlediska klasické MVC architektury, tvoří právě a pouze view, tedy vrstvu pohledu, která prezentuje data uživateli. A pravděpodobně jste si všimli, že v současné době je jednou z nejoblíbenějších a nejžádanějších knihoven s velkou komunitou vývojářů.“ <sup>7</sup>

„Na úvod je také důležité říct něco o aspektech této knihovny a o vlastnostech aplikací v ní vytvořených. Základním stavební kámen zde tvoří tzv. komponenty (*components*), což jsou různé znovupoužitelné HTML elementy se zapouzdřenou funkcionalitou, jejichž skládáním vzniká komplexní UI aplikace. Tyto komponenty pak mají své vlastnosti (*props*) a zpravují svůj vnitřní stav (*state*). Tento deklarativní způsob práce s daty aplikace vede k více předvídatelnému chování i lehčímu ladění a to může být právě jeden z důvodů, proč je tato knihovna tak populární. Také si díky tomu dobře rozumí s dalšími podobně zaměřenými knihovnami jako je např. Redux.“ <sup>8</sup>

### 3.1.2.3. Vue.js

„Vue.js je frontend framework pro tvorbu reaktivních webových aplikací. Má vlastní systém komponentů, routingu a dalších pár vychytávek. Může se psát v JavaScriptu, ale má také nativní podporu pro TypeScript. Byl vytvořen vývojářem Evan You.“ <sup>9</sup>

<sup>8</sup> Lekce 1 – Úvod do React [online]. Dostupné z: <https://www.itnetwork.cz/javascript/react/zaklady/uvod-do-react>

<sup>9</sup> Lekce 1 – Úvod do Vue.js [online]. Dostupné z: <https://www.itnetwork.cz/javascript/vuejs/uvod-do-vuejs>

„Proč by jsme zrovna měli psát web ve Vue namísto oblíbeného Reactu a nebo enterprise Angularu? Spoustu začátečníků se k Vue obrací kvůli jednoduchosti. Místo toho, aby jsme jako v Reactu psali celý frontend v JSX, můžeme psát normální HTML a tomu dát potom život pomocí Vue. Kromě toho má Vue.js optimalizovaný výkon a jeho minifikovaná verze má pouze kolem 20KB.“<sup>9</sup>

Dle zkušeností autora této práce je Vue.js svou architekturou nejvíce podobné známému a hojně používanému frameworku React.

#### 3.1.2.4. jQuery

jQuery patřilo svého času mezi nejrozšířenější JavaScript knihovnu, kterou znal každý, kdo se nějakým způsobem zajímal o Front-end webových aplikací. První verze jQuery vyšla již v roce 2006 a představovala tehdy takovou menší revoluci, neboť konečně bylo možné DOM zapouzdřit do předem otestovaných jQuery funkcí a tím se vyhnout velkým problémům s nekompatibilitou tehdejších webových prohlížečů.

V dnešní době však jQuery pomalu ztrácí na svém významu, neboť se v něm sice dá pořád dobře pracovat s DOMem, ale jQuery nenabízí (na rozdíl od frameworků jako např. Angular či React) takové možnosti pro napsání dobře strukturované MVC<sup>10</sup> nebo MVVM<sup>11</sup> aplikace.

„Knihovna má jednoduchou syntaxi, výrazně zjednodušuje manipulaci s obsahem stránky, reakci na události, animace a používání AJAXu. Její největší síla ale pravděpodobně spočívá v obrovském množství jQuery pluginů, které můžeme pro své stránky stáhnout a získáme tak bez práce interaktivní galerie, carousely, rozšířené formulářové prvky a podobně. I když doba její největší popularity již pominula, stále by měla patřit do základního povědomí frontendistů a je nadále masově užívána nejpopulárnějšími frameworky jako je např. Samotný CSS framework Bootstrap. Konkurenci jQuery v poslední době představují komplexnější vícevrstvé frameworky jako je např. Angular nebo React.“<sup>12</sup>

#### 3.1.2.5. PrimeNG

PrimeNG je komplexní knihovna různých UI komponent pro Angular. Ve své původní verzi v podstatě vychází z PrimeFaces<sup>13</sup>, které poskytují UI komponenty

---

<sup>10</sup> Model-View-Controller

<sup>11</sup> Model-View-ViewModel

<sup>12</sup> Lekce 1 – Úvod do jQuery [online]. Dostupné z:

<https://www.itnetwork.cz/javascript/jquery-zaklady/javascript-tutorial-funkcionalni-programovani-a-jquery-webova-kalkulacka>

<sup>13</sup> PrimeFaces [online]. Dostupné z: <https://www.primefaces.org/>

aplikacím napsaným v Javě. Nabízí tedy různé prvky, které se dají použít v Angular aplikaci ve formě Angular komponent. Obsahuje například komponenty pro formulářové prvky, tlačítka, vizualizace dat (např. stromové struktury či tabulky), různé panely (např. Toolbar, Fieldset, Scrollpanel atd.), upload souborů, menu, grafy, notifikace, fotogalerie či komponentu pro Drag&Drop <sup>14</sup>. Všechny komponenty mají propracované API, pomocí kterého lze individuálně upravit chování daných komponent. <sup>15</sup>

### **3.1.3. Middleware a webové služby**

Webové služby slouží v moderních webových aplikacích jako tzv. „mezivrstva“ pro komunikaci a výměnu dat mezi Front-endem a Back-endem. Architektura takových aplikací je obvykle navržena tak, že se domluví společné API <sup>16</sup> pro Front-end a Back-end. V rámci tohoto API se nadefinují webové služby pro jednotlivé operace s daty. Následně podle pravidel architektury klient-server pošle Front-end HTTP request na daný endpoint a Back-end vrátí HTTP response. Pro přenos dat mezi Front-endem a Back-endem se jako datový formát používá nejčastěji JSON. A z protokolů pro webové služby se nejvíce používají REST nebo SOAP.

#### **3.1.3.1. JSON**

„JSON (JavaScript Object Notation) je jednoduše řečeno formát pro přenos dat, podobně jako SOAP nebo XML-RPC. Na rozdíl od těchto dvou formátů však formát JSON není založen na formátu XML. Jedná se o kód v JavaScriptu, který je volně založen na definicích a formátech ve stylu jazyka C. Ačkoli má tento formát v názvu slovo JavaScript, nabízí pro mnoho programovacích jazyků pracujících na straně serveru parsers. Díky tomu a také své netěžkopádnosti se stal populární alternativou k XML pro komunikaci mezi webovým prohlížečem a serverem.“ <sup>17</sup>

#### **3.1.3.2. XML-RPC**

Jedná se o protokol, který využívá XML k zakódování své výzvy a HTTP jako transportní mechanismus, který umožňuje aplikacím napsaným v různých programovacích jazycích komunikaci mezi různými počítačovými architekturami a jejich operačními systémy. Také se obecně používá na XML pro vzdálené volání procedur, a to nezávisle na specifickém protokolu. <sup>18</sup>

---

<sup>14</sup> Chytnout a táhnout

<sup>15</sup> PrimeNG [online]. Dostupné z: <https://primefaces.org/primeng/>

<sup>16</sup> Application Programming Interface

<sup>17</sup> CHOW, S. Programujeme Mashup aplikace pro Web 2.0 v PHP. Brno: Computer Press, 2008. s. 229

<sup>18</sup> XML-RPC [online]. Dostupné z: <https://it-slovník.cz/pojem/xml-rpc>

### 3.1.3.3. SOAP

„SOAP, dříve známý jako Simple Object Access Protocol (zkratka přišla až ve verzi 1.2), se zrodil krátce po příchodu XML-RPC, Byl vytvořen skupinou vývojářů s kořeny ve společnosti Microsoft. Zajímavé je, že tvůrce XML-RPC.“<sup>19</sup>

„Podobně jako XML-RPC je i SOAP protokol postavený na XML. SOAP však napravuje spoustu nedostatků XML-RPC – jmenovitě absenci uživatelských datových typů, lepší podporu znakových sad a elementární zabezpečení. Jednoduše řečeno je to mocnější a flexibilnější protokol než REST i XML-RPC. Naneštěstí s touto flexibilitou jdou ruku v ruce i určité nevýhody. Protokol SOAP je mnohem složitější a náročnější na použití. Ačkoli může SOAP pracovat sám o sobě, je mnohem užitečnější ve spojení s dalším standardem postaveným na XML nesoucím název WDSL( Web Services Descriptor Language).“<sup>20</sup>

Proto je pro práci s protokolem SOAP dobré také znát XDSL a XSD.

### 3.1.3.4. REST

REST (Representational State Transfer) je v současnosti velmi populární pro využití ve webových službách. Nejedná se o formální standard, nýbrž spíše o styl architektury.

„V praxi je REST jednoduchý a flexibilní. Podobně jako v případě XML-RPC odešle klient požadavek serveru. Zde ale veškerá podobnost končí. Požadavky REST jsou jednoduše požadavky HTTP. Může se použít kterákoli z pěti standardních metod protokolu HTTP, kterými jsou GET, POST, PUT, DELETE a HEAD. Poslední tři jsou ve světě vývoje webových aplikací poměrně vzácné a nejinak je tomu i v případě REST. Převažují metody POST a GET. Mnoho rozhraní API však aktuálně používá GET i v případech, kdy je vhodnější použít POST. V každém případě však požadavky REST pracují velmi podobně jako když webový prohlížeč otevře stránku z webového serveru.

Akceptované parametry jsou definované v rozhraní API. V případě REST využívající metodu GET se budou parametry nacházet v URL. Nijak se to neliší od běžného předávání parametrů metodou GET jejich připojením k URL. Není třeba naše požadavky formátovat do XML.

Podobně návratová hodnota závisí zcela na API. Mnoho API vrací určitou strukturu XML. Jiné jednoduše vrací řetězec bez jakékoli strukturalizace, Druhý jmenovaný způsob je častý obzvlášť u odpovědi, které obsahují pouze jednu hodnotu namísto více hodnot. Nevrací se žádná strukturalizovaná chybová hlášení. API definuje, jakým způsobem bude

---

<sup>19</sup> CHOW, S. Programujeme Mashup aplikace pro Web 2.0 v PHP. Brno: Computer Press, 2008. s. 81

<sup>20</sup> CHOW, S. Programujeme Mashup aplikace pro Web 2.0 v PHP. Brno: Computer Press, 2008. s. 82

informovat o chybě. Je zcela na nás, abychom se rozhodli, jakým způsobem zpracujeme tuto odpověď.“<sup>21</sup>

REST se při volání REST požadavku ze strany klienta na server obvykle používá ve spojení s AJAX<sup>22</sup>.

„AJAX (Asynchronous JavaScript And XML) je označení technologie, pomocí které můžeme z JavaScriptu provádět **HTTP** požadavky. Je to jako bychom do adresního řádku prohlížeče zadali nějakou webovou adresu. Prohlížeč odešle HTTP požadavek a stáhne **data** na dané adrese (např. HTML kód webové stránky). AJAX nám umožňuje toto provést přímo v javascriptovém kódu a hlavně **na pozadí**, aniž by se stránka, na které se nacházíme, musela přenačíst.“<sup>23</sup>

XML v názvu AJAXu může být zavádějící, protože pomocí AJAXu lze posílat a přijímat v podstatě jakýkoliv datový formát. Přičemž nejčastěji se v souvislosti s moderními JavaScript frameworky pro tento účel používá JSON.

### 3.1.4. Back-end technologie

Předmětem této diplomové práce je především analýza a implementace Front-end technologií s ohledem na zobrazení na jednotlivých koncových zařízeních (mobil, tablet, desktop). Nicméně žádný Front-end webová aplikace nemůže existovat bez příslušného Back-endu. A proto je vhodné uvést také alespoň základní popis nejdůležitějších Back-end technologií. Pro vývoj na straně Back-endu se obvykle používá jeden z těchto programovacích jazyků: JAVA, .NET, node.js, PHP a Python.

JAVA a .NET se používají zpravidla na velké korporátní systémy (např. v bankách), kde je požadavek na vysokou robustnost systémů.

Na menší projekty se naopak může použít node.js, PHP a nebo poslední dobrou získává velkou oblibu i Python.

Pokud je daná aplikace navržena tak, že je od sebe striktně oddělen Front-end a Back-end a mezitím je nějaký middleware, tak je z pohledu Front-endu vlastně v podstatě jedno, jaké technologie jsou použity na Back-endu. A to z důvodu, že takový Front-end pouze stahuje či posílá data pomocí definovaných webových služeb a jak tyto webové služby následně spolupracují s Back-endem již není pro Front-end příliš podstatné. Ale samozřejmě pokud se podíváme na systém jako celek (tj. na Front-end a Back-end dohromady), tak z hlediska například škálovatelnosti, rozšiřitelnosti,

---

<sup>21</sup> CHOW, S. Programujeme Mashup aplikace pro Web 2.0 v PHP. Brno: Computer Press, 2008. s. 41

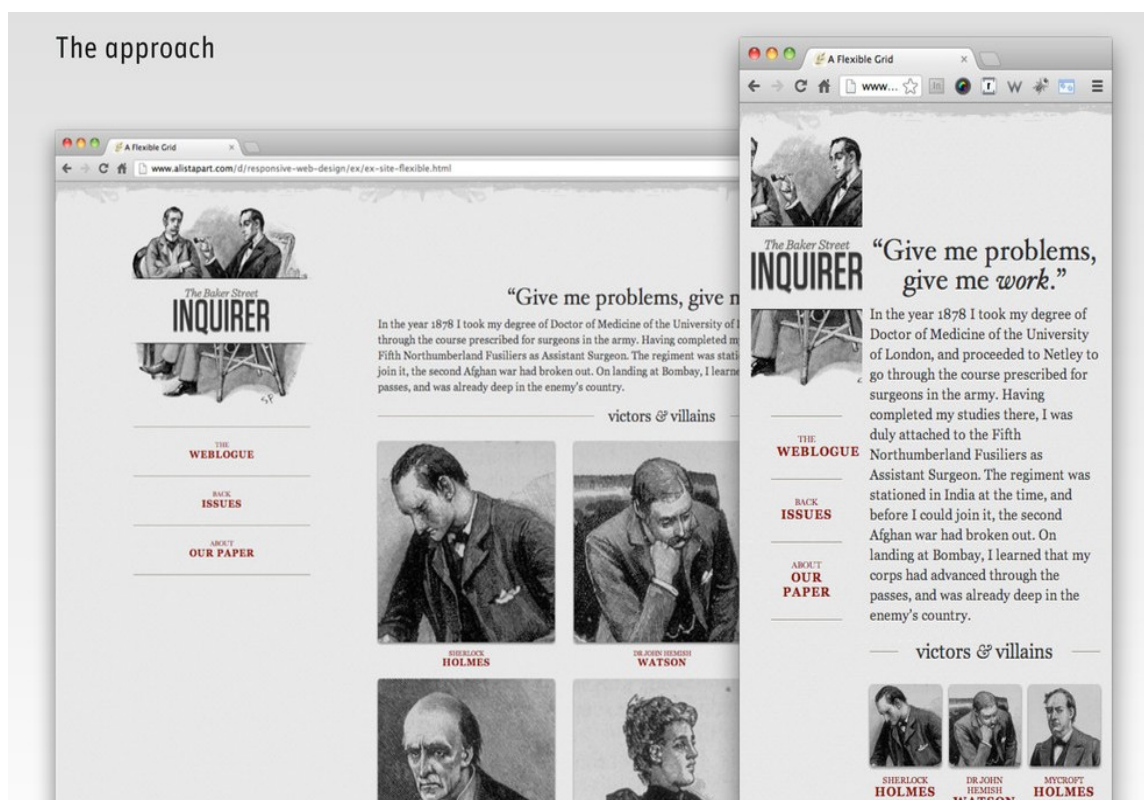
<sup>22</sup> Asynchronous JavaScript and XML

<sup>23</sup> Lekce 11 – AJAX v JavaScriptu – Základní dotazy [online]. Dostupné z: <https://www.itnetwork.cz/javascript/oop/ajax-v-javascriptu-zakladni-dotazy>

znovupoužitelnosti, výkonnosti atd. hraje volba technologie na Back-endu důležitou roli při vývoji daného systému/aplikace.

## 3.2. Responzivní design

„V květnu 2010 napsal Etham Marcotte článek pro web A List Apart s titulkem „Responsive Web Design“. Přístup, který zde popisoval, byl sice jednoduchý, ale zároveň revoluční. Použil k tomu tři základní nástroje – mediální dotazy (media queries), plovoucí struktury (fluid grids) a škálovatelné obrázky (scalable images) – web, který se báječně zobrazoval na mnoha různých rozlišeních.“<sup>24</sup> Všechny tyto tři důležité nástroje pro tvorbu responzivního designu budou popsány v této kapitole.



Obrázek: První responzivní web od Ethama Marcote<sup>25</sup>

### 3.2.1. CSS a responzivní design

#### 3.2.1.1. Mediální dotazy (Media Queries)

Mediální dotazy (Media Queries) jsou důležitou součástí CSS. V CSS stylech se zapisují pomocí pravidla @media.

Pravidla @media se v poprvé objevilo v CSS2 a sloužilo tehdy k tomu, aby bylo možné rozlišit styly pro různá zařízení. Konkrétně pravidlo nabízelo rozlišit styly pro

<sup>24</sup> KADLEC, Tim. Responzivní design profesionálně. 1. vydání. Brno: Zoner Press, 2014. s. 21

<sup>25</sup> A List Apart [online]. Dostupné z: <https://alistapart.com/>



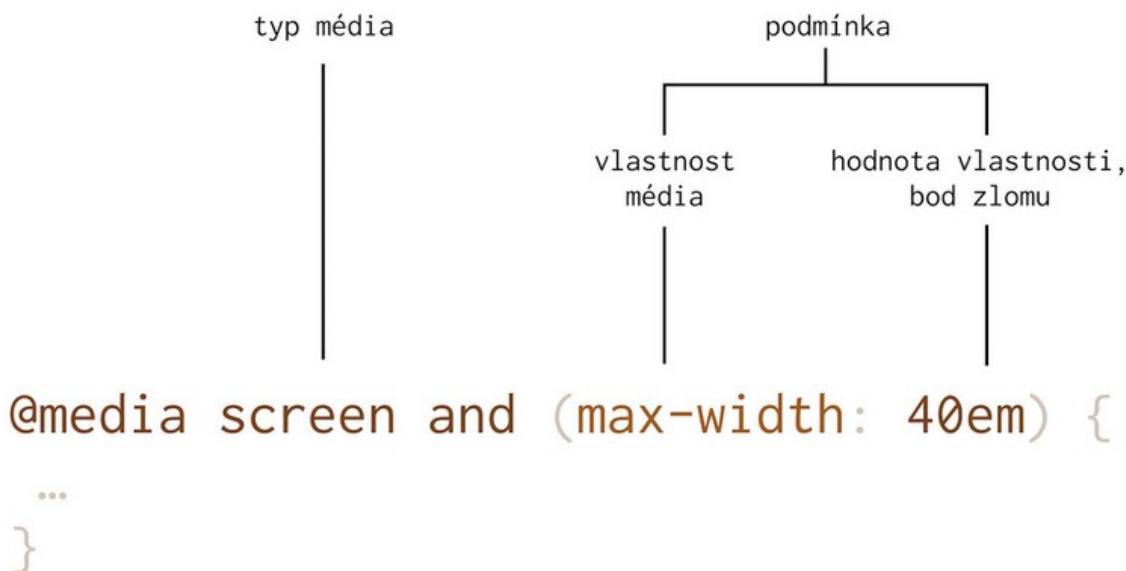
obrazovky (screen), tisk (print) a čtečky obrazovky (screenreaders). Nejčastěji se v té souvislosti pravidlo @media používalo pro stylování tiskové verze webu.<sup>26</sup>

S příchodem CSS3 došlo u Media Queries k vylepšení o bližší specifikaci vlastnosti médií, na kterých se daný web zobrazuje. Od CSS3 tedy mohou být Media Queries využity pro kontrolu (a tím i pro napsání vlastních stylů pro danou podmínku) u mnoha věcí jako např.:

- výšku a šířku viewportu
- výšku a šířku obrazovky zařízení
- orientaci (tj. jestli je mobil či tablet pracuje v režimu portrait nebo landscape)
- rozlišení

Využívání těchto možností je velmi oblíbené a důležité při stylování webu podle konceptu responzivního designu. Je totiž díky tomu možné nadefinovat tzv. breakpointy. Breakpointy v praxi fungují tak, že stanovují jak má být ten či onen prvek při dané podmínce (Media Query) nastýlován.

Anatomie Media Query je složena z typu média (media type, kde výchozí hodnotou je „all“, tedy, že se daná query týká všech zařízení) a podmínky obsahující vlastnosti média (media features). U media feature se obvykle vlastnost (např. max-width) a její definovaná hodnota či rozmezí více hodnot.



Obrázek: Anatomie Media Query<sup>26</sup>

<sup>26</sup> CSS3 Media Queries [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/css3-media-queries>

V praxi tak můžeme nadefinovat, že nadpis h1 má velikost fontu 2 em. Ale při maximální šířce obrazovky do 40 em má mít tento element velikost pouze 1.5 em. Zápis takového pravidla bude vypadat následovně:

```
h1 { font-size: 2em }

@media only screen and (max-width: 40em) {
  h1 { font-size: 1.5em }
}
```

**Obrázek: Media Query** <sup>27</sup>

### 3.2.1.2. Plovoucí struktury (Fluid Grids)

„Umístovat jednotlivé prvky designu v rámci nějaké mřížky je neobyčejně oblíbená praktika, která je o mnoho dekád starší než web samotný. Mřížky pomáhají docílit nejenom vyváženosti správných rozestupů, ale také přehledného uspořádání prvků na stránce. Dobře implementovaný mřížkový systém (grid system) způsobuje, že web vypadá méně chaoticky, je lépe čitelný a lépe se také prohledává.“ <sup>28</sup>

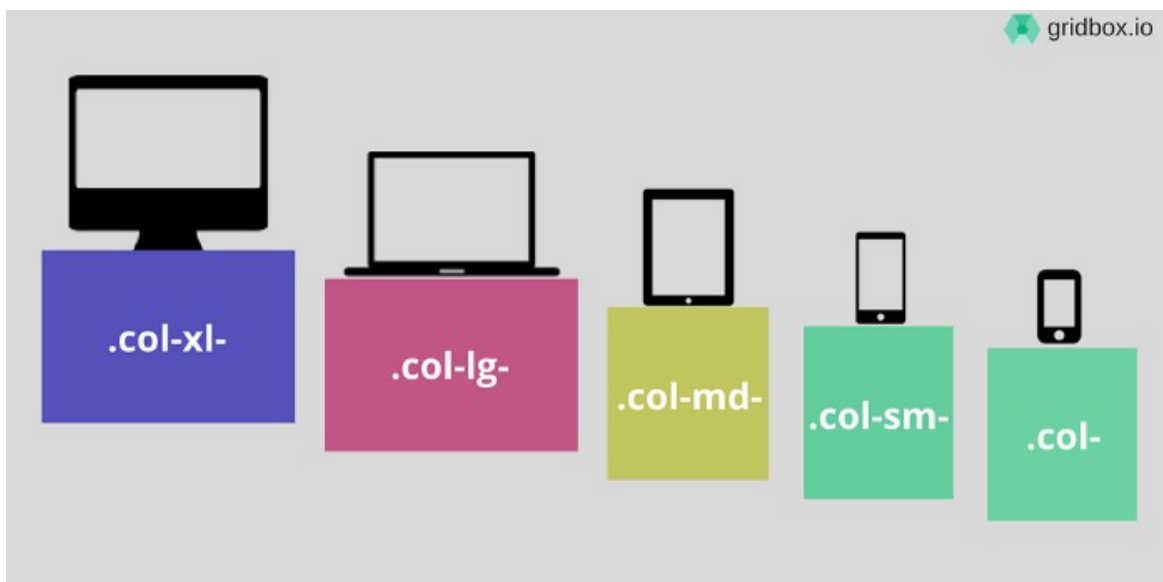
Další důležitou částí responzivního designu je tedy tzv. Fluid Grid. Koncept Fluid Gridu je postaven na myšlence, že se obrazovka dané webové stránky rozdělí na sloupce a řádky. Sloupců, na které se může obrazovka rozdělit je obvykle 12 a důležité je, že šířka těchto sloupců musí být tzv. fluidní. To znamená, že se musí automaticky přizpůsobovat šířce obrazovky, na které se příslušná webová stránka zobrazuje.

Myšlenka tohoto sloupcového layoutu je taková, že pokud budu mít např. Webovou stránku formulářem, tak:

- Na mobilu se formulářové pole zobrazí přes celou šířku obrazovky (tj. využije se k tomu všech 12 sloupců).
- Na tabletu, kde je obrazovka větší, tak už můžeme to samé pole zobrazit jen na určitou část obrazovky a vedle něj třeba dát další pole (tj. každé formulářové bude zabírat polovinu šířky obrazovky, tzn. 6 sloupců).
- Na desktopu je obrazovka ještě větší, takže je možné například zobrazit každé pole vedle sebe jen na třetinu stránky (tj. každé zabere 4 sloupce).

<sup>27</sup> CSS3 Media Queries [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/css3-media-queries>

<sup>28</sup> KADLEC, Tim. Responzivní design profesionálně. 1. vydání. Brno: Zoner Press, 2014. s. 42



**Obrázek: Mřížka (Grid System) na různých zařízeních** <sup>29</sup>

COL-3				COL-3				COL-3				COL-3					
COL-4						COL-4						COL-4					
COL-6								COL-6									
COL-2			COL-2			COL-2			COL-2			COL-2			COL-2		
COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1	COL-1		

**Obrázek: Fluid Grid pro různá zařízení** <sup>30</sup>

Implementace sloupcového layoutu úzce souvisí s mediálními dotazy (Media Queries), viz. předchozí kapitola. Šířky jednotlivých sloupců jsou totiž nadefinovány v CSS jako třídy a jsou zpravidla zapouzdřené v mediálním dotazu (Media Query) s příslušným breakpointem. Tyto třídy pak mají specifický systém jejich pojmenování a tak přiřazují se na stránce k jednotlivým HTML elementům tak, jak je potřeba.

Systém pojmenování tříd a rozdělení má v aplikaci obvykle na starost nějaký UI framework. Nejpoužívanějším UI frameworkům je podrobně věnována kapitola 4.3.

Obecně je v názvu této třídy nějak zakomponován breakpoint, kterého se daná třída týká a velikost sloupce.

Takže například:

- CSS třída sloupce pro malá zařízení (mobily) o velikosti 6 sloupců je v Bootstrapu pojmenována „col-sm-6“.

<sup>29</sup> Bootstrap 4 vs Foundation 6 Grid System [online]. Dostupné z: <https://medium.com/gridbox/bootstrap-4-vs-foundation-6-grid-system-5874e0e87a95>

<sup>30</sup> Bootstrap 4 vs Foundation 6 Grid System [online]. Dostupné z: <https://medium.com/gridbox/bootstrap-4-vs-foundation-6-grid-system-5874e0e87a95>

- CSS třída sloupce pro velký zařízení (desktopy) o velikosti 4 sloupců je v Bootstrapu pojmenování „col-lg-4“.

Takto vypadá nejčastěji používaný koncept plovoucí struktury (Fluid Grid), který se využívá při návrhu responzivního designu.

Možností, jak navrhnout plovoucí strukturu (Fluid Grid) je však více:<sup>31</sup>

- Využití zbytku – nadefinuje se pevná velikost jednoho či více sloupců a zbylé místo na obrazovce se věnuje sloupci, který automaticky dopočítá svou šířku.
- Využití procent – přesně se nadefinuje procentuální šířka sloupců podle potřeb daného layoutu.

### 3.2.1.3. Škálovatelné obrázky (Scalable Images)

Dalším velmi důležitým aspektem při budování kvalitního responzivního webu jsou škálovatelné obrázky (resp. obecně responzivní média jako taková). Zde největší problém spočívá v tom, že různá zařízení, pro které se daný responzivní web vyvíjí, mají zpravidla odlišné možnosti výpočetního výkonu a rychlosti připojení k internetu. Nejlépe v obou těchto ohledech většinou vychází desktopy, nejhůře naopak mobilní telefony. Tablety jsou něco mezi tím – z hlediska výpočetního výkonu bývají rychlejší než mobily, ale k internetu se pomocí tabletů obvykle připojujeme stejně jako u mobilů.

Při načítání každého webu obvykle je zpravidla z hlediska výkonnosti nejsložitější načíst bitmapové obrázky, webové fonty a případně videa, pokud na webu nějaká jsou. Taktéž mohou obrázky na malém displeji mobilů působit poněkud přepláceně a rušit uživatele od čtení důležitých částí webového obsahu.

Z těchto dvou důvodů je velmi žádoucí obrázky na ve verzi pro malá zařízení nějakým způsobem omezit – a to buď je vůbec nenačítat a nebo alespoň zmenšit jejich velikost.

Pro řešení prvního případu (vůbec obrázky nenačítat) se logicky nabízí nastavit jim v CSS vlastnost `display: none`. „Řešení CSS a `display: none` není životaschopné. Sice skryje obrázky, takže nejsou na webové stránce vůbec vidět, přesto ale vytváří HTTP požadavky na server a stahují se do prohlížeče. Pokud chcete, aby se obrázky zobrazovaly jen nad nějakým konkrétním breakpointem, bude lepší, když je budete načítat podmíněně a až poté, co se webová stránka načte.“<sup>32</sup>

<sup>31</sup> W3.CSS Responsive fluid Grid [online]. Dostupné z: [https://www.w3schools.com/w3css/w3css\\_grid.asp](https://www.w3schools.com/w3css/w3css_grid.asp)

<sup>32</sup> KADLEC, Tim. Responzivní design profesionálně. 1. vydání. Brno: Zoner Press, 2014. s. 117

U obrázků, které lze v mobilní verzi postrádat, je tedy doporučeno z HTML šablony úplně odstranit načítání obrázků pomocí klasického tagu `<img src="" />` a místo toho dané obrázky načítat prostřednictvím datového atributu `data-src` z HTML5, který se následně zpracuje v JavaScriptu.

**Obrázek 1: Ukázka využití datového atributu `data-src` pro načtení obrázků**

```
1 <ul class="slats">
2   <li data-src="images/ball.jpg" class="group">
3     <a href="#">
4       <h3>Kicker connects on record 13 field goals</h3>
5     </a>
6   </li>
7   <li data-src="images/goal_post.jpg" class="group">
8     <a href="#">
9       <h3>Your favorite team loses to that team no one likes</h3>
10    </a>
11  </li>
12  <li data-src="images/ball_field.jpg" class="group">
13    <a href="#">
14      <h3>The Scarecrows Win 42-0</h3>
15    </a>
16  </li>
17 </ul>
```

**Zdroj: Responzivní design, str. 95**

Následně je pak potřeba JavaScriptu vybrat všechny HTML elementy, které obsahují atribut `data-src` a u nich vytvořit obrázek prostřednictvím instance z třídy `Image()`.

**Obrázek 2: Ukázka načtení obrázku z `data-src` pomocí JavaScriptu**

```
1 if (window.matchMedia( query: "(min-width: 37.5em)").matches) {
2   // načte obrázky
3   var lazy = Utils.q('[data-src]');
4   for (var i = 0; i < lazy.length; i++) {
5     var source = lazy[i].getAttribute('data-src');
6     // vytvoří obrázek
7     var img = new Image();
8     img.src = source;
9     // vloží ho do odkazu
10    lazy[i].insertBefore(img, lazy[i].firstChild);
11  }
12 }
```

**Zdroj: Responzivní design, str. 96**

Metoda `matchMedia` v JavaScriptu slouží k tomu, aby se obrázky mohly načíst podmíněně podle zadané Media Query. V tomto případě se načtou, pokud obrazovka má minimální šířku 37.5em.

Tímto způsobem lze analogicky načítat různé velké obrázky pro různé breakpointy a tím se v případě malých zařízení docílíme toho, že pokud se budou obrázky načítat, tak se budou načítat v menším rozlišení, než na větších zařízeních s lepším výkonem a internetovým připojením.

Co se týče videí, tak tady je nejlepší vyřešit situaci tak, že pro velká zařízení se bude video na dané stránce automaticky načítat. A na menších zařízeních se naopak místo samotného videa vytvoří odkaz, na který když uživatel klikne, tak se otevře podstránka a až tam se příslušné video načte.

### **3.2.2. Princip Mobile First**

V předchozích 3 kapitolách bylo popsáno, že návrh responzivního designu má svá specifika pro různé typy zařízení. Ta specifika spočívají především ve výkonnosti jednotlivých zařízení (jak procesoru, tak internetového připojení). Mobily disponují zpravidla nejnižším výkonem a desktopy naopak nejvyšším. To v praxi znamená, že by responzivní verze pro mobily měla být mnohem více “osekaná“, než verze pro tablety či dokonce desktopy.

Z těchto důvodů je při vývoji responzivního webu velmi doporučováno začít mobilní verzi, zde naimplementovat ty nejdůležitější prvky, který musí být na daném webu k dispozici i pro uživatele, kteří se na web budou dívat na mobilu. A následně nad touto verzí přidávat další prvky, jejichž vizuální podoba či dokonce jejich samotné zobrazení už bude podmíněně nějaký breakpointem (logicky vyšším, než jaká je velikost displeje na mobilu).

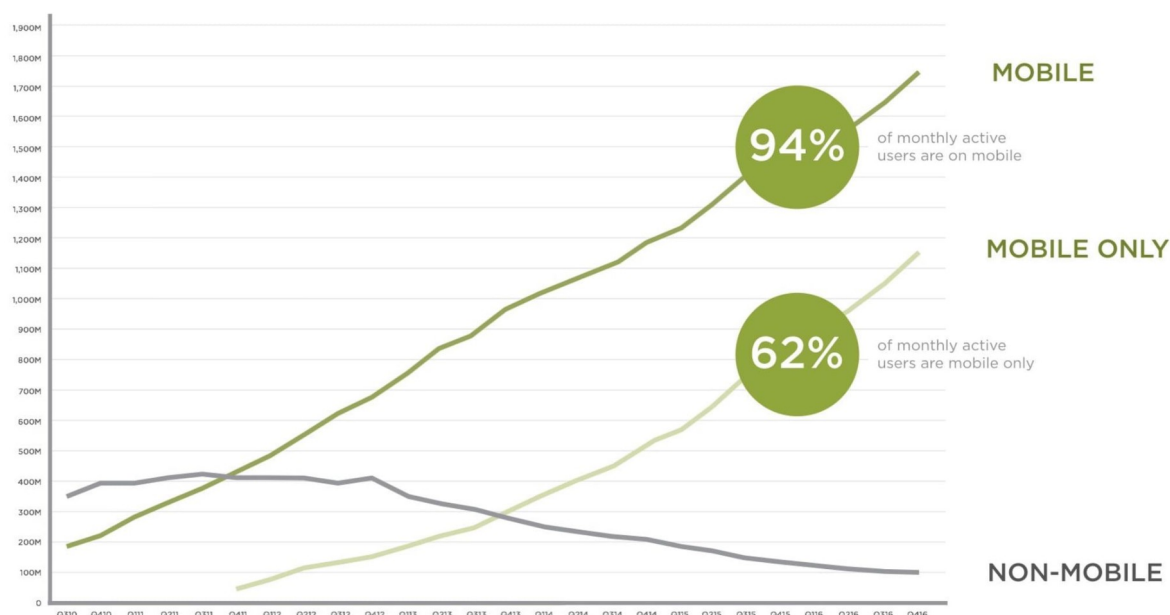
Tento postup, kdy se začíná na responzivní web navrhovat pro mobily se nazývá jako „Mobile First“.

„Mobile First je způsob návrhu uživatelského rozhraní, který z pohledu důležitosti staví mobilní zařízení minimálně na úroveň tradičních počítačů s velkými obrazovkami. Kromě reflexe nástupu smartphonů na trh má Mobile First ještě jeden – daleko zajímavější – vedlejší účinek. Učí nás navrhovat jednodušší a myslím že i lepší rozhraní. Autor myšlenky, Luke Wroblewski, ji definoval asi takto:

Designéři, navrhujte nejprve pro mobily. Prudce se šíří mezi uživateli. Nutí zaměřit pozornost na to nejdůležitější. A rozšiřují naše možnosti.“<sup>33</sup>

Důležitost toho přístupu spočívá především ve faktu, že mobily opravdu i podle statistiky návštěvnosti webů bývají mnohdy používanější, než desktop.

Luke Wroblewski zveřejnil statistiku, že již v roce 2016 přistupovala na Facebook 94 % uživatelů přes mobilní zařízení. A 62 % jich dokonce na Facebook přistupovalo pouze prostřednictvím mobilního zařízení.



Obrázek: Návštěvnost Facebook z různých zařízení v roce 2016<sup>34</sup>

Princip Mobile First v praxi znamená, že se doporučuje nastýlovat verzi pro mobily bez podmínění jakýmikoliv Media Querys a jejich příslušnými breakpointy. A za těmito vlastně „defaultními“ styly teprve udělat Media Queries s příslušnými breakpointy. A do těchto Media Queries následně nastýlovat design pro zařízení s větším displejem.

Tento přístup podporují i nejpoužívanější UI frameworky. Např. Bootstrap doporučuje sloupcový layout stylovat s využitím tříd „col-xx“ jako „defaultní“ verzi, která se bude zobrazovat na mobilech. A pokud budeme třeba v desktopové verzi chtít udělat udělat sloupcový layout jinak, tak k této třídě „col-xx“ přidáme ještě třídu „col-lg-xx“ jako nadstavbu nad původním layoutem.<sup>35</sup>

Z těchto poznatků je zřejmé, že se při návrhu responzivních webů velmi hodí využívat zmiňovaný princip Mobile First. V praxi je to však poněkud složitější. Neboť

<sup>33</sup> Co je „Mobile First“? Ale doopravdy [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/mobile-first>

<sup>34</sup> Co je „Mobile First“? Ale doopravdy [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/mobile-first>

<sup>35</sup> Symbol „xx“ je proměnná, za kterou se dosadí šířka sloupce z intervalu 1 až 12 dle Grid System Bootstrapu.

pokud se při vývoji nějakého projektu používá tento princip, tak to znamená, že tímto způsobem musí uvažovat nejen Front-end vývojář, ale téměř všichni členové vývojového týmu – minimálně ještě UI/UX designer, SCRUM Master, Business analytik a Product Owner. A to důvodu, že Front-end vývojář může nakódovat kvalitní responzivní design s dodržováním principu Mobile First jen tehdy, pokud k tomu má podporu v týmu a pokud k tomu dostane od UI/UX designera patřičné podklady. U návrhů UI/UX se totiž v takovém případě musí s principem Mobile First počítat a začínat UX návrhy navrhovat také od mobilu a skončit u desktopu.

Autor této práce může ze svojí vlastní komeční praxe potvrdit, že toto bývá mnohdy problém. Neboť zvláště pokud je daný vývojový tým složen buď z méně zkušených členů a nebo jestliže je tým složen členů pocházejících z různých firem (z případě IT Bodyshoppingu), tak se mnohdy web začíná od desktopu a až časem (třeba když do týmu přijde nový člen s dostatečně seniorními znalostmi) se zjišťuje, že tento postup není správný.

Ostatně i přední odborníci na tuto problematiku Martin Michálek a Luke Wroblewski říkají, ať se hlavně responzivní weby nenavrhují podle opačného principu „Desktop First“.

„Uvědomme si, že Wroblewski s myšlenkou přišel v roce 2009, jen dva roky po uvedení prvního iPhone. Zvolání „Mobile First!“ vzniklo jako reakce na v té době převládající postup. Weby se navrhovaly jen pro velké displeje. Postupem zvaným „Desktop First“.

Mobilní rozhraní se pak vymýšlelo až v implementační fázi, nebo dokonce až po implementaci rozhraní pro velké displeje. Nastávaly ohromné vývojářské i designéřské potíže. Ve výsledku pak velké kompromisy v uživatelském rozhraní na mobilech. Že je téma stále žhavé, ukazuje fakt, že v mnoha českých webařských týmech jde stále o aktuální způsob práce. To bolí!“<sup>36</sup>

### 3.2.3. UI frameworky

UI frameworky jsou frameworky založené zpravidla na CSS. Slouží k výraznému usnadnění návrhu webů podle pravidel responzivního designu, které byly popsány v předchozích kapitolách. Kvalitní UI framework obvykle nabízí vývojářům minimálně tyto možnosti:

- Mřížku (Grid System)
- Zlomové body (Breakpoint)

---

<sup>36</sup> Co je „Mobile First“? Ale doopravdy [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/mobile-first>



- Škálovatelné obrázky (Scalable Images)
- Podporu principu Mobile First
- Podporu CSS preprocesorů
- Vlastní ovládací prvky (tlačítka, textová pole, navigace, carousely atd.) podporující responzivní design
- Možnost customizace
- Širokou podporu různých webových prohlížečů

Hojně používané jsou v současnosti UI frameworky Twitter Bootstrap <sup>37</sup>, ZURB Foundation <sup>38</sup> a Skeleton <sup>39</sup>. Možnosti těchto frameworků jsou detailně analyzovány ve vícekriteriální analýze variant, která je vypracována v kapitole 4.3 této práce.

### 3.3. Model vícekriteriální analýzy variant

V metodice této práce je uvedeno, že u jednotlivých JavaScript a UI frameworků budou analyzovány jejich možnosti prostřednictvím vícekriteriální analýzy variant. Vícekriteriální analýza v tomto případě slouží k tomu, aby bylo možné určit nejlépe hodnocený JavaScript a UI framework. A to prostřednictvím objektivního exaktního srovnání na základě kritérií, jež jsou v případě této práce stanoveny v metodice.

Vybrané varianty a příslušná kritéria lze zapsat do tzv. Kritériální matice Y, kde prvek  $y_{ij}$  vyjadřuje hodnocení i-té varianty podle j-tého kritéria. <sup>40</sup>

Obrázek 3: Kritériální matice

$$\begin{matrix} & \mathbf{f}_1 & \mathbf{f}_2 & \dots & \mathbf{f}_k \\ \mathbf{a}_1 & \left( \begin{matrix} y_{11} & y_{12} & \dots & y_{1k} \end{matrix} \right) \\ \mathbf{a}_2 & \left( \begin{matrix} y_{21} & y_{22} & \dots & y_{2k} \end{matrix} \right) \\ \vdots & \left( \begin{matrix} \dots & \dots & \dots & \dots \end{matrix} \right) \\ \mathbf{a}_p & \left( \begin{matrix} y_{p1} & y_{p2} & \dots & y_{pk} \end{matrix} \right) \end{matrix}$$

Zdroj: Šubrt

Kritéria se rozdělují na více skupin.

**Podle povahy mohou být kritéria:**

<sup>37</sup> Bootstrap [online]. Dostupné z: <https://getbootstrap.com/>

<sup>38</sup> Foundation [online]. Dostupné z: <https://get.foundation/>

<sup>39</sup> Skeleton [online]. Dostupné z: <http://getskeleton.com/>

<sup>40</sup> ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. s. 162

1. Maximalizační – při rozhodování se vychází z toho, že nejlepší varianty podle tohoto kritéria mají nejvyšší hodnoty.
2. Minimalizační – opak maximalizačního kritéria, nejlepší varianty mají naopak nejnižší hodnoty podle tohoto kritéria.<sup>41</sup>

#### **Dle kvantifikovatelnosti mohou být:**

1. Kritéria kvantitativní – hodnoty variant podle takových kritérií tvoří objektivně měřitelné údaje. Protože se tato kritéria také někdy nazývají jako objektivní.
2. Kritéria kvalitativní – hodnoty variant podle těchto kritérií nelze objektivně změřit a jejich hodnoty jsou tak mnohem více subjektivní. Velmi často jde o hodnoty subjektivně odhadnuté uživatelem (subjektivní kritéria). V těchto případech se používají různé bodovací stupnice nebo relativní hodnocení variant.

Pro řešení problému je velmi důležité, že a jak je některé kritérium preferováno před jiným.<sup>42</sup> To je zajištěnou váhou kritéria, která je obecně hodnota z intervalu  $(0; 1)$  a vyjadřuje relativní důležitost tohoto kritéria v porovnání s ostatními kritérii. Součet vah všech kritérií je roven jedné. Váhy kritérií lze stanovit také např. Saatyho metodou.

#### **3.3.1. Saatyho metoda<sup>43</sup>**

Saatyho metoda kvantitativního párového porovnání kritérií. Pro ohodnocení párových porovnání kritérií se používá devítibodová stupnice a je možné používat i mezistupně (hodnoty 2, 4, 6, 8):

- 1 – rovnocenná kritéria i a j
- 3 – slabě preferované kritérium i před j
- 5 – silně preferované kritérium i před j
- 7 – velmi silně preferované kritérium i před j
- 9 – absolutně preferované kritérium i před j

Každá dvojice kritérií se porovná a míra preference i-tého kritéria před j-tým kritériem se zapíše to tzv. Saatyho matice  $S = (s_{ij})$ .

<sup>41</sup> ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. s. 163

<sup>42</sup> ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. s. 164

<sup>43</sup> ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. s. 174

**Obrázek 4: Saatyho matice**

$$\begin{matrix} & f_1 & f_2 & \dots & f_k \\ f_1 & \begin{bmatrix} 1 & s_{12} & \dots & s_{1k} \end{bmatrix} \\ f_2 & \begin{bmatrix} 1/s_{12} & 1 & \dots & s_{2k} \end{bmatrix} \\ \vdots & \vdots \\ f_k & \begin{bmatrix} 1/s_{1k} & 1/s_{2k} & \dots & 1 \end{bmatrix} \end{matrix}$$

**Zdroj: Šubrt, str. 175**

Saatyho matice je čtvercová a reciproká, tj. platí, že  $s_{ij} = 1 / s_{ji}$ . To v praxi znamená, že se u preferovaného kritéria řádku  $i$  před sloupcem  $j$  nad hlavní diagonálou zapisují celá čísla v rozmezí 1 až 9. A pod hlavní diagonálu se naopak do nepreferovaných kritérií zapisují převrácené hodnoty, např.  $1/9$ ,  $1/7$  atd. Pod hlavní diagonále Saatyho matice jsou vždy hodnoty 1, protože jedno konkrétní kritérium nemůže být preferováno před sebou samým.

Váhy kritérií  $v_j$  lze podle Saatyho odhadnout pomocí normalizovaného geometrického průměru řádků  $b_i$  Saatyho matice (metoda logaritmických nejmenších čtverců).

$$b_i = \sqrt[n]{\prod_{j=1}^n s_{ij}}$$

Samotné váhy  $v_i$  se pak vypočítají normalizací hodnot  $b_i$ .

$$v_i = \frac{b_i}{\sum_{i=1}^n b_i}$$

### 3.3.2. Analýza metodou váženého součtu

„Metoda váženého součtu vyžaduje kardinální informace, kritériální matici  $Y$ , a vektor vah kritérií  $v$ . Konstruuje celkové hodnocení pro každou variantu, a tak ji lze použít jak pro hledání jedné nejvýhodnější varianty, tak pro uspořádání variant od nejlepší po nejhorší.“<sup>44</sup>

Algoritmus pro výpočet metodou váženého součtu má 2 kroky, konkrétně:

<sup>44</sup> ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. s. 186

1. Určení ideální varianty H s ohodnocením  $(h_i, \dots, h_n)$  a bazální varianty D s ohodnocením  $(d_i, \dots, d_n)$
2. Vytvoření standardní kritériální matice R, jejíž prvky se získají pomocí vzorce:

$$r_{ij} = \frac{y_{ij} - d_j}{h_j - d_j}$$

## 4. Vlastní práce

### 4.1. Analýza požadavků

Funkční a nefunkční požadavky na aplikaci HealthCare-App byly analyzovány z důvodu lepšího popisu a pochopení toho, co má být implementováno.

#### 4.1.1. Funkční požadavky

**Tabulka 1: Funkční požadavky**

Komponenta	Funkce	Popis
Login (Přihlášení)	Přihlášení se do systému	Zadání e-mailu a hesla, které je v systému registrováno
Profile (Registrace)	Registrace uživatele do systému	Zadání e-mailu, jména, příjmení, data narození a telefonu
Overview (Homepage)		Přivítání uživatele v aplikaci a možnost vstoupit do dotazníku
Questionare (Dotazník)	Zadání nového stavu	V dotazníku je sada připravených otázek. Uživatel na ne postupně krok za krokem odpovídá. Odpověď je možná pomocí checkboxů a radiobuttonů. U otázek mohou být ikony (koláč, šipky s trendem) a případně ilustrativní obrázky. Otázky jsou rozděleny do 8 kategorií a pozadí v dotazníku má odlišné barvy (v závislosti na kategorii dané otázky).
Statistics (Statistiky)	Zobrazují vývoj a pokrok v léčbě prostřednictvím různých typů grafů.	
	Pavučinový teorém	Zobrazují se jednotlivé kategorie a v nich součet všech otázek jednotlivých kategorií.
	Trendový graf	Graf součtu všech příznaků jdoucích v čase.
Medication (Medikace)	Seznam medikamentů	Zobrazuje seznam medikace a jejich dávkování, které pacient má brát.
AddMedication	Přidat medikaci	Přidá novou medikaci. Zadat u ní lze: název, typ dávkování (ml, mg, kapek, tablet), množství dávkování a periodicitu dávkování.

**Zdroj: vlastní zpracování**

#### 4.1.2. Nefunkční požadavky

Nefunkční požadavky definují především požadavky týkající se kvality výsledné ukázkové aplikace a jsou rozděleny na požadavky z oblasti:

1. architektury
2. frameworků a integrovaných nástrojů

##### **Požadavky z oblasti architektury:**

- Implementovat aplikaci podle konceptu Single-Page aplikace
- Implementovat i18n pro snadnou úpravu jazykových mutací bez nutnosti výrazných zásahů v HTML šablonách a výkonné logice aplikace
- Navrhnout responzivní design podle principu Mobile First

##### **Požadavky z oblasti frameworků a integrovaných nástrojů:**

- Využít nějaký moderní JavaScript framework
- Využít nějaký moderní UI framework
- Napsat ukázkou unit testů s využitím testovacího nástroje, který podporuje pro aplikaci zvolený JavaScript framework

## 4.2. Porovnání JavaScript frameworků

Jak již bylo řečeno na v kapitole 2.1, tak důležitou součástí této práce je porovnání možností, jaké nabízí jednotlivé JavaScript frameworky a výběr toho, který nejlépe odpovídá na požadavky ukázkové aplikace HealthCare-App.

Porovnávány mezi sebou budou tyto frameworky: Angular, React, Vue, Ember a Knockout.

Konkrétně tyto frameworky byly pro analýzu vybrány z těchto důvodů:

- Všechny nabízí nějakou možnost pro vytváření vlastních UI komponent.
- Všechny jsou licencovány jsou open-source pod MIT licenci, která je jednou v nejbenevolentnějších licencí.<sup>45</sup>
- Všechny v současnosti mohou být použity ve spojení s TypeScriptem.
- Všechny obsahují nějaké konfigurační nástroje, jež umožňují individuální konfiguraci frameworku pro potřeby konkrétního projektu.

Výběr vhodného frameworku byl proveden ve 2 sobě jdoucích fázích, kde v první fázi byly z výběru vyloučeny ty frameworky, který mají velmi nízkou podporu a aktivitu vývojářské komunity, jež za daným frameworkem stojí. Tato aktivita komunity lze snadno zjistit z GitHubu podle počtu contributorů a množství commitů v určitém časovém období. Byla provedena analýza aktivity u frameworků za poslední měsíc a rok. Tato analýza má za cíl z výběru vyřadit ty frameworky, které jsou velmi málo rozvíjeny či podporovány.

Druhá fáze analýzy se týká zhodnocení z hlediska kvality a kompatibility s ohledem na požadavky aplikace těch frameworků, které ve výběru zůstaly i po první fázi. Zde konkrétně jde o zhodnocení vhodnosti použití zbývajících frameworků na základě provedené vícekritériální analýzy variant na základě předem stanovených kritérií a použitím metody váženého součtu.

---

<sup>45</sup> Obecné závěry k licenčním podmínkám MIT [online]. Dostupné z: <https://www.root.cz/specialy/licence/obecne-zavery-k-licencnim-podminkam-mit/>

#### 4.2.1. Fáze č. 1 - Výběr podle aktivity vývojářské komunity

Velmi důležitým aspektem při výběru frameworku pro využití na projektu je aktivita vývojářské komunity či firmy, která za daným frameworkem stojí. Aktivitou se rozumí, kolik lidí v komunitě či firmě se stará o rozvoj konkrétního frameworku a jak intenzivně se frameworku tito lidé věnují – např. jak často vydávají nové verze, fixují chyby v předchozích verzích, jak se věnují tvorbě dokumentace atd.

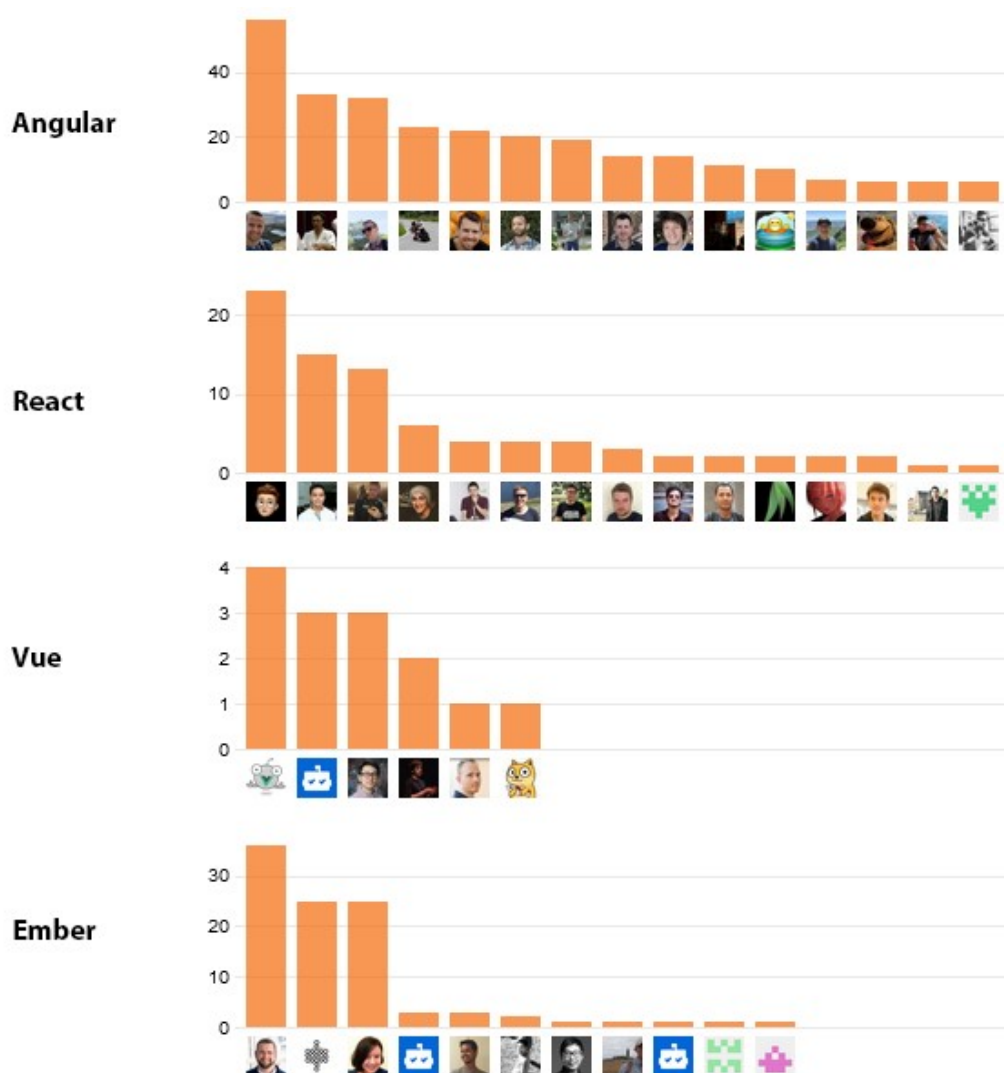
To je důležité především z důvodu, že čím více se firma/komunita frameworku věnuje, tak tím je vyšší pravděpodobnost, že budou ve frameworku včas odhalovány případné chyby či různé nedodělky a také s tím logicky souvisí fakt, že čím více se frameworku komunita věnuje, tak tím více vývojářů příslušný framework zná a umí v něm vyvíjet. To je velmi důležité, neboť v průběhu klasického životního cyklu projektu se obvykle na projektu vystřídá více různých vývojářů a pokud je projekt napsán nad frameworkem, který není ze strany vývojářů příliš známý, tak to obvykle vede k tomu, že po určitém čase je jakákoliv úprava na projektu časově i finančně náročná, protože se kód projektu již nikdo z vývojářů pořádně nevyzná. A takový stav obvykle vede k tomu, že se projekt začne přepisovat do nějakého z tohoto pohledu lepšího řešení. Přepis obvykle stojí spoustu peněz, a proto je opravdu dobré na začátku projektu vybrat framework, který je všemi zúčastněnými stranami co nejvíce podporován.

Tím, že jsou všechny frameworky pod MIT licencí, tak jsou k dispozici volně ke stažení na GitHubu. Proto v případě výběru frameworku podle aktivity vývojářské komunity se lze snadno podívat to statistik GitHubu, kolik lidí na daném frameworku za určité období pracovalo a kolik vytvořilo commitů.

Ze statistik GitHubu je patrné, že poslední měsíc (viz. Obrázek 3) mají Angular, React a Ember stabilně podporu komunity minimálně 5x vyšší, než Vue. Naopak Vue má pouze pár contributorů (autora frameworku a k tomu 5 lidí) a jejich aktivita je v průběhu celého období v podstatě sporadická. U Knockoutu není za poslední měsíc žádná aktivita.



**Obrázek 5: Frameworky – aktivita za poslední měsíc**



**Zdroj: GitHub**

U statistik za uplynulý rok je aktivita u jednotlivých frameworků zhruba podobná jako u statistik za poslední měsíc.

**Obrázek 6: Frameworky – aktivita za poslední rok**



**Zdroj: GitHub**

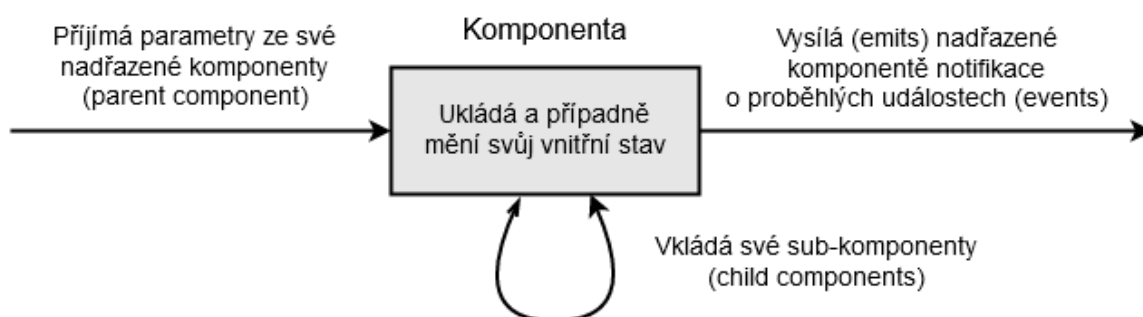
Toto jsou dostatečně pádné důvody k tomu, proč v dalším kroku metodiky výběru vhodného frameworku lze uvažovat již jen o Angularu, Reactu nebo Emberu. A mezi Angularem, Reactem a Emberem bude z metodického hlediska proveden výběr již na základně vhodnější architektury frameworku.

#### 4.2.2. Fáze č. 2 – Zhodnocení kvality a kompatibility zbylých frameworků

Po fázi č. 1 byly z výběru vyloučeny frameworky Vue a Knockout, neboť mají v současnosti jen velmi malou podporu vývojářské komunity. Naopak ve výběru stále zůstaly frameworky Angular, React a Ember. Ve fázi č. 2 jsou tyto frameworky analyzovány a zhodnoceny prostřednictvím vícekriteriální analýzy variant. Vícekriteriální analýza variant začíná stanovením kritérií pro hodnocení, kterým jsou následně přiděleny váhy pomocí Saatyho metody. Poté je u vybraných frameworků provedena samotná vícekriteriální analýza variant, ze které se nakonec vybere kompromisní varianta. A to pomocí metody váženého součtu.

Architektura a filozofie toho, jak se v aplikaci skládají UI komponenty je u všech těchto frameworků podobná. Všechny frameworky mají komponenty, které obsahují sub-komponenty. Sub-komponenty mohou z parent komponent přijímat vstupní data nebo naopak ze sebe vysílat notifikace o proběhnutých událostech do parent komponent. Zpravidla je možné mezi parent a child komponentami posílat jak data, tak i volat metody. Způsob implementace této výměny dat mezi komponentami je však u každého frameworku mírně odlišný.

**Obrázek: Obecná schéma komponenty**



**Zdroj: vlastní zpracování**

Uvnitř komponenty jsou standardní HTML tagy. A případně, že je potřeba zavolat nějakou komponentu, tak je také obvykle volána ve formátu vlastního HTML tagu, do kterého jsou příslušné komponentě případně předány nějaké vstupní parametry.

Volba kritérií pro hodnocení daného frameworku pokrývá kvalitu a kompatibilitu frameworku s ohledem na funkční a nefunkční požadavky aplikace. Nutno podotknout, že

samotná volba kritérií, kterými je framework hodnocen je do určité míry subjektivní. A vychází z vývojářských zkušeností autora této práce.

Následující tabulka znázorňuje, jaká kritéria byla zvolena pro provedení vícekritériální analýzy variant:

**Tabulka 2: Kritéria pro vícekritériální analýzu JavaScript frameworků**

Číslo	Kritérium	Jednotky
1	Velikost frameworku	MB
2	Architektura a API	Bodové hodnocení
3	Nástroje	
4	Komponenty	

**Zdroj: vlastní zpracování**

Kritérium č. 1 (Velikost frameworku) je kvantitativní a má minimalizační povahu. Data pro hodnocení variant u tohoto kritéria jsou získávány z naklonováním (git clone) jednotlivých frameworků z GitHubu.

Ostatní kritéria jsou naopak kvalitativní a mají maximalizační povahu. Data pro hodnocení variant u těchto kritérií jsou získávány do značné míry subjektivně a to na základě vývojářských zkušeností autora této práce a studia dokumentace jednotlivých frameworků a požadavků na výslednou aplikaci. Bodové ohodnocení variant zde může nabývat číselných hodnot od 1 do 10, kde 1 je nejhorší a 10 nejlepší.

### **Velikost frameworku**

Velikost konkrétního frameworku je důležitá především pro rychlost načítání výsledné webové aplikace. Neboť logicky platí, že čím je framework větší, tím se pomaleji načítá. Velikost daného frameworku je stanovena jako velikost, kterou frameworku zabírá po naklonování příslušného repozitáře z GitHubu.

### **Architektura a API**

Architektura a API představuje souhrnné kritérium, ve kterém je diskutováno, jak je vůbec v konkrétním frameworku navržena základní struktura, nad je vyvíjena aplikace. Zde je důležité vzít v úvahu především tyto aspekty:

- Komunikace frameworku s Back-endem
- Způsob vkládání (inject) služeb (service) do komponent
- Jak jsou (pokud vůbec) odděleny HTML šablony od výkonné logiky aplikace

- Jak je daný framework striktní či naopak volnější z hlediska dodržování definované architektury při vývoji aplikace.

### **Nástroje**

Nástroje představují všechny možné integrované nástroje, které vývojářům usnadňují práci s frameworkem. Nebo různé zásuvné moduly, jež mohou doplnit základní funkcionalitu daného frameworku. Frameworky jsou zpravidla vybaveny nástrojem CLI, který umožňuje lepší správu jednotlivých částí vyvíjené aplikace - např. vytvořit projekt, komponentu, službu (service) atd.

### **Komponenty**

Komponenty v analýze řeší, jakým způsobem daný framework umožňuje pracovat s komponentami. A to ve smyslu jejich zanořování ve stromové struktuře nebo předávání dat či informací o vyslaných notifikacích mezi komponentami. Také je u komponent důležitá, jak je vyřešena v HTML šablonách syntaxe pro komunikaci mezi HTML šablonami a výkonnou logikou aplikace.

#### **4.2.2.1. Angular**

O Angular <sup>46</sup> se stará vývojářská komunita zastřešovaná společností Google. První verze AngularJS byla vydána již v roce 2010. AngularJS je však jako framework velmi rozdílný oproti všem verzím Angular 2+. Angular verze 2 byl po dlouhém očekávání vydán Googlem v roce 2016 a jakékoliv následující verze Angularu jsou již podobné této verzi 2. V době psaní této práce byla poslední verzí verze 10.

### **Velikost frameworku**

Velikost Angularu v základní konfiguraci je 335 MB.

### **Architektura a API**

Architektura Angularu je navržena tak, že jsou poměrně striktně od sebe oddělené moduly, komponenty a služby. Moduly slouží pro import všech důležitých částí aplikace (tj. především komponenty a služby). Pokud chceme v nějaké komponentě zavolat službu, tak se musí v nainjectovat v konstruktoru dané komponenty pomocí Dependency Injection

<sup>47</sup>.

---

<sup>46</sup> Angular [online]. Dostupné z: <https://angular.io/>

<sup>47</sup> Dependency Injection in Angular [online]. Dostupné z: <https://angular.io/guide/dependency-injection>

API má v sobě rovněž funkcionalitu pro routování mezi jednotlivými komponentami prostřednictvím různých URL adres.

Hodnocení: 10 bodů

### **Nástroje**

Angular je standardně vybaven těmito nástroji:

- CLI, kterým jde vytvořit nový projekt, spustit a buildovat projekt, vygenerovat komponentu, službu, interface, modul, direktivu atd. <sup>48</sup>
- testovací prostředí pro psaní e2e a unit testů
- velké množství pluginů pro různorodé záležitosti, např. Bootstrap pro stylování, RxJS pro asynchronní zpracování či NgRx pro state management <sup>49</sup>.

Hodnocení: 9 bodů

### **Komponenty**

U komponent je v Angularu doporučeno od sebe důsledně oddělení HTML šablony, CSS styly a výkonnou JS (či TS) logiku.

Do child komponent je z parenta možné předávat data pomocí decoratoru `@Input()` <sup>50</sup>. A opačným směrem je možné vysílat notifikace o proběhlé události a to díky decoratoru `@Output()`, který je typu `EventEmitter` <sup>51</sup>.

Komponenty jsou obvykle definovány jako třída v TypeScriptu, která může implementovat nějaký interface. Angular podporuje v rámci komponenty používat anotace. Nejdůležitější údaje o komponentě jsou uvedeny v anotaci `@Component`. Zde najdeme typicky tyto položky:

- `selector` – definuje, pod jakým názvem se bude komponenta volat v šabloně parent komponenty.
- `templateUrl` – obsahuje buď samotnou HTML šablonu komponenty a nebo častěji je zde uvedena relativní cesta k dané HTML šabloně.
- `styleUrls` – odkazuje na CSS styly, které mají být v komponentě použity.

---

<sup>48</sup> CLI Overview and Command Reference [online]. Dostupné z: <https://angular.io/cli>

<sup>49</sup> Správa vnitřního stavu aplikace

<sup>50</sup> Sharing data between child and parent directives and components [online]. Dostupné z: <https://angular.io/guide/inputs-outputs>

<sup>51</sup> EventEmitter [online]. Dostupné z: <https://angular.io/api/core/EventEmitter>

Komponenty mají standardně implementovaný životní cyklus. Nejdůležitější metody z životního cyklu komponent jsou: `ngOnChanges()`, `ngOnInit()`, `ngDoCheck()`, `ngAfterContentInit()`, `ngAfterContentChecked()`, `ngAfterViewInit()`, `ngAfterViewChecked()` a `ngOnDestroy()` <sup>52</sup>.

Šablony obsahují tzv. Template syntax <sup>53</sup>, který umožňuje pracovat s:

- strukturálními direktivami – např. `*ngIf`, `*ngFor`, `*ngSwitch`
- data-binding – slouží k přenosu dat mezi výkonnou logikou komponenty a DOMem
- pipes – přetransformují zadaná data ještě před tím, než jsou zobrazena, hodí se např. pro `il8n`.

**Obrázek 7: Ukázka implementace služby (service) v Angularu**

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class HelloService {
7    |
8    getHelloMessage(applicationName: string): string {
9      return 'Hello ' + applicationName;
10   }
11
12 }
13
```

**Zdroj: vlastní zpracování**

<sup>52</sup> Hooking into the component lifecycle [online]. Dostupné z: <https://angular.io/guide/lifecycle-hooks>

<sup>53</sup> Introduction to components and templates [online]. Dostupné z: <https://angular.io/guide/architecture-components>

**Obrázek 8: Ukázka nadřazené (parent) komponenty a volání její sub-komponenty**

```
1 <div>
2   <app-hello-world [applicationName]="['HealthCare-App']" />
3 </div>
```

**Zdroj: vlastní zpracování**

**Obrázek 9: Ukázka implementace sub-komponenty**

```
1 import { Component, Input, OnInit } from '@angular/core';
2 import { HelloService } from './hello.service';
3
4 @Component({
5   selector: 'app-hello-world',
6   templateUrl: './hello-world.component.html',
7   styleUrls: ['./hello-world.component.scss']
8 })
9 export class HelloWorldComponent implements OnInit {
10
11   @Input() applicationName: string;
12
13   constructor(public helloService: HelloService) { }
14
15   ngOnInit() {
16     this.sayHello();
17   }
18
19   sayHello(): void {
20     let result: string = this.helloService.getHelloMessage(this.applicationName);
21     console.log(result);
22   }
23 }
```

**Zdroj: vlastní zpracování**



Hodnocení: 10 bodů

#### 4.2.2.2. React

O React <sup>54</sup> se stará vývojářská komunita v rámci společnosti Facebook. První verze vyšla v roce 2013 a jednalo se v podstatě odpověď na tehdy bezkonkurenční framework AngularJS. AngularJS byl totiž v podstatě prvním masově používaným JavaScript frameworkem, který využíval MVx <sup>55</sup> architekturu. V době psaní této práce je k dispozici verze 17 jako poslední verze.

React jako JS framework je podstatně „štíhlejší“ než výše popisovaný Angular. Sám o sobě obsahuje v podstatě jen systém pro práci s komponentami a dále CLI a React Router a i18n. Nicméně ve standardní základní instalaci Reactu už toho moc o moc více k dispozici není a všechny ostatní věci je potřeba doinstalovat pomocí externích pluginů.

#### **Velikost frameworku**

Velikost Reactu v základní konfiguraci je 180 MB.

#### **Architektura a API**

Struktura React aplikace rozhodně není tak striktně definována jako v Angularu, což zároveň představuje jeden z největších rozdílů mezi Angularem a Reactem. Služby například není nutné injectovat do komponent pomocí Dependency Injection, ale stačí je jednoduše nad definicí třídy (komponenty) naimportovat jako jakýkoliv jiný externí soubor. Co se týče API ve výchozí konfiguraci Reactu, tak má v sobě jen pluginy pro routování a i18n.

Hodnocení: 8 bodů

#### **Nástroje**

React nabízí tyto nástroje:

- CLI, které umožňuje vytvořit nový projekt a nastavit prostředí pro automatizované testování; nový projekt se standardně vytváří z Create React App <sup>56</sup>.

---

<sup>54</sup> React [online]. Dostupné z: <https://reactjs.org/>

<sup>55</sup> Obecný pojem pro MVC, MVP a MVVM

<sup>56</sup> Create a New React App [online]. Dostupné z: <https://reactjs.org/docs/create-a-new-react-app.html>

- Velké množství různých externích pluginů pro stylování komponent (Bootstrap, Ant Design atd.) či pro state management (Redux <sup>57</sup>).

Hodnocení: 7 bodů

### **Komponenty**

React umožňuje pracovat a komponentami v podstatě podobně jako Angular. Akorát zde jsou mnohem volnější pravidla pro implementaci. Taktéž je zde systém parent a child komponent, které si mezi sebou mohou předávat data či volat metody. React pro tento účel obsahuje tzv. `props` <sup>58</sup>. Pokud potřebujeme nadefinovat nějaká data související s vnitřním stavem komponenty, tak to React umožňuje pomocí `state` <sup>59</sup>.

Komponenty se v Reactu zapisují to s využitím JSX souboru (nebo TSX, pokud používáme TypeScript), který obsahuje speciální syntaxi pro definování šablon. To znamená, že je komponentách v podstatě v jednom souboru dohromady smíchaná výkonná logika komponenty a její šablona. Komponenty v Reactu mají svůj definovaný životní cyklus a využívají se k tomu především tyto metody: `render()`, `componentDidMount()`, `shouldComponentUpdate()`, `componentDidUpdate()`, `componentWillUnmount()` a `componentDidCatch()`. Každá komponenta musí z těchto metod obsahovat minimálně metodu `render()`, která ve stromové struktuře vrací HTML šablonu dané komponenty. To je zásadní rozdíl oproti Angularu, kde jsou tyto 2 vrstvy od sebe oddělené.

---

<sup>57</sup> Redux Fundamentals, Part 5: UI and React [online]. Dostupné z: <https://redux.js.org/basics/usage-with-react>

<sup>58</sup> Components and Props [online]. Dostupné z: <https://reactjs.org/docs/components-and-props.html>

<sup>59</sup> State and Lifecycle [online]. Dostupné z: <https://reactjs.org/docs/state-and-lifecycle.html>

**Obrázek 10: Ukázka implementace služby (service) v Reactu**

```
1  class HelloWorldService {  
2  
3      getHelloMessage(applicationName: string): string {  
4          return 'Hello ' + applicationName;  
5      }  
6  
7  }  
8  
9  export default HelloWorldService;
```

**Zdroj: vlastní zpracování**

**Obrázek 11: Ukázka implementace nadřazené (parent) komponenty v Reactu**

```
1  import * as React from 'react';  
2  import { Component } from 'react';  
3  
4  import HelloWorld from '../HelloWorld/HelloWorld';  
5  
6  class Greetings extends Component {  
7  
8      render() {  
9          return (  
10             <div>  
11                 <HelloWorld applicationName={'HealthCare-App'} />  
12             </div>  
13             );  
14      }  
15  
16  }  
17  
18  export default Greetings;
```

**Zdroj: vlastní zpracování**

**Obrázek 12: Ukázka implementace sub-komponenty v Reactu**

```
1 import * as React from 'react';
2 import { Component } from 'react';
3
4 import HelloWorldService from './HelloWorldService';
5
6 interface IProps {
7     applicationName: string;
8 }
9
10 interface IState {
11     greetings: string;
12 }
13
14 class HelloWorld extends Component<IProps, IState> {
15
16     private helloWorldService = new HelloWorldService();
17
18     constructor(props: IProps) {
19         super(props);
20     }
21
22     sayHello(): void {
23         let result: string = this.helloWorldService.getHelloMessage(this.props.applicationName);
24         this.setState( state: {
25             greetings: result
26         });
27         console.log(this.state.greetings);
28     }
29
30     render() {
31         return (
32             <div>{this.state.greetings}</div>
33         );
34     }
35 }
36
37 export default HelloWorld;
```

**Zdroj: vlastní zpracování**

Hodnocení: 8 bodů

#### **4.2.2.3. Ember**

Ember <sup>60</sup> vyvíjí a podporuje komunika nezávislých contributovů a první verze byla vydána v roce 2011. V době psaní této psání je aktuální poslední verze 3.

---

<sup>60</sup> Ember.js [online]. Dostupné z: <https://emberjs.com/>

## **Velikost frameworku**

Velikost Emberu v základní konfiguraci je 77 MB.

## **Architektura a API**

Jedná se o klasickou MVC aplikaci, kde je kód rozdělen na komponenty (View), modely a controllery. Komponenty obsahují převážně HTML kód a nějaká složitější aplikační logika je v implementována v controllerech či modelech. Modely slouží pro práci s daty a controllery jsou ke komunikaci mezi komponentami a modely. Velmi podobně byl z hlediska architektury navržen starý AngularJS. Framework automaticky umožňuje routování na front-endu.

Hodnocení: 5 bodů

## **Nástroje**

Ember nabízí tyto nástroje:

- CLI, které umožňuje vytvořit nový projekt, spustit projekt, vygenerovat komponentu, service, routu či spustit testy.
- Existují k Emberu různé pluginy pro stylování komponent, např. Material-UI. Je jich však výrazně méně, než pro Angular či React.

Hodnocení: 6 bodů

## **Komponenty**

Komponenty mají vlastní syntaxi, která umožňuje používat strukturální direktivy a spouštění události. Tím je možné si mezi komponentou a příslušným controllerem předávat data. Podobně jaké u Angularu a Reactu je i zde možné pracovat s hierarchií komponent ve struktuře parent – child. Ale přístupu parent či naopak child komponentě je v Emberu poněkud komplikovanější – je nutné k tomu používat speciální syntaxi v šablonách.

```

app/templates/components/blog-post.hbs
1 <article class="blog-post">
2   <h1>{{title}}</h1>
3   <p>{{yield}}</p>
4   <p>Edit title: {{input type="text" value=title}}</p>
5 </article>

app/templates/index.hbs
1 {{#each model as |post|}}
2   {{#blog-post title=post.title}}
3     {{post.body}}
4   {{/blog-post}}
5 {{/each}}

app/routes/index.js
1 import Route from '@ember/routing/route';
2
3 export default Route.extend({
4   model() {
5     return this.store.findAll('post');
6   }
7 });

```

Obrázek: Ukázka komponent v Emberu a jejich zanoření <sup>61</sup>

Hodnocení: 5 bodů

#### 4.2.2.4. Vyhodnocení variant a nalezení nejlepší kompromisní varianty

##### Přehled ohodnocení variant podle jejich kritérií

V u jednotlivých variant (frameworku), bylo ke každému kritériu zadáno hodnocení. V následující tabulce jsou přehledně uvedeny všechny varianty a jejich ohodnocení dle jednotlivých kritérií.

**Tabulka 3: Ohodnocení variant podle jejich kritérií**

	Kritérium			
	1	2	3	4
Angular	335	10	9	10
React	180	8	7	8
Ember	77	5	6	5

**Zdroj: vlastní zpracování**

Tato tabulka představuje nejdůležitější vstupní data pro provedení vícekritériální analýzy variant u jednotlivých JavaScript frameworků.

<sup>61</sup> Defining a Component [online]. Dostupné z: <https://guides.emberjs.com/v3.4.0/components/defining-a-component/>

### Stanovení vah kritérií

Stanovení vah kritérií probíhá pomocí Saatyho metody, jak bylo uvedeno v kapitole 3.3.1. Je k tomu potřeba si stanovit, která kritéria výběru kompromisní varianty mají vyšší váhu před ostatními kritérii. Dle zkušeností autora této práce je u JavaScript frameworků důležitá zejména architektura a API. Dále pak také komponenty. Proto mají pro následnou analýzu nejvyšší váhu kritéria č. 2 a 4. V následující tabulce jsou přehledně uvedeny váhy pro jednotlivá kritéria.

**Tabulka 4: Stanovení vah kritérií dle Saatyho metody**

		Kritérium				Geometrický průměr ( $b_i$ )	Váha ( $v_i$ )
Kritérium	1	1	2	3	4		
	1	1	1/9	1/5	1/7	0,270	0,044
	2	9	1	5	3	3,708	0,603
	3	5	1/5	1	3	1,627	0,265
	4	7	1/3	1/3	3	0,541	0,088
Součet						6,145	1

**Zdroj: vlastní zpracování**

### Analýza metodou váženého součtu

U analýzy metody váženého součtu je důležité stanovit pro jednotlivá kritéria ideální varianty H a bazální varianty D. Toto stanovení úzce souvisí s tím, jestli je dané kritérium minimalizační nebo maximalizační. Kritérium č. 1 (Velikost frameworku) je minimalizační. Proto je zde ideální varianta nejnížší hodnota ze všech variant a bazální naopak hodnota nejvyšší. Ostatní kritéria u variant jsou naopak maximalizační, a proto je u nich situace opačná. Ideální a bazální varianty jsou uvedeny v následující tabulce:

**Tabulka 5: Stanovení ideální a bazální varianty**

	Kritérium			
	1	2	3	4
Ideální varianta	77	10	10	10
Bazální varianta	335	5	6	5

**Zdroj: vlastní zpracování**

Na základě těchto hodnot se následně vypočítá kritériální matice R. Podle vzorce, který je uveden v kapitole 3.3.2 se vypočítávají postupně dílčí užítky jednotlivých variant a jejich kritérií do doby, až z toho vznikne kompletní kritériální matice R.

Takže například prvek pro variantu 2 a kritérium 1 se vypočítá dosazením do vzorce a to následovně:

$$r_{11} = \frac{180 - 335}{77 - 335} = 0,601$$

Hodnota 180 je hodnota  $y_{ij}$  – tedy v tomto případě hodnota 1. kritéria u 2. varianty. Hodnota 335 představuje bazální variantu  $d_j$  a hodnota 77 ideální variantu  $h_j$ .

**Tabulka 6: Kriteriační matice R**

	Kritérium			
	1	2	3	4
Angular	0,000	1,000	0,750	1,000
React	0,601	0,600	0,250	0,600
Ember	1,000	0,000	0,000	0,000

**Zdroj: vlastní zpracování**

Pro výpočet celkového užítu u jednotlivých variant (a tím i pro nalezení kompromisní varianty) je nyní potřeba provést skalární součin mezi dílčími užítky  $r_{ij}$  pro danou variantu z matice R a jejich váhou kritérií  $v_i$ .

**Tabulka 7: Celkový užitek variant a jejich pořadí**

Varianta	Celkový užitek variant	Pořadí
Angular	0,890	1.
React	0,507	2.
Ember	0,044	3.

**Zdroj: vlastní zpracování**

### Výsledné vyhodnocení

Byl vypracován popis a srovnání jednotlivých JavaScriptových frameworků v souvislosti s výběrem vhodného frameworku pro vývoj ukázkové aplikace HealthCare-App. Do výběru bylo zahrnuto celkem 5 frameworků, konkrétně: Angular, React, Vue, Ember a Knockout. Výběr vhodného frameworku proběhl dvoufázově. V první fázi byly vyřazeny frameworky, které podle GitHubu mají velmi malou podporu vývojářských komunit, jež se o dané frameworky starají. Prvními vyřazenými tedy jsou Vue a Knockout.

Následně pro druhou fázi zůstaly ve výběru Angular, React a Ember. U těchto zbývajících frameworků již bylo provedeno srovnání podle možností, které vývojářům nabízí a to na základě provedení vícekritériální analýzy variant.

Provedením vícekritériální analýzy variant s využitím metody váženého součtu vyšel framework Angular jako variant s nejlepším hodnocením celkového užítu. A to s hodnotou 0,890. Proto na základě této analýzy byl právě **Angular** vybrán jako vhodný framework pro implementaci ukázkové aplikace HealthCare-App.



### 4.3. Porovnání UI frameworků

UI frameworky jsou obvykle frameworky postavené nad CSS a jejich přidaná hodnota spočívá ve skutečnosti, že vývojářům výrazně pomáhají při návrhu responzivního designu u webových aplikací. V této analýze jsou mezi sebou srovnávány frameworky: Twitter Bootstrap <sup>62</sup>, ZURB Foundation <sup>63</sup> a Skeleton <sup>64</sup>. Bootstrap a Foundation jsou poměrně rozsáhlé frameworky disponující velkým množstvím různých CSS tříd, jež usnadňují vývoj Front-endu webových aplikací. Oba tyto frameworky jsou si mezi sebou do značné míry podobné a jsou v mezi sebou v podstatě konkurenční. Skeleton je naopak jako framework výrazně „štíhlejší“, což znamená, že nabízí méně možností. Ale také je na druhou stranu i výrazně menší, což se může hodit při načítání webových stránek.

Výběr vhodného UI frameworku je proveden pomocí vícekritériální analýzy variant. Pro nalezení kompromisní varianty jsou stanovena kritéria, jež jsou uvedena v této tabulce:

**Tabulka 8: Kritéria pro vícekritériální analýzu UI frameworků**

Číslo	Kritérium	Jednotka
1	Velikost frameworku	kB
2	Mřížka a zlomové body	Bodové hodnocení
3	Responzivní ovládací prvky	
4	Customizace	
5	Podpora webových prohlížečů	

**Zdroj: vlastní zpracování**

Kritérium č. 1 (Velikost frameworku) je kvantitativní a má minimalizační povahu. Data pro hodnocení variant u tohoto kritéria jsou získávány stažením CSS souboru a příp. i JavaScript souboru (pokud je u frameworku k dispozici) jejich minifikované podobě z CDN <sup>65</sup>.

Ostatní kritéria jsou naopak kvalitativní a mají maximalizační povahu. Data pro hodnocení variant u těchto kritérií jsou získávány do značné míry subjektivně a to na základě vývojářských zkušeností autora této práce a studia dokumentace jednotlivých frameworků a požadavků na výslednou aplikaci. Bodové ohodnocení variant zde může nabývat číselných hodnot od 1 do 10, kde 1 je nejhorší a 10 nejlepší.

#### **Velikost frameworku**

<sup>62</sup> Bootstrap [online]. Dostupné z: <https://getbootstrap.com/>

<sup>63</sup> Foundation [online]. Dostupné z: <https://get.foundation/>

<sup>64</sup> Skeleton [online]. Dostupné z: <http://getskeleton.com/>

<sup>65</sup> Content Delivery Network

Velikost konkrétního frameworku je důležitá především pro rychlost načítání výsledné webové aplikace. Neboť logicky platí, že čím je framework větší, tím se pomaleji načítá. Velikost daného frameworku je pro jednotlivé varianty součtem CSS a příp. i JavaScript souboru v jejich minifikované podobě. Tyto soubory jsou staženy z CDN.

### **Mřížka a zlomové body**

Mřížka a zlomové body jsou jednoznačně nejdůležitější součástí každého UI frameworku, který podporuje návrh responzivního designu. Jak bylo popsáno v kapitole Teoretická východiska, tak mřížka (Grid System) slouží k tomu, aby možné do buněk mřížky (Grid System) na webu umístit různé prvky a následně těmto buňkám měnit svou velikost (zpravidla šířku) v závislosti na použitém koncovém zařízení.

Zlomové body (Breakpoints) pak slouží k tomu, aby bylo možné při různých šířkách obrazovky a na základě definovaných mediálních dotazů (Media Query) měnit design jednotlivých prvků. U zlomových bodů se obvykle hodí, pokud framework podporuje nějaký CSS preprocesor, neboť pak se dají zlomové body v preprocesoru snadno nadefinovat (či použít výchozí).

### **Responzivní ovládací prvky**

Responzivní ovládací prvky slouží k tomu, aby bylo možné responzivní web vybavit nějakým komplexním ovládacím prvkem, který je již od základu navržen pro práci na různých koncových zařízeních. Typicky se může jednat např. horní menu. Horní menu se pro desktop (a někdy i pro tablet) zobrazuje zpravidla vodorovně (tj. položky menu jsou umístěny vedle sebe) v místě hlavičky webu. A toto menu je obvykle trvale zobrazeno – tj. nezakrývá se na základě kliknutí. Naopak o mobilu je naopak hlavní menu zobrazeno jen po kliknutí na určité tlačítko a po kliknutí se zobrazí svisle (tzn. položky pod sebou).

V tomto kritériu je analyzováno, zda-li jednotlivé frameworky ve své výchozí instalaci nabízejí nějaké takové responzivní ovládací prvky.

### **Customizace**

Jedná se o kritérium, kde se posuzuje, zda-li a jak je možné provádět hromadné úpravy CSS stylů na celém webu. Zde se velmi hodí, pokud daný framework podporuje alespoň 1 CSS preprocesor. Taktéž je zde analyzováno, jestli jdou styly frameworku nějak individuálně nakonfigurovat ještě před jeho stažením a poté stáhnout již takto upravenou verzi frameworku.

## **Podpora webových prohlížečů**

Podpora webových prohlížečů je důležitá, neboť je často tím zásadním faktorem, který může (v případě nedostatečné podpory) odrazit uživatele aplikace od jejího používání. U UI frameworků je především stěžejní zaměřit se na to zda-li framework podporují spíše novější nebo naopak starší verze prohlížečů. A pokud ho podporují i starší verze prohlížečů, tak jak si konkrétní prohlížeče poradí s některými moderními prvky frameworku (např. s responzivním designem atd.).

### **4.3.1. Twitter Bootstrap**

Twitter Bootstrap je UI framework, za jehož vývojem stojí vývojářská komunita společnosti Twitter, jak již jeho název napovídá. Konkrétně se o jeho vývoj zasadili Jacob Thornton a Mark Otto, kteří v roce 2010 vytvořili jeho první verzi, jež měla původně sloužit pro vývoj interních nástrojů Twitteru. Bootstrap je v současnosti nejpopulárnější HTML, CSS a JS framework pro navrhování responzivních a Mobile First aplikací. Je pod licencí MIT, takže je ke stažení v podstatě volně.<sup>66</sup> V době psaní této psaní byla aktuální verzí Bootstrapu verze 4. Dokumentace k Bootstrapu je k nalezení na [getbootstrap.com](https://getbootstrap.com).

#### **Velikost frameworku**

Velikost Bootstrapu v základní konfiguraci je 238 kB.

#### **Mřížka a zlomové body**

Bootstrap má systém mřížky (Grid System), která standardně obsahuje 12 sloupců. Lze však v konfiguraci nastavit v podstatě jakýkoliv počet sloupců.

Standardně jsou implementovány 4 zlomové body, které umožňují vytvářet pomocí mřížky odlišné designy pro 4 různé velikosti displeje (small, medium, large a extra large). Každý zlomový bod je podle zmiňované velikosti displeje pojmenován určitou zkratkou a to konkrétně: sm, md, lg a xl. Následující tabulka znázorňuje přesné detaily ohledně možností implementace odlišného designu pro zařízení a jejich přesné zlomové body.

---

<sup>66</sup> Bootstrap 3 Tutorial [online]. Dostupné z: <https://www.w3schools.com/bootstrap/default.asp>

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

**Obrázek 13: Mřížka a zlomové body v Bootstrapu**

V rámci konfigurace Bootstrapu (a to jak je verze s CSS preprocesorem, tak i bez něj) je možné tyto hodnoty jednotlivých zlomových bodů individuálně nastavovat. Lze teoreticky přidávat i nové zlomové body. Nicméně s touto možností pravděpodobně nebylo při vývoji Bootstrapu počítáno, a proto k tomu není v zdrojových souborech nic připraveno a přidávání či případné odebrání zlomových bodů je zde poměrně komplikované.

Hodnocení: 9 bodů

### **Responzivní ovládací prvky**

Bootstrap má poměrně velké množství různých předpřipravených ovládacích prvků, které lze na web jednoduše naimplementovat a případně podle potřeb změnit v CSS jejich stylování. Jako příklad lze vyjmenovat tyto prvky: notifikace, drobečková navigace, carousel, formuláře, seznamy, modální okno, menu, stránkování, tooltips a další.

Co je vše nevýhodou je fakt, že tyto ovládací prvky nelze např. do projektu v Angularu automaticky implementovat jako samostatnou autonomní komponentu. Naopak se můžeme do HTML šablony buď vložit celý HTML kód daného prvku. A nebo si sám vytvořit komponentu a tam tento celý HTML kód vložit.

Všechny tyto ovládací prvky jsou samozřejmě plně responzivní.

Hodnocení: 9 bodů

### **Customizace**

Možnosti customizace jsou takové, že v podstatě u všech prvků lze upravit jejich CSS. A to jak pomocí manuální úpravy CSS stylů, kdy se výchozí CSS styl přetáhne vlastním stylem. Nebo pomocí konfiguračního nástroje, který výchozí CSS styl

automaticky upraví ještě před stažením frameworku do projektu. Konfigurace v tomto případě probíhá tak, že uživatel do formuláře zadá, jak přesně si přeje, aby jím upravený Bootstrap vypadal. K tomu jsou k dispozici na internetu různé volně použitelné formulářové konfigurační nástroje. Jeden z nich je k dispozici na URL adrese <https://bootstrap.build/app>

Hodnocení: 10 bodů

### **Podpora webových prohlížečů**

Bootstrap podporuje v podstatě všechny webové prohlížeče, které v současné době využívá větší množství uživatelů. A to, jak na mobilu, tak na desktopu. Následující tabulky přehledně znázorňují, které prohlížeče Bootstrap podporuje.

	Chrome	Firefox	Safari	Android Browser & WebView	Microsoft Edge
Android	Supported	Supported	N/A	Android v5.0+ supported	Supported
iOS	Supported	Supported	Supported	N/A	Supported
Windows 10 Mobile	N/A	N/A	N/A	N/A	Supported

**Obrázek 14: Podpora Bootstrapu na mobilních prohlížečích**

	Chrome	Firefox	Internet Explorer	Microsoft Edge	Opera	Safari
Mac	Supported	Supported	N/A	N/A	Supported	Supported
Windows	Supported	Supported	Supported, IE10+	Supported	Supported	Not supported

**Obrázek 15: Podpora Bootstrapu na desktopových prohlížečích**

Hodnocení: 9 bodů

### 4.3.2. ZURB Foundation

ZURB Foundation je UI framework, který je svou koncepcí velmi podobný Bootstrapu. Taktéž se jedná o framework, který umožňuje navrhovat responzivní webové aplikace pro jakékoliv zařízení, médium a přístupnost. Foundation je rodina responzivních UI frameworků, které usnadňují vývoj responzivních webů, aplikací a e-mailů. Foundation je sémantický a kompletně customizovatelný.<sup>67</sup> V době psaní této práce byla aktuální verze 6. Foundation je pod licencí MIT a dokumentace je k dispozici na URL adrese [get.foundation](http://get.foundation).

#### **Velikost frameworku**

Velikost Foundation v základní konfiguraci je 307 kB.

#### **Mřížka a zlomové body**

Mřížka (Grid System) je ve Foundation řešena stejně jako v Bootstrapu. Tedy, že ve výchozí konfiguraci je k dispozici 12 sloupců, které se dají podle potřeby přidávat nebo naopak odebírat.

Zlomové body (Breakpoints) jsou taktéž ve Foundation podobné jako Bootstrapu. Ovšem s tím rozdílem, že Foundation má pouze 2 zlomové body. Konkrétně jsou zde zlomové body standardně definovány takto:

- Small – jakákoliv velikost menší než 640 px
- Medium – jakákoliv velikost větší nebo rovna 640 px a menší než 1024
- Large – jakákoliv velikost větší nebo rovna 1024 px<sup>68</sup>

Dle zkušenosti autora této práce jsou pouze 2 zlomové body pro návrh kvalitního responzivního designu poměrně málo. Protože tak nelze u menších rozměrů odlišit např. horší mobil s nízkým rozlišením oproti lepšímu mobilu s vyšším rozlišením. A naopak u větších obrazovek nelze nastavit specifický design např. pro rozlišení nad 1200 px. Dle názoru autora má tedy Bootstrap lépe pokryto zlomové body s ohledem na různá koncová zařízení.

Hodnocení: 7 bodů

#### **Responzivní ovládací prvky**

---

<sup>67</sup> Foundation [online]. Dostupné z: <https://get.foundation/>

<sup>68</sup> Media Queries [online]. Dostupné z: <https://get.foundation/sites/docs/media-queries.html>

Foundation nabízí, podobně jako Bootstrap, poměrně velké množství různých ovládacích prvků. Například: tlačítka, menu, stránkování, drobečkovou navigaci, dropdown, tabulky, media objekt (pro přehrání videa), fotogalerii, tooltip a další. Všechny tyto prvky podporují responzivní zobrazení a je možné je v CSS přestylavat.

Stejně jako v Bootstrapu mají tyto ovládací prvky tu nevýhodu, že se nejedná o samostatné komponenty. To znamená, že se do projektu vkládá celý HTML kód s daným prvkem. Případně se např. u Angular projektu může tento HTML kód zobecnit a vložit do samostatné komponenty. Avšak tu je nutné pro tento účet připravit – není to takto připraveno automaticky ve výchozí konfiguraci.

Hodnocení: 9 bodů

### **Customizace**

Customizace je možná prostřednictvím přetížení výchozích CSS stylů. Není však možné provádět customizaci prostřednictvím nějakého interaktivního konfiguračního formuláře ještě před samotným stažením frameworku. Toto je nevýhoda oproti Bootstrapu, neboť tam lze provést konfiguraci prostřednictvím interaktivního konfiguračního formuláře.

Hodnocení: 8 bodů

### **Podpora webových prohlížečů**

Jsou podporovány všechny důležité webové prohlížeče, minimálně 2 verze zpět. Detailní přehled podpory jednotlivých prohlížečů je uveden v následující tabulce:

Chrome	✓ Last Two Versions
Firefox	
Safari	
Opera	
Mobile Safari <sup>1</sup>	
IE Mobile	
Edge	
Internet Explorer	✓ Versions 9+
Android Browser	✓ Versions 4.4+

**Obrázek 16: Podpora Foundation v jednotlivých webových prohlížečích**

Hodnocení: 9 bodů

#### 4.3.3. Skeleton

Skeleton je oproti dvou výše uvedeným UI frameworkům (Bootstrap a Foundation) nejmenší a má nejméně možností. Jak je uvedeno i v jeho dokumentaci, tak by se měl použít v případě, že vývojář pracuje na menším projektu nebo zkrátka nemá pocit, že potřebuje všechnu rozsáhlou funkcionalitu větších frameworků. Skeleton styluje jen hrstku standardních HTML prvků a obsahuje mřížku.<sup>69</sup> V době psaní této práce je aktuální verze Skeletonu verze 2. Skeleton je pod licencí MIT a jeho dokumentace je k dispozici na adrese [getskeleton.com](http://getskeleton.com).

##### **Velikost frameworku**

Velikost Skeletonu v základní konfiguraci je 25 kB.

##### **Mřížka a zlomové body**

Skeleton je vybaven standardní mřížkou o 12 sloupcích. Není zde však možné snadno přidávat či odebírat sloupce tak, jako to jde u Bootstrapu nebo Foundation.

Zlomových bodů (Breakpoints) obsahuje Skeleton celkem 5. Následující tabulka znázorňuje jednotlivé zlomové body:

<sup>69</sup> Skeleton [online]. Dostupné z: <http://getskeleton.com/>



- Desktop HD: 1200px
- Desktop: 1000px
- Tablet: 750px

- Phablet: 550px
- Mobile: 400px

### Obrázek 17: Zlomové body ve Skeleton

Je zde dodržován princip Mobile First, což v praxi znamená, že pokud není zadán žádný mediální dotaz (Media Query), tak se daný styl týká zobrazení na všech zařízeních. A až, když je nějaký mediální dotaz zadán, tak se příslušná úprava designu promítne ve vyšších zlomových bodech.

Hodnocení: 7 bodů

#### Responzivní ovládací prvky

Skeleton neobsahuje ve své výchozí instalaci žádné předpřipravené responzivní ovládací prvky. Jedná je o velmi jednoduchá framework, který obsahuje v podstatě jen mřížku (Grid System) a zlomové body (Breakpoints).

Hodnocení: 0 bodů

#### Customizace

Customizace ve Skeletonu možná je. Akorát tím, že Skeleton obsahuje v podstatě jen mřížku (Grid System), tak lze přestylovat jen tuto mřížku. Žádné responzivní ovládací prvky Skeleton neobsahuje, tak zde není co stylovat.

Hodnocení: 3 body

#### Podpora webových prohlížečů

Tím, že je Skeleton jako framework velmi jednoduchý, tak funguje bez problému nejen na všech aktuálně používaných prohlížečích. Ale často i s jejich staršími verzemi. Co se týče Internet Exploreru, tak je podporován od verze 9 a výše.

Hodnocení: 10 bodů

#### 4.3.4. Vyhodnocení variant a nalezení nejlepší kompromisní varianty

##### Přehled ohodnocení variant podle jejich kritérií

V u jednotlivých variant (frameworku), bylo ke každému kritériu zadáno hodnocení. V následující tabulce jsou přehledně uvedeny všechny varianty a jejich ohodnocení dle jednotlivých kritérií.

**Tabulka 9: Ohodnocení variant podle jejich kritérií**

	Kritérium				
	1	2	3	4	5
Bootstrap	238	9	9	10	9
Foundation	307	7	9	8	9
Skeleton	25	7	0	3	10

**Zdroj: vlastní zpracování**

Tato tabulka představuje nejdůležitější vstupní data pro provedení vícekritériální analýzy variant u jednotlivých UI frameworků.

##### Stanovení vah kritérií

Stanovení vah kritérií probíhá pomocí Saatyho metody, y; Dle zkušeností autora této práce je u UI frameworků důležitá zejména mřížka a zlomové body a pak také nabídka responzivních ovládacích prvků u daného frameworku. Proto mají pro následnou analýzu nejvyšší váhu kritéria č. 2 a 3. V následující tabulce jsou přehledně uvedeny váhy pro jednotlivá kritéria.

**Tabulka 10: Stanovení vah kritérií dle Saatyho metody**

		Kritérium					Geometrický průměr ( $b_i$ )	Váha ( $v_i$ )
Kritérium	1	1	2	3	4	5		
	1	1	1/9	1/7	1/3	1/5	0,254	0,033
	2	9	1	3	7	5	3,936	0,510
	3	7	1/3	1	5	3	2,036	0,264
	4	3	1/7	1/5	1	1/3	0,491	0,064
	5	5	1/5	1/3	3	1	1,000	0,130
<b>Součet</b>							<b>7,718</b>	<b>1</b>

**Zdroj: vlastní zpracování**

##### Analýza metodou váženého součtu

U analýzy metody váženého součtu je důležité stanovit pro jednotlivá kritéria ideální varianty H a bazální varianty D. Toto stanovení úzce souvisí s tím, jestli je dané kritérium minimalizační nebo maximalizační. Kritérium č. 1 (Velikost frameworku) je minimalizační. Proto je zde ideální varianta nejnížší hodnota ze všech variant a bazální

naopak hodnota nejvyšší. Ostatní kritéria u variant jsou naopak maximalizační, a proto je u nich situace opačná. Ideální a bazální varianty jsou uvedeny v následující tabulce:

**Tabulka 11: Stanovení ideální a bazální varianty**

	Kritérium				
	1	2	3	4	5
Ideální varianta	25	9	9	10	10
Bazální varianta	307	7	0	3	9

**Zdroj: vlastní zpracování**

Na základě těchto hodnot se následně vypočítá kritériální matice R. Podle vzorce, který je uveden v kapitole 3.3.2 se vypočítávají postupně dílčí užítky jednotlivých variant a jejich kritérií do doby, až z toho vznikne kompletní kritériální matice R.

Takže například prvek pro variantu 1 a kritérium 1 se vypočítá dosazením do vzorce a to následovně:

$$r_{11} = \frac{238 - 307}{25 - 307} = 0,245$$

Hodnota 238 je hodnota  $y_{ij}$  – tedy v tomto případě hodnota 1. kritéria u 1. variantu. Hodnota 307 představuje bazální variantu  $d_j$  a hodnota 25 ideální variantu  $h_j$ .

**Tabulka 12: Kritériální matice R**

	Kritérium				
	1	2	3	4	5
Bootstrap	0,245	1,000	1,000	1,000	0,000
Foundation	0,000	0,000	1,000	0,714	0,000
Skeleton	1,000	0,000	0,000	0,000	1,000

**Zdroj: vlastní zpracování**

Pro výpočet celkového užitku u jednotlivých variant (a tím i pro nalezení kompromisní varianty) je nyní potřeba provést skalární součin mezi dílčími užítky  $r_{ij}$  pro danou variantu z matice R a jejich váhou kritérií  $v_i$ .

**Tabulka 13: Celkový užitek variant a jejich pořadí**

Varianta	Celkový užitek variant	Pořadí
Bootstrap	0,846	1.
Foundation	0,309	2.
Skeleton	0,162	3.

**Zdroj: vlastní zpracování**

## Výsledné vyhodnocení

Byla provedena vícekritériální analýza variant s využitím metody váženého součtu, která sloužila pro výběr nejvhodnějšího UI frameworku v ukázkové aplikaci. Na základě této analýzy získal nejlepší hodnocení (celkový užitek varianty) framework **Twitter Bootstrap**. Tento UI framework byl tedy použit pro ukázkovou aplikaci HealthCare-App.

#### 4.4. Implementace ukázkové aplikace

Na základě porovnání nejvhodnějších frameworků (viz. kapitola 4.4 a 4.5) bude pro implementaci ukázkové aplikace použit JavaScript framework Angular a UI framework Twitter Bootstrap.

Dále bude ukázková aplikace postavena z tohoto technologického stacku: HTML5, CSS3, SCSS, PrimeNG.

##### 4.4.1. Angular a jeho konfigurace

Stran Angularu bude použita verze 8. Framework Angular v současnosti nabízí 2 možnosti práce s formuláři:

1. Reactive Forms
2. Template Driven Forms

Pro implementaci formulářů bude v ukázkové aplikaci použita modernější varianta, kterou představují Reactive Forms <sup>70</sup>.

Protože autor této práce v budoucnu plánuje aplikaci rozšířit v rámci komerčního vývoje v rámci svého zaměstnání, tak je důležité např. Aby aplikace mohla v budoucnu mít více jazykových mutací. Implementace jazykových mutací ve vývoji software obvykle nazývá zkratkou i18n <sup>71</sup>. A pro i18n bude v aplikaci použito ngx-translate <sup>72</sup>, což je standardní knihovna z Angularu.

##### 4.4.2. Způsob využití CSS

Na základě provedené vícekritériální analýzy variant byl pro aplikaci jako UI framework vybrán Twitter Bootstrap v aktuální verzi 4.

Při vývoji HTML šablon je využit standardní 12sloupcový layout z Grid Systemu Bootstrapu, který v aplikaci výrazně přispívá k návrhu standardního responzivního layoutu pomocí principu Mobile First.

---

<sup>70</sup> Reactive forms [online]. Dostupné z: <https://angular.io/guide/reactive-forms>

<sup>71</sup> Localization vs. Internationalization [online]. Dostupné z: <https://www.w3.org/International/questions/qa-i18n>

<sup>72</sup> ngx-translate [online]. Dostupné z: <https://github.com/ngx-translate/core>

Stylování CSS stylů bylo provedeno pomocí preprocesoru SCSS. Možnosti CSS preprocesorů byly popsány v kapitole 3.1.1.2. SCSS byl použit z důvodu, že se dnes v oblasti CSS preprocesorů jedná v podstatě o standard, které dokáže efektivně odstranit všechny nedostatky standardního CSS v podobě, jak ho před mnoha lety definovalo konsorcium W3C <sup>73</sup>.

## **4.5. Implementace jednotlivých komponent aplikace**

### **4.5.1. Komponenta Login**

Přihlašovací formulář se autentizuje pomocí tzv. „Fake Back-endu“, který ukládá hodnota na straně klienta do LocalStorage <sup>74</sup> ve webovém prohlížeči.

Byl převzat z tutoriálu „Angular 8 - User Registration and Login Example & Tutorial“. <sup>75</sup>

V případě opravdové aplikace by samozřejmě pro tento účel bylo nutné vytvořit nějaký plně funkční Back-end. To však dle autora této práce mimo rozsah zadání, neboť tato práce se zabývá převážně Front-endem webových aplikací.

V přihlašovacím formuláři jsou drobné odlišnosti mezi mobilní a desktopovou verzí. Na mobilní verzi jsou formulářová pole na celou šířku stránky a textové popisky (label) jsou nad nimi. Taktéž dolní tlačítka „Přihlásit“ a „Registrovat“ zabírají dohromady celou šířku stránky. V desktopové verzi jsou naopak formulářová pole vedle jejich textových popisků (label) a dolní tlačítka zabírají jen část šířky obrazovky.

---

<sup>73</sup> HTML & CSS [online]. Dostupné z: <https://www.w3.org/standards/webdesign/htmlcss.html>

<sup>74</sup> Window.localStorage [online]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

<sup>75</sup> Angular 8 – User Registration and Login Example & Tutorial [online]. Dostupné z: <https://jasonwatmore.com/post/2019/06/10/angular-8-user-registration-and-login-example-tutorial>

**Obrázek 18: Komponenta Login v mobilní verzi**

The image shows a mobile app interface for 'HealthCare-App'. At the top, the app name is centered. Below it, there are two input fields: one for 'E-mail:' and one for 'Heslo:'. Under the password field, there are two buttons: a solid blue 'Přihlásit' button and a white 'Registrovat' button with a blue border. At the bottom of the screen, a footer contains the text 'Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka'.

HealthCare-App

E-mail:

Heslo:

Přihlásit

Registrovat

Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka

**Zdroj: vlastní zpracování**



**Obrázek 19: Komponenta Login v desktopové verzi**

The screenshot shows a web application titled "HealthCare-App". Below the title, there is a login form with two input fields: "E-mail:" and "Heslo:". Below these fields are two buttons: a blue button labeled "Přihlásit" and a white button with a blue border labeled "Registrovat". At the bottom of the page, there is a footer text: "Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka".

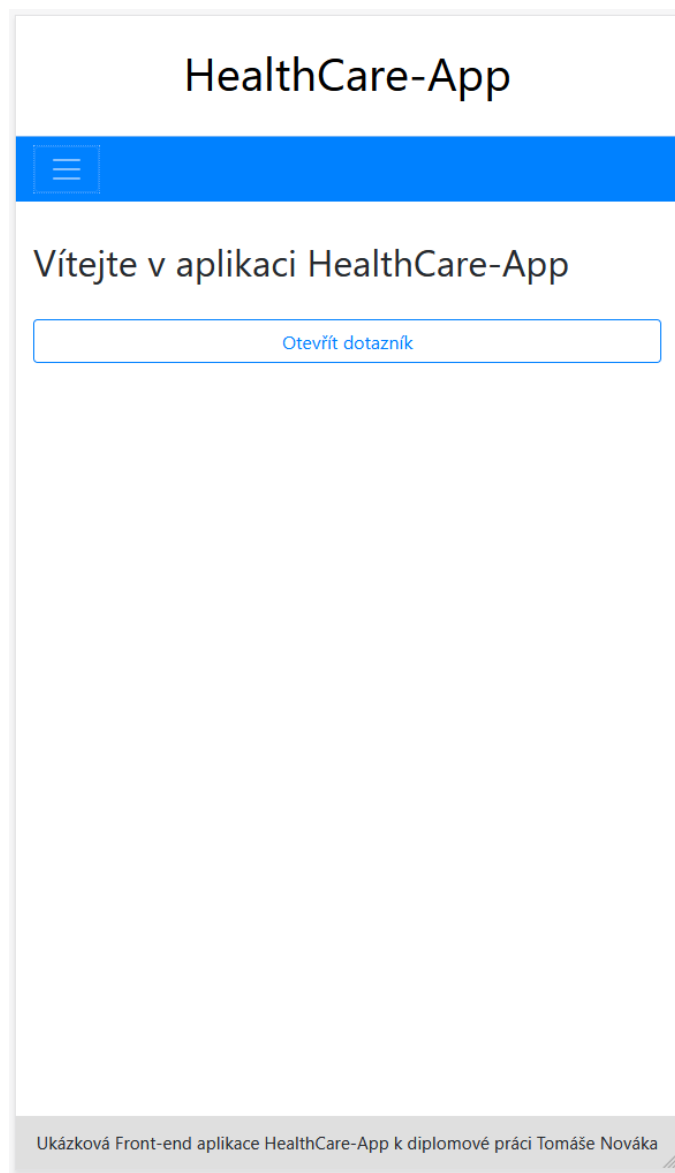
**Zdroj: vlastní zpracování**

#### **4.5.2. Komponenta Overview**

Komponenta Overview slouží v aplikaci v podstatě jako homepage pro přihlášené pacienty. V obsahové části stránky je umístěn uvítací nadpis a pod ním tlačítko „Otevřít dotazník“, které umožňuje otevřít nový dotazník. Je zde drobná odlišnost mezi mobilní a desktopovou verzí, která spočívá ve faktu, že na mobilní verzi je tlačítko „Otevřít dotazník“ umístěno přes celou obrazovku. Na desktopu vyplňuje jen třetinu šířky obrazovky.

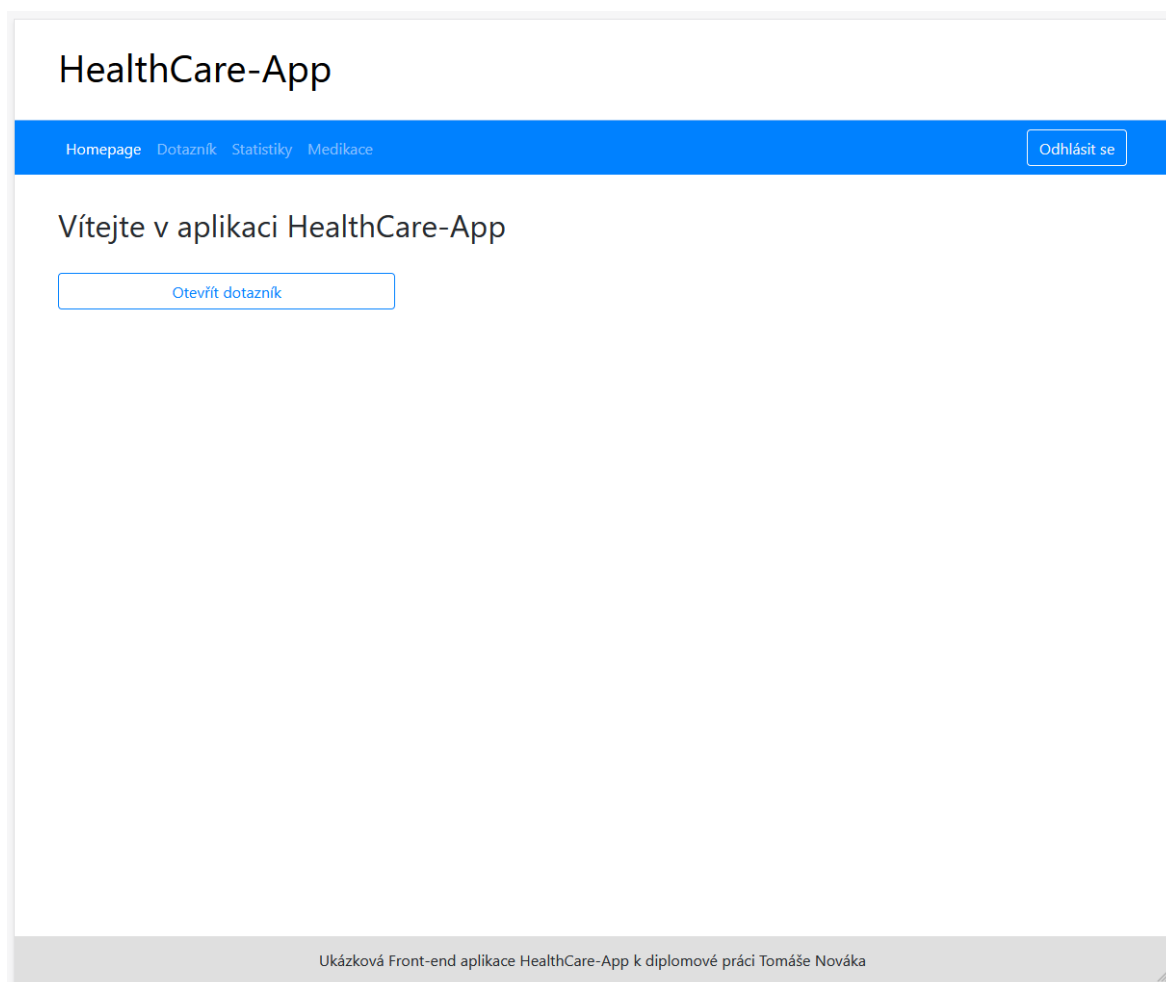


**Obrázek 20: Komponenta Overview v mobilní verzi**



**Zdroj: vlastní zpracování**

**Obrázek 21: Komponenta Overview v desktopové verzi**



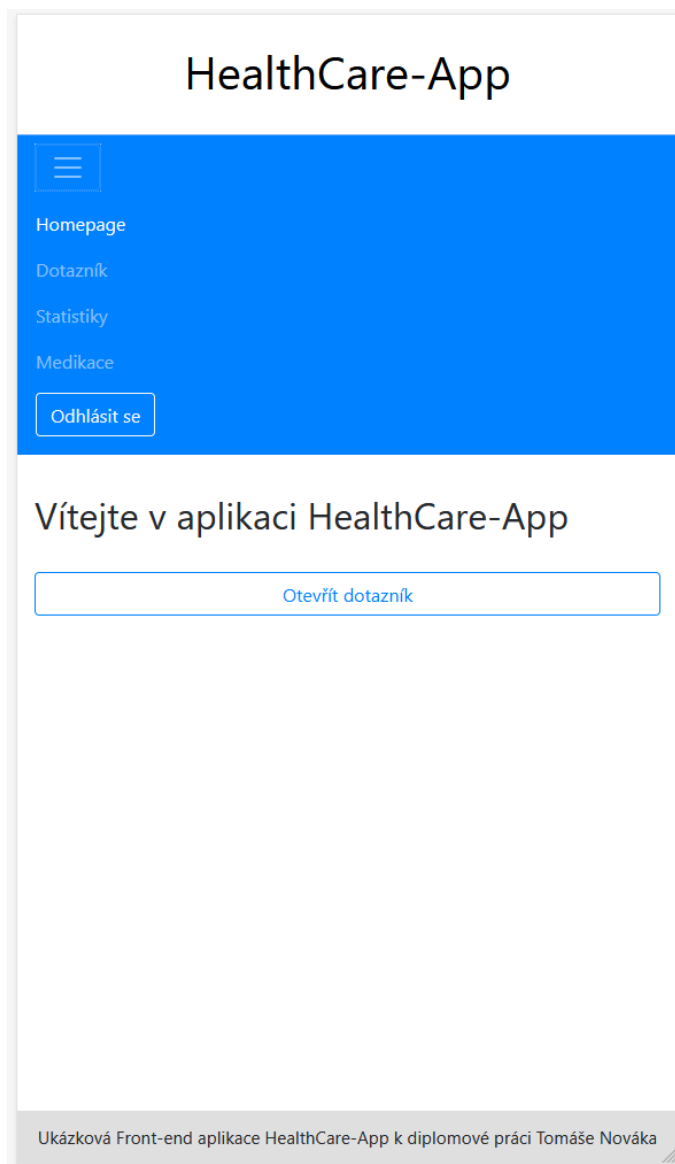
### **Zdroj: vlastní zpracování**

Významný rozdíl v mobilní a desktopové verzi je však patrný horním menu, které je k dispozici ve všech komponentách za komponentou Login. Tedy ve všech komponentách určených pro přihlášené uživatele. Zatímco na desktopu (a i tabletu) je toto hlavní menu umístěno v modrém boxu vodorovně přes celou obrazovku, tak na mobilu je toto menu navrženo tak, že ve výchozím zobrazení webu se uživatelům samotné menu vůbec nezobrazuje. Ve výchozím zobrazení se zobrazuje jen tlačítko, které se někdy nazývá jako „Hamburger“. Po kliknutí na toto tlačítko se rozbalí menu, které je orientováno svisle – tzn. jednotlivé položky menu jsou zobrazeny pod sebou. Toto je v podstatě standardní způsob, jak se dělá hlavní menu pro mobilní verze webů. Neboť displej mobilu je zpravidla malý a uživatel nepotřebuje mít k dispozici pořád. Proto je zde vhodné zobrazit menu až po kliknutí na příslušné tlačítko.<sup>76</sup>

<sup>76</sup> KADLEC, Tim. Responzivní design profesionálně. 1. vydání. Brno: Zoner Press, 2014. s. 84



**Obrázek 22: Hlavní menu v mobilní verzi**



**Zdroj: vlastní zpracování**

#### **4.5.3. Komponenta Questionare**

Komponenta Questionare slouží, jak už jeho název napovídá k tomu nejdůležitějšímu, co tato aplikaci pacientům nabízí. A sice možnost vyplňovat jednotlivé otázky týkající se pacientova zdravotního stavu. Otázky mohou být, podle diagnózy daného pacienta logicky strukturovány do více skupin. Tyto skupiny jsou následně zohledněny ve vizualizaci v pavučinovém teorému v komponentě Statistics (viz. kapitola 4.7.4).

Samotná otázka, na kterou pacient odpovídá může mít různou strukturu. Obvykle jsou v záhlaví otázky umístěny tyto prvky:

1. Progress bar indikující, jak daleko je pacient ve vyplňování svého dotazníku
2. Tlačítka sloužící k navigaci na předchozí či další otázku
3. Konkrétní otázka

Pod otázkou jsou zpravidla buď ve formě checkboxů nebo ve formě radio buttonů uvedené jednotlivé možnosti odpovědi na otázku. U jednotlivých odpovědí může být nějaký doprovodný obrázek pro lepší pochopení (např. šipka nahoru či dolů u otázek dotazujících se na to, jestli se nějaký symptom zlepšil či naopak zhoršil). Rovněž může být u otázky nějaký doprovodný obrázek (např. schéma hlavy u otázek týkajících se bolesti hlavy).

Na obrazovce jsou určité odlišnosti v mobilní a desktopové verzi. Tlačítka pro navigaci mezi otázkami jsou u mobilní verze před celou šířku obrazovky. Avšak u desktopové verze zabývají jen levou 1 třetinu obrazovky.

Další odlišností je případný doprovodný obrázek vázající se na otázku. U mobilní verze je až pod možnostmi otázek, aby u samotných možností otázek zbytečně nezabíral místo, kterého je na displeji mobilu málo. Ale u desktopové verze, kde je místa mnohem více, tak může být v pravé části obrazovky vedle otázky, která je umístěna vlevo.

**Obrázek 23: Komponenta Questionare v mobilní verzi**

HealthCare-App

Predchozí otázka

Další otázka

Aktuální bolest hlavy

☐ Zlepšující se hlava

☐ Stále stejná

☒ Zhoršující se

Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka

**Zdroj: vlastní zpracování**

**Obrázek 24: Komponenta Questionare v desktopové verzi**

HealthCare-App

Homepage Dotazník Statistiky Medikace

Odhlásit se

Předchozí otázka Další otázka

Aktuální bolest hlavy

☐ Zlepšující se hlava

☐ Stále stejná

☒ Zhoršující se

Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka

**Zdroj: vlastní zpracování**

#### 4.5.4. Komponenta Statistics

Komponenta Statistics je úzce provázána s komponenta Questionare. Slouží totiž ke grafické vizualizaci dat získaných od pacientů v rámci pravidelného vyplňování dotazníků. Tato komponenta obsahuje 2 typy grafů:

1. Spojnicový
2. Pavučinový teorém.

Spojnicový graf funguje tak, že se sečtou body ze všech odpovědí na otázky a výsledkem tohoto součtu je konkrétní číselná hodnota k určitému datu (datum vyplňování dotazníku). Spojnicový graf má na ose x čas a na ose y tyto součtové hodnoty bodů z otázek. A pak jsou tyto součtové hodnoty snadno znázorněny ve spojniciovém grafu. Když se takový hodnot do grafu zavede více (tj. vyplní se více dotazníků), tak lze sledovat trend,

jak se vyvíjí v čase pacientův klinický stav. Pro spojnicový graf byla využita komponenta Line Chart <sup>77</sup> pocházející z PrimeNG.

Druhý graf – pavučinový teorém – funguje na podobném principu, protože také zobrazuje data získaná z dotazníků, jež vyplňují pacienti. Avšak zobrazuje trochu jiná data, než spojnicový graf. Jak již bylo uvezeno v popisu komponenty Questionare, tak jednotlivé otázky jsou zde logicky roztrženy do různých kategorií. Někdy může být žádoucí vědět, jak se pacient zlepšil či zhoršil nikoliv celkově vše všech dotazovaných symptomech, ale jak se zlepšil či zhoršil v jednotlivých kategoriích. A přesně k tomu slouží tento pavučinový teorém. Pavučinový teorém sečte body z odpovědí na otázky vždy jen v rámci té určité kategorie symptomů a ten zobrazí v grafu příslušné kategorie symptomů. Pro pavučinový teorém byla využita komponenta Radar <sup>78</sup> pocházející z PrimeNG. Tyto komponenty sloužící pro práci grafy fungují tak, že se jim předá nějaký JSON s daty. V tomto případě s daty týkající se odpovědí na otázky z komponenty Questionare. A komponenta Line Chart nebo Radar se následně postará o grafickou vizualizaci těchto dat z JSONu. JSON data se volají pomocí jako HTTP požadavek prostřednictvím k tomu připravené služby (service). Tato služba (service) je to komponenty Statistics vložena (inject) pomocí Dependency Injection, což je v Angularu standardní postup, jak se v komponentách pracuje se službami (service) <sup>79</sup>.

**Todo: tady přidat screenshot z kódu, jak se tam v DI volá service**

**Obrázek: Ukázka volání služby (service) pomocí Dependency Injection [zdroj: vlastní]**

Co se týče rozdílů v zobrazení mezi jednotlivými zařízeními, tak v této komponenta nejsou příliš velké rozdíly mezi tím, jak se zobrazuje na mobilu a jak na desktopu. Komponenty pro grafy jsou navrženy natolik chytře, že jsou plně fluidní – to znamená, že dokáží flexibilně přizpůsobit svojí šířku a výšku v závislosti na rozlišení daného koncového zařízení. Jediným rozdílem zde v podstatě spočívá v tlačítkách, jež slouží pro přepínání jednotlivých typů grafů. Na mobilu se tato tlačítka zobrazují dohromady přes celou šířku obrazovky. A na desktopu vyplňují obě tlačítka dohromady jen polovinu obrazovky.

---

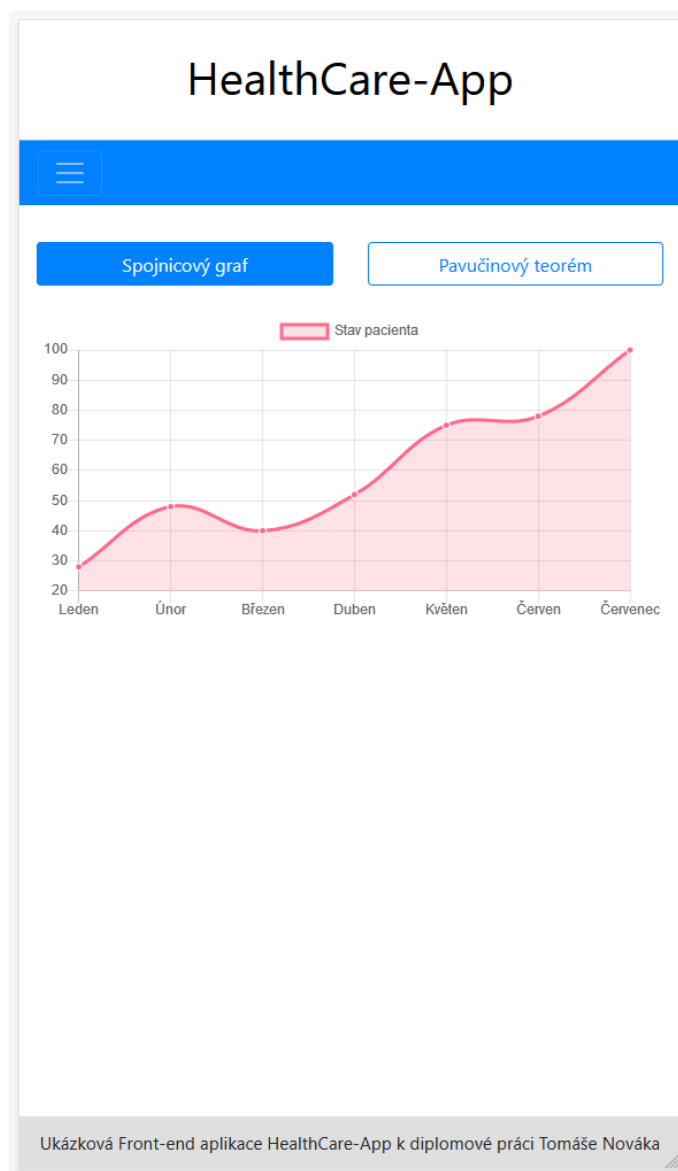
<sup>77</sup> Line Chart [online]. Dostupné z: <https://primefaces.org/primeng/showcase/#/chart/line>

<sup>78</sup> Radar Chart [online]. Dostupné z: <https://primefaces.org/primeng/showcase/#/chart/radar>

<sup>79</sup> Dependency Injection in Angular [online]. Dostupné z: <https://angular.io/guide/dependency-injection>



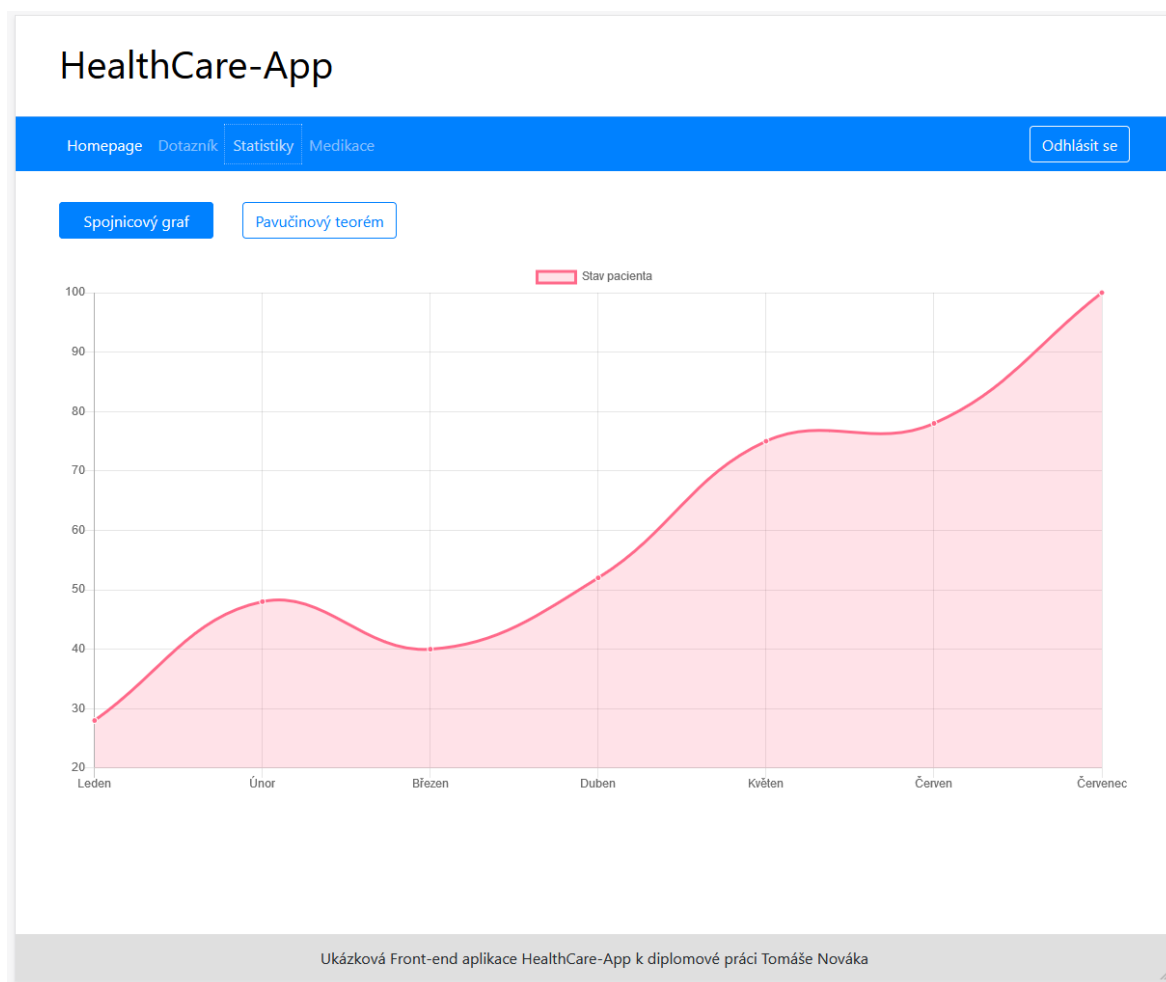
**Obrázek 25: Komponenta Statistics v mobilní verzi – spojnicový graf**



**Zdroj: vlastní zpracování**

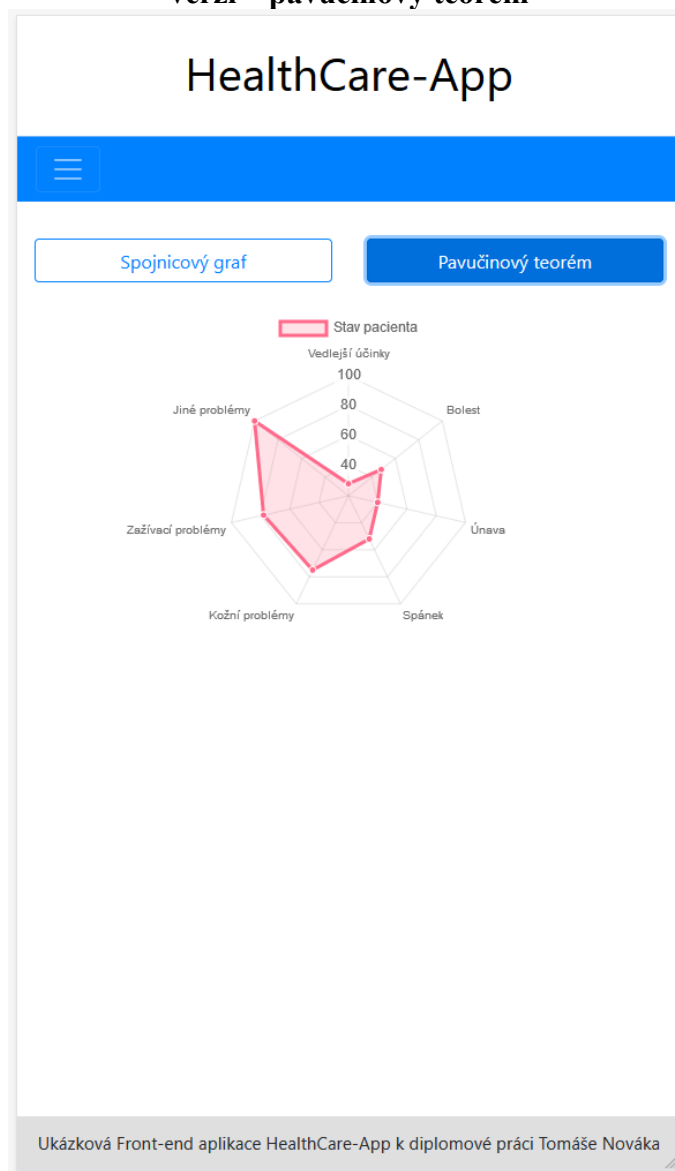


**Obrázek 26: Komponenta Statistics v desktopové verzi – spojnicový graf**



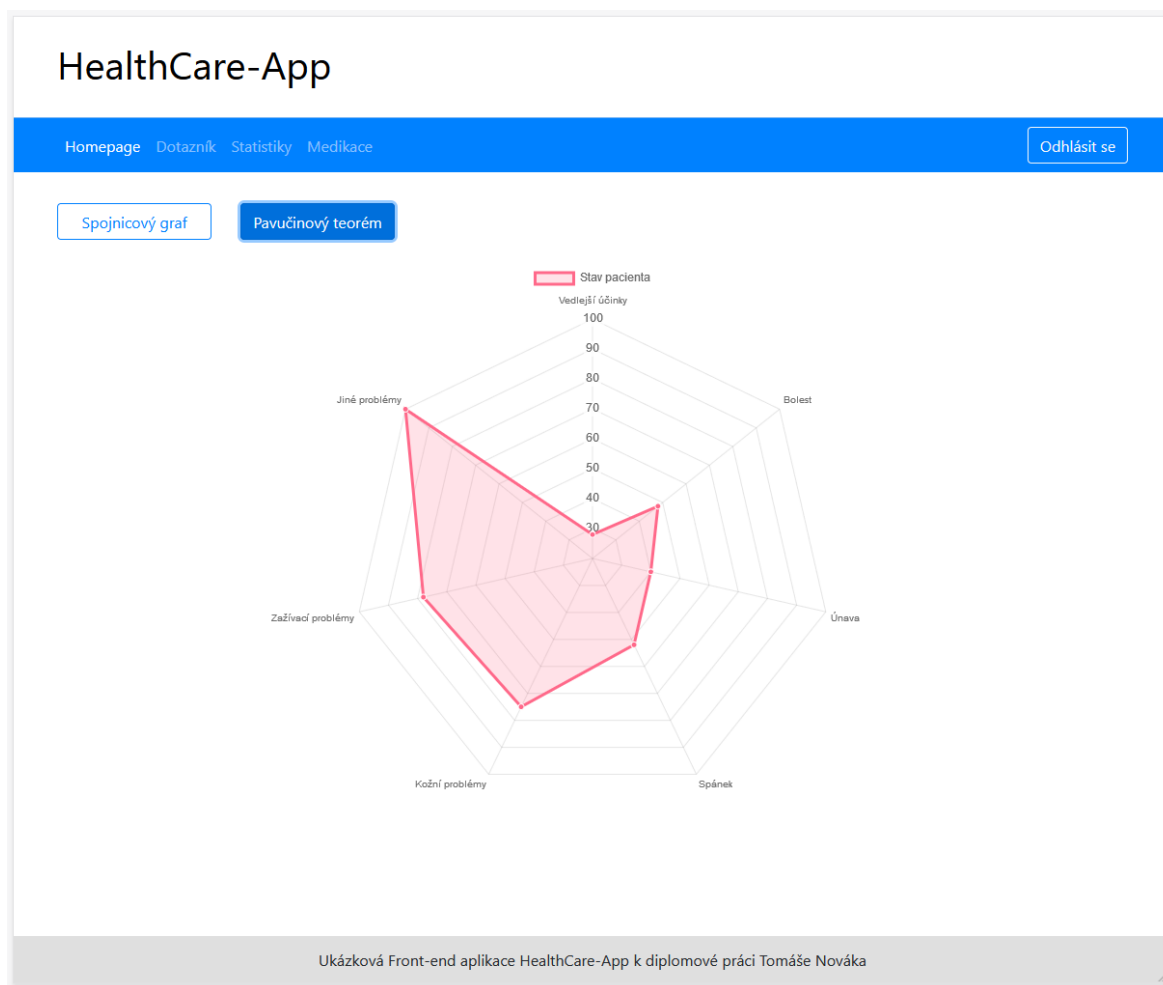
**Zdroj: vlastní zpracování**

**Obrázek 27: Komponenta Statistics v mobilní verzi – pavučinový teorém**



**Zdroj: vlastní zpracování**

**Obrázek 28: Komponenta Statistics v desktopové verzi – pavučinový teorém**



**Zdroj: vlastní zpracování**

#### **4.5.5. Komponenta Medication**

V komponentě Medication slouží pro správu medikace, kterou pacient v rámci své léčby užívá. Komponenta je rozdělena na 2 části:

1. Seznam všech medikací
2. Přidání nové medikace

Seznam všech medikací je v podstatě jednoduchá tabulka sloužící k výpisu medikamentů, které pacient užívá. Tabulka má tyto sloupce:

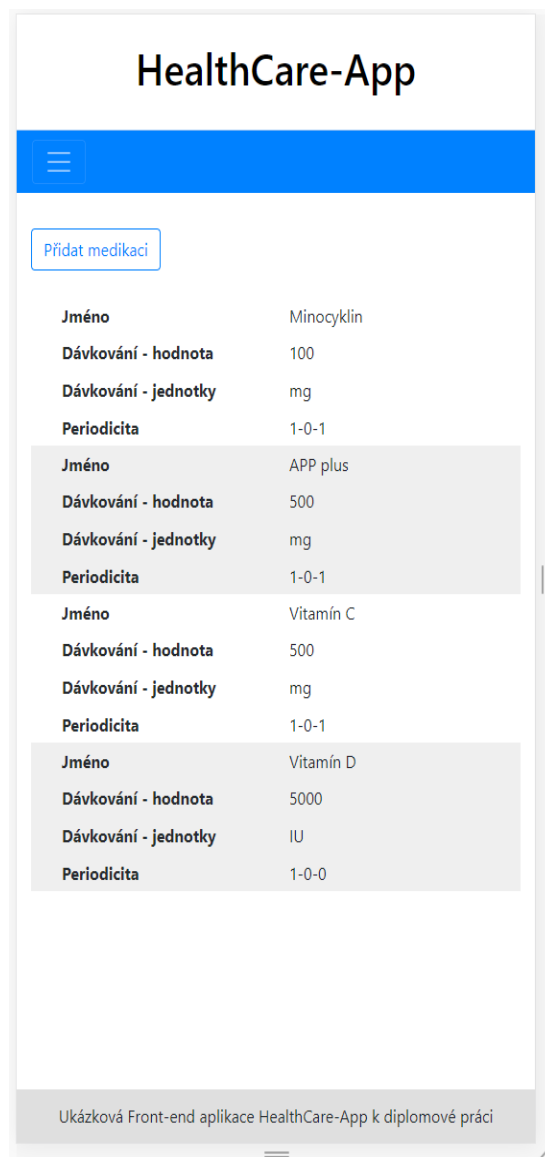
- Název
- Dávkování – hodnota
- Dávkování – jednotky

- Začátek medikace
- Konec medikace
- Periodicita

U této tabulky bylo důležité vyřešit její responzivitu. Tabulky totiž obvykle představují problém při návrhu mobilních verzí responzivních webů. A to z důvodu, že tabulky mají zpravidla sloupce umístěné vedle sebe a celá tabulka má tak orientaci na šířku. Kdežto mobily mají displej orientovaný na výšku. Pro řešení tohoto problému byla použita komponenta Table z PrimeNG <sup>80</sup>. Tato komponenta je velmi užitečná v tom, že ve svém API má implementovanou funkci „Responsive“. Při zapnutí této funkce se na mobilních verzí webů u tabulek automaticky otáčí sloupce a řádky. Tím se docílí žádoucího jevu, kdy se na mobilu zobrazují sloupce v tabulce pod sebou a tabulka pak má celkově orientaci na výšku.

---

<sup>80</sup> Table [online]. Dostupné z: <https://primefaces.org/primeng/showcase/#!/table>



**Obrázek: Komponenta Medication v mobilní verzi [zdroj: vlastní]**

# HealthCare-App

[Homepage](#) [Dotazník](#) [Statistiky](#) [Medikace](#) [Editace profilu](#)

[Odhlásit se](#)

[Přidat medikaci](#)

Jméno	Dávkování - hodnota	Dávkování - jednotky	Periodicita
Minocyclin	100	mg	1-0-1
APP plus	500	mg	1-0-1
Vitamín C	500	mg	1-0-1
Vitamín D	5000	IU	1-0-0

Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka

## Obrázek: Komponenta Medication v desktopové verzi [zdroj: vlastní]

Možnost přidání nové medikace je implementována tak, že po kliknutí na tlačítko „Přidat medikaci“ se otevře nová komponenta AddMedication, která umožňuje zadat informace o nové medikaci. Formulář je implementován jako standardní Reactive form v Angularu. Po úspěšném zadání nové medikace se tato komponenta zavře.



# HealthCare-App

Přidat medikaci

Název:

Dávkování - hodnota:

Dávkování - jednotka:

Začátek medikace:

dd . mm . rrrr

Konec medikace:

dd . mm . rrrr

Periodicita:

Uložit

<b>Jméno</b>	Minocyklin
<b>Dávkování - hodnota</b>	100
<b>Dávkování - jednotky</b>	mg

Ukázková Front-end aplikace HealthCare-App k diplomové práci

**Obrázek: Formulář pro přidání nové mobilní verzi [zdroj: vlastní]**

Přidat medikaci

Název:

Dávkování - hodnota:

Dávkování - jednotka:

Začátek medikace:

dd . mm . rrrr

Konec medikace:

dd . mm . rrrr

Periodicita:

Uložit

Jméno	Dávkování - hodnota	Dávkování - jednotky	Začátek medikace	Konec medikace	Periodicita
Minocyklin	100	mg	1.11.2020	30.11.2020	1-0-1
APP plus	500	mg	1.11.2020	30.11.2020	1-0-1
Vitamín C	500	mg	1.11.2020	30.11.2020	1-0-1
Vitamín D	1000	IU	1.11.2020	30.11.2020	1-0-0

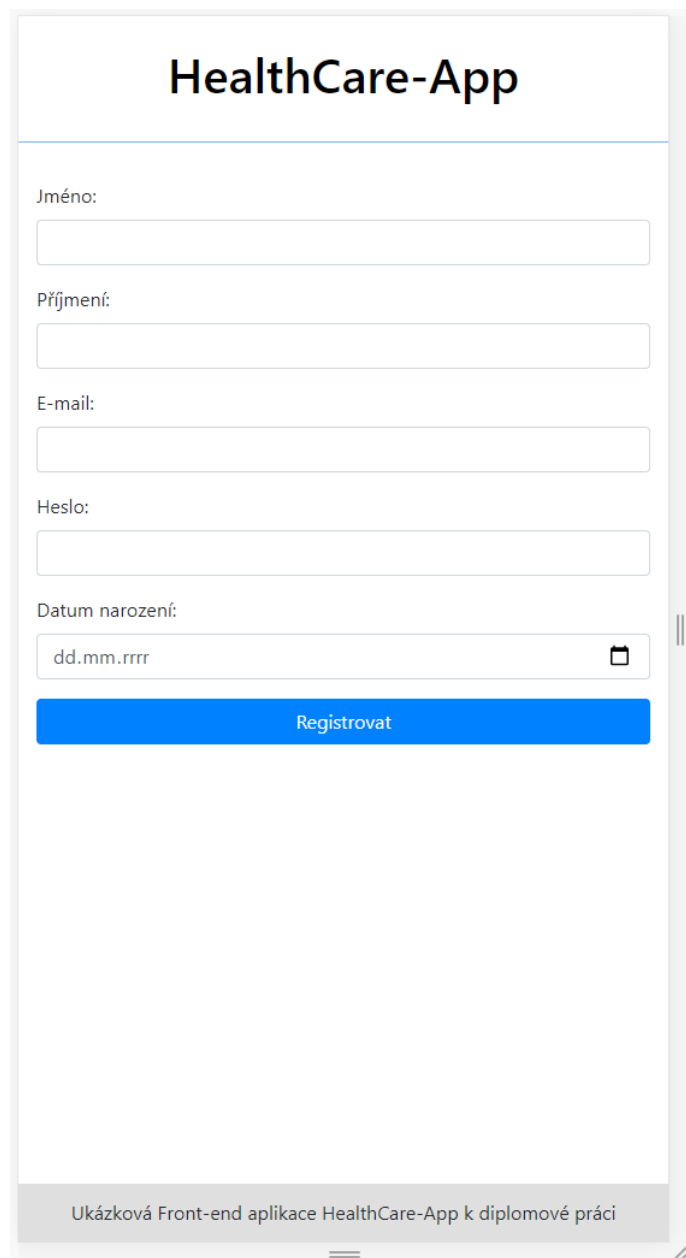
**Obrázek: Formulář pro přidání nové medikace v desktopové verzi [zdroj: vlastní]**

#### 4.5.6. Komponenta Profile

Komponenta Profile slouží pro registraci nového pacienta do aplikace. Tato komponenta je vybavena jednoduchým formulářem, který obsahuje data o daném pacientovi. Formulář byl navržen podobně jako u komponenty Login – tj. jako „Reactive form“ v Angularu.

Odlišnosti mezi mobilní a desktopovou vezi jsou u této komponenty podobné jako u komponenty Login. To znamená, že na mobilu jsou formulářové prvky přes celou šířku obrazovky, textové popisky (label) nad nimi a tlačítko „Registrovat“ je na mobilu zobrazeno také přes celou obrazovku.





The image shows a mobile application interface for registration. At the top, the title "HealthCare-App" is centered in a large, bold, black font. Below the title, there is a vertical stack of input fields, each preceded by a label: "Jméno:", "Příjmení:", "E-mail:", "Heslo:", and "Datum narození:". The "Datum narození:" field includes a date format placeholder "dd.mm.rrrr" and a calendar icon. A prominent blue button with the text "Registrovat" is positioned below the date field. At the bottom of the screen, a light gray footer bar contains the text "Ukázková Front-end aplikace HealthCare-App k diplomové práci". The entire interface is framed by a light gray border, with a hamburger menu icon on the right side.

# HealthCare-App


Jméno:

Příjmení:

E-mail:

Heslo:

Datum narození:




Registrovat

Ukázková Front-end aplikace HealthCare-App k diplomové práci

**Obrázek: Komponenta Profile v mobilní verzi [zdroj: vlastní]**

# HealthCare-App

---

Jméno:	<input type="text"/>
Příjmení:	<input type="text"/>
E-mail:	<input type="text"/>
Heslo:	<input type="password"/>
Datum narození:	<input type="text" value="dd.mm.rrrr"/> 
<input type="button" value="Registrovat"/>	

Ukázková Front-end aplikace HealthCare-App k diplomové práci Tomáše Nováka

**Obrázek: Komponenta Profile v desktopové verzi [zdroj: vlastní]**

## 4.6. Testování

Byly provedeny 2 druhy testování aplikace HealthCare-App:

1. Manuální testování pro různá koncová zařízení
2. Unit testy

### 4.6.1. Manuální testování pro různá koncová zařízení

Ukázková webová aplikace HealthCare-App byla otestována na prohlížečích, které jsou uvedené v analýze UI frameworků (v kapitole 4.3) jako podporované prohlížeče pro framework Twitter Bootstrap. Testování je zaměřeno na testování responzivního designu pro různě široké obrazovky. Testy byly záměrně zvoleny tak, aby šířky obrazovek byly identické s výchozími zlomovými body (breakpoint) v Bootstrapu, na kterém je Front-end aplikace postaven. Uživatelské testování tedy bylo provedeno pro obrazovky těchto šířek: 576 px, 768 px, 992 px a 1200 px. Testováno bylo zaměřeno na zobrazení všech komponent aplikace, jež jsou popsány v kapitole 4.5. Výsledek testování je takový, že se všechny moduly v aplikaci zobrazují správně a podle očekávání ve všech testovaných rozlišeních. Screenshots z jednotlivých testů jsou uvedeny v přílohách č. 1 až 4.

### 4.6.2. Unit testy

Unit testy slouží pro automatické otestování určité komponenty pro ověření, zda-li s určitým vstupem vrací v testu požadovaný výstup. V Angularu je doporučeno pro psaní unit testů používat Jasmine test framework<sup>81</sup>. Předmětem této práce není pokrýt unit testy celou funkcionalitu aplikace. Nicméně pro ukázkou byl napsán unit test pro otestování komponenty Login, kde bylo provedeno automatizované testování načtení komponenty, vykreslení (render) formuláře a následné vytvoření instance od objektu FormGroup.

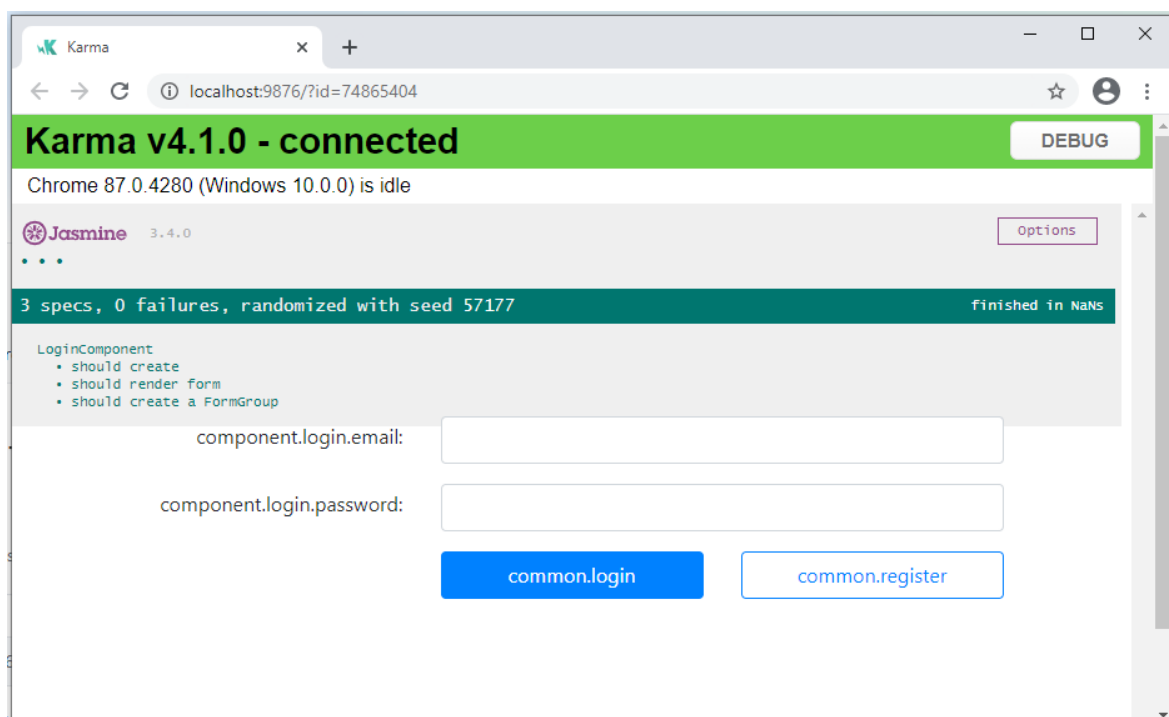
---

<sup>81</sup> Jasmine Documentation [online]. Dostupné z: <https://jasmine.github.io/>

```

15 describe( description: 'LoginComponent', specDefinitions: () => {
16     let component: LoginComponent;
17     let fixture: ComponentFixture<LoginComponent>;
18
19     beforeEach(async( fn: () => {
20         TestBed.configureTestingModule( moduleDef: {
21             declarations: [ LoginComponent ],
22             imports: [
23                 TranslateModule.forRoot(),
24                 ReactiveFormsModule,
25                 HttpClientTestingModule,
26             ],
27             providers: [
28                 {provide: Router, useValue: {} },
29                 {provide: ActivatedRoute, useValue: {} },
30                 {provide: MessageService, useValue: {} }
31             ]
32         })
33         .compileComponents();
34     }));
35
36     beforeEach( action: () => {
37         fixture = TestBed.createComponent(LoginComponent);
38         component = fixture.componentInstance;
39         fixture.detectChanges();
40     });
41
42     it( expectation: 'should create', assertion: () => {
43         expect(component).toBeTruthy();
44     });
45
46     it( expectation: 'should render form', assertion: () => {
47         const compiled = fixture.debugElement.nativeElement;
48         expect(compiled.querySelector('form')).toBeTruthy();
49     });
50
51     it( expectation: 'should create a FormGroup' assertion: () => {

```



Obrázek 30: Výsledek spuštěných unit testů pro komponentu Login

## 5. Výsledky a diskuse

### 5.1. Výsledky analýzy JavaScript frameworků

V oblasti JavaScript frameworků byla provedena dvoufázová analýza, kde se srovnávaly jednotlivé frameworky. Do výběru byly zařazeny frameworky Angular, React, Vue, Ember a Knockout. Za vývojem a následnou údržbou u všech těchto frameworků stojí nějaká vývojářská komunita. Komunitu může buď zastřešovat nějaká velká softwarová firma (např. Google nebo Facebook) a nebo se může jednat o komunitu nezávislých vývojářů. V první fázi byly z výběru vyřazeny ty frameworky, jejichž vývojářská komunita má malý počet přispěvatelů (contributor), příp. pokud tito přispěvatelé za uplynuté časové období (sledováno typicky za poslední rok a také za poslední měsíc). Z první fáze analýzy plyne, že dle informací z příslušného repozitáře na GitHubu mají ze strany vývojářské komunity velmi malou podporu frameworky Vue a Knockout. Proto byly tyto frameworky z dalšího výběru vyřazeny. Zbývají tedy frameworky Angular, React a Ember. U těchto zbývajících frameworků byla vybrána ta nejvhodnější varianta na základě provedené vícekritériální analýzy variant s využitím metody váženého součtu.

U vícekritériální analýzy variant výběr proběhl tak, že byla stanovena kritéria, která byla ohodnocena určitým počtem bodů. Kvantitativní (tedy objektivně měřitelné) je zde pouze kritérium č. 1 (Velikost frameworku). Ostatní kritéria (tj. Architektura a API, Nástroje a Komponenty) jsou naopak kvalitativní, což znamená, že jsou byly u jednotlivých variant přiřazovány body více subjektivně a to na základě vývojářských zkušeností autora této práce.

Všechny zmiňované frameworky umožňují pracovat s vlastními komponentami, které je možné do sebe ve stromové struktuře zanořovat a předávat mezi nimi data či notifikace o proběhlých událostech. Jako nejslabší z této trojice vychází Ember. Ember je svou architekturou sice relativně podobný Angularu. Avšak ta podoba je patrná spíše ve srovnání se starým AngularJS první verze, než se současným Angularem verze 2+. Ostatně první verze Emberu vyšla již v roce 2011 (rok po vydání prvního AngularJS) a v dnešní době se již moc nerozvíjí.

Proto Ember dostal ve vícekritériální analýze poměrně málo bodů. Tím dosáhl v celkovém užitku variant hodnoty pouze 0,044 a skončil ve vícekritériální analýze variant na 3. místě.



Zbývají tedy poslední 2 varianty a těmi jsou Angular a React. Zde autor této práce musí konstatovat, že se v současnosti jedná o 2 „hlavní“ JavaScript frameworky, které si mezi sebou do určité míry konkurují. A proto žádný z nich není vyloženě špatný a volba jednoho z nich je víceméně subjektivní.

Největší rozdíl mezi Angularem a Reactem spočívá v tom, že Angular má mnohem více oddělenou jednotlivé vrstvy aplikace a s tím souvisí i fakt, že Angular má více striktní pravidla co a jak v aplikaci implementovat. React je v těchto ohledech naopak mnohem volnější a závisí v podstatě na konkrétních vývojářích, jak svou aplikaci v Reactu z hlediska architektury uchopí.

V praxi to znamená, že v Angularu má vývojář již ve výchozí instalaci k dispozici téměř vše důležité, co potřebuje mít pro vývoj kvalitní aplikace, a nemusí zdaleka instalovat tolik různých pluginů jako v Reactu. U komponent jsou v Angularu od sebe důsledně odděleny HTML šablony, CSS styly a výkonná logika daných komponent. Dále veškeré služby (service), které se v komponentách v Angularu používají, tak musí být do dané komponenty vkládány (inject) pomocí návrhového vzoru Dependency Injection.

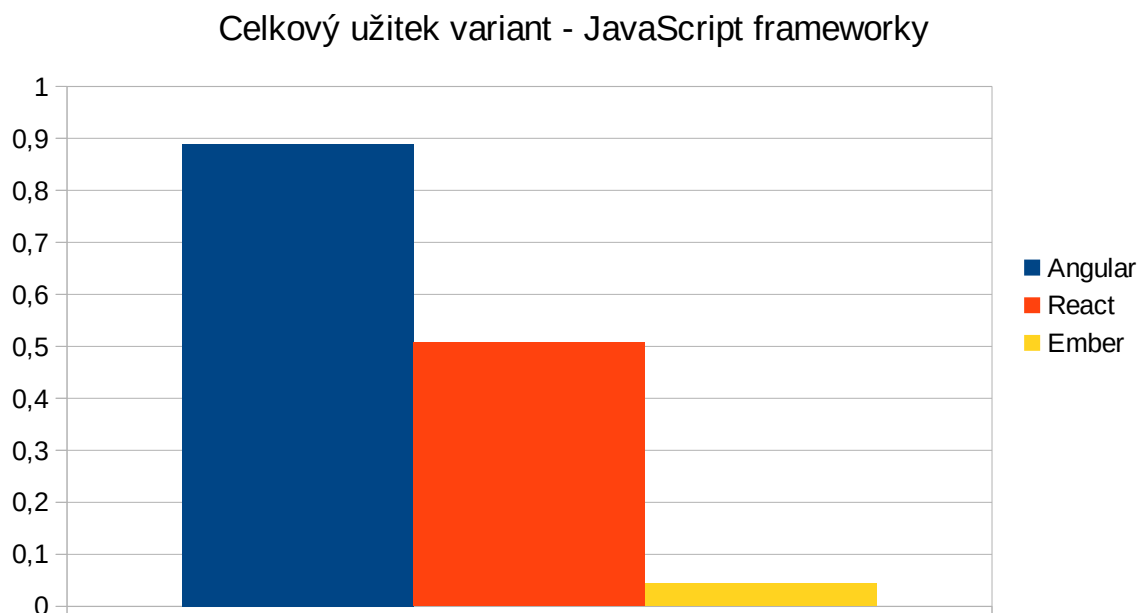
V Reactu tato pravidla naopak vyžadována nejsou a mnohdy ani nejsou při vývoji aplikací žádoucí. U komponent React používá JSX (nebo TSX) soubory. A to znamená, že spojeny dohromady HTML šablony a výkonná logika. Jediné nutné pravidlo u React komponent je to, že každá komponenta musí obsahovat metodu `render()`, ve které se nadefinuje celá HTML šablona dané komponenty a ta se vrátí jako v podstatě návratová hodnota z této metody. React je, na rozdíl od Angularu, ve své základní instalaci pouze jednoduchá knihovna pro práci s komponentami, do které se veškeré pluginy musí dodatečně doinstalovávat.

Taktéž CLI má Angular propracované lépe, neboť v CLI od Reactu jde v podstatě pouze vytvořit nový projekt React Create App.

Nelze jednoznačně říci, který z těchto přístupů je lepší a který horší. Protože jak řešení Angularu, tak i řešení Reactu, mají své výhody a nevýhody. Autor této práce má ze své komerční vývojářské praxe zkušenosti, že učitelská křivka vývojářů je u Reactu o něco strmější, než u Angularu. To v praxi znamená, že vývojář, který nezná ani jeden z těchto frameworků, tak je zpravidla schopen rychleji něco vyvinout v Reactu, než v Angularu. Neboť Angular má skutečně mnoho různých pravidel, které než dotyčný vývojář pochopí, tak to nějaký čas trvá. Na druhou stranu zkušenosti autora této práce říkají, že tato rychlá učitelská křivka Reactu má i své nevýhody. V Reactu totiž jde poměrně rychle napsat nějaká jednodušší aplikace. Avšak pokud dotyčný vývojář nezná přesně, které pluginy na tu či onu funkcionalitu použít, tak výsledek často bývá aplikace nepřiliš vysoké kvality. Proto v

Reactu je důležité, aby paradoxně z důvodu jeho jednoduchosti a velké volnosti pracovali spíše senioři, kteří mají dostatek zkušeností a vědí, jak správně aplikaci v Reactu napsat.

Následující graf znázorňuje hodnoty celkového užítu jednotlivých variant, který byl vypočten právě metodou váženého součtu. Spolu s konkrétními hodnotami tento graf vlastně zobrazuje i pořadí, jak dopadly jednotlivé frameworky ve vícekritériální analýze.



**brázek 31: Celkový užitek variant – JavaScript frameworky**

Co se týče výsledků vícekritériální analýzy variant, tak použitím metody váženého součtu dosáhl Angular celkového užítu 0,890 a 0,507. Autora této práce poněkud překvapilo, že je zde poměrně velký rozdíl mezi hodnotou celkového užítu Angularu oproti celkovému užítu Reactu.

Každopádně autor této práce tedy na základě provedené vícekritériální analýzy preferuje framework v podobě robustního řešení, které ve svém základu obsahuje všechny důležité prvky pro vývoj aplikace. A proto ukázková aplikace HealthCare-App, jež je součástí této práce byla vyvinuta právě v **Angularu**.

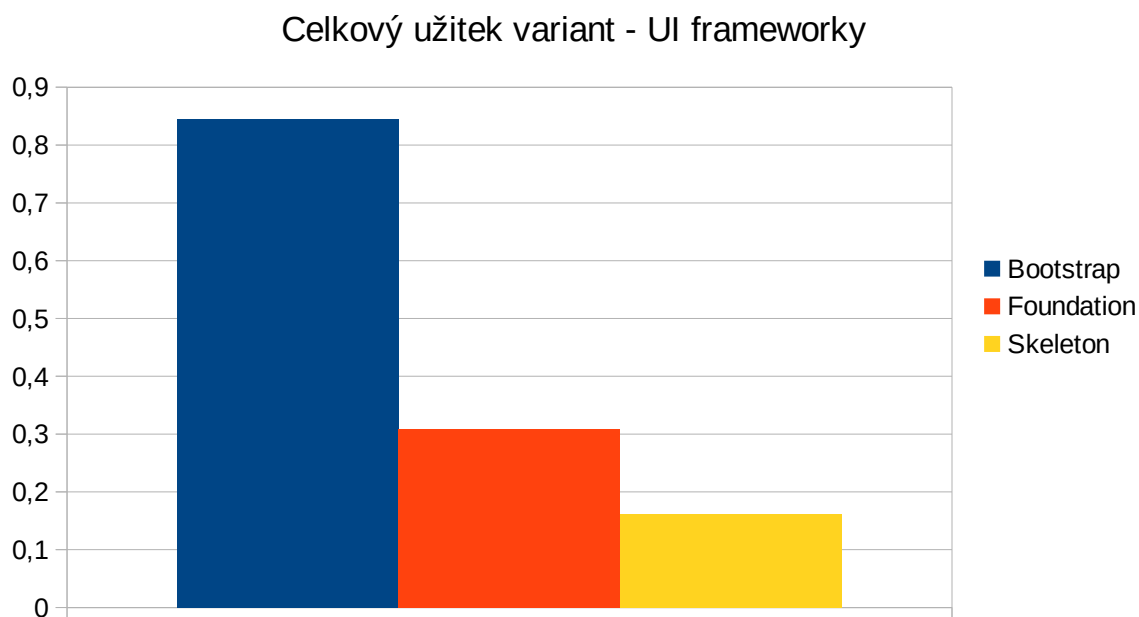
## **5.2. Výsledky analýzy UI frameworků**

Volba vhodného UI frameworku je dalším důležitým aspektem, který je potřeba řešit při vývoji Front-endu responzivních webových aplikací. Byl vypracován popis možností jednotlivých UI frameworků a následně byla pro výběr vhodného frameworku provedena vícekritériální. Analýza variant s využitím metody váženého součtu.

Jako varianty byly výběru byly zahrnuty tyto frameworky: Twitter Bootstrap, ZURB Foundation a Skeleton. U vícekritériální analýzy variant výběr proběhl tak, že byla stanovena kritéria, která byla ohodnocena určitým počtem bodů. Kvantitativní (tedy objektivně měřitelné) je zde pouze kritérium č. 1 (Velikost frameworku). Ostatní kritéria (tj. Mřížka a zlomové body, Responzivní ovládací prvky, Customizace a Podpora webových prohlížečů) jsou naopak kvalitativní, což znamená, že jsou byly u jednotlivých variant přiřazovány body více subjektivně a to na základě vývojářských zkušeností autora této práce.

Všechny uvedené varianty mají v sobě implementovanou mřížku (Grid System) a nějaké zlomové body (Breakpoints), což je tím nejdůležitějším kritériem pro vývoj responzivních webových aplikací. Bootstrap a Foundation jsou si mezi sebou do určité míry podobné. Avšak Foundation má ve vícekritériální analýze méně bodů, neboť nabízí o něco méně možností, než Bootstrap. Taktéž Skeleton nabízí méně možností, než Bootstrap a dokonce i než Foundation. Protože ve Skeletonu má vývojář k dispozici v podstatě jen mřížku (Grid System) a zlomové body (Breakpoints). Jakékoliv responzivní ovládací prvky či možnosti customizace ve Skeletonu nejsou. Výhodou Skeletonu na druhou stranu je, že jako framework zabírá pouze 25 kB, což je oproti ostatním variantám jednoznačně nejméně.

Následující graf znázorňuje výsledek vícekritériální analýzy variant – konkrétně celkový užitek variant, který byl vypočten metodou váženého součtu. Kromě konkrétních hodnot celkového užitku variant tento graf znázorňuje i výsledné pořadí v hodnocení jednotlivých variant.



**brázek 32: Celkový užitek variant – UI frameworky**

Co se týče výsledků vícekriteriální analýzy, tak nejhůře dopadl u Skeletonu – jeho celkový užitek dosahuje hodnoty pouze 0,162. Na druhém místě skončil ZURB Foundation s výsledkem 0,309. A nejlépe dopadl Twitter Bootstrap, jehož celkový užitek varianty je 0,846. Autora této práce překvapil poněkud velký rozdíl v hodnotách celkového užitku mezi frameworky Foundation a Bootstrap, neboť tyto dva frameworky si mezi sebou do určité míry konkurují. Nicméně Foundation opravdu nabízí o něco méně možností, než Bootstrap. A proto bylo objektivní mu v některých kritériích udělit méně bodů.

Na základě provedení vícekriteriální analýzy variant byl tedy jako UI framework pro ukázkovou aplikaci HealthCare-App použit **Twitter Bootstrap**, který dosáhl v metodě váženého součtu ze všech variant nejlepšího výsledku.

## 6. Závěr

Tématem této diplomové práce byl Front-end design webových aplikací. Hlavním cílem práce bylo analyzovat a porovnávat způsoby tvorby moderních webových aplikací s nimi souvisejících technologií zaměřených na Front-end design. Dílčími cíli práce bylo porovnat mezi sebou možnosti nejrozšířenějších JavaScript a UI frameworků a na základě analyzovaných poznatků vytvořit ukázkovou aplikaci.

Jedná se tedy o práci zaměřenou především na moderní trendy v oblasti návrhu webových aplikací. Těmi jsou především responzivní design a koncept tzv. Single-Page aplikací.

Navrhovat weby s ohledem na pravidla responzivního designu je důležité především z toho důvodu, že uživatelé si v dnešní době weby zdaleka neprohlíží jen na desktopech. Naopak, jak bylo v práci popsáno, tak převládají uživatelé, kteří si weby prohlíží na mobilních telefonech. A pro tuto skupinu uživatelů je potřeba navrhovat design webů poněkud odlišně a v ideální případě při tom dodržovat princip Mobile First.

V práci byla provedena srovnávací analýza celkem pěti JavaScript frameworků. Jednalo se o frameworky Angular, React, Vue, Ember a Knockout. Zde proběhl dvoufázový výběr té nejvhodnější varianty. V první fázi byly vyřazeny ty frameworky, které mají velmi nízkou podporu zastřešující vývojářské komunity. Jednalo se o frameworky Vue a Knockout. U zbylých variant (Angular, React a Ember) byla provedena vícekritériální analýza variant s použitím metody váženého součtu, jejímž cílem bylo nalézt variantu s nejvyšším celkovým užitekem (tj. kompromisní variantu). Tou se stal na základě této analýzy Angular, který tak byl použit pro vývoj ukázkové aplikace.

Kromě analýzy JavaScript frameworků byla v práci provedena také srovnávací analýza celkem tří UI frameworků, jejichž využití v aplikaci velmi výrazně pomůže při návrhu responzivního designu. Konkrétně se jednalo o UI frameworky Twitter Bootstrap, ZURB Foundation a Skeleton. Taktéž zde byla provedena vícekritériální analýza a jejím výsledkem je, že Twitter Bootstrap má v této analýze nejvyšší celkový užitek varianty a představuje tedy kompromisní variantu. Jako UI framework tak byl vybrán právě Bootstrap.

Ukázková aplikace nese pracovní název „HealthCare-App“ a jedná se o aplikaci z oblasti zdravotnictví. Aplikace je zaměřena na monitorování zdravotního stavu pacientů formou pravidelně vyplňovaných dotazníků ohledně klinického stavu ze strany pacienta. Hlavní přínos takové aplikace spočívá ve zlepšení komunikace mezi lékařem a jeho pacienty v případě, že dotyčný lékař u těchto pacientů léčí nějaký dlouhodobý chronický

zdravotní problém. Neboť v takovém případě je se hodí, pokud má lékař k dispozici průběžně získávaná data o zdravotním stavu svého pacienta. Dalším přínosem této aplikace může být fakt, že díky pravidelnému vyplňování dotazníků ze strany pacientů se lékař může prioritně dozvědět, pokud dotyčný pacient má nějaký závažný problém (např. náhlé výrazné zhoršení zdravotního stavu) a může na něj tak prioritně i reagovat.

U aplikace byl v rámci této práce navržen Front-end s využitím frameworků Angular a Twitter Bootstrap. Aplikace samotná je však z hlediska reálného využití spíše ve fázi konceptu. Současný koncept aplikace jako data používá mocky ve formátu JSON. V případě, že by byla aplikace nasazena v produkčním prostředí (např. u nějakého lékaře), tak k ní musí být naprogramován ještě Back-end, který bude obsluhovat jednotlivé požadavky přicházející z Front-endu. Autora této práce nápad na realizaci aplikace tohoto typu velmi zaujal. A proto je poměrně pravděpodobně, že bude na vývoji aplikace pracovat i po odevzdání této diplomové práce. Předmětem pokračujících prací bude pak především implementace Back-endu.

Závěrem této práce byl vypracovaný Front-end aplikace otestován a to dvěma způsoby. První způsob testování představovalo uživatelské testování na různých rozlišeních obrazovky s cílem otestovat chování aplikace na různých koncových zařízeních. V této souvislosti bylo provedeno testování na obrazovkách o šířce 576 px, 768 px, 992 px a 1200 px. Následně byla realizována ukázka testování v podobě unit testů. Pro tento účel byl napsán ve frameworku Jasmine unit test pro komponentu Login.

## 7. Seznam použitých zdrojů

1. Angular 8 – User Registration and Login Example & Tutorial [online]. Dostupné z: <https://jasonwatmore.com/post/2019/06/10/angular-8-user-registration-and-login-example-tutorial>
2. Angular - documentation [online]. Dostupné z: <https://angular.io/>
3. Angular – One framework. Mobile & Desktop. [online]. Dostupné z: <https://github.com/angular/angular>
4. CASTRO, Elizabeth a Bruce HYSLOP. HTML5 a CSS3: názorný průvodce tvorbou WWW stránek. 1. vydání. Brno: Computer Press, 2012. 440 s. ISBN 978-80-251-3733-8.
5. Co to je „Mobile First“? Ale doopravdy [online]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/mobile-first>
6. CSS Media Queries [online]. Dostupné z: [https://www.w3schools.com/css/css3\\_mediaqueries.asp](https://www.w3schools.com/css/css3_mediaqueries.asp)
7. Ember [online]. Dostupné z: <https://github.com/emberjs/ember.js>
8. Ember.js – A framework for ambitious web developers [online]. Dostupné z: <https://emberjs.com/>
9. FINGER, A. *User interface of system ERIAN based on web technologies*. Praha, 2019. Diplomová práce. MFF UK. Vedoucí práce Martin Nečaský.
10. HTML & CSS [online]. Dostupné z: <https://www.w3.org/standards/webdesign/htmlcss.html>
11. HOROWITZ, Richard I. *How Can I Get Better?: An Action Plan for Treating Resistant Lyme & Chronic Disease*. 1. vydání. New York: St. Martin's Press, 2017. 480 s. ISBN 978-1-250-07054-8.
12. CHOW, S. *Programujeme Mashup aplikace pro Web 2.0 v PHP*. Brno: Computer Press, 2008. 282 s. ISBN 978-80-251-2057-6.
13. Jasmine Documentation [online]. Dostupné z: <https://jasmine.github.io/>
14. jQuery - documentation [online]. Dostupné z: <https://jquery.org/>
15. KADLEC, Tim. *Responzivní design profesionálně*. 1. vydání. Brno: Zoner Press, 2014. 248 s. ISBN 978-80-7413-280-3.
16. Knockout [online]. Dostupné z: <https://github.com/knockout/knockout>
17. KOPECKÝ, L. *Analýza CSS frameworků pro tvorbu responzivního designu webových stránek*. Praha, 2018. Diplomová práce. PEF ČZU. Vedoucí práce Jan Jarolímek.

18. Localization vs. Internationalization [online]. Dostupné z:  
<https://www.w3.org/International/questions/qa-i18n>
19. LUBBERS, Peter, Brian ALBERS a Frank SALIM. HTML5: programujeme moderní webové aplikace. 1. vydání. Brno: Computer Press, 2011. 304 s. ISBN 978-80-251-3539-6.
20. MARGORÍN, Marián. JQuery bez předchozích znalostí. 1. vydání. Brno: Computer Press, 2011. 256 s. ISBN 978-80-251-3379-8.
21. ngx-translate [online]. Dostupné z: <https://github.com/ngx-translate/core>
22. NOVÁK, T a kol. *Projekt - Řízení IT projektů*. Praha, 2020. Semestrální práce. PEF ČZU.
23. React - documentation [online]. Dostupné z: <https://reactjs.org/>
24. React [online]. Dostupné z: <https://github.com/facebook/react>
25. Sym Collect [online]. Dostupné z: <https://symcollect.de/>
26. ŠUBRT, Tomáš. Ekonomicko-matematické metody. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. 351 s. ISBN 978-80-7380-345-2.
27. Twitter Bootstrap - documentation [online]. Dostupné z:  
<https://getbootstrap.com.org/>
28. Vue [online]. Dostupné z: <https://github.com/vuejs/vue>
29. Vue.js - documentation [online]. Dostupné z: <https://vuejs.org/>
30. W3.CSS Responsive Fluid Grid [online]. Dostupné z:  
[https://www.w3schools.com/w3css/w3css\\_grid.asp](https://www.w3schools.com/w3css/w3css_grid.asp)
31. ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. 1. vydání. Brno: Computer Press, 2015. 184 s. ISBN 978-80-251-4573-9.



## **8. Přílohy**

### **8.1. Příloha č. 1**

**(todo)** Tady budou screenshoty výsledků testování pro rozlišení 576 px

## **8.2. Příloha č. 2**

**(todo)** Tady budou screenshoty výsledků testování pro rozlišení 768 px

### **8.3. Příloha č. 3**

**(todo)** Tady budou screenshoty výsledků testování pro rozlišení 992 px

#### **8.4. Příloha č. 4**

**(todo)** Tady budou screenshoty výsledků testování pro rozlišení 1200 px

## **8.5. Příloha č. 5**

### **Návod na instalaci aplikace HealthCare-App**

Text...