Aplikace pro vyřizování pohovorů

Bc. Marek Bařina

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek Bařina**
Osobní číslo:	**A15836**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Aplikace pro vyřizování pohovorů**
Téma anglicky:	**Application for Interviewing Employee Candidates**

Zásady pro vypracování:

1. **Definujte funkční a nefunkční požadavky aplikace pro vyřizování pohovorů.**
2. **Požadavky analyzujte a vytvořte specifikaci pro implementaci.**
3. **Prostudujte dostupné technologie pro implementaci uživatelského rozhraní a serverové části.**
4. **Vyberte technologii, která nejlépe splňuje požadavky zadání.**
5. **Pomocí vybrané technologie aplikaci implementujte.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **STEFANOV, Stoyan. React: Up & Running: Building Web Applications. O'Reilly Media, 2016. ISBN 978-1491931820.**
2. **AQUINO, Chris a Todd GANDEE. Front-end web development: the Big Nerd Ranch guide. ISBN 978-0134433943.**
3. **SIMPSON, Kyle. You Don't Know JS: Up & Going. O'Reilly Media, 2015. ISBN 978-1491924464.**
4. **UGURLU, Tugberk., Alexander. ZEITLER a Ali. KHEYROLLAHI. Pro ASP.NET web API: HTTP web services in ASP.NET. Berkeley, CA: Apress, 2013. Expert's voice in .NET. ISBN 978-1430247258.**
5. **KURTZ, Jamie. ASP.NET MVC 4 and the Web API: building a REST service from start to finish. Expert's voice in ASP.NET. ISBN 978-1430249771.**
6. **ASP.NET: Web API [online]. 2015 [cit. 2017-02-02]. Dostupné z: https://www.asp.net/web-api/overview**
7. **TutorialsPoint: Learn ReactJS [online]. 2015 [cit. 2017-02-02]. Dostupné z: https://www.tutorialspoint.com/reactjs/**

Vedoucí diplomové práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. února 2017**

Termín odevzdání diplomové práce: **16. května 2017**

Ve Zlíně dne 3. února 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Cílem této práce byl vývoj aplikace pro vyřizování pohovorů. Analyzoval jsem funkční a nefunkční požadavky aplikace a sestavil návrh implementace včetně UML diagramů. Během tohoto vývoje jsem také porovnal nejběžnější používané technologie pro vývoj klientské a serverové části aplikací a vybral z nich technologie nejvhodnější pro použití v mém řešení. Poté jsem navrženou aplikaci implementoval a popsal využité knihovny a zdrojový kód.

Klíčová slova: Vývoj softwaru, C#, .NET, Javascript, ReactJS, MS SQL

ABSTRACT

The aim of this thesis was the development of applications for interviews. I analyzed the functional and non-functional requirements of the application and prepare the implementation design, including all diagrams. During this development, I also compared the most common technologies used to develop client-side and server-side applications, and chosed the most apropriate technologies which I used in my solution. I then implemented the application and described the used libraries and source code.

Keywords: Software development, C#, .NET, Javascript, ReactJS, MS SQL

Chtěl bych poděkovat své rodině za podporu během studia, protože bez nich bych se studiu věnovat nemohl. Dále bych chtěl poděkovat vedoucímu své práce Ing. Tomáši Dulíkovi za možnost pracovat na tomto zajímavém tématu.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

„Kdo víno má a nepije, kdo hrozny má a nejí je, kdo ženu má a nelíbá, kdo zábavě se vyhýbá, na toho vemte bič a hůl, to není člověk, to je vůl.“

- *Jan Werich*

OBSAH

ÚVOD

Téma této práce jsem si zvolil při příležitosti vývoje aplikace pro zefektivnění recruitment procesu ve firmě, ve které pracuji jako .Net developer. Tato aplikace dostala interní název InterviewMe. V této práci jsem popsal postup vývoje softwaru od analýzy požadavků až po implementaci. V teoretické části jsem se zaměřil na obecné věci týkající se analýzy, návrhu a vývoje softwaru a porovnal jsem běžně používané technologie pro implementaci serverové a klientské části aplikace. Toto rozdělení je hlavně z důvodu, že jeden z hlavních nefunkčních požadavků aplikace bylo kompletní oddělení těchto částí jakožto i udržitelnost kódu. Nefunkční požadavky aplikace byly hlavním faktorem při výběru technologií a frameworků pro uživatelské rozhraní a serverovou část.

Funkcionální požadavky na aplikaci byli především správa kandidátů, vytváření testů, možnost testy přiřazovat ke kandidátům a hodnotit je, správa otevřených pozic, možnost aby kandidát mohl testy pomocí aplikace vyplňovat a další detailnější požadavky.

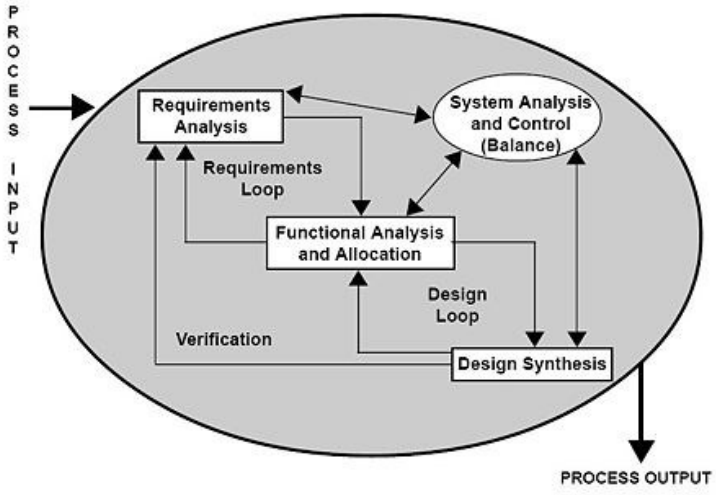
I. TEORETICKÁ ČÁST

1 ANALÝZA POŽADAVKŮ

Analýza požadavků je prvním stádiem v procesu systémového inženýrství a procesu softwarového vývoje. Je to inženýrská disciplína pro stanovení požadavků uživatelů a určení softwarových systémů. Systematická analýza požadavků je spíše známa jako: „*requirements engineering"* (RE). Někdy je také (špatně) pojmenovaná jako sběr požadavků, nebo specifikace požadavků. Pojem analýza požadavků může být také pokládán za analýzu v právem slova smyslu (jako příklad opaku k sběru nebo dokumentaci požadavku). Analýza požadavků v systémovém a softwarovém inženýrství, pojímá ty úkoly, které vstupují do rozhodování o potřebách a podmínkách, které jsou kladeny na nový, nebo změněný produkt. Také musí brát v úvahu různé protichůdné požadavky účastnících se stran tzv. **stakeholderů** jako uživatelů, nebo jiných účastníků využívajíc výsledných efektů.

Existuje mnoho definic analýzy požadavků, nicméně všechny sdílejí myšlenku, že požadavky zahrnují zjištění toho, co lidé chtějí od počítačového systému a pochopení toho, co jejich požadavky znamenají z hlediska designu. Analýza požadavků úzce souvisí se softwarovým inženýrstvím, které se více zaměřuje na proces navrhování systému, který uživatelé chtějí. Možná nejsrozumitelnější shrnutí pochází od Barryho Boehma: "navrhování správné věci", na rozdíl od softwarového inženýrství "navrhovat věci správně" (Boehm, 1981). Nuseibeh a Easterbrook (2000) poskytují komplexnější definici: „analýza požadavků softwarových systémů je proces objevování účelu systému tím, že identifikujeme zúčastněné strany a jejich potřeby a dokumentujeme je ve formě, která je přístupná analýze, komunikaci a následné implementaci" [1].

Analýza požadavků sdílí mnoho konceptů a technik interakce člověka s počítačem (HCI), zejména návrh orientovaný na uživatele, participační design a interakce. Rozlišuje se však od HCI vzhledem k rozsahu konstrukce. Například socio-technický design je zřídka zmíněn v analýze požadavků, kde organizační a lidská část systému je explicitním cílem požadavků a designu systému. Druhý rozdíl spočívá v tom, že se HCI soustředí na design jako takový a interakční návrh, kde jsou uživatelské požadavky považovány za součást procesu průzkumu návrhu, prototypování a dialogu s uživatelem o vyhodnocení, spíše než jako lineární proces "Specifikovat-Navrhnout-Implementovat" upřednostňovaný v analýze požadavků. Nicméně v analýze požadavků se jistě přebírá iterativní návrh, prototypování a proces vyhodnocení (obvykle se používá termín validace v terminologii požadavků) [1].



Obrázek 1 – Zjednodušený proces analýzy požadavků a tvorby návrhu [2]

Analýza požadavků je kritická ve vztahu k úspěšnému dokončení vývoje systému. Požadavek musí být proveditelný, měřitelný, testovatelný, musí se vztahovat ke konkrétnímu byznys požadavku, nebo příležitosti a také musí být definovaný dostatečné detailně pro účely návrhu systému.

1.1 Činnosti a procesy v analýze požadavků

Základními činnostmi v procesu analýzy požadavků jsou průzkum, shromažďování informací (požadavků), analýza, modelování a validace.

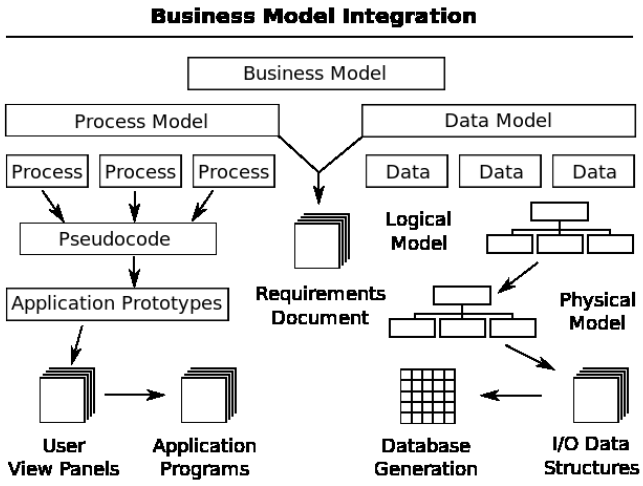
1.1.1 Průzkum

Začátek procesu často začíná nejasným projevem záměru. Prvním problémem je stanovení hranice vyšetřování a mimo jiné rozsah zamýšleného systému. Bohužel je to zřídka snadný úkol, protože klienti často nevědí přesně co chtějí a znalosti o zamýšleném systému jsou nejasné. Průzkum má tendenci být iterační aktivitou, neboť hranice jsou jasnější se zvyšujícím se pochopením oblasti sdílené všemi zainteresovanými stranami (stakeholdery). Tento proces je však špatně pochopen a jen málo výzkumu se přímo zabýval tímto obtížným problémem [1].

Vezměte například systém, který by pomohl epidemiologům při jejich výzkumu. Zainteresované strany by mohly zahrnovat lékařské analytiky se zájmem o epidemiologii, stejně jako lékaře. Rozsah možných nástrojů pro podporu rozhodování by mohl zahrnovat sběr dat, přípravu dat, statistickou analýzu, vizualizaci, grafy, mapy a také skupinovou diskuzi o výsledcích. Vlastník systémů a rozsah nebyly zřejmé, protože projekt byl zahájen v rámci programu

výzkumu elektronických věd podporovaného vládou Spojeného království, s uživateli, kteří byli akademickými výzkumníky epidemiologie a kteří také spolupracovali s lékaři z místních nemocnic [1].

V obecné oblasti průzkumu poskytuje podnikatelské modelování (enterprise modeling) způsob popisu podnikatelského kontextu, aby se objevily požadavky ve velkém (tjn. všechny cíle, účely a politiky). Workshopy v metodě brainstormingu KJ, pojmenované podle jeho vynálezce Jiro Kawakita a *„Rapid Applications Development"* jsou současným trendy v této oblasti. Obhajují používání seznamů a neformálních map problémového prostoru, i když tyto metody nabízejí jen málo systematické vedení. Podrobnější určení rozsahu průzkumu bylo zkoumáno Jacksonem a Zavem, kteří navrhli techniky pro stanovení hranice systému zkoumáním povinností a úkolů zamýšleného systému při reagování na události v reálném světě, ačkoli to nepomáhá omezovat vyšetřování, která začíná obecnými prohlášeními záměrů uživatelů [1].



Obrázek 2 – podnikatelské modelování [3]

Průzkumu je nejlépe dosaženo diskusí se všemi zainteresovanými stranami a zdokumentováním cílů systému na vysoké úrovni. Vypracování rozsahu působnosti má tendenci soustředit pozornost uživatelů na to, kde by mělo ležet systémové vyšetřování, a pomáhá identifikovat alespoň počáteční prostor pro systém [1].

1.1.2 Shromažďování informací

Většina technik pro tuto činnost byla vypůjčena z analýzy systémů, např. Rozhovory, pozorování, dotazníky, analýza textu a dokumentů. Byly použity techniky získávání znalostí, jako jsou repertoární sítě a protokolová analýza, ale kromě předběžné studie Maiden a Rugga

(1994) nebyly systematicky zkoumány zásluhy různých technik zachycování faktů. Zajímavou vznikající oblastí je využití etnografických a spojených pozorovacích metod (Goguen & Linde, 1993; Luff et al., 1993). Většina z nich však není schopna poskytnout explicitní návod k zachycení nebo analýze faktů, což vedou softwarové inženýry k tomu, aby navrhli své vlastní rychlé a špinavé přístupy [1].

1.1.3 Analýza

Analýza a modelování obecně vycházejí z přístupů shora dolů a soustředí se na rozklad cílů. Analýza je často motivována základními otázkami, takzvaných pět "W" (z anglického jazyka):

- Jaký je účel systému (cíle)?
- Jaké objekty jsou zahrnuty?
- Kde je systém umístěn?
- Kdy by se měly věci dělat?
- Proč je systém nezbytný (cíle nebo problémy, které hodlá řešit)?

Cílově orientovaná analýza používá scénáře k objevení překážek nebo potenciálních problémů způsobených externími činiteli, s nimiž se systém musí vyrovnávat. Z překážek lze formulovat cíle pro udržení a vyvarovat se a opravám problémů. Jiné metody rozkladu cílů sledují taxonomický přístup a pokoušejí se analyzovat cíle v kontextu doménových modelů. Pro analýzu problémů metodika měkkých systémů poskytuje prostředky neformálního modelování a analytického přístupu k objevování problémově orientovaných požadavků. Neformální diagramy a náčrty, které mohou být označovány jako doménové modely nebo bohaté obrázky, se používají k dokumentaci analýzy v průběhu jejího postupu [1].

1.1.4 Modelování

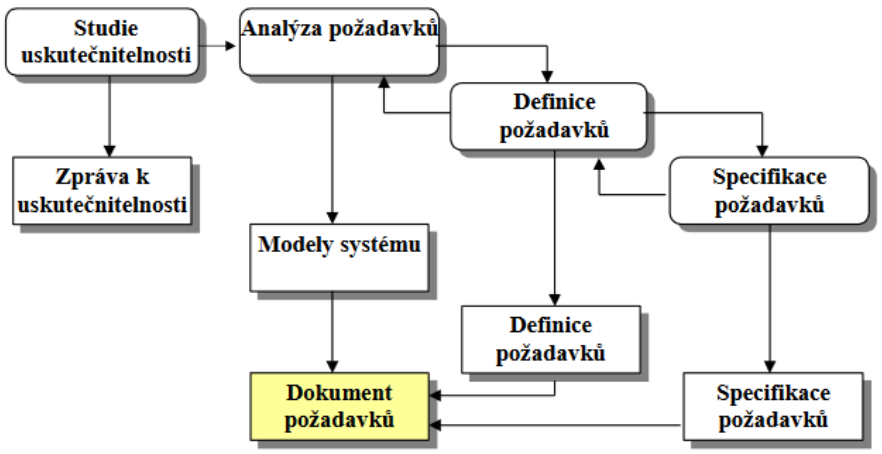
Tato aktivita využívá výstup z analýzy, strukturuje fakta a reprezentuje je v notacích. Analýza požadavků si pro tuto činnost vypůjčila metody strukturovaného vývoje systému a konceptuální modelování. Neformální modelové notace, jako jsou diagramy datových toků, diagramy případů užití a schémata vztahů mezi entitami jsou široce používány. Mnoho formálních přístupů k modelování bylo převzato z softwarového inženýrství, ačkoli efektivnost těchto technik ještě nebyla prokázána v průmyslové praxi. Analýza a modelování jsou často prokládány pro vypracování požadavků, neboť prostřednictvím jejich zastoupení se zvyšuje porozuměním problematické oblasti [1].

1.1.5 Validace

Tato klíčová aktivita v analýze požadavků, ačkoli byla rozsáhle prozkoumána, je stále problematická. Validace neboli ověření předpokládá, že uživatelé pochopí důsledky specifikace požadavků a poté souhlasí, tj. potvrdí, že přesně odrážejí jejich přání. Současným stavem techniky jsou techniky průzkumu, v nichž jsou v dílně designérů a uživatelů kritizovány polo-formální specifikace, jako jsou diagramy datových toků. Návody mají zásluhu na včasné validaci specifikací, zatímco prototypy jsou pravděpodobně silnější, protože uživatelé silněji reagují na skutečný pracovní systém. Bohužel prototypy stále přinášejí náklady na výstavbu a špatně organizované používání prototypů může být škodlivé. Nicméně prototypy v kombinaci s technikami pro shromažďování a vyhodnocování zpětné vazby uživatelů mohou být vysoce účinné. Celkově je proces validace špatně pochopen a vysvětlení je důležitou, přesto často opomíjenou složkou. Některý výzkum vysvětlující komplexní požadavky ukázal, že je nutná kombinace vizualizace, příkladů a simulace (Carroll et al., 1994; Maiden & Sutcliffe, 1994). Scénářové reprezentace a animované simulace pomáhají uživatelům vidět důsledky chování systému a tím zlepšují validaci. Navíc prototypy se scénáři jsou silným prostředkem vyvolání zpětné vazby. Technika dotazovacího cyklu (Potts et al., 1994) přistupuje k validaci porovnáním skriptů představovaného chování v reálném světě proti požadovanému chování ve specifikaci. Validace je stále špatně pochopena a je zapotřebí dalšího výzkumu, aby se zjistilo, jak interagují vysvětlení, reprezentace a porozumění uživatelů se specifikacemi systému [1].

1.2 Blokové schéma procesu analýzy požadavků

Na obrázku níže můžeme vidět detailnější schéma procesu analýzy požadavků.



Obrázek 3 – schéma procesu analýzy požadavků [5]

1.3 Typy požadavků

Požadavky jsou zadání v přirozeném jazyce, příp. diagramy udávající požadované služby systému a omezení. Jsou vytvořeny na základě informace od zákazníka. Požadavky jsou kategorizované několika způsoby. Následující kategorizace požadavků je z technického pohledu [2, 6].

Požadavky zákazníků: Specifikace požadavků a předpokladů, které určují očekávání od systému vzhledem na sledované cíle, prostředí, omezení a metriky efektivnosti a vhodnosti (MOE / MOS). Zákazníci jsou ti, kteří vykonávají osm primárních funkcí systémového inženýrství, se zvláštním důrazem na operátora jako na klíčového zákazníka [6].

1.3.1 Funkční požadavky

Umožňují nám jako výrobci softwaru základní orientaci v potřebách zákazníka a představují jakýsi most mezi prostředím a terminologií zákazníka a prostředím vývoje software. Pokud má zákazník jasnější představu o způsobech použití budoucího software, popíšeme tzv. případy užití (neboli Use cases) [7, 2].

1.3.2 Nefunkční požadavky

Definují vlastnosti a omezení systému jako celku a týkají se produktu i procesu vývoje (kvalita, udržovatelnost atd). I tyto požadavky mají mnohdy kritický vliv na aplikaci, ale jejich samotným úkolem však není podpořit byznys cíle, ale vyvinout kvalitní stabilní aplikaci a její kvalitu měřit podle kritérií, kterých si u dané aplikace zákazník nejvíce cenní. Mezi základní požadavky patří výkon, škálovatelnost, spolehlivost, rozšiřitelnost, udržitelnost, spravovatelnost a bezpečnost. Při volbě požadavků, které jsou pro systém nejdůležitější, musejí architekt ve spolupráci se zákazníkem počítat s tím, že se jednotlivé požadavky vylučují navzájem. Chceme-li například navrhnout a realizovat systém, který si bude zakládat na vysokém výkonu, musíme obětovat požadavky, jako jsou udržitelnost a rozšiřitelnost. Musíme tedy počítat s tím, že rychlost naší aplikace bude vykoupena například tím, že v budoucnu budeme muset investovat více prostředků pro rozšíření dané aplikace o novou funkčnost [7, 8].

Typy nefunkčních požadavků [5]:

- požadavky na produkt
  1. na použitelnost
  2. na efektivnost
     a. výkonnostní
     b. prostorové
  3. na spolehlivost
  4. na přenositelnost
- požadavky na proces
  1. na dodání
  2. na implementaci
  3. na standardy
- externí požadavky
  1. na součinnost (interoperability)
  2. etické
  3. legislativní
     a. ochrana soukromí
     b. bezpečnostní

Nefunkční požadavky musí být ověřitelné a měřitelné.

Rychlost: transakce/s, doba odezvy,

Velikost: kód, požadavky na diskový prostor, paměť RAM,

Použitelnost: doba zaškolení, rozsah nápovědy,

Spolehlivost: střední doba bezporuchového provozu, pravděpodobnost nedostupnosti, četnost poruch a chyb,

Robustnost: doba obnovy po poruše, pravděpodobnost zničení dat při poruše,

Přenositelnost: procento závislého kódu, počet cílových systémů.

2 UML

UML (Unified Modeling Language) je soubor grafických notací, který se používá pří vývoji softwaru. V oblasti analýzy a návrhu se stal standardem, a proto je pro programátory důležité, aby se v něm orientovali. UML je použito v mnoha materiálech, v dokumentacích a podobně. Hlavně nám ale může sloužit jako užitečný nástroj k usnadnění návrhu a vývoje informačního systému.

2.1 Use Case

V letech 1992-1997 se většina lidí, kteří psali o případech užívání, vyhýbala tomu, aby říkala, co je přesně. Nicméně prohlásili, jak užitečné jsou pro projekty. Přestože se zdálo, že nikdo nemůže říci, co to je, nebo pojmenovat rozdíl mezi případem použití a scénářem, základní a přitažlivý nápad zůstal [9]:

„Napište krátký textový popis toho, jak systém interaguje s okolím, když provádí jistou funkci pro jednoho z jeho uživatelů, a zachyťte, jak se má systém chovat, když se něco děje špatně.“ [9]

Tato myšlenka byla užitečná tehdy, a je stále užitečná, bez ohledu na to, jak formálně nebo neformálně by text mohl být napsán [9].

Mezi různými myšlenkami, které v té době rostly, navrhovali a doporučovali téměř všechny kombinace a permutace odpovědí na základní otázky [9]:

Je použití případu požadavkem nebo jen popisem činnosti?

Je scénář jen jiný název pro případ použití?

Je případ použití formální, polo-formální nebo neformální struktura?

Existuje propojovací struktura pro případy použití, nebo prostě přicházejí ve skupinách?

Pro některé lidi to byl pouze jiný název pro scénář, krátký popis někoho, kdo používal tento systém (Martin Fowler často říkal, "myslím, že případ použití musí být švédské označení pro scénář"). Podle této myšlenky, případ užití nemá žádnou vnitřní strukturu a případy užití jen sedí ve skupinách. Lidé, kteří takto popisovali případy užití, získali dobrou hodnotu za úsilí, které vynaložili, a mnoho z nich doporučuje psát takové neformální případy užití [9].

Jiní lidé, zejména výzkumní pracovníci a návrháři nástrojů CASE (Computer Assisted Software Engineering) považovali tyto neformální pracovní produkty za neúplné, potřebují matematický základ a podporu v nástroji CASE. Vygenerovali jazyky, notace a nástroje, aby používaly případy "přesné". Tato myšlenková škola se tak dobře neoplatila. Lidé chtějí poměrně neformální médium, v němž vyjadřují své počáteční myšlenky na systém. Prostě nebudou používat formální nástroje. Nemají pocit, že se v průběhu projektu vyplácí další pracovní síla [9].

Dalším problémem, se kterým se formalisté setkávají, je manipulace se všemi variantami, které musí systém zvládnout. Například: "Pro cukrářský stroj, jak specifikuji, že člověk může dát tři čtvrtiny nebo patnáct centů? Nebo čtvrtina následuje deset centů, nebo deset centů, za kterými následuje čtvrtina? Nebo některou z dalších desítek způsobů, jimiž může osoba vložit minci? ". Správná odpověď byla a je, "Jen napište, že ten člověk dává peníze."

Poté, co byli v různých časech lidé uvězněni v příliš hodně, a naopak příliš málo formálním zpracování, je vybrána prostřední cesta: polo-formální text sedící v polo-formální struktuře. Pomocí těchto polo-formálních struktur, můžeme [9]:

- Ujistit se, že případy použití jsou opravdu požadavky a potřebují základní strukturu
- Umožnit lidem psát co chtějí, když potřebují.

Překvapením je, že druhý cíl je důležitější než první.

Strukturování případů užití s cíli

Jak můžeme vidět na ukázkovém obrázku níže, je případ užití textovým popisem činností aktérů. Velká otázka na počátku devadesátých let byla, jak tyto společné akce souvisí dohromady [9].

Vysvětlení Ivara Jacobsona a jeho příklady, poskytují dvě rady. První a největší je, že případ užití popisuje aktéra, který se pokouší dosáhnout cíle pomocí systému. To znamená, že pokud jmenujeme jednoho z aktérů jako primárního aktéra, pak všechny akce souvisejí s tím, že daný aktér dosáhl nějakého cíle, který byl jeho zájmem [9]

Spojení případů použití s cíli aktérů je velmi důležité, protože posunuje pozornost z funkčních seznamů, které se většina programátorů obává a vrací je zpět uživatelům no to co uživatelé skutečně chtějí dosáhnout se softwarem, a jaké jsou jejich cíle při používání Softwaru. Pokud software podporuje tyto cíle, software přinese největší obchodní hodnotu [9].

Druhým radou je, že cíle někdy selhávají. Ať už získáte smlouvu, získáváte peníze z bankomatu, vložíte kartu ATM do čtečky karet nebo dokonce přesunete do dalšího pole při vyplňování online formuláře, může být jakýkoliv cíl neúspěšný. Přemýšlení a psaní o způsobu řešení selhání v dokumentu požadavků šetří projektovým týmům v průběhu projektu spoustu peněz [9].

Případy užití jsou proto strukturováno do dvou částí: sled akcí, kdy vše funguje dobře, následované různými malými sekvencemi popisujícími, a co se stane, když se různé cíle a subcíle nezdaří [9].

Dokonce i s těmito radami byl jeden neočekávaný důsledek práce s cílem. Cíle jsou v různých velikostech. V jednom okamžiku můj cíl může být získání významné obchodní smlouvy. Okamžitějším cílem (zaměřeným na menší časový interval) je přijetí klienta na oběd. Dalším okamžitým cílem (s ještě menším časovým rozpětím) je dostat hotovost na oběd [9].

Pokud to shrneme je případ užití sada několika akcí, které vedou k dosažení určitého cíle. Use Case může být přidání komentáře k článku, založení nového uživatele nebo např. vytisknutí dokumentu. Definuje tedy jednu funkcionalitu, kterou by měl navrhovaný systém umět. Ta v sobě obsahuje další akce, např. přidání komentáře bude obsahovat ověření uživatele, validaci zadaných dat, zápis do databáze a podobně. To v diagramu zachyceno již nebude. UML často hovoří o tzv. blackboxu (černé skříňce), kde skryjeme vnitřní logiku a pracujeme pouze s komponentami. Tento princip přesně využívá UC diagram [4].

2.2 Model případů užití (Use Case model)

Use Case Diagram (česky diagram případů užití) zobrazuje chování systému tak, jak ho vidí uživatel. Účelem diagramu je popsat funkcionalitu systému, tedy co od něj uživatel nebo my očekáváme. Diagram vypovídá o tom, co má systém umět, ale neříká, jak to bude dělat. Proto je to většinou první diagram, který při návrhu informačního systému vytváříme. Je důležité se nejprve shodnout na tom, co má náš systém (nebo aplikace, hra, cokoli) umět. Až potom má smysl se ptát, jak to vlastně uděláme [4, 9].

2.2.1 Aktér

Aktér je role, která komunikuje s jednotlivými případy užití. V této roli může být obsazen uživatel nebo externí systém. Aktérem tedy může být např. Uživatel, Administrátor, SMS

server nebo dokonce Čas. Aktér inicializuje nějaký případ užití (např. Uživatel vloží příspěvek do fóra). Zde hovoříme o tom, že je aktér aktivní. Aktér sám však může být případem užití iniciován (např. externí SMS server je iniciován případem užití Poslat SMS). V tomto případě hovoříme o pasivním aktérovi a zakresluje ho v diagramu napravo [4, 9].

Aktéry znázorňujeme jako postavu z čar s názvem napsaným pod ní.

Obrázek 4 – Aktér [4]

2.2.2 Specifikace případů užití

Samotný UC diagram nám ukazuje, jaké funkcionality systém obsahuje a jak (kým) jsou spouštěny. Kromě názvů jednotlivých případů užití o nich však nevíme vůbec nic. Tento problém řeší Use Case Specifikace. Jedná se o doplňující dokument, který je k Use Case diagramu přiložen. Nemá žádnou pevně definovanou podobu, může být ve formě tabulky nebo prostého textu.Specifikace obsahuje jednotlivé případy užití, ke každému z nich definuje několik bodů. Nejprve si je popíšeme, potom si zkusíme část specifikace k našemu modelu vytvořit. Vyjmenujme si tedy jednotlivé části specifikace pro jeden případ užití [4]:

1. Krátký popis (Brief Description)

V první části krátce popíšeme případ užití, stačí 1 až 2 věty. Měl by vysvětlovat, jakou má funkčnost pro uživatele přidanou hodnotu, proč ji uživatel spouští. Víme, že Use Case diagram popisoval funkčnost právě z pohledu uživatele. Toto pravidlo zachováme i v UC specifikaci. Příkladem by mohl být popis: "Use Case umožňuje vytisknout vybraný dokument" [4].

2. Aktéři (Actors)

Další část jmenuje aktéry, kteří se případu užití účastní. Příkladem mohou být Systém a Uživatel [4].

3. Podmínky pro spuštění

Každý případ užití může mít definované určité podmínky, které musí být pro jeho spuštění splněny. Ty můžeme uvést v této části jeho specifikace. Tyto podmínky můžeme také vynechat. Příkladem podmínek pro spuštění může být nainstalovaná tiskárna nebo internetové připojení. Bez těchto náležitostí nemá UC smysl a nebude ani spuštěn [4].

4. Základní tok

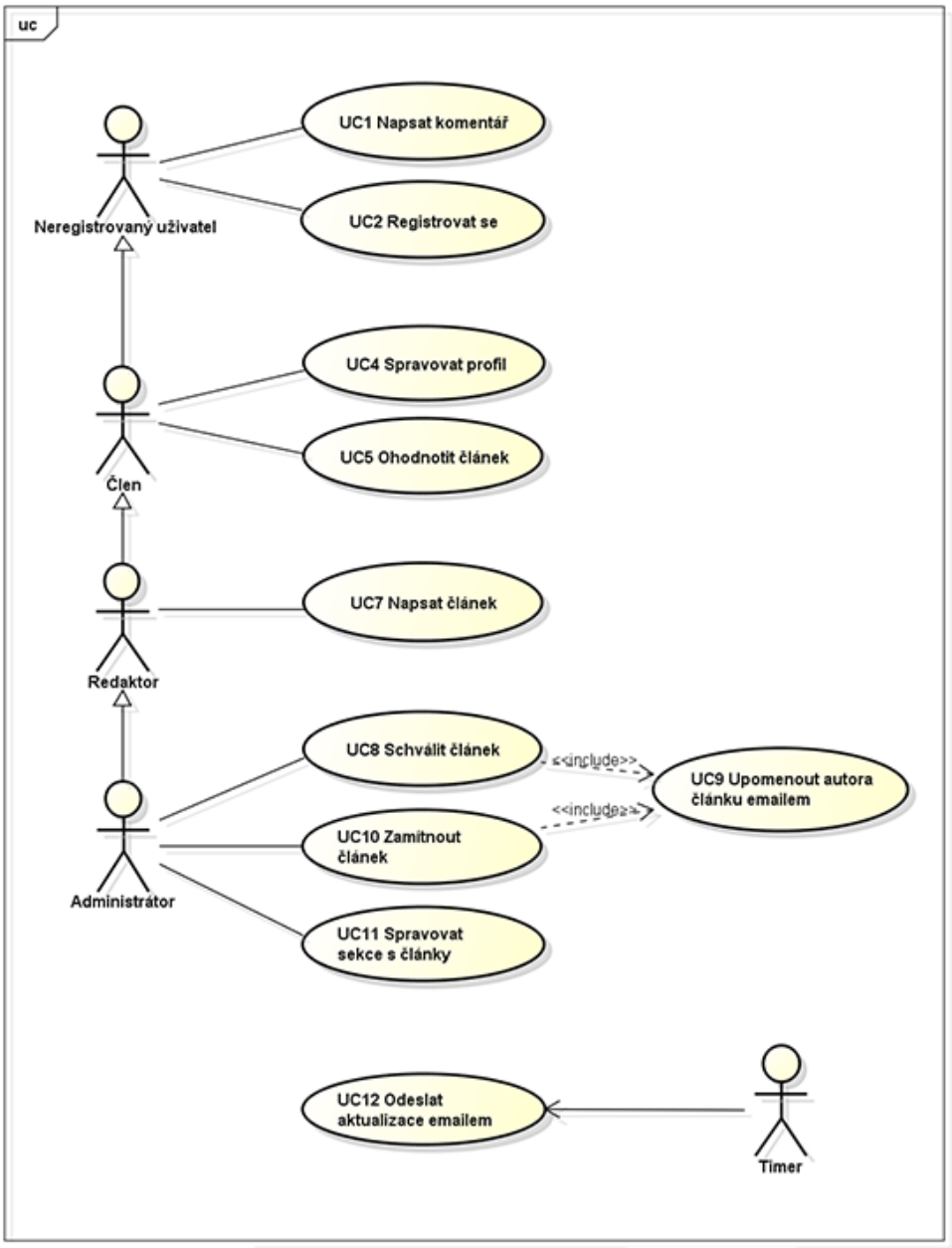
Konečně se dostáváme k základnímu toku. V jeho bodech je popsána interakce mezi aktéry a jednotlivými případy užití. Body zapisujeme jako scénář, ve kterém se střídají vždy aktér a systém. Opět máme na paměti, že popisujeme z pohledu uživatele a toho, jaký pro něj má případ užití význam. Častou chybou je popisovat co systém zobrazí, co uživatel zapíše do formuláře a podobně. Popis by měl však být úplně odstíněn od toho, jak systém vypadá, měl by se zaměřit na to, jak funguje. Základní tok neřeší možné chyby a předpokládá bezproblémový průběh, kde v posledním kroku dojde ke splnění cíle případu užití. Příklad si ukážeme dále [4].

5. Alternativní toky

Specifikace může obsahovat několik alternativních toků (scénářů), které umožňují reagovat na odchylky od scénáře hlavního. Jedná se o poruchy nebo chyby, ať již ze strany uživatele (špatně zadal heslo) nebo systému (nepodařilo se vytisknout dokument). Alternativní tok se vždy vztahuje k určitému bodu hlavního toku a řeší jeho nestandardní verzi. Většinou je na konci odkázán opět na nějaký bod hlavního toku, kde zas hlavní tok pokračuje dále [4].

6. Podmínky pro dokončení

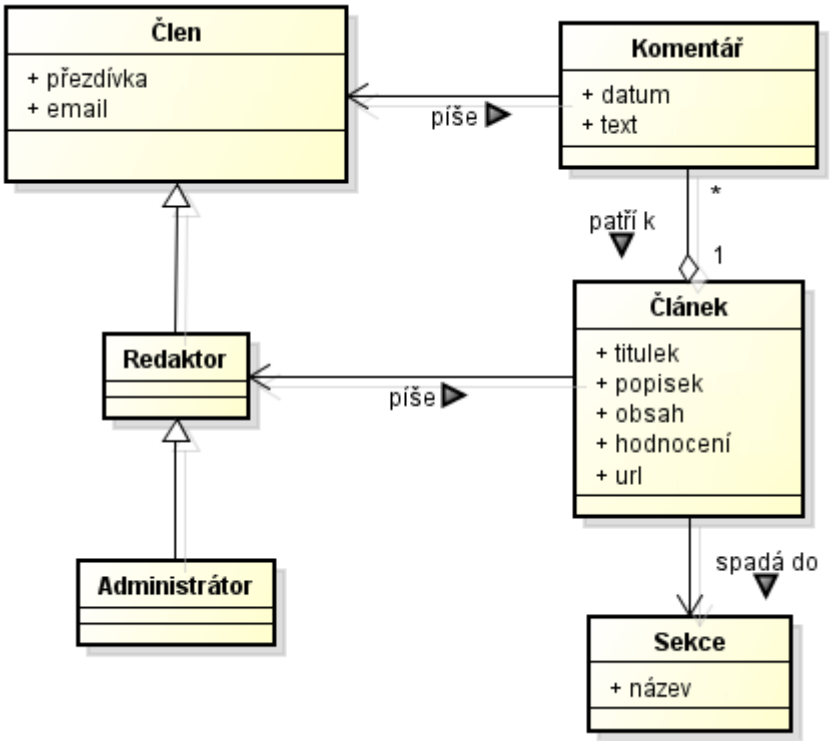
Podobně, jako může mít případ užití podmínky přes spuštěním, může mít i podmínky pro dokončení. Podmínkami může být např. že vše proběhlo v pořádku nebo že chyby byly zaznamenány do chybového logu [4].



Obrázek 5 – ukázka diagramu případů užití [4]

2.3 Doménový model

Doménový model je formou class diagramu, tedy diagramu tříd. Třídy v doménovém modelu jsou však značně zjednodušené, neobsahují metody a mají pouze důležité atributy. Model je tedy jakýsi náčrt základních entit systému a vztahů mezi nimi. Je **platformě nezávislý** (není určen pro konkrétní programovací jazyk) a atributy nemají datové typy [4].



Obrázek 6 – ukázka doménového modelu [4]

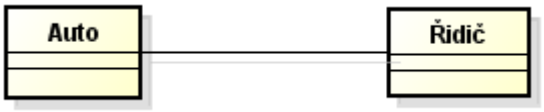
Při tvorbě doménového modelu vycházíme ze zadání klienta. Z něj identifikujeme klíčové entity a vztahy mezi nimi. Tyto entity zakreslíme do modelu jako třídy.

2.3.1 Vztahy v doménovém modelu

UML nabízí několik druhů vztahů, ty základní si vyjmenujeme.

Asociace (Association)

Asociace určuje základní vztah mezi dvěma entitami. Ty mohou existovat nezávisle na sobě. Zakreslujeme ji jako jednoduchou plnou čáru. Příklad jednoduché asociace mezi dvěma entitami může být Auto a Řidič. Vztah by se znázornil takto:



Obrázek 7 – asociace [4]

Jako výchozí je směr na obě strany, tedy že první entita má odkaz na druhou, a naopak druhá na první. Toto chování můžeme změnit přidáním jednoduché šipky, která směr specifikuje a způsobí, že odkaz si uchovává pouze ta instance, na kterou nesměřuje šipka. Je možné vytvořit asociaci i mezi třemi třídami, ale tím se nebudeme zabývat [4].

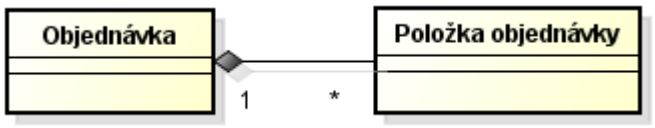
Agregace (Aggregation)

Agregace reprezentuje vztah typu celek - část. Znázorňuje se jako jednoduchá plná čára, zakončená na jedné straně prázdným kosočtvercem. Ten je umístěn u té entity, která reprezentuje celek (např. sekce s články). Z hlediska implementace je to tak entita, která drží kolekci. Entita reprezentující část může existovat sama o sobě a být součástí i jiných kolekcí. Příkladem agregace může být již zmíněná sekce, obsahující články. Čísla na konci vazby znamenají tzv. multiplicitu, přesněji, že sekce obsahuje libovolný počet článků a článek patří alespoň do 1 sekce [4].

Obrázek 8 – agregace [4]

Kompozice (Composition)

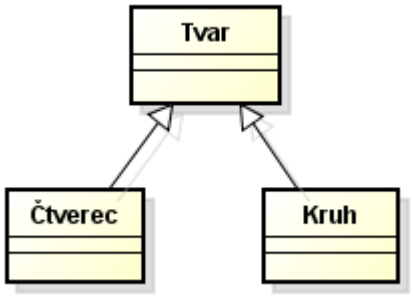
Kompozice je podobná agregaci, avšak reprezentuje silnější vztah. Entita části nemá bez celku smysl. Pokud zanikne celek, zanikají automaticky i jeho části. Kompozici zakreslujeme stejně jako agregaci, kosočtverec je ovšem plný. Tato vazba bývá matoucí a doporučil bych se jí spíše vyhýbat a nahradit ji agregací. U entity reprezentující celek musí být multiplicita vždy 1. Příkladem může být Objednávka a Položka objednávky. Zatímco článek z minulého příkladu dává bez sekce ještě nějaký smysl, položka objednávky bez objednávky smysl nedává. Proto je zde použita kompozice [4].



Obrázek 9 – kompozice [4]

Generalizace (Generalization)

Posledním vztahem, který si zde uvedeme, je generazilizace. Z hlediska implementace se jedná o dědičnost. Jedna entita dědí vlastnosti a chování jiné. S touto vazbou jsme se již setkali u Use Case diagramu. Generalizaci zakreslujeme jako plnou čáru, zakončenou na jedné straně prázdnou uzavřenou šipkou (nebo chcete-li trojúhelníkem). Šipka je na straně entity, ze které se dědí. Příkladem může být třída Tvar, ze které dědí třídy Čtverec a Kruh [4].



Obrázek 10 – generalizace [4]

Multiplicita

Vraťme se ještě k multiplicitě (neboli násobnosti). Multiplicitu můžeme uvést u vazeb asociace, agregace a kompozice (zde pouze z jedné strany).

Vraťme se k příkladu sekce-článek:

Multiplicitu zde čteme takto: Sekce může mít libovolný počet článků (to poznáme podle hvězdičky u třídy Článek). Článek patří do 1 až libovolně sekcí (to poznáme podle 1..* u Sekce). Pojďme si nyní uvést jednotlivé možné zápisy multiplicity [4]:

- **1 (číslo)** - Označuje konkrétní hodnotu (zde právě 1).
- **(hvězdička)** - Označuje libovolný počet (tedy i 0). Místo hvězdičky můžeme v některých materiálech nalézt symbol N.
- **1..* (interval)** - Pomocí 2 teček můžeme označit interval. Do něj vkládáme nám již známé symboly, např.: 2..6 nebo 1..* nebo 0..1.

Zápisy můžeme dokonce i slučovat, např. takto: 1, 2, 3, 7..*. Tento zápis označuje multiplicitu 1, 2, 3 nebo 7 a více. Pokud není multiplicita uvedena, označuje to výchozí hodnotu 1 [4].

3 TECHNOLOGIE PRO TVORBU KLIENTSKÉ ČÁSTI APLIKACÍ

Rozdělení na technologie běžně používané zvlášť na klientskou část aplikace serverovou je především z důvodu jednoho z hlavních nefunkčních požadavků na tuto aplikaci, a to kompletní separace těchto částí. I když mnoho níže uvedených technologií by samozřejmě zvládlo obojí [4].

3.1 Webový klienti

3.1.1 PHP

PHP Hypertext Preprocessor je skriptovací jazyk běžící na straně serveru (server-side) speciálně navržený pro potřeby webové stránky. Pro skriptování na straně klienta je používán JavaScript. Pomocí PHP jsme schopni vytvořit dynamické webové stránky pro uživatelské rozhraní zobrazené na straně prohlížeče jako HTML s CSS. Ale také k vytvoření celé logiky aplikace včetně databázového přístupu. PHP od verze 5.3 je velmi dobře objektově orientovaný jazyk a je Open Source. PHP byla původně zkratka pro „Personal Home Page“ neboli osobní domovská stránka.

Klady [10, 11, 12, 15]:

- PHP je specializované na webové stránky.
- **Strmá křivka učení**: Začít a pracovat s PHP je docela jednoduché. Vytváření webových stránek je téměř příliš jednoduché. To je kvůli jeho původu jako nástroj pro vytváření osobních domovských stránek a tlumočení formulářů.
- **Funkcionální a objektově orientované programování**: Oba způsoby jsou nyní dobře podporovány v aktuální verzi PHP s anonymními funkcemi, také známými jako lambda nebo uzávěry, které nyní získávají status prvotřídního objektu.
- **Obrovský ekosystém**: Ekosystém kolem PHP je obrovský. Mnoho z toho je kvůli popularitě WordPress a Magento. Existují desítky bezplatných a placených online školicích míst pro PHP i lokální tréninková místa po celém světě.
- **Velká komunita softwaru s otevřeným zdrojovým kódem**: S příchodem skladatele asi před pěti lety explodovala komunita OSS s více než 70 000 knihovnami s otevřeným zdrojovým kódem.
- Poměrně slušná dokumentace.

- **Množství propojitelných frameworků**: Po vzniku PHP Framework Interop Group před přibližně pěti lety komunita PHP konečně přijala interoperabilitu. To umožňuje psát kód pro jeden framework, který lze snadno přenést do jiného. To také znamená, že můžete použít mnoho stejných knihoven v rámci různých frameworků. Interoperabilita také znamená sdílení jádrových komponent a je snazší vytvářet a udržovat framework, přičemž se zaměřuje spíše na případ použití pro váš framwork než na klíčové komponenty, které musí každý framework poskytovat. To vedlo k množství opravdu dobrých frameworků jako Symfony2, Laravel, Silex a zend-expressive, které sdílejí společné knihovny.
- **Automatizační nástroje**: Je k dispozici poměrně dobrý systém automatizačních nástrojů pro testování a nasazení PHP aplikací, které jsou napsány v PHP.
- **Ladění**: díky službě Xdebug má PHP ladění první třídy včetně vzdáleného ladění, které se stává čím dál důležitější v souvislosti s rozvíjejícím se virtuálním strojem světa Vagrant a Docker, které se používají na vývojové pracovní stanici. Být schopen snadno ladit vzdáleně je obrovské plus.
- Rozsáhlý soubor funkcí v základní knihovně PHP (přes pět a půl tisíce), další funkce v PECL.
- Nativní podpora mnoha databázových systémů.
- **Multiplatformnost** (zejména Linux a Microsoft Windows).
- Možnost využití nativních funkcí operačního systému (možná nekompatibilita s jiným OS).
- Obrovská podpora na hostingových službách – PHP je fakticky standardem, který najdeme všude.
- Poměrně slušná dokumentace.
- **Velmi svobodná licence**, která (v protikladu k např. GPL) neobsahuje copyleft.

Zápory [10, 11, 12, 15]:

- **Interpretovaný jazyk**: Stejně jako Python a Ruby je to interpretovaný jazyk, který je kompilován do Opcode. To může být do jisté míry překonáno ukládáním do mezipaměti Opcode, ale doba náběhu může být problematická.
- **Vláknové zpracování**: I po příchodu správce procesů FastCGI (FPM) je to stále soubor vláknových spouštěčů. To představuje problémy se škálovatelností, které postihují všechny prostředí využívající vlákna jiných jazyků, jako jsou Java Servlets,

Python WSGI, Ruby Rack nebo Microsoft IIS. Každý podproces má vlastní paměť a máte konečné množství paměti, takže jste omezen pamětí na počet připojení, které můžete podporovat. Většina jazyků disponuje pevným asynchronním I / O řešením. PHP má ReactPHP a Icicle, ale nemá významnou základnu.

- **Globální rozšíření**: PHP vyžaduje rozšíření, mosty mezi kódem C a PHP musí být instalovány globálně do spustitelného souboru PHP. Vyžadují úpravy globálního konfiguračního souboru, aby byly přístupné. To vyžaduje, aby se při nasazování aplikací podílel správce systému. To také ztěžuje používání rozšíření u hostovaných aplikacích. Většina ostatních interpretovaných jazyků poskytuje podporu k instalacím rozšíření správce balíčků (package manager).
- **Malá podpora IoT**: Většina interpretovaných jazyků, s výjímkou Node.js, mají minimální komunitu internetu věcí (IoT), ale pro PHP se zdá, že neexistuje žádná. Pokud existuje, skrývá se. To může být způsobeno chybějícím zavedeným asynchronním I / O frameworkem, který je potřebný pro správné zpracování I / O.
- **Nábor PHP vývojářů**: Vzhledem k nižší bariéře vstupu a žádné skutečné představě o tom, co může být „Senior PHP Developer“, je opravdu těžké najít dobré talenty v PHP na úrovni seniorů. Najdete zde vývojáře nejvyšší úrovně, ale bude to mnohem těžší najít než jiné jazyky.
- **Vnímání Jazyka**: Vzhledem k nízké bariéře vstupu a populárním, ale starším kódovým základům, jako je WordPress, má vnější svět tendenci sledovat PHP jako nejistý a začátečnický jazyk. Nemyslím si, že PHP se za deset let považuje za super a inovátorské. Toto vnímání může způsobit skutečné problémy pro vývojáře i firmy. V případě vývojářů na nejvyšší úrovni může být vaše hodnota a prodejnost jako vývojáře omezena, pokud váš životopis je silně orientován na PHP. Vývojáři zřídkakdy dostanou příležitost pracovat na něčem, co se objevuje, protože PHP je zřídkakdy jazykem, který se nejprve využívá pro vývoj rozvíjejících se technologií nebo procesů. Podívejte se na koncept „IoT“ výše. Jako společnost může být vaše hodnota a důvěryhodnost negativně ovlivněna na základě vnímání PHP. Pokud se snažíte získat kapitál nebo prodat svou firmu, bude mít aplikace Node.js nebo Java s průměrnými vývojáři úrovně zřejmě více finančně životaschopné než mít tým nejlepších inženýrů PHP za stejnou cenu.

- Jazyk PHP byl dlouho definován pouze svou implementací, oficiální specifikace jazyka byla oznámena na konci července 2014.
- Nekonzistentní pojmenování funkcí, např.: strpos(), strchr(), ale str_replace(), str_pad().
- Nejednotné názvosloví skupin funkcí, např.: mysql_XXXX, imap_XXXX, json_XXXX (s podtržítkem) versus imageXXXX, bcXXXX, gzXXXX (bez podtržítka).
- Nejednotné pořadí parametrů, např.: array_map() vs. array_filter().
- Ačkoliv jazyk podporuje výjimky, jeho knihovna je používá jen zřídka.
- Slabší podpora Unicode, pouze přes PHP knihovnu (ve verzích po PHP 5 má být Unicode řetězec jako základní typ).
- Ve standardní distribuci chybí ladící (debugovací) nástroj.
- Po zpracování požadavku neudržuje kontext aplikace, vytváří jej vždy znovu (oslabuje výkon).

3.1.2 PERL

Perl je interpretovaný programovací jazyk vytvořený Larry Wallem v roce 1987. S rozvojem internetu se Perl stal velmi populárním nástrojem pro tvorbu CGI skriptů (v praxi používaný zejména na WWW serverech). Perl zahájil svou éru jako skriptovací jazyk, náhrada jazyka AWK a interpretru SH. Největšího rozšíření dosáhl ve verzi 4 z roku 1991. Verze 5 přinesla četná vylepšení, především výkonné datové struktury a možnost objektového programování. V poslední době získal Perl oblibu mimo jiné v bioinformatice [15].

Umožňuje psát krátké programy jednoduše a rychle, a přitom nebrání v psaní těch složitých. Jeden ze způsobů je přitom obvykle velmi stručný, takže Perl získal nezaslouženou pověst jazyka, ve kterém se tvoří nesrozumitelný a neudržovatelný kód. Tato kritika ale není oprávněná, Perl je vhodný k řešení malých i velkých problémů. Schopnosti a nástroje, které se používají u velkých projektů, lze použít i v krátkých skriptech [15].

Výhody [15]:

- Interpretovaný jazyk – rychlý vývoj bez nutnosti kompilace a linkování.
- Přes 18 000 volně dostupných modulů třetích stran v CPAN (Comprehensive Perl Archive Network).

- Pojmenování, kategorizace, dokumentace, testování a instalace modulů jsou standardizovány a zpřístupňují prakticky veškerá dostupná rozhraní a knihovny.
- Efektivita programování.
- Automatická práce s pamětí (není třeba explicitně alokovat a uvolňovat paměť).
- Pokročilé datové typy např. asociativní pole neboli hash.
- Svobodný software, licencován pod Artistic License či GNU.
- Snadné spojování již hotových komponent (modulů).
- Reference na statické, dynamické i anonymní datové struktury.
- Umožňuje procedurální, funkcionální i objektově orientované programování.
- Snadná práce s textem a značkovacími jazyky (XML, HTML…).
- Perl podporuje znakovou sadu Unicode a je (byl) Y2K kompatibilní.
- Stabilita – mnoho let vyvíjený programovací jazyk.
- Perl umí zacházet se zakódovanými webovými daty, mezi něž patří např. transakce u elektronického obchodování.
- Perl může být součástí web serverů, čímž může dojít ke zrychlení až o 2 000 %.
- Modul umožňuje web serveru Apache vložení Perlu s výhodami, jako u PHP.
- Rozsáhlá dokumentace a literatura, komunita kolem Perlu, konference, …

Nevýhody [15]:

- Nedisciplinovaný programátor může velmi snadno vytvářet nesrozumitelný kód.
- Při některých aplikacích se může projevit neefektivnost dynamicky typovaného jazyka ve srovnání se staticky typovanými jazyky, zejména spotřeba paměti.
- Kruhové odkazy a problematika jejich destrukce.
- Mnozí tvrdí, že je to jazyk nevhodný pro výuku programování.

3.1.3 JavaScript

Vedle HTML a CSS je JavaScript (standardizovaný jako ECMAScript) považován za jednu z velkých tří hlavních komponent webových stránek. Zaměstnaný většinou webových stránek, JavaScript je skriptovací jazyk, který obvykle běží v prohlížeči a dělá webové stránky dynamické a interaktivní. Jeho syntaxe (zápis zdrojového textu) patří do rodiny jazyků C/C++/Java. Ale JavaScript je od těchto jazyků zásadně odlišný, sémanticky (vnitřně) jde o jiný jazyk. Slovo Java je součástí jeho názvu pouze z marketingových důvodů. Program v

JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele.

Výhody [13, 14]:

- **Javascript je spuštěn na straně klienta**: To znamená, že kód je spuštěn na uživatelském procesoru namísto webového serveru, čímž se na webovém serveru šetří šířku pásma připojení k síti a zátěž procesoru i paměti.
- **Javascript je poměrně snadný jazyk**: Jazyk Javascript je poměrně snadné se učit a obsahuje syntaxi, která je blízká angličtině. Používá model DOM, který poskytuje různým objektům spoustu předpřipravených funkcí, což vytváří možnost pro vytvoření skriptu k vyřešení vlastního cíle.
- **Javascript je pro koncového uživatele poměrně rychlý**: Jelikož je kód spuštěn na počítači uživatele, zpracování výsledků je dokončeno téměř okamžitě v závislosti na operaci.
- **Snadné ladění a testování**: kód JavaScriptu je interpretován řádek po řádku. Chyby jsou uvedeny spolu s číslem řádku. Je velmi snadné najít chybu v kódu, opravit ji a otestovat (pokud se nejedná o chybu sémantickou).
- **Programování založené na událostech**: JavaScript je jazyk založený na událostech. To znamená, že při výskytu určité události se provádí určitý segment kódu. Například segment kódu může být spuštěn, když uživatel klikne nebo přesune myš nad objekt atd.
- **Procedurální schopnosti**: JavaScript poskytuje všechny možnosti procedurálního jazyka. Poskytuje kontrolu stavu, smyčky a větvičky, které lze provést na webové stránce.
- **Nezávislost platformy**: JavaScript je jazyk nezávislý na platformě. Každý prohlížeč podporující jazyk JavaScript může chápat a interpretovat kód JavaScript. Jakýkoli kód jazyka JavaScript lze spustit na různých typech hardwaru.

Nevýhody [13, 14]:

- **Problémy s bezpečností**: Úryvky jazyka Javascript, jakmile jsou připojeny k webovým stránkám, se okamžitě spustí na klientských serverech, a proto mohou být také

použity k zneužití uživatelského systému. Zatímco určitá omezení jsou stanovena moderními webovými standardy v prohlížečích, škodlivý kód může být stále vykonáván v souladu s nastavenými omezeními.

- **Vykreslení jazyka Javascript se liší**: Různé vykreslovací nástroje mohou způsobovat různé rozložení, což bude mít za následek nesrovnalost z hlediska funkčnosti a rozhraní. Zatímco nejnovější verze JavaScriptu, co se týče vykreslování, byly zaměřeny na univerzální standard, některé varianty stále existují.

3.1.3.1 ReactJS

React je JavaScriptová knihovna pro vytváření webových komponent. V pomyslném MVC představuje "V" neboli view vrstvu a dal by se tak přirovnat například Latte v Nette. React je však daleko více. Přináší totiž zásadní změnu paradigmatu. S Reactem už nepíšeme kód, který něco mění, ale kód, který popisuje, jak má vypadat výsledek, což je řádově snažší úloha. Dvojnásob to pak platí, pokud tím výsledkem je "těžká váha" v podobě DOMu [17].

React spatřil světlo světa v květnu 2013. Opensourcoval ho Facebook, který ho už několik let před tím sám interně používal a vylepšoval. Prvotní vydání se však dočkalo velkého výsměchu. Odezva byla dokonce tak špatná, že Facebook chvíli uvažoval i o jeho stáhnutí. Terčem kritiky se stalo především míchání "HTML a programování". Podobné obavy nedávno vyjádřili i někteří prominentní čeští webaři. Postupně se však ukázalo, že došlo k pouze nepochopení základního konceptu, a nejen FE vývojáři si začali rychle osvojovat a užívat nové fundamenty, které React přinesl [17].

K dnešnímu dni (květen 2016 má 6600 commitů, 41 000 stargazerů a 685 contributorů a je tak jedním z nejoblíbenějších a nejaktivnějších repozitářů na GitHubu. Facebook během této doby uvolnil i další JS projekty jako React Native (React pro iOS a Android) či Immutable.js (immutable kolekce). Zajímavou knihovnou je také GraphQL, což je dotazovací jazyk, kterým v komponentách popíšete, jaká data ze serveru potřebují. Doplňuje ho Relay, který GraphQL umí na straně webu zpracovat a je tak zajímavou alternativou pro datovou komunikaci server-client, která dnes představuje nejtěžší část webové aplikace [17].

Klasické server-side webovky jsou vcelku jednoduché. **Server totiž nemusí udržovat žádný stav**. Přijde mu požadavek od uživatele, poslepuje dohromady nějaké řetězce (část z nich načte třeba z databáze) a celé to pak pošle uživateli do prohlížeče, který to rozparsuje a sestaví DOM [17].

Jednosměrný datový tok

Vlastnosti, neboli sada nezměnitelných hodnot, jsou předávány vykreslovací funkci komponenty jako vlastnosti v její značce HTML. Komponenta nemůže přímo měnit žádné vlastnosti, které jí byly předány, ale může je předat callback funkci, která upraví tyto hodnoty. Tento mechanismu je vyjádřen jako "data tečou dolů, akce nahoru" [19].

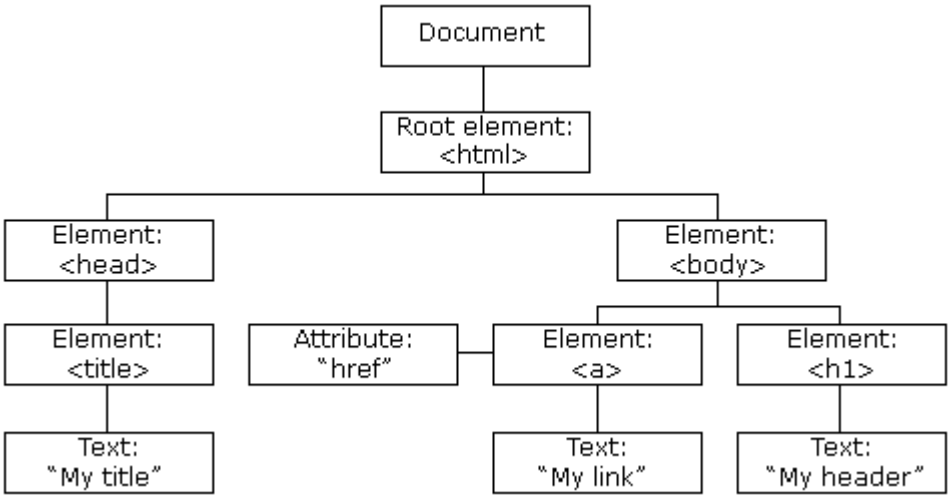
JSX

Komponenty v Reactu jsou typicky psány v JSX, což je syntaxe rozšíření jazyka JavaScript umožňující citovat HTML a používat syntaxi značek HTML k vykreslení sub-komponentů. Toto je rozšíření gramatiky specifické pro React v jazyk JavaScript, jako je zaniklý E4X. Syntaxe HTML je zpracována do volání JavaScriptu v rámci Reactu. Vývojáři mohou také psát v čistém jazyce JavaScript. JSX je podobná další syntaxi rozšíření vytvořené společností Facebook pro PHP a to XHP [19].

Co je DOM ?

Dokumentový model dokumentu DOM (W3C Document Object Model) je platformové a jazykově neutrální rozhraní, které umožňuje programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styl dokumentu [18].

Po načtení webové stránky vytvoří prohlížeč objektový model dokumentu na stránce. Model HTML DOM je vytvořen jako strom objektů:



Obrázek 11 - Model HTML DOM [18]

S objektovým modelem získává JavaScript veškerou potřebnou sílu k vytvoření dynamického kódu HTML [18]:

- může změnit všechny prvky HTML na stránce
- může změnit všechny atributy HTML na stránce
- může změnit všechny styly CSS na stránce
- může odstranit stávající prvky HTML a atributy
- může přidat nové prvky a atributy HTML
- může reagovat na všechny stávající události HTML na stránce
- může na stránce vytvářet nové události HTML

DOM je standardem W3C (World Wide Web Consortium). DOM definuje standard pro přístup k dokumentům. Objektový model dokumentu DOM (W3C Document Object Model) je platformové a jazykově neutrální rozhraní, které umožňuje programům a skriptům dynamicky zpřístupňovat a aktualizovat obsah, strukturu a styl dokumentu [18].

Standard W3C DOM je rozdělen na 3 různé části [18]:

- Core DOM - standardní model pro všechny typy dokumentů
- XML DOM - standardní model pro XML dokumenty
- HTML DOM - standardní model pro HTML dokumenty

Výhody ReactJS [17, 19, 20]:

- **Nejzřejmější výhoda pro začátečníka je ta, že React nás prakticky úplně odstíní od DOMu**. V React komponentách pouze deklarativně zadefinujeme strukturu (HTML) skládáním JS funkcí. Jinými slovy, popíšeme, jak má vypadat výsledná stránka na základě příchozích dat.
- **Optimální renderování**: React si z dodaých dat poskládá svůj vlastní virtuální DOM, který pak pomocí chytrých algoritmů porovnává s tím skutečným DOMem a když najde rozdíly, tak ho nejefektivnějším možným způsobem aktualizuje. V prohlížeči vždy uvidíte aktuální pohled vzhledem k dodaným datům.
- **Je snadné vědět, jak je každá komponenta vykreslena**: Stačí se podívat na funkci vykreslení (což je jediná funkce každá komponenta musí mít definovanou).
- Zajišťuje srozumitelnost a usnadňuje údržbu.
- Můžete použít React s jakýmkoli frameworkem (Backbone.js, Angular.js), protože je to pouze zobrazovací vrstva.
- Nyní je hodně poptávky po vývojářích se zkušeností s ReactJS.

- Využití Flux architektury a její implementace React-Redux je poměrně jednoduché ale velmi užitečné při implementaci aplikace.
- Velká a velmi se rozrůstající komunita stejně jako velké množství knihoven a hotových komponent.
- React-Bootstrap aneb obecně použitelné responzivní komponenty.
- Testování je snadné testovat a můžete také integrovat některé nástroje, jako je Jest.
- JSX usnadňuje čtení kódu vašich komponent. Je také velmi snadné vidět rozložení nebo jak jsou komponenty vzájemně spojeny.
- Strmá křivka učení.

Za prvé, mnoho vývojářů přechází na ReactJS kvůli mnoha výhodám, které má v porovnání s ostatními UI frameworky. Je těžké vypsat nevýhody, protože s nimi existuje mnoho výhod [19].

Nevýhody ReactJS [19, 20]:

- Knihovna samotná je poměrně velká.
- Při porovnávání s monolitickým frameworkem, jako je AngularJS, zjistíte, že neexistuje předdefinovaný způsob, jak strukturu vaší aplikace (jako josu služby, řídicí jednotky a views v Angularu). To znamená, že je odpovědností developera, aby našel své vlastní způsoby, jak účinně spravovat několik částí aplikace bez předem definované struktury. To může vést k významným režijním a dlouhým vývojovým časům, kdy navrhovaná struktura je špatná nebo neúčinná.
- Je to jen zobrazovací vrstva, stále musíte připojit svůj kód pro požadavky Ajax, události a tak dále.

3.1.3.2 AngularJS

AngularJS je framework s otevřeným zdrojovým kódem, vyvíjený společností Google a komunitou programátorů. AngularJS je často považován za framework na tvorbu SPA projektů. Tento framework implementuje architektonický vzor MVC. Ve frameworku se nacházejí šablony pohledu, které jsou reprezentovány kódem v jazyce HTML / HTML5. V šablonách se využívá deklarativní přístup programování, pomocí kterého se pohledy propojují s daty (modely). Deklarativní přístup programování je realizován pomocí tzv. direktiv. Pohled vzniká, když do šablony vložíme data z příslušného modelu.

Jinou logickou jednotkou v kontextu MVC architektury jsou kontrolery, které obsluhují určitou část uživatelského rozhraní. Kontroler se tedy váže k určitému HTML elementu, přičemž může být navázán i k více. V elementu je vytvořen kontext kontroleru, co v praxi znamená, že uvnitř elementu je možné využívat funkcionalitu kontroleru. V kontroleru se typicky nachází služba *$scope,* která reprezentuje daný kontext. Tento kontext je uložen do hierarchické struktury aplikace, a tedy máme jeden rodičovský kontext *rootScope*, pod kterým jsou následně jeho potomci. Modelem je objekt reprezentující datovou jednotku, zpravidla uloženou v kontextu kontroleru, pomocí které klientská část aplikace uchovává aktuální data [21].

Synchronizace mezi modelem a rozhraním je realizována pomocí techniky oboustranného provázání dat (Two Way Data-Binding). Obdoby tohoto mechanismu se nacházejí také ve frameworcích Ember.js a KnockouJS. První směr je z rozhraní do modelu. Toto zajišťuje automatický přenos dat z rozhraní do modelu. V praxi to znamená, že když změníme vstup na rozhraní, tato změna se promítne do modelu daného kontextu. Druhá, a tudíž opačná cesta je fakt, že po změně modelu se tato změna promítne do všech částí rozhraní, kde je model použitý [21].

Součástí frameworku je také systém na využití jednotlivých modulů. V praxi to funguje tak, že na tvorbu aplikace se využívá jádro frameworku AngularJS, a následně se přidávají potřebné moduly, jako moduly na práci se směrováním, animacemi a podobně. Programátor si může vytvářet i moduly vlastní. AngularJS jako framework však poskytuje mnohem více než jen rozdělení kódu do logických vrstev, propojení vrstev, a tedy zjednodušení přístupu. AngularJS poskytuje sadu užitečných funkcí, direktiv, služeb a podobně [21].

Výhody [21]:

- Automatické obousměrné provázání rozhraní s modelem.
- Velká komunita a výborná dokumentace.
- Poskytuje velmi dobrou strukturu - kontrolery, direktivy, služby, moduly, filtry a podobně.
- Jako šablona slouží jazyk HTML.
- Vysoce dobrá testovatelnost kódu.
- Synchronizace se serverem - modely mohou být provázány se serverem pomocí Restful API.

Nevýhody [21]:

- Komplexnost zápisu direktiv v HTML může být nepřehledná a vést k syntaktickým chybám.
- Zavádění do stávajících projektů vyžaduje obvykle změnu struktury aplikace.
- V případě velkého počtu interaktivních elementů na stránce je AngularJS pomalý (Potřeba neustálé synchronizace).

3.1.4 ASP.NET

ASP.NET je součást .NET Frameworku pro tvorbu webových aplikací. Je nástupcem technologie ASP (Active Server Pages) a přímým konkurentem JSP (Java Server Pages). Ačkoliv název ASP.NET je odvozen od starší technologie pro vývoj webů ASP, obě technologie jsou velmi odlišné. ASP.NET je založen na CLR (Common Language Runtime), který je sdílen všemi aplikacemi postavenými na .NET Frameworku. Programátoři tak mohou realizovat své projekty v jakémkoliv jazyce podporujícím CLR, např. Visual Basic.NET, JScript.NET, C#, Managed C++, ale i mutace Perlu, Pythonu a další. Aplikace založené na ASP.NET jsou také rychlejší, neboť jsou před-kompilovány do jednoho či několika málo DLL souborů, na rozdíl od ryze skriptovacích jazyků, kde jsou stránky při každém přístupu znovu a znovu parsovány [15, 16].

Koncept ASP.NET WebForms ulehčuje programátorům přechod od programování klasických aplikací pro Windows do prostředí webu: stránky jsou poskládány z objektů, ovládacích prvků (Controls), které jsou protějškem ovládacích prvků ve Windows. Při tvorbě webových stránek je možné používat ovládací prvky jako tlačítko (Button), nápis (Label) a další. Těmto prvkům lze přiřazovat určité vlastnosti, zachytávat na nich události atd. Tak, jako se ovládací prvky pro Windows samy kreslí do formulářů na obrazovku, webové ovládací prvky produkují HTML kód, který tvoří část výsledné stránky. ASP.NET MVC je další oficiální framework postavený na technologii ASP.NET. Tento framework umožňuje snadněji vyvíjet aplikace podle architektury Model-View-Controller [15, 16].

Výhody [15, 16]:

- Vyšší rychlost aplikace (díky kompilovanému kódu).
- Jednodušší odladění chyb při samotném vývoji.
- Uživatelsky definované ovládací prvky lze použít jako šablony (redukce kódu).
- Velké množství ovládacích prvků a knihoven tříd (rychlejší vývoj aplikací).

- Možnost využít mnoha programovacích jazyků.
- Schopnost cachovat celou stránku nebo její část.
- Od verze 2 generuje ASP.NET validní HTML 4.0 / XHTML 1.0/1.1 a JavaScript.

Nevýhody [15, 16]:

- Špatná rozšířenost mezi hostingy a vysoká cena za hostování.
- Uzavřený kód ve vlastnictví Microsoftu.
- Špatná, případně žádná podpora pod jiným operačním systémem než MS Windows.
- Session jednotlivých verzí není kompatibilní.
- Vysoká závislost na JavaScriptu.

3.1.4.1 ASP.NET WebForms

Ačkoliv webový protokol HTTP je sám o sobě bezstavový, událostmi řízené programování zachování stavu (uchování kontextu mezi jednotlivými požadavky) vyžaduje. ASP.NET tento problém řeší kombinací HTML a JavaScriptu pomocí dvou základních technik. ViewState uchovává informace mezi postbacky (opakovaným odesíláním formuláře na server) v zakódovaném tvaru ve skrytých formulářových polích. Jeho výhodou je, že využívá pouze HTML a nevyžaduje žádnou speciální podporu na straně serveru ani klienta. Nevýhodou je, že se mezi serverem a klientem přenáší větší objem dat, zejména je-li ViewState využíváno nesprávně [15, 16].

Session State oproti tomu ukládá veškeré informace na straně serveru a předává (typicky jako cookie nebo součást URL) pouze jednoznačný identifikátor. To sice zmenšuje objem přenášených dat, ale klade vyšší nároky na výkon serveru. Pokud se sessions používají nesprávně, může být server náchylný i k Denial of Service útokům. Oproti ASP umožňuje ASP.NET ukládání session state do samostatného procesu nebo na SQL server. To zjednodušuje použití session ve webových farmách, zvyšuje výkon a umožňuje stav zachovat i při restartu serveru [15, 16].

3.1.4.2 ASP.NET MVC

Na přelomu roku 2007 a 2008 ohlásila firma Microsoft plán na vývoj ASP.NET MVC frameworku. Tento framework umožňuje tvorbu webových aplikací podle softwarové architektury Model-View-Controller. ASP.NET MVC má představovat alternativu oproti

WebForms. Narozdíl od WebForms aplikace vytvořené pomocí ASP.MVC nevyžadují ViewState a dají se snadněji testovat. V současné době se ASP.NET MVC nachází ve verzi 3 Beta (5. února 2011). Microsoft ujistil komunitu, že vydáním ASP.NET MVC nekončí vývoj WebForms (časem kdo ví) [15, 16].

3.2 Desktopový klienti

3.2.1 Java

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a představila 23. května 1995. Jde o jeden z nejpoužívanějších programovacích jazyků na světě. Podle TIOBE indexu je Java nejpopulárnější programovací jazyk. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE). Tyto technologie se jako celek nazývají platforma Java. Dne 8. května 2007 Sun uvolnil zdrojové kódy Javy (cca 2,5 miliónů řádků kódu) a Java bude dále vyvíjena jako open source [23].

Výhody [22]:

- **Učení jazyka Java je snadné**: Přestože toto prohlášení může podnítit mnoho protikladných názorů, zůstává to fakt. Dokonce i když člověk nemá programovací zázemí a nikdy se nenaučil úvodní programovací jazyky jako C ++, učením se konceptů Java to nebude překážkou. Bez nutnosti používat a chápat magické znaky, jako jsou „Generics Angle Brackets“ atd., podporuje Java anglickou syntaxi a příkazy. Po pochopení počátečních hodin se zbytek často stává jednodušším.
- **Používá koncepci OOP**: Aplikace, které jsou vyvinuty pomocí konceptu Java OOP (Object Oriented Programming), jsou kompetentnější, protože jsou rozšiřitelné, škálovatelné a flexibilní. Má bohatou knihovnu standardních návrhových vzorů a dalších osvědčených postupů. Otevřené zdroje, jako je Spring, atd., Používají koncepty objektově orientovaného programování, což je ještě přizpůsobivější pro vývoj aplikací v Javě.

- **Platformě nezávislá**: Od doby, kdy Java získala popularitu, tj. od devadesátých let minulého století, jeho nezávislost na platformě způsobila, že se jedná o vysoce žádanou technologii. Tato funkce doplnila slovníčku „Write Once Run Anywhere“ v pravém slova smyslu, protože otevřela dveře mnoha novým vývojům. Až do dnešního dne je to i nadále důvodem, proč je mnoho aplikací Java postaveno na systému Windows a běží na systému UNIX.
- **Komunita a podpora**: Podpora komunity Javy byla blaho všech programátorů. Mnoho různých fór pro posílání dotazů, jako Stackoverflows a další uživatelské skupiny vždy rozšířily podporu a pomoc při všech tématech. Poskytování a konzultace o vývoji aplikací Java od odborníků zdarma přispělo k propojení jedné z těchto největších a nejbohatších komunit.

Nevýhody [22]:

- **Paměť a rychlost**: Vzhledem k tomu, že JAVA vyžaduje vysokou paměťovou kapacitu a využívá více paměti, a má pomalejší výkon ve srovnání s jinými jazyky.

3.2.1.1 Swing

Swing je knihovna uživatelských prvků na platformě Java pro ovládání počítače pomocí grafického rozhraní. Knihovna Swing poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí Swingu je možno vytvářet okna, dialogy, tlačítka, rámečky, rozbalovací seznamy atd [24].

3.2.1.2 SWT

Standard Widget Toolkit (SWT) je knihovna grafických uživatelských prvků pro platformu Java. Původně byla tato knihovna vyvinuta firmou IBM a nyní je ve správě nadace Eclipse Foundation, spolu s projektem Eclipse IDE. Je alternativou k AWT a Swing - GUI nástrojů od společnosti Sun Microsystems jako součást Java Platform, Standard Edition.

Knihovna SWT je napsaná v Javě. K vykreslovaní prvků GUI používá nativní knihovny operačních systémů prostřednictvím JNI (Java Native Interface), a to obdobným způsobem jako programy psané s použitím nativního API operačního systému. Programy používající SWT jsou přenosné, ale implementace „toolkitu“, ačkoliv je napsán v Javě, je unikátní pro každou platformu [25].

3.2.2 C# (.Net)

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework, později schválený standardizačními komisemi ECMA (ECMA-334) a ISO (ISO/IEC 23270). Microsoft založil C# na jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi). C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení (PDA a mobilní telefony) atd [27].

Vlastnosti [27]:

- V C# neexistuje vícenásobná dědičnost – to znamená, že každá třída může být potomkem pouze jedné třídy. Toto rozhodnutí bylo přijato, aby se předešlo komplikacím a přílišné složitosti, která je spojena s vícenásobnou dědičností. Třída ale může implementovat libovolný počet rozhraní.
- Neexistují žádné globální proměnné a metody, všechny musí být deklarovány uvnitř tříd. Náhradou za globální proměnné a metody jsou statické metody a proměnné veřejných tříd.
- V objektově orientovaném programování se z důvodu dodržení principu zapouzdření často používá vzor, kdy k datovým atributům třídy lze zvenčí přistupovat pouze nepřímo, a to pomocí dvou metod: metody get (accessor) a metody set (mutator). V C# lze místo toho definovat tzv. property, která zvenčí stále funguje jako datový atribut, ale uvnitř obsahuje prostor pro definici obou těchto metod. Výhodou je jednodušší práce s datovým atributem při zachování principu zapouzdření.
- C# je typově bezpečnější než C++. Jediné předdefinované implicitní konverze jsou takové, které jsou považovány za bezpečné. Příkladem budiž rozšiřování celočíselných typů (např. z 32bitového na 64bitový) nebo konverze z odvozeného typu na typ rodičovský. Neexistuje však implicitní konverze z celočíselných typů na boolean ani implicitní konverze mezi výčtovými a celočíselnými typy.
- C# nepotřebuje a ani neobsahuje dopřednou deklaraci – pořadí deklarace metod není důležité.
- Jazyk C# je case sensitive – rozlišuje mezi velkými a malými písmeny. Identifikátory „hodnota“ a „Hodnota“ tedy nejsou, na rozdíl od VB .NET, ekvivalentní.

3.2.2.1 .NET a jeho Historie

(„dotnet“ podle anglického dot NET = tečka NET, NET pochází z network, síť) je zastřešující název pro soubor technologií v softwarových produktech, které tvoří celou platformu, která je dostupná nejen pro Web, Windows i Pocket PC. Common Language Infrastructure je standardizovaná specifikace jádra .NET. Základní komponentou je Microsoft .NET Framework, prostředí potřebné pro běh aplikací a nabízející jak spouštěcí rozhraní, tak potřebné knihovny. Pro vývoj .NET aplikací vydal Microsoft Visual Studio .NET [28].

.NET Framework 4.7

Dne 5. dubna 2017 společnost Microsoft oznámila, že .NET Framework 4.7 byl integrován do aktualizace Windows 10 Creators Update, přičemž byly naplánovány samostatné instalátory pro další verzi systému Windows. Aktualizace pro aplikaci Visual Studio 2017 byla k tomuto datu vydána za účelem přidání podpory pro cílení .NET Framework 4.7. Slíbený samostatný instalátor byl vydán velmi nedávno a to dne 2. května 2017 [28].

Nové funkce v rozhraní .NET Framework 4.7 zahrnují [28]:

- Vylepšená kryptografie s kryptografií eliptické křivky
- Zlepšete podporu zabezpečení vrstvy Transport Layer (TLS), zejména pro verzi 1.2
- Podpora podpory vysokého DPI ve formách Windows
- Více podpory dotyku a stylus v systému Windows Presentation Foundation (WPF)
- Nové tiskové rozhraní API pro WPF

.NET Framework 4.6

.NET Framework 4.6 byl oznámen dne 12. listopadu 2014. Byl propuštěn 20. července 2015. Podporuje nový kompilátor Just-in-Time (JIT) pro 64bitové systémy s názvem RyuJIT, který nabízí vyšší výkon a podporu pro sady instrukcí SSE2 a AVX2. Formáty WPF a Windows Forms obdržely aktualizace pro scénáře s vysokým rozlišením DPI. Podpora TLS 1.1 a TLS 1.2 byla přidána do WCF. Tato verze vyžaduje systém Windows Vista SP2 nebo novější [28].

.NET Framework 4.5

Tato verze .NET Frameworku vyšla 15. srpna 2012. Zároveň vyšlo i Visual Studio 2012 a nové verze C# a VB .NET. Od této verze .NET Frameworku je ukončena podpora Windows XP (potřebujete tedy Windows Vista nebo novější). Největšími novinkami jsou [29]:

- Async a Await metody
- Podpora UTF-16 v konzoli
- Vylepšení ZIP komprese
- Podpora HTML 5 v ASP .NETu
- Přidání nového programovací rozhraní pro HTTP aplikace a to v System.Net.Http a System.Net.Http-.Headers namespacech
- Přidání podpory pro URI

.NET Framework 4.5.1

Tato aktualizace .NET Frameworku přinesla hlavně podporu multiplatformních aplikací. Zájemce o podrobnější informace odkáži na MSDN [29].

.NET Framework 4.5.2

Zatím nejnovější verze .NET Frameworku přináší změnu velikosti kontrolek ve Windows Formech na základě nastavení DPI v systému. Dále tato aktualizace přinesla nové API pro ASP .NET, které vylepšuje práci s hlavičkami a stavovými kódy. Další podrobnější informace můžete nalézt na MSDN [29].

.NET Framework 4.0

Tato verze .NET Frameworku vyšla 12. dubna 2010 spolu s Visual Studiem 2010 a CLR 4.0, C# 4.0 a VB .NET 10.0 [29].

.NET Framework 4.0 přinesl mnoho novinek a mezi nejvýznamnější patří [29]:

- Výrazné rozšíření paralelního programování. K tomu rozšíření pomohly technologie PLINQ (Parallel LINQ) a TPL (Task Parallel Library).
- Přidání nových funkcí do C# a VB .NETu. Například dynamický výběr metod, pojmenované argumenty a volitelné argumenty.
- Přidání podpory pro výpočty s libovolnou přesností (System.Numeric-s.BigInteger) a pro práci s komplexními čísly (System.Numeric-s.Complex)

.NET Framework 3.5

Verze 3.0 .NET Frameworku byla vypuštěna do světa 19. listopadu 2007. Stejně jako .NET Framework 3.0 běží i verze 3.5 na CLR 2.0. V této verzi přibylo několik nových metod a vlastností pro třídy v .NET Frameworku, ale největší novinkou v této verzi je bezpochyby Language Integrated Query, zkráceně LINQ. Vyšly také nové verze C# (3.0) a Visual Basic

.NETu. (9.0). Spolu s touto verzí .NET Frameworku vyšel i .NET Compact Framework 3.5. Pro tuto verzi .NET Frameworku vyšlo Visual Studio 2008. Tato verze .NET Frameworku je již obsažena v Windows 2008 R2 a Windows 7. Service pack pro .NET Framework vyšel 11. srpna 2008. Obsahem tohoto service packu byly určité změny ve WPF. Avšak největšími novinkami, které tento service pack přinesl, jsou ADO .NET Entity Framework, ADO .NET Data Services a ASP .NET MVC Framework [29].

.NET Framework 3.0

.NET Framework 3.0 byl vydán 21. listopadu 2006. Je součástí operačních systému Windows Vista a Windows Server 2008. Je také k dispozice ke stažení pro Windows XP se SP2 a Windows Server 2003. Tato verze .NET Frameworku používá stejnou verzi CLR jako .NET Framework 2.0 [29].

Nejvýraznějšími novinkami v .NET Frameworku 3.0 jsou [29]:

- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- Windows Workflow Foundation (WWF), slouží k modelování procesů, což je jedním z moderních přístupů vývoji aplikací. Více informací můžete nalézt v oficiální dokumentaci.
- zrušení podpory Windows CardSpace. Tato technologie uchovávala informace o uživateli a mohla je například předat aplikaci při přihlašování.

.NET Framework 2.0

Verze .NET Frameworku 2.0 byla vydána 22. ledna 2006. Tato verze byla vydána spolu s Visual Studiem 2005 a Microsoft SQL Serverem 2005. Jedná se o poslední verzi, která je spustitelná na Windows 98 a Windows ME [29].

.NET Framework přišel s následujícími novinkami [29]:

- CLR 2.0
- C# 2.0 a Visual Basic .NET 8.0
- Podpora 64 bitových aplikací a hardwarových platforem
- ASP .NET:
- Nové kontrolky
- Master Page
- Možnost vývoje stránek bez psaní kódu

- Možnost různého vykreslování kontrolek na základě prohlížeče
- Parciální třídy
- Nullovatelné typy
- Anonymní metody
- Genericita a generické kolekce

.NET Framework 1.1

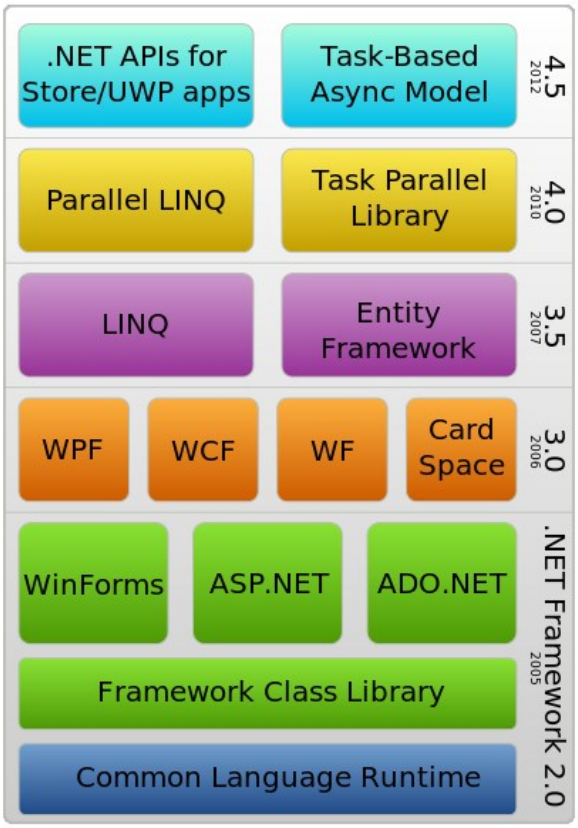
Verze 1.1 byla prvním verze větší aktualizace .NET Frameworku 1.0. Byla také první verzí .NET Frameworku, která již byla obsažena ve Windows a to konkrétně ve verzi Windows Server 2003. Byla vydána 3. dubna v roce 2003 a podpora pro ní byla ukončena v říjnu roku 2008. Tuto verzi také obsahovalo druhé vydání Visual Studia .NET 2003.

Tato verze .NET Frameworku sebou přinášela mimo jiné [29]:

- ASP .NET Mobile Controls, což je sada ovládacích prvků určená pro navigaci na stránce pomocí mobilní zařízení (mobilní telefony, PDA).
- Začlenění podpory ODBC přímo do .NET frameworku v namespacu System.Data.Od-bc;. Tato knihovna byla dříve k dispozici na internetu ke stažení v případě potřeby.
- Podpora protokolu IPv6. Knihovny pro práci s protokolem IPv6 byly začleněny do namespacu System.Net;.
- Side-by-Side execution, což je technologie, která umožní nainstalovat program s více verzemi kódu, aby si aplikace mohla následně vybrat, na které verzi CLR poběží, nebo které komponenty využije. Více informací naleznete v oficiální dokumentaci.
- Změny v zabezpečení .NET Frameworku
- .NET Compact Framework, což je verze .NET Frameworku pro malé zařízení
- Nová verze C# 1.2.

.NET Framework 1.0

První verze .NET frameworku vyšla 13. února 2002. Obsahovala C# 1.0, Visual Basic .NET 7.0 a samozřejmě CLR 1.0, bez které by samotný .NET framework nefungoval. Zároveň vyšlo samozřejmě i vývojové prostředí, konkrétně Visual Studio .NET. První verze .NET Frameworku podporovala Windows NT, Windows 98, Windows ME, Windows 2000 a Windows XP, ale nebyla součástí ani jednoho z těchto systému, musela se vždy doinstalovat. Podpora byla ukončena 10. července 2007 [29].



Obrázek 12 – obsah verzí .NET frameworku [26]

3.2.2.2 WinForm a WPF

WPF (Windows Presentation Foundation) je framework pro komplexní tvorbu bohatých formulářových aplikací, který je součástí .NET frameworku od verze 3.0. Disponuje širokou paletou formulářových prvků a také umožňuje bohaté stylování vzhledu aplikace [30].

Komponentová architektura

Framework nám nabízí spoustu hotových komponent, ze kterých formulář jednoduše poskládáme. Jedná se tedy o různá tlačítka, pole, posuvníky, popisky a další komponenty, které Microsoft nazval controls (česky asi kontrolky). Nic nám samozřejmě nebrání v tvorbě vlastních kontrolek, když by nám nějaká nestačila, ale to se nestává příliš často. Během seriálu si mnoho kontrolek popíšeme a naučíme se s nimi pracovat [30].

WPF a Windows Forms

Kromě WPF je v .NET frameworku stále přítomný starší formulářový framework Windows Forms. Ačkoli Microsoft Windows Forms ještě neoznačil jako zastaralý a v současné době se paralelně používají oba frameworky, WPF je technologicky mnohem dále. I když mnoho existujících aplikací stále používá Windows Forms, nové aplikace již prakticky nemá smysl vyvíjet v ničem jiném, než právě ve WPF [30].

Proč WPF vzniklo ?

Technologie jdou dopředu a je tlak na stále graficky bohatší aplikace. Zde starší Windows Forms selhává zejména z těchto důvodů [30]:

- V posledních několika letech na trhu začaly dominovat mobilní zařízení, ale WF aplikaci je problém uzpůsobit jejich fyzické velikosti kvůli slabé podpoře DPI. Nelze jednoduše používat tu samou aplikaci např. na mobilu, tabletu a na stolním PC s fullHD rozlišením. Proto WPF zavádí jako jednotku délky tzv. DIP (Device Independent Pixel) a čistě vektorovou grafiku, aby výsledná aplikace vypadala vždy na každém zařízení dobře.
- WF ukládá u každé kontrolky její absolutní pozici na formuláři, což se pro návrh složitějších formulářů nehodí. Na webu se pro tvorbu prezentační vrstvy (to je ta část

aplikace, se kterou uživatel komunikuje) velmi osvědčil jazyk HTML. C# se zde stejně jako Java inspiruje a zavádí definici formulářů pomocí jazyka XML, v našem případě přesněji XAML. Kromě lepšího oddělení prezentace a logiky přináší jednoduchou podporu tzv. bindingů, které umožňují napojit vlastnosti objektů přímo na kontrolky formuláře. Daní za kvalitnější aplikaci je více práce s návrhem formuláře a nutnost ovládání dalšího jazyka, i když je velmi jednoduchý.

- Používání původního vykreslovacího rozhraní Windows (GDI) je pomalé a přináší s sebou mnoho omezení. WPF používá k vykreslování formulářů Direct3D. To je rozhraní pro akcelerovanou grafiku, jehož plné využití poprvé přinesly Windows Vista. WPF aplikace jsou tedy svižnější a méně zatěžují procesor. Díky nezávislosti na GDI se je možné odpoutat od strohé palety základních kontrolek operačního systému a vytvářet graficky bohaté aplikace. Nemáme prakticky žádná omezení, můžeme např. vkládat obrázky do tlačítek, do položek comboboxu, cokoli zprůhlednit, naše aplikace skinovat nebo dokonce jednoduše animovat pomocí tzv. storyboardů. Sice rozhodně nebudeme vytvářet omalovánky a naše aplikace by měla vypadat stále jako aplikace, občas se však hodí ostylovat si nějakou část přesně jak to potřebujeme. Níže máte takovou extravagantní ukázku toho, jak může vypadat formulář v WPF. K jeho tvorbě byl použit nástroj Expression Blend.

4 TECHNOLOGIE PRO TVORBU SERVEROVÉ ČÁSTI APLIKACÍ

Pro tvorbu tzv. backendu aplikací se používá řada běžně známých jazyků a technologií, my pro tuto aplikace chceme svůj backend mít ve formě webové služby a mít tak její API dostupné i pro jiné klientské aplikace.

4.1.1 JavaScript

JavaScript jako takový jsem již popsal v předchozí kapitole, pro se zaměříme jen na serverové technologie. Dnes je JavaScript rychle rostoucí jako server-side technologie díky vydání Node.js v roce 2009.

4.1.1.1 NodeJS

Node.js je softwarový systém navržený pro psaní vysoce škálovatelných internetových aplikací, především webových serverů. Programy pro Node.js jsou psané v jazyce JavaScript, hojně využívající model událostí a asynchronní I/O operace pro minimalizaci režie procesoru a maximalizaci výkonu. Node.js se skládá z V8 JavaScript engine od společnosti Google a několika standardních knihoven. Node.js vytvořil v roce 2009 Ryan Dahl, jeho následný rozvoj byl sponzorován firmou Joyent, jeho zaměstnavatelem [31].

Platforma Node.js je JavaScriptové běhové prostředí postavené na JavaScriptovém interpretu V8 od Google Chrome. Její funkcí je vytvářet vysoce škálovatelné aplikace, běžící na severové části aplikace, nebo síťové aplikace. Pro vytvoření souběžnosti (tj. spouštění více procesů najednou) na serverové části aplikace se používají hlavně dvě techniky: a to tzv. vlákna (angl. threads) nebo události (angl. events). Vlákna se používají např. ve webovém serveru Apache. Nicméně problém s vlákny je ten, že při větším počtu se snižuje výkon celé aplikace, navíc nastává velká spotřeba operační paměti. V druhém případě existují tzv. emitory, které generují události, dále existuje smyčka, jež postupně načítá tyto události nebo čeká až nějaké nastanou a potom pro každou událost spustí její obslužnou funkci, která vrátí nějaký stav. Tímto způsobem může být více akcí navázáno pouze na jedno vlákno. Rozdíl času při spouštění obslužné funkce a přepínáním mezi vlákny je značný. Tato architektura se nazývá událostmi řízená (angl. event-driven architecture). Node.js je jednovláknový a implementuje tzv. asynchronní, událostmi řízené, běhové prostředí pro Javascipt. Asynchronní neboli neblokující, tj. např. při načítání dat se nečeká na to, až se operace dokončí a hned se přechází na další akci. Až se dokončí načítání, výsledek se analyzuje ve smyčce, která zpracovává události [32].

Dalšími důležitými znaky Node.js jsou [32]:

- znovupoužitelný kód
- transport dat ve formátu JSON
- rychlost, neboť se využívá interpret V8

4.1.2 Java

Javu jako takovou jsem již popsal v předchozí kapitole, pro se zaměříme jen na webové služby.

4.1.2.1 Jersey

Jersey RESTful Web Services Framework je open source framework pro vývoj RESTful Web Services v Javě. Poskytuje podporu rozhraní API JAX-RS a slouží jako referenční implementace JAX-RS (JSR 311 & JSR 339) [34].

Následující komponenty jsou součástí Jersey [34]:

- Core Server: Pro budování RESTful služby založené na anotaci (jersey-core, jersey-server, jsr311-api)
- Core Client: Pomáhá vám komunikovat se službami REST (jersey-client)
- Podpora JAXB
- Podpora JSON
- Integrační modul pro Spring a Guice

Oproti zbytku frameworků je relativně nový (2012). Mezi ostatními implementacemi JAX-RS se jedná o zdaleka nejmodulárnější. Jedná se o referenční implementaci od společnosti Oracle. Na vývoji frameworku se téměř výhradně podílí čeští vývojáři z Pražské pobočky Oracle. Je dostupný v balíku s GlassFish serverem [33].

Výhody:

- Vysoký výkon a využití aktuálních návrhářských postupů
- Dobrý URI routing a URI building
- Integrace s dalšími frameworky jako Grizzly, Netty či Spring
- Nevyžaduje servlet kontejner
- Hladká integrace JUnit testů

Nevýhody:

- Chaotická implementace dependency injection od verze 2.0
- Verze 1.x využívají starou implementaci JAX-RS
- Mnoho online zdrojů je pro verze 1.x, a jsou tím pádem nevhodné pro 2.x
- Chybí bezpečnostní protokol OAuth 2.0

4.1.3 C# (.NET)

C# jako takový jsem již popsal v předchozí kapitole, pro se zaměříme jen na webové služby.

4.1.3.1 .NET Web API

Služba HTTP není určena pouze pro zobrazování webových stránek. Je také výkonnou platformou pro vytváření API, které vystavují služby a data. HTTP je jednoduchý, flexibilní a všudypřítomný. Téměř každá platforma, o které si myslíte, má knihovnu HTTP, takže služby HTTP mohou oslovit širokou škálu klientů, včetně prohlížečů, mobilních zařízení a tradičních desktopových aplikací [35].

Rozhraní ASP.NET Web API usnadňuje sestavování služeb HTTP, které jsou poskytovány širokému spektru klientů, včetně prohlížečů a mobilních zařízení. Představuje ideální platformu pro sestavování aplikací RESTful v rozhraní .NET Framework. Podobně jako knihovna ASP.NET MVC využívá kontrolery a akce. Kontrolery slouží jako vstupní brány pro jednotlivé typy poskytovaných entit, akce potom jako jednotlivé operace, jež jsou definovány nad danými entitami. Knihovna nese název Microsoft.AspNet.WebApi a je možné ji stáhnout a nainstalovat jako NuGet balíček [35].

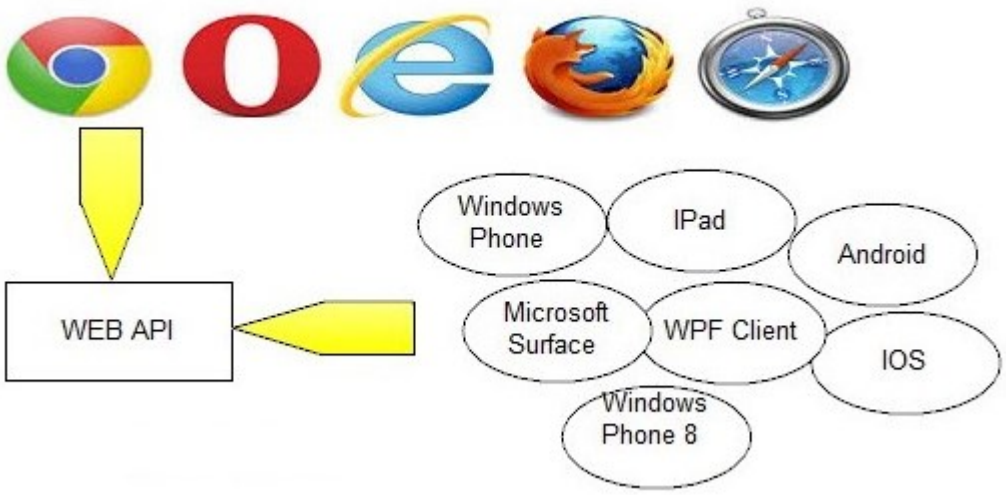
Asp.Net Web API podporuje konvenční CRUD akce, protože pracuje s požadavky HTTP GET, POST, PUT a DELETE. Aplikace může být hostována v rámci aplikace nebo ve službě IIS. [36]

Životní cyklus HTTP zprávy nalezneme v příloze P I.

Výhody [36]:

- ASP.NET Web API může být backendem pro nativní aplikace (postavené pro konkrétní platformu) běžící na mobilních zařízeních, kde SOAP není podporován, protože HTTP je běžný faktor ve všech platformách. Také nativní aplikace běžící na platformách jiných, než Windows mohou použít ASP.NET Web API jako backend.

- Z prvního bodu můžeme říci, že pokud potřebujeme Webovou službu a nepotřebujeme SOAP, je nejlepší volbou ASP.Net Web API.
- ASP.NET Web API bude vhodná pro webové aplikace bohatých klientů, které silně používají AJAX k získání podnikové nebo datové vrstvy. Klientskou aplikací může být cokoliv, co je schopno porozumět protokolu HTTP. Nemá nudnou a rozsáhlou konfiguraci, jako je služba WCF REST.
- ASP.NET Web API nám umožňuje velmi jednoduchou tvorbu služeb. Pomocí WCF REST je tvorba služeb obtížnější než pomocí ASP.NET Web API.
- Aplikační rozhraní API ASP.NET je založeno pouze na protokolu HTTP a snadno se definuje, vystavuje a využívá způsobem REST
- ASP.NET Web API je lehká architektura.
- ASP.NET Web je open source. Zdrojový kód ASP.NET MVC byl vydán pod licencí Apache 2.0.



Obrázek 13 – ASP.NET Web API [36]

II. PRAKTICKÁ ČÁST

5 ANALÝZA POŽADAVKŮ A NÁVRH APLIKACE

Součástí analýzy požadavků bude především pochopení účelu systému, jeho funkcionální specifikace a vytvoření datového modelu, který bude v systému využíván. Mimo toho ale také prozkoumání ostatních požadavků a systém a vybrání vhodných technologií z hlediska těchto požadavků a z hlediska jejich vhodnosti pro daný účel.

5.1 Požadavky

Nejprve ovšem vypsání všech počátečních požadavků na tuto aplikaci.

5.1.1 Funkční požadavky

- Přihlášení do aplikace bude provedeno pomocí firemního doménového účtu (souvisí s prvním bodem nefunkčních požadavků) a to pomocí ověření účtu přes jinou firemní službu starající se o databázi firemních účtů (TimurServices).
- Pomocí aplikace bude možná správa práv jednotlivých uživatelů pro administrátory.
- V aplikaci budou dvě úrovně oprávnění Administrátor a Reviewer.
- Pomocí aplikace bude možná správa kandidátů a všech informací o nich. Jejich prohlížení, vytváření, editace a mazání. Reviewer bude moct pouze prohlížet.
- Informace o kandidátech budou rozděleny do dvou skupin. Administrátor bude mít plný přístup ke všem, včetně jejich editace a Reviewer pouze k těm základním (konkrétní informace budou vypsány při tvorbě datového modelu).
- Každý uživatel bude moct ke kandidátům psát komentáře.
- Ke každému kandidátovi bude možné nahrát soubory a přistupovat k nim (CV atd.).
- Ke každému kandidátovi bude možné přiřadit test.
- Pomocí aplikace bude možná správa testů. Jejich prohlížení, vytváření, editace a mazání. Reviewer bude moct pouze prohlížet.
- Pomocí aplikace bude možná správa otevřených pozic. Jejich prohlížení, vytváření, editace a mazání. Reviewer k nim vůbec nebude moct přistupovat.
- V aplikaci bude zobrazena základní statistika o kandidátech a otevřených pozivích.
- Pomocí aplikace bude možné pro kandidáta vyplnit test.
- Pomocí aplikace bude možné test ohodnotit.

5.1.2 Nefunkční požadavky

- Aplikace bude přístupná pouze v rámci intranetu firmy.

- Aplikace bude mít oddělenou klientskou a serverovou část.
- Aplikace bude moct být používána na PC a tabletu. Tj. bude mít dostatečně responzivní uživatelské rozhraní.
- Aplikace musí být dobře udržovatelná a rozšiřitelná.

5.2 Analýza a specifikace implementace

V této kapitole se budu zabývat analýzou požadavků uvedených výše. Sestavím potřebné diagramy případů užití a doménové modely a ostatní specifikace jako podklad pro implementaci aplikace.

Požadavky jsou poměrně jasné ale jsou zde věci nebo spíše otázky které je třeba zodpovědět a z vypsaných požadavků nejsou jasně patrné. Nyní odpovím na základní otázky uvedení v teoretické části v kapitole analýzy požadavků, podle informací získaných od stakeholderů. Ještě před tím je potřeba uvést kdo jsou hlavními stakeholdery této aplikace, je to HR oddělení a management.

Otázky jsou následující:

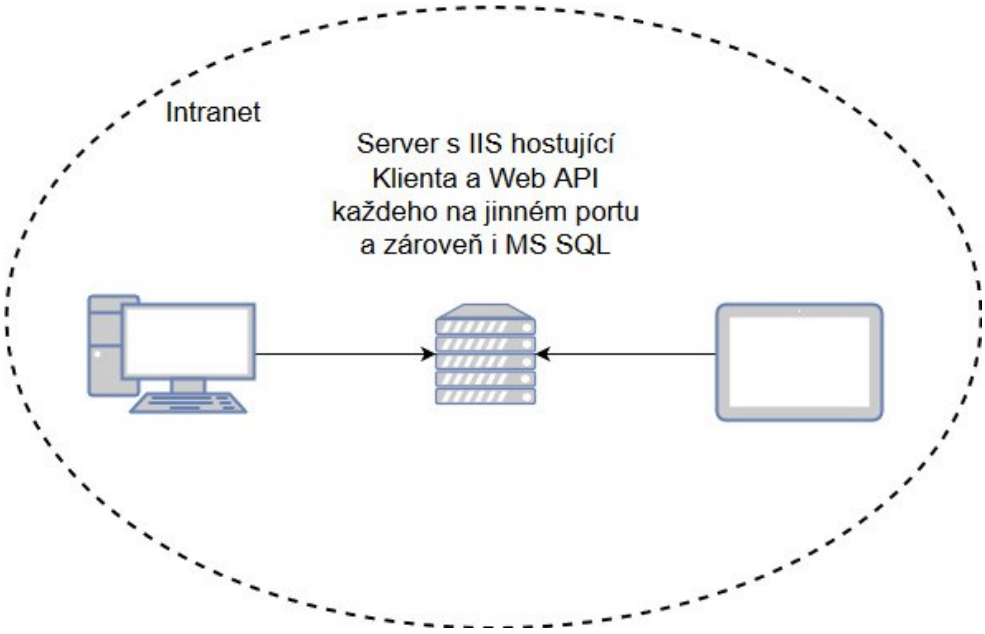
- Jaký je účel systému (cíle)?
- Jaké objekty jsou zahrnuty?
- Kde je systém umístěn?
- Kdy by se měly věci dělat?
- Proč je systém nezbytný (cíle nebo problémy, které hodlá řešit)?

Účelem systému je zefektivnit náborový proces kandidátu na nové pozice ve firmě. Hlavními zúčastněnými jsou HR oddělení a management. Během pohovorů a během zpracování dat o možných kandidátech bude možné díky tomuto systému udržovat databázi uchazečů se všemi informacemi na jednom místě a také umožní, aby HR mohlo snadno nechat kandidáta vyplnit test a nebylo díky tomu potřeba nikoho s odbornými znalostmi přímo u pohovoru. Systém je umístěn na serveru a dostupný pouze v rámci intranetu firmy.

5.2.1 Výběr technologií pro klienta a serverovou část aplikace

Jako první krok bude nejlepší vybrat technologie, které při implementaci aplikace použiji. Vzhledem k požadavku, aby aplikace byla použitelná jak na PC, tak i na tabletu, by bylo v hodné, aby zvolná technologie měla dobrou podporu multiplatformnosti. Další požadavek na dobrou udržitelnost také napovídá, že chce vybrat takovou technologii a framework, který má dobrou základnu a jen tak jeho podpora nezmizí. Dále je třeba také vzít v úvahu distribuci aplikace, ta by sice nebyla problémem, ale pokud zvolím pro klientskou část aplikace webovou technologii, nebudu ji muset vůbec řešit, jelikož aplikace bude spustitelná na každém zařízení s prohlížečem bez nutnosti její instalace. Vzhledem k tomu, že v požadavcích není žádné kritérium, které by zabraňovalo použití webového klienta, vyberu tedy z webových technologií. Mám na výběr mezi frameworky jazyků PERL, PHP, JavaScript a technologií ASP.NET. Vzhledem rostoucí a rostoucí oblibě **JavaScriptu** a velmi dobrým frameworkům jako jsou React a Angular, jsem zvolil právě tuto cestu. Nakonec jsem zvolil jednoduch a velmi efektivní **ReactJS**. Ostatní technologie a frameworky by samozřejmě šli pro vývoj této aplikace také použít. Ale vzhledem k tomu, že žádný požadavek kromě dobré udržitelnosti aplikace není pro rozhodování při výběrů důležitý přihlédl jsem hlavně k výhodám Reactu popsaných v teoretické části a také k radám kolegů a komunity vývojářů. React se totiž oproti ostatní technologiím zaměřuje pouze na zobrazovací vrstvu aplikace (uživatelské rozhraní) a na jednotlivé komponenty aplikace. To je vše, co od klienta potřebuji, vše ostatní bude mít na starosti serverová část aplikace.

Pro serverovou část bude nejvhodnější zvolit technologii pro tvorbu RESTful webové služby. Na výběr je mezi Node.js, Javou (např. Jersey) a ASP.NET Web API. Pro serverovou část jsem si vybral **ASP.NET Web API** a to hlavně kvůli dostupnosti IIS a MS SQL jejichž licence firma má a klient by stejně běžel na IIS. Navíc je možné využít dalších výborných knihoven jako například Entity framework pro přístup k databázi a operacemi nad ní.



Obrázek 14 – Schéma okolí systému

5.2.2 Diagramy případů užití (Use Case)

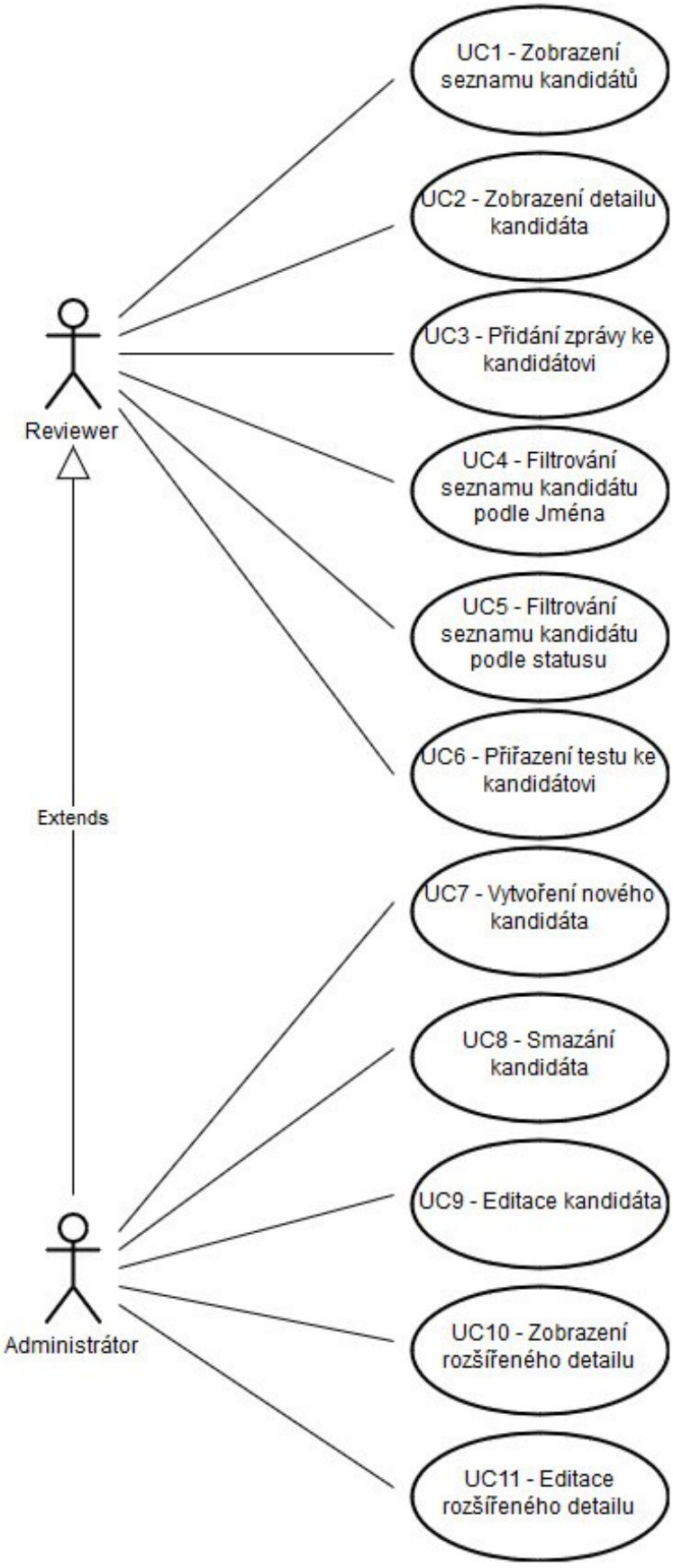
V této kapitole vytvořím všechny potřebné případy užití systému pro čisté pochopení veškeré funkcionality.

Pro vytváření diagramů případů užití jsem využil bezplatného online diagram buildera Draw.io. Dostupný na stránce: www.draw.io.

Draw.io není nejlepším vývojovým diagramem nebo grafickou aplikací, ale je nejlepší, která nabízí zcela bezplatnou úroveň služeb. Draw.io má placenou verzi pro server Confluence, což je způsob, jakým je aplikace zisková, ale všechny její další verze, včetně webové verze online jsou zdarma. Tento nástroj vám pomáhá vytvářet diagramy a další vizuální prvky mnohem jednodušeji, než kdybyste používali vektorový software. Při použití s Disku Google má aplikace Draw.io dobrou podporu pro spolupráci v reálném čase, takže více než jedna osoba může pracovat na diagramu současně.

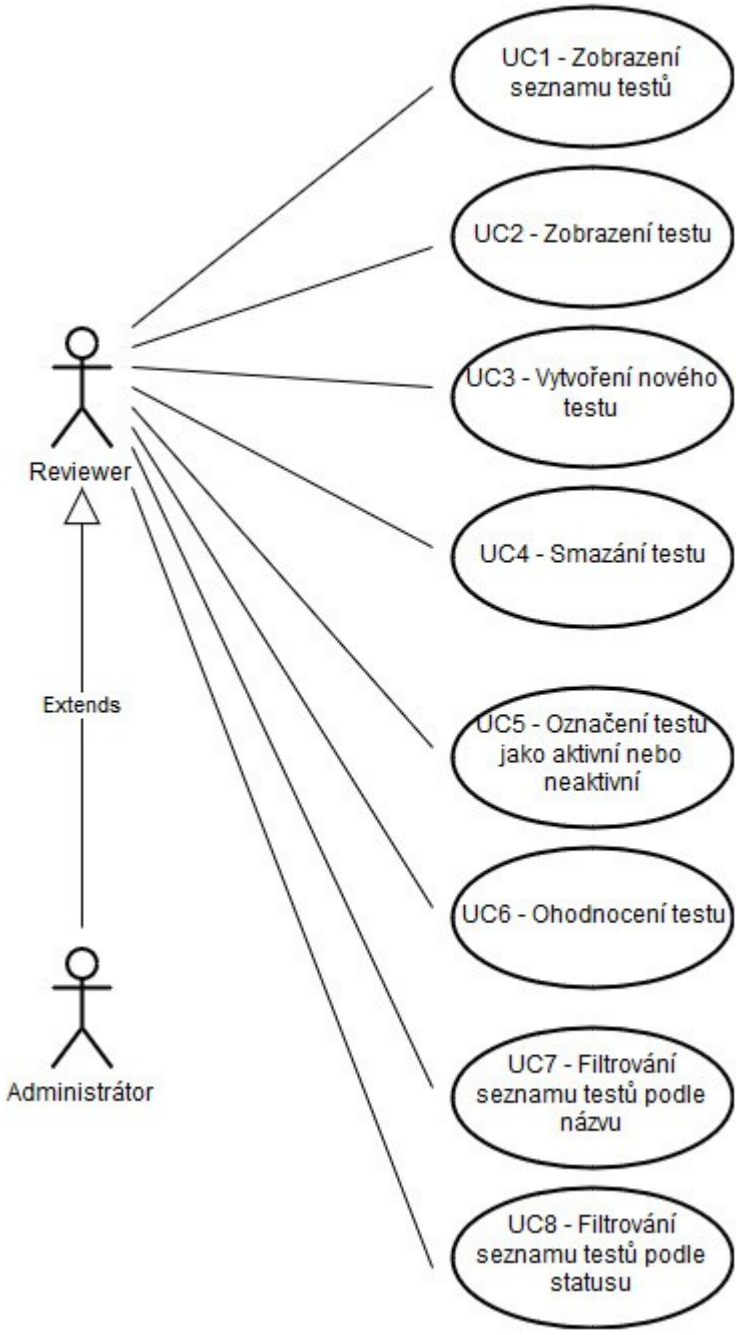
Díky diagramům případů užití bychom měli být schopni určit všechny jednotlivé operace, které bude náš systém vykonávat a také kdo je bude vykonávat.

Nejprve diagram případů užití ohledně správy kandidátů.



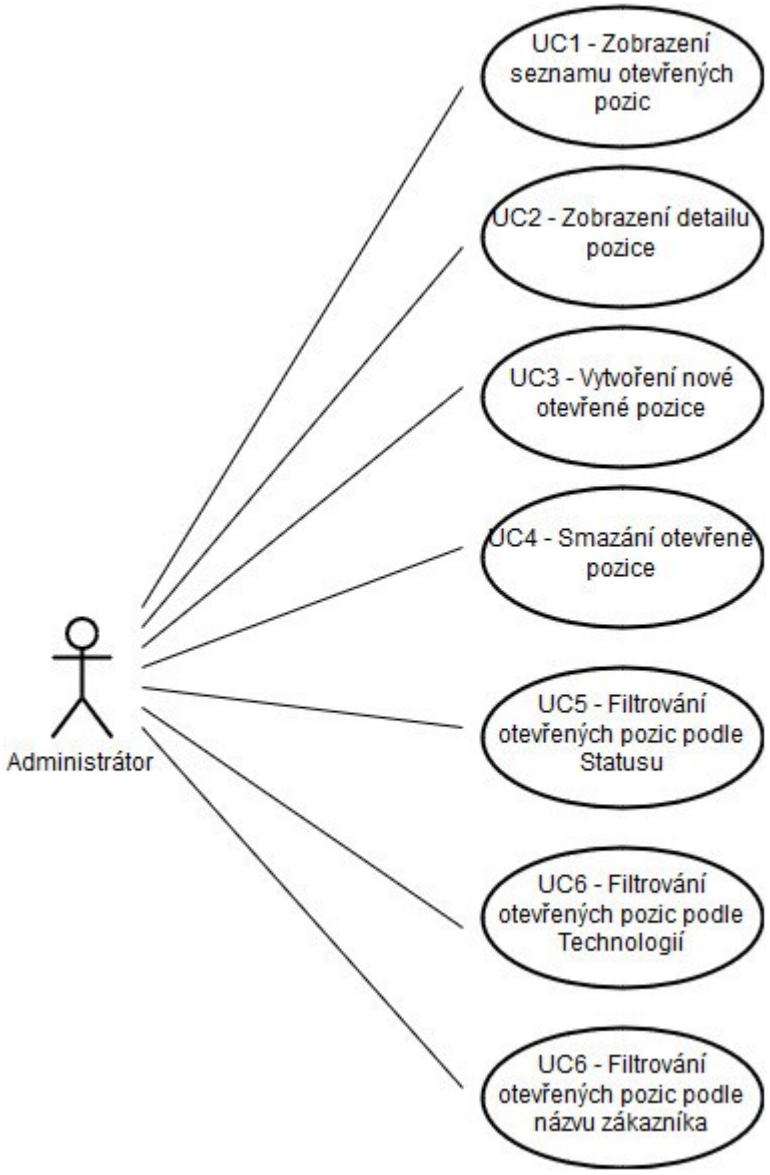
Obrázek 15 – Use Case diagram pro správu kandidátů

Následuje druhý obsáhlejší diagram případů užití ohledně správy testů.

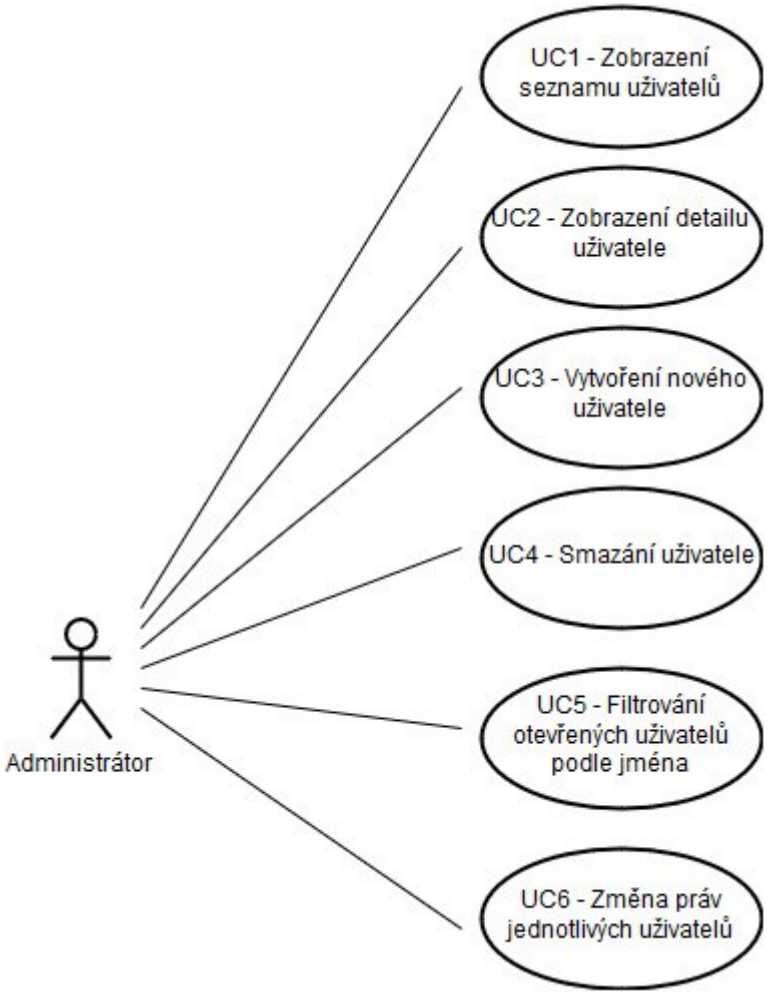


Obrázek 16 - Use Case diagram pro správu testů

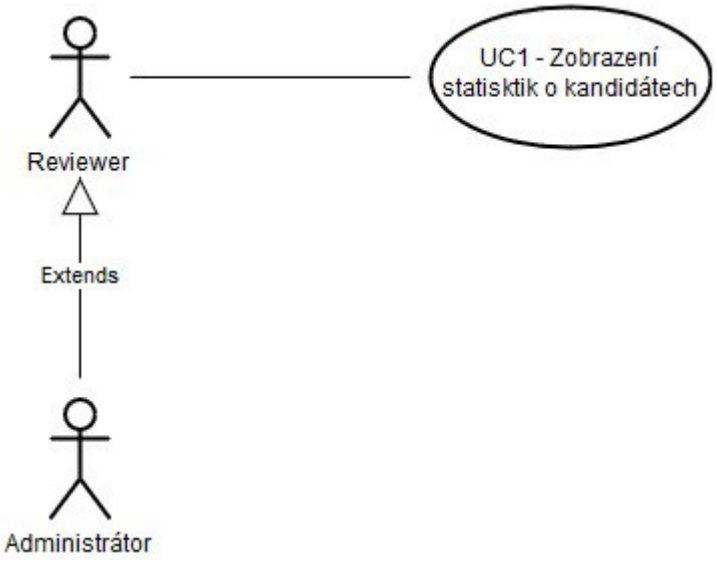
Správa otevřených pozic je dostupná pouze uživateli s administrátorskými právy.



Obrázek 17 - Use Case diagram pro správu otevřených pozic



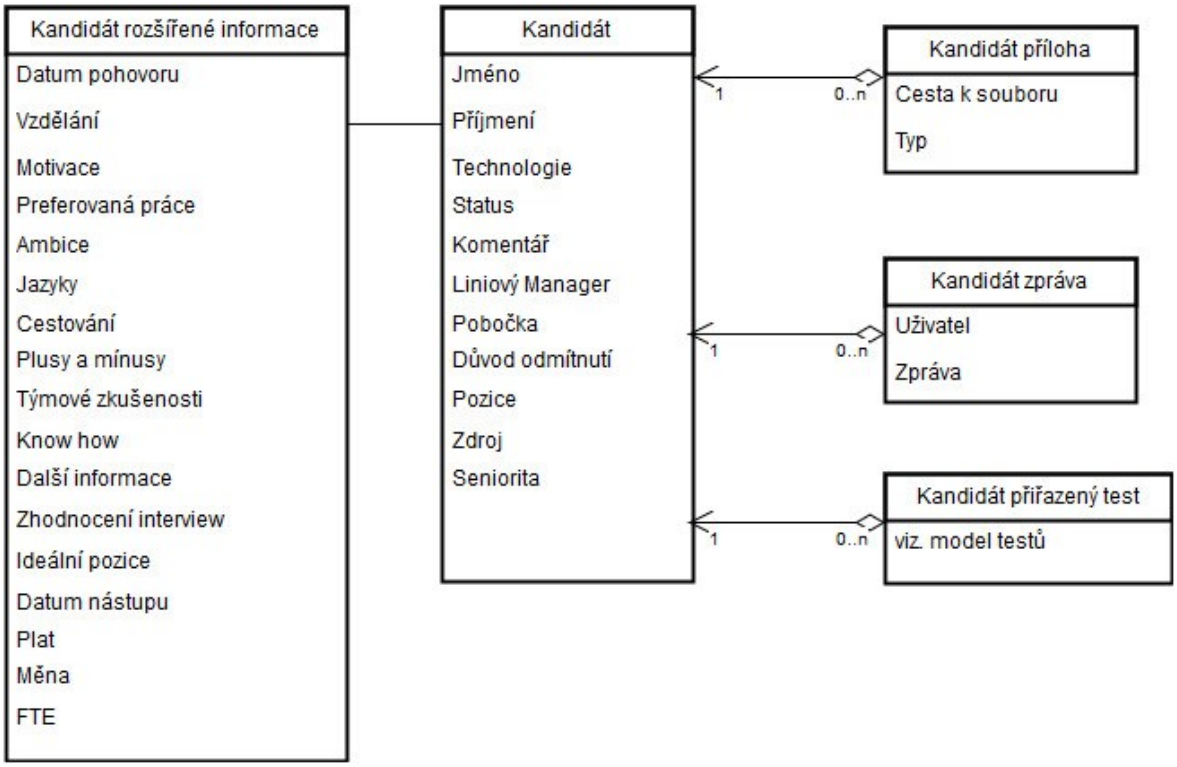
Obrázek 18 - Use Case diagram pro správu uživatelů



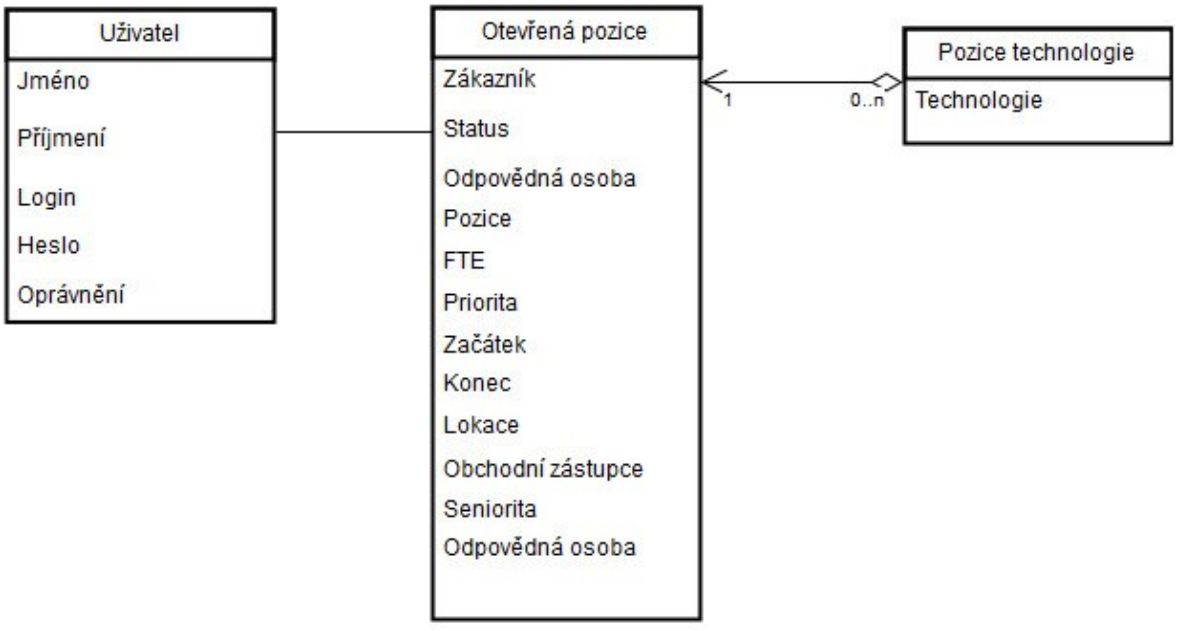
Obrázek 19 - Use Case diagram pro zobrazení statistik

5.2.3 Doménový model

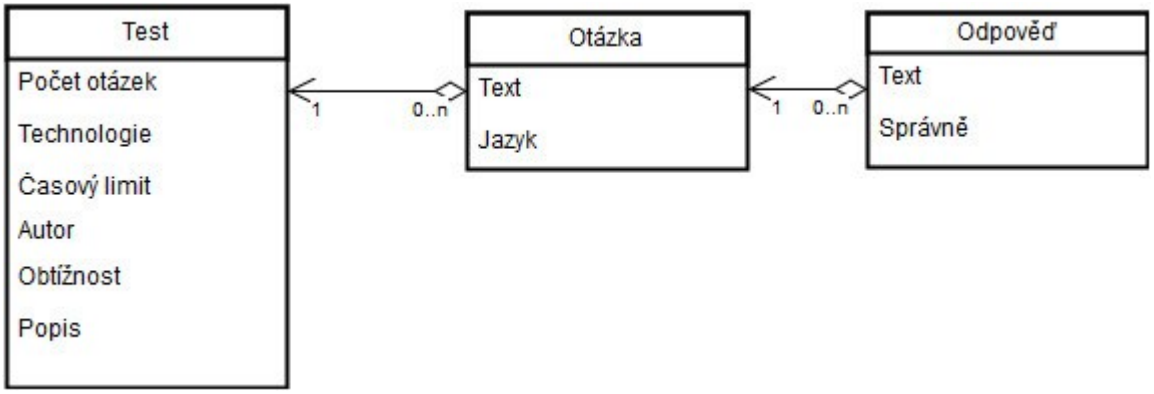
V této sekci specifikuji objekty, které budou v aplikaci využívány a všechny jejich doposud známé atributy. K sestavení doménových modelů jsem opět využil draw.io.



Obrázek 20 – doménový model kandidáta



Obrázek 21 – doménový model pozice a uživatele

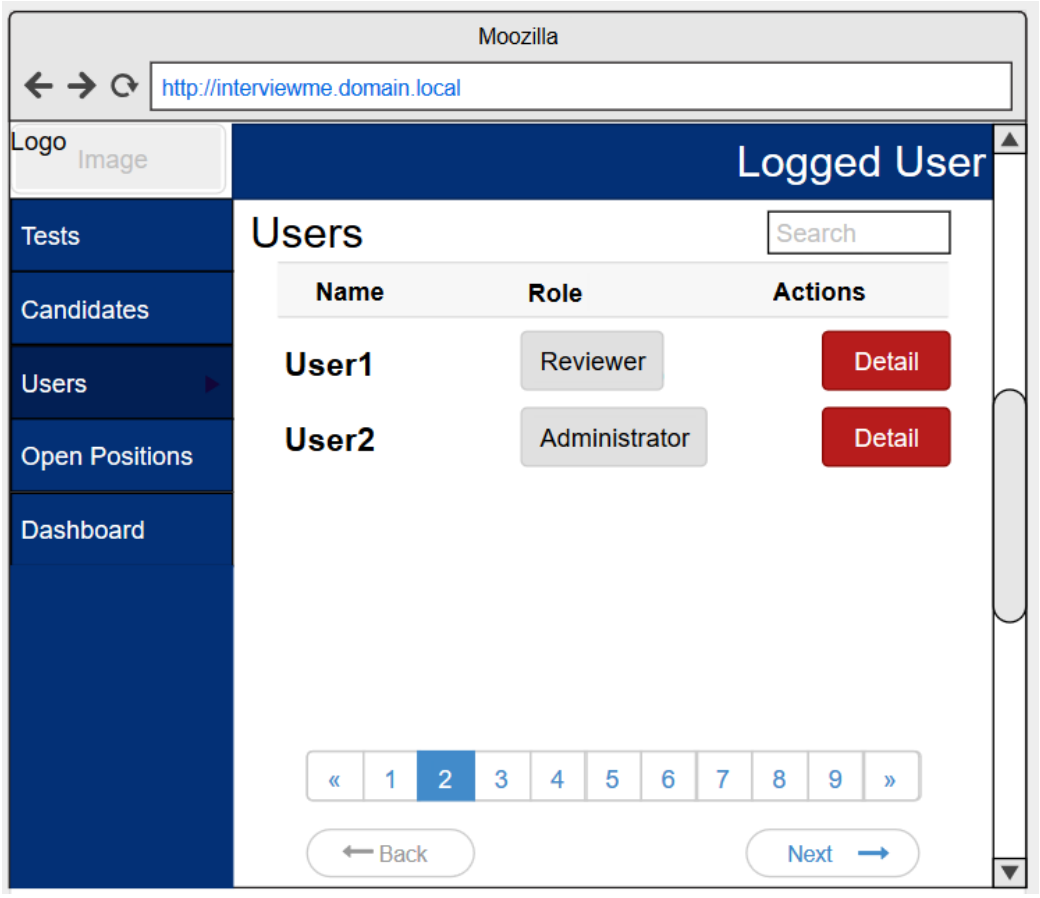


Obrázek 22 – doménový model testu

5.2.4 Návrh GUI

Zde je základní náčrt, ze kterého budu vycházet při tvorbě uživatelského rozhraní. Aplikace je poměrně jednoduchá z hlediska layoutu. Pro uživatelské rozhraní mi postačí hlavička, logo aplikace, navigační lišta a oblast, ve které se budou zobrazovat seznamy a formuláře.

Pro tvorbu návrhu uživatelského rozhraní jsem použil základní verzi moqups.com, což je online designer pro uživatelské rozhraní.

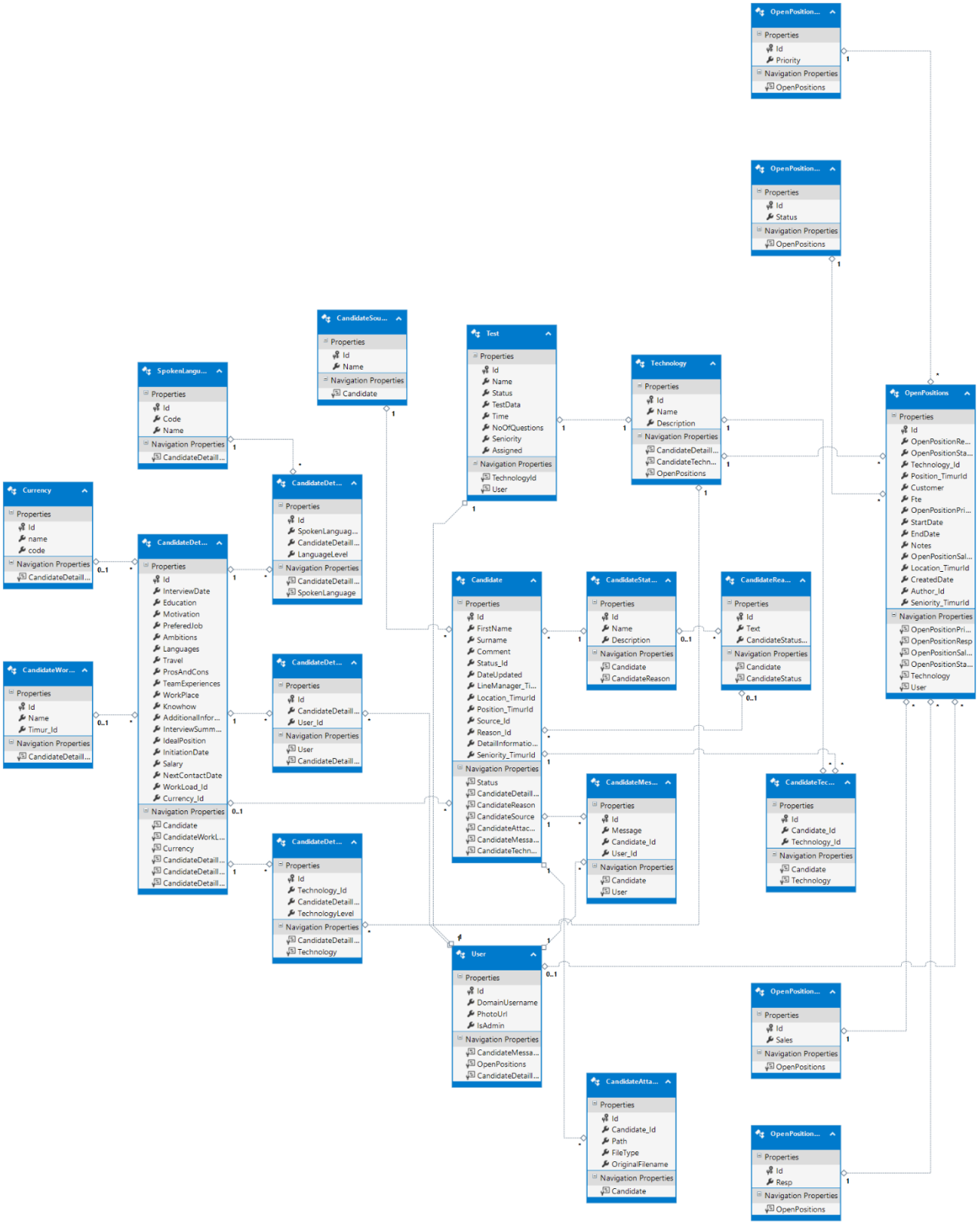


Obrázek 23 – návrh GUI aplikace

6 IMPLEMENTACE

6.1 Databázový model

Zde je ukázka diagramu databázového modelu vygenerovaného pomocí Visual Studia.



6.2 Ukázka implementace klienta

V této části se pokusím nastínit kód klienta. Využiji k tomu seznam kandidátu nejdříve ukáži, jak data zpracovává a zobrazuje klient s ukázkou zdrojového kódu a poté jak serverová část.

Nejprve ale začnu s havní komponentou App, její kód je následující:

```
class App extends Component {
  render() {
    const {
      sidebarVisibility,
      children,
     } = this.props;

    return (
      <div>
        <AppHeader />
        <AppNavigation sidebarVisibility={sidebarVisibility} />
        <MainContentWrapper sidebarVisibility={sidebarVisibility}>
          { children }
          <AppFooter />
        </MainContentWrapper>
      </div>
    );
  }
}
```

Na tomto kusu kódu vidíme zakladní kompontu reactu která obsahuje vše ostatní. Každá komponenta musí implementovat funkci render. Jak můžeme vidět v této funkci z props přebírají dva objekty a to sidebarVisibility a children. Ty se dostastaly do props pomocí namapování state na props díky využívání redux uložiště pomocí metody connect.

```
const mapStateToProps = (state) => {
  const { SliderNavigationReducer } = state;
  return {
    sidebarVisibility: getSidebarVisibility(SliderNavigationReducer),
  };
};
export default connect(mapStateToProps)(App);
```

Dále vidíme, že funkce render vrací jeden uzavřený tag (vždy to musí být jeden uzavřený tag), obsahující další komponenty, které jsou naimportované z ostatních souborů a mají stejnou strukturu jako komponenta App. Z tohoto příkladu můžeme vidět jak je zápis

přehledný a čitelný jasně vidíme že komponenta App obsahuje hlavičku, navigační boční lištu a hlavní část a patičku.

Teď se podíváme na navigaci:

```
export default class AppNavigation extends Component {
  render() {
    const { sidebarVisibility } = this.props;
    const iconStyle = { width: '20px' };

    return (
      <StyledSidebar sidebarVisibility={sidebarVisibility}>
        <NavigationItem>
          <span>MAIN NAVIGATION</span>
        </NavigationItem>
        <NavigationItem>
          <IndexLink to="/" activeClassName="active">
            <FontAwesome title="Tests" name="files-o" style={iconStyle} />
            <span>Tests</span>
          </IndexLink>
        </NavigationItem>
        <NavigationItem>
          <Link to="/candidates" activeClassName="active">
            <FontAwesome title="Candidates" name="fa fa-child" style={iconStyle} />
            <span>Candidates</span>
          </Link>
        </NavigationItem>
        <NavigationItem>
          <Link to="/users" activeClassName="active">
            <FontAwesome title="Users" name="users" style={iconStyle} />
            <span>Users</span>
          </Link>
        </NavigationItem>
        <NavigationItem>
          <Link to="/dashboard" activeClassName="active">
            <FontAwesome title="Dashboard" name="tachometer" style={iconStyle} />
            <span>Dashboard</span>
          </Link>
        </NavigationItem>
        <NavigationItem>
          <Link to="/positions" activeClassName="active">
            <FontAwesome title="Open positions" name="flag" style={iconStyle} />
            <span>Open positions</span>
          </Link>
        </NavigationItem>
      </StyledSidebar>
    );
  }
}
```

Jak vidíme i zde je vše přehledné, většina komponent je použita z již hotový a ostylovaných komponent, které jsou k reactu dnes velmi snadno k nalezení nejdůležitější částí kódu výše- jou právě linky. Link změní naše url a díky tomu react-router vybere pro wrapper hlavního obsahu správnou page.

Toto je část nastavení react routeru:

```
export default function createRoutes() {
  return (
    <Route path="/" component={App}>
      <IndexRoute component={TestsPage} />
      <Route path="candidates" component={CandidatesPage} />
      <Route path="candidates/createCandidate" component={CreateCandidatePage} />
      <Route path="candidates/detailCandidate/:id" component={CandidateDetailPage} />
      <Route path="candidates/editCandidate/:id" component={CandidateEditPage} />
```

Teď se podíváme na samotnou stránku seznamu kandidátů.

```
class CandidatesPage extends Component {
  static propTypes = {
    candidates: PropTypes.array,
    loaded: PropTypes.bool,
    fetchCandidates: PropTypes.func.isRequired,
    asyncViewerFetch: PropTypes.func.isRequired,
  };

  componentWillMount() {
    const {
      fetchCandidates,
      asyncViewerFetch,
    } = this.props;
    fetchCandidates();
    asyncViewerFetch();
  }

  render() {
    const { candidates, loaded, viewer } = this.props;

    return (
      <div>
        <h1>Candidates</h1>
        <StyledContainer primary>
          <div className="clear">
            {viewer.isAdmin && <StyledButton
              className="button danger"
              to="/candidates/createCandidate"
            >Add new Candidate
            </StyledButton>}
```

```
        </div>
        <Loader loaded={loaded}>
          <CandidatesList candidates={candidates} />
        </Loader>
      </StyledContainer>
    </div>
  );
 }
}
```

Zde vidíme novou funkci a to componentWillMount ta se zavolá až po vytvoření komponentu stránky. A zavolá funkci fetchCandidates.

Redux funguje tak, že se pomocí funkce dispatch vyvolá nějaká akce/příkaz (zde načtení všech kandidátů), ta prolítne reducerama což jsou úplně obyčejné funkce, které umí měnit podobu stavu redux úložiště a následně se tento nový stav uloží a komponenta se automaticky překreslí. Důležité je, že Redux si vlastně drží stav všech komponent u sebe v jednom velkém globálním úložišti (store) a akce resp. reducery slouží pro alespoň trošku rozumné ovládání jeho obsahu. Zároveň redux úložiště nemá z hlediska kódu nic společného s Reactem. Redux je ke komponentě nějak napojen, ale může fungovat úplně bez Reactu.

Funkce fetchCandidates slouží pro vytvoření objektu reprezentující nějakou akci. Ta se potom spouští pomocí dispatch. Akce je obyčejný JS objekt, který si nese informaci o typu akce + nějaká další dodatečná data. Pokud však chci data teprve načíst, je nutné vytvářet místo objektu funkci, která tak učiní:

```
export const fetchCandidates = () => (dispatch) => {
  dispatch(candidatesFetch());
  return api.get('/candidates/all')
      .then(
        response => dispatch(candidatesSuccess(response.data)),
        error => dispatch(candidatesFailure(error)),
      );
};
```

Tím se nám data dostala do redux uložiště, poté do props příslušné komponenty a ta byla překreslena.

6.3 Ukázka implementace serverové části Web API

Základní třídou ASP.Web API je:

```
public class WebApiApplication : HttpApplication
    {
        private readonly IWindsorContainer _container;

        public WebApiApplication()
        {
            _container = new WindsorContainer().Install(
                new Api.CompositionRoot.WindsorInstaller(),
                new Business.CompositionRoot.WindsorInstaller());
        }

        protected void Application_Start()
        {
            GlobalConfiguration.Configure(c => WebApiConfig.Register(c, _conta-
iner));

            GlobalConfiguration.Configuration.Services.Replace(
                typeof(IHttpControllerActivator),
                new WindsorCompositionRoot(_container));

            ProfileRegistration.RegisterProfiles(_container.ResolveAll<Profile>());
        }

        public override void Dispose()
        {
            _container.Dispose();
            base.Dispose();
        }
    }
```

Jak vidíme aplikace využívá dependency injection container WindsorContainer. Konfigurace dependency injection by se měla vždy provádět kořenu naší aplikace v tomto případě Global.asax čili v této funkci. Termín pro takové místo je „composition root“.

Nyní dost o DI a podívejmě se jak je aplikace nakonfigurována.

```
public static void Register(HttpConfiguration config, IWindsorContainer windsorCon-
tainer)
        {
            config.EnableCors(new EnableCorsForOriginsAttribute(GetAllowedOri-
gins()));
            config.Filters.Add(windsorContainer.Resolve<IAuthorizationFilter>(na-
meof(AuthorizationFilter)));
            config.Services.Replace(typeof(IExceptionLogger), windsorContainer.Re-
solve<IExceptionLogger>());
#if !DEBUG
            config.Services.Replace(typeof(IExceptionHandler), windsorContainer.Re-
solve<IExceptionHandler>());
#endif

            config.Formatters.JsonFormatter.SerializerSettings.ContractResolver =
                new CamelCasePropertyNamesContractResolver();
            config.MapHttpAttributeRoutes();
            config.Routes.MapHttpRoute(
```

```
                "ActionApi",
                "api/{controller}/{action}",
                new { action = RouteParameter.Optional }
            );
        }
```

V první řadě je povolen CORS aby bylo API dostupné i z jinné domény, např pro jinné klienty, i já volám jinné API pro ověření doménových účtu, a bez tohoto povolení by to později nešlo.

Dále je aplikován autorizační filtr ten kontroluje jestli v kontextu požadavku je atribut admin pokud ne pokračuje se ve zpracování požadavku pokud ano je zkontrolováno jestli má uživatel administrátorská práva (ukážu pozdeji v kódu). Dále je JSON formátovač nastaven na Camel Case. A jako poslední URL namapovení podle názvu kontroleru a akce, kde akce je nastavena jako optional.

Nyní se ty podíváme na ukázku kontroleru kandidátů.

```
public class CandidatesController : ApiController
    {
        private readonly ICandidateCrudFacade _candidateFacade;
        private readonly ICandidateStatusFacade _candidateStatusFacade;
        private readonly ICandidateSourceFacade _candidateSourceFacade;
        private readonly ICandidateReasonFacade _candidateReasonFacade;

        public CandidatesController(
            ICandidateCrudFacade candidateFacade,
            ICandidateStatusFacade candidateStatusFacade,
            ICandidateSourceFacade candidateSourceFacade,
            ICandidateReasonFacade candidateReasonFacade)
        {
            _candidateFacade = candidateFacade;
            _candidateStatusFacade = candidateStatusFacade;
            _candidateSourceFacade = candidateSourceFacade;
            _candidateReasonFacade = candidateReasonFacade;
        }

        /// <summary>
        /// Retrieves all currently tracked candidates.
        /// </summary>
        /// <returns>A list of <see cref="CandidateListDTO"/>.</returns>
        [HttpGet]
        public IEnumerable<CandidateListDTO> All()
        {
            return _candidateFacade.GetList();
        }

        /// <summary>
        /// Returns detailed information about a candidate specified by <paramref
name="candidateId"/>.
        /// </summary>
        /// <param name="candidateId">ID of candidate.</param>
        /// <returns>Detailed information about candidate represented by <see
cref="CandidateDTO"/> object.</returns>
        [HttpGet]
        public CandidateDTO Detail(int candidateId)
```

```
        {
            return _candidateFacade.GetDetail(candidateId);
        }

        /// <summary>
        /// Returns list of possible values for candidate status.
        /// </summary>
        /// <returns>A list of <see cref="CandidateStatusDTO"/> objects.</returns>
        [HttpGet]
        public IEnumerable<CandidateStatusDTO> Statuses()
        {
            return _candidateStatusFacade.GetCandidateStatuses();
        }

        /// <summary>
        /// Returns list of possible values for candidate source.
        /// </summary>
        /// <returns>A list of <see cref="CandidateSourceDTO"/> objects.</returns>
        [HttpGet]
        public IEnumerable<CandidateSourceDTO> Sources()
        {
            return _candidateSourceFacade.GetCandidateSources();
        }

        /// <summary>
        /// Returns list of possible values for candidate rejection reason.
        /// </summary>
        /// <returns>A list of <see cref="CandidateReasonDTO"/> objects.</returns>
        [HttpGet]
        public IEnumerable<CandidateReasonDTO> Reasons()
        {
            return _candidateReasonFacade.GetRejectReasons();
        }

        /// <summary>
        /// Saves information about candidate.
        /// </summary>
        /// <param name="candidate">Candidate to save.</param>
        [Admin]
        [HttpPost]
        public void Save(CandidateDTO candidate)
        {
            _candidateFacade.Save(candidate);
        }

        /// <summary>
        /// Deletes candidate.
        /// </summary>
        /// <param name="id">Candidate id.</param>
        [Admin]
        [HttpPost]
        public void Delete(int id)
        {
            _candidateFacade.Delete(id);
        }
    }
```

Výše vidíme kompletní kontroler pro správu kandidátů, ten kromě základních CRUD operací má ještě fasády pro vrácení všech možných statusům, zdrojů atd., které se využívají při

vytváření kandidáta. Metody CRUD fasády poté pomocí entity frameworku a automapperu načtou data z databáze a namapují je na výstupní DTO objekt.

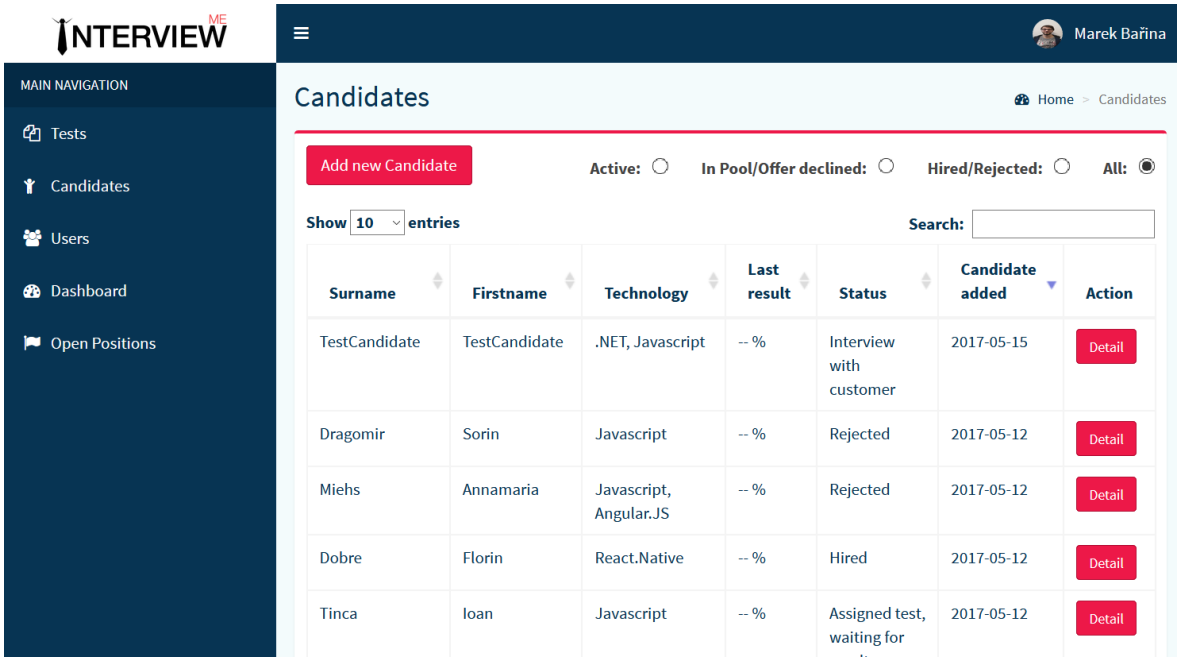
Ještě si můžeme všimnout atributu admin u metod Save a Delete.

Takto například může vypadá odpověď API na url „{domain}/api/candidates/all“ :

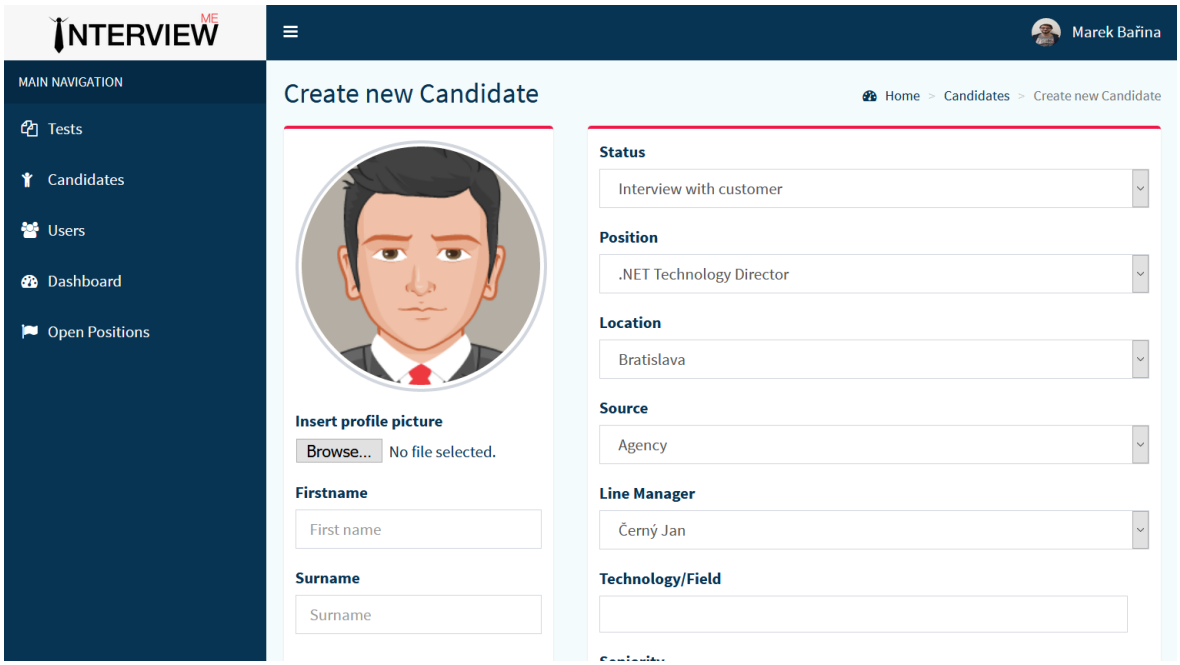

```
− <ArrayOfCandidateListDTO>
  − <CandidateListDTO>
      <DateUpdated>2017-05-11T11:53:45.62</DateUpdated>
      <FirstName>Destini</FirstName>
      <Id>2308</Id>
      <LastResult i:nil="true"/>
      <Status>Interview with customer</Status>
      <Surname>Auer</Surname>
    − <Technologies>
        <d3p1:string>React</d3p1:string>
        <d3p1:string>JavaScript</d3p1:string>
      </Technologies>
    </CandidateListDTO>
  − <CandidateListDTO>
      <DateUpdated>2016-11-01T10:41:57.973</DateUpdated>
      <FirstName>Joany</FirstName>
      <Id>2311</Id>
      <LastResult i:nil="true"/>
      <Status>Interview with customer</Status>
      <Surname>Gerlach</Surname>
    − <Technologies>
        <d3p1:string>JavaScript</d3p1:string>
        <d3p1:string>Java</d3p1:string>
      </Technologies>
    </CandidateListDTO>
  − <CandidateListDTO>
      <DateUpdated>2016-11-16T21:06:15.38</DateUpdated>
      <FirstName>Verla</FirstName>
      <Id>2314</Id>
      <LastResult i:nil="true"/>
      <Status>Interview with customer</Status>
      <Surname>Yost</Surname>
```

7 UKÁZKA APLIKACE

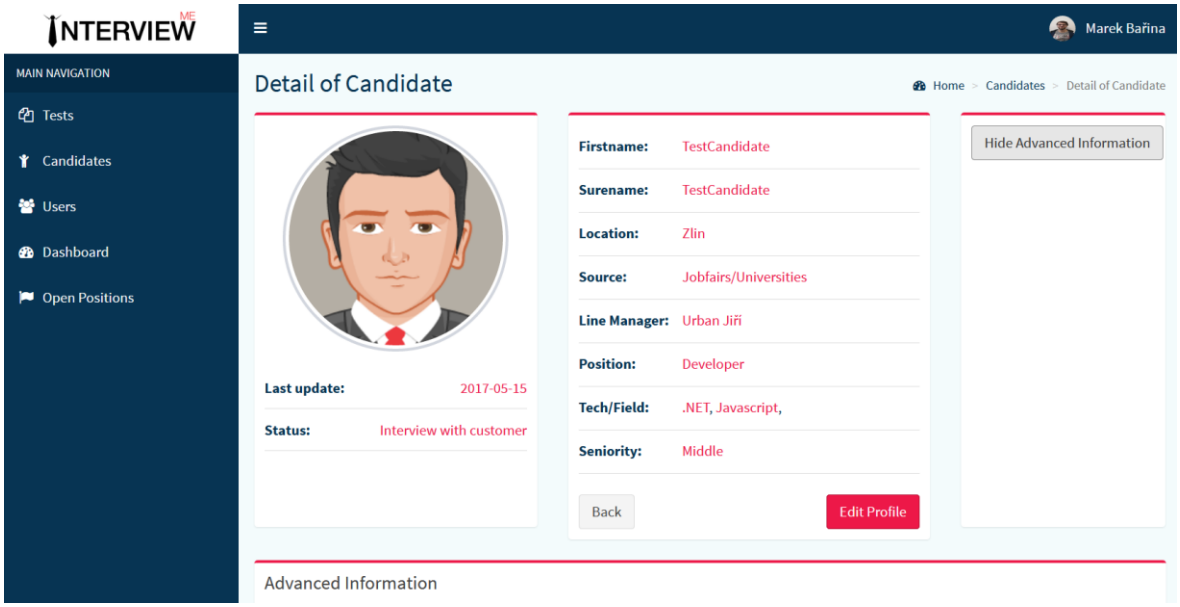
Zde uvedu některé ukázkové screenshoty aplikace.



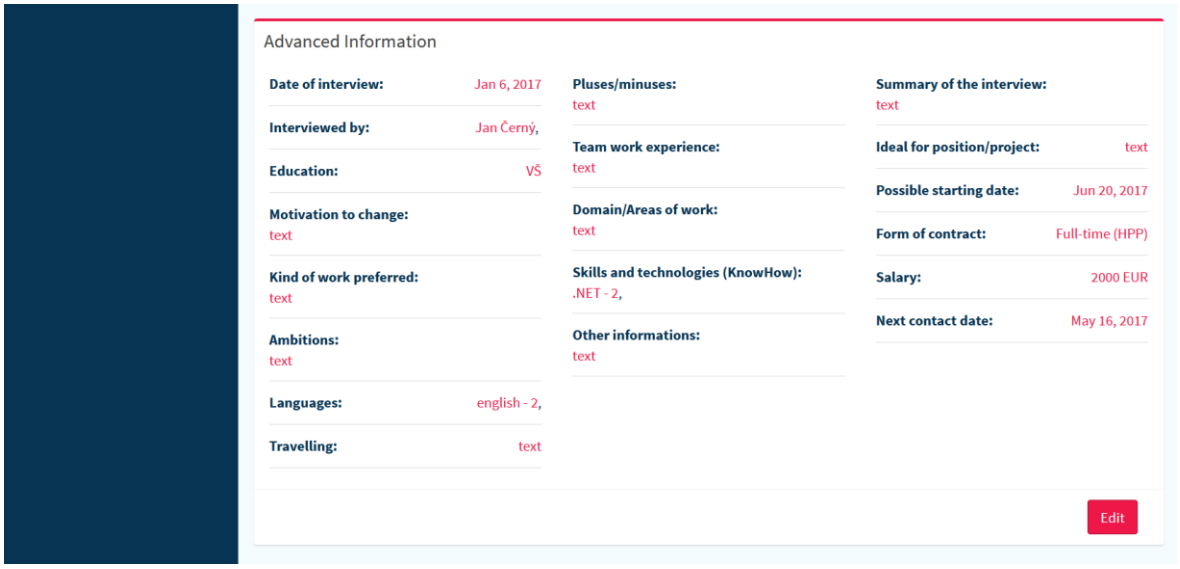
Obrázek 24 – ukázka seznam kandidátů



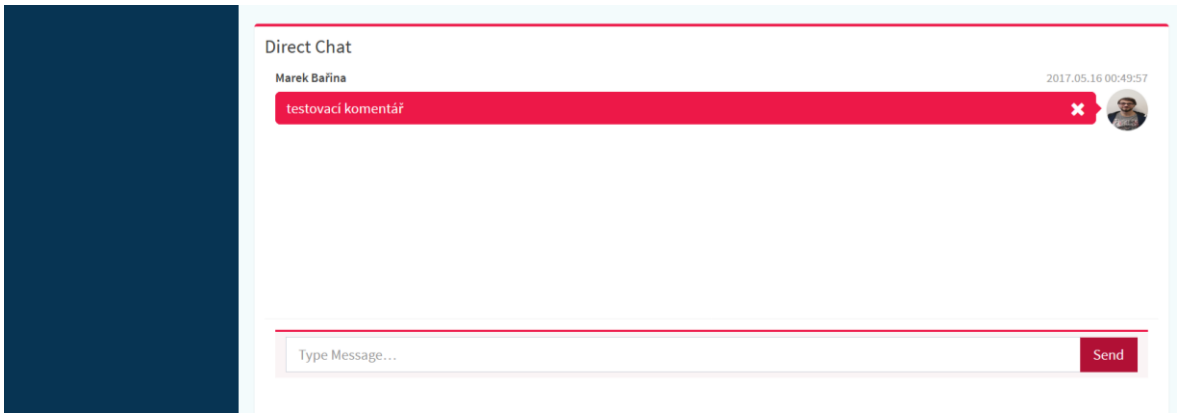
Obrázek 25 – ukázka vytvoření kandidáta



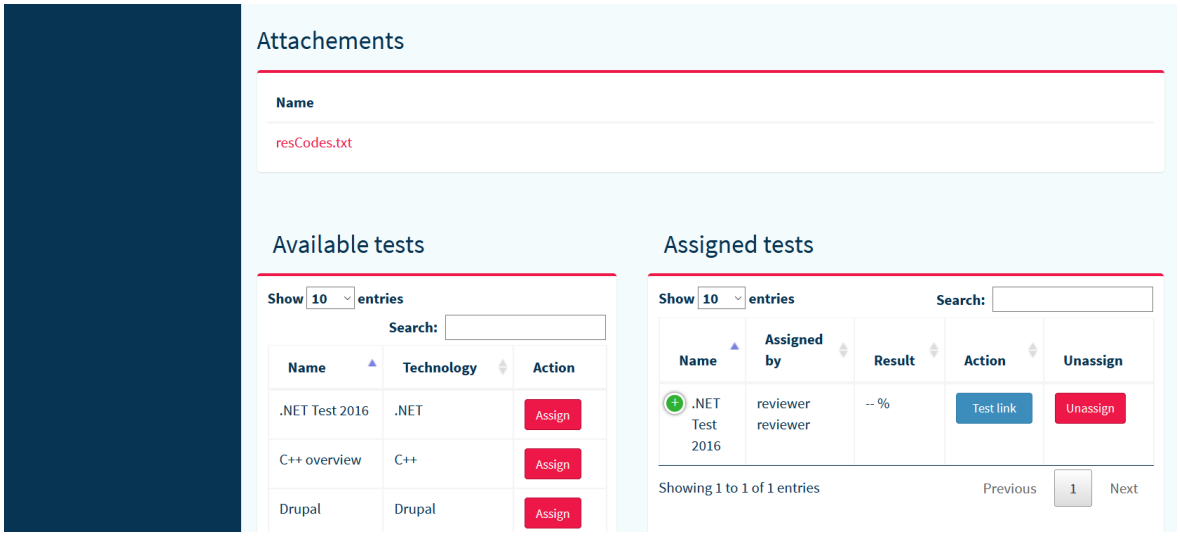
Obrázek 26 – ukázka detail kandidáta



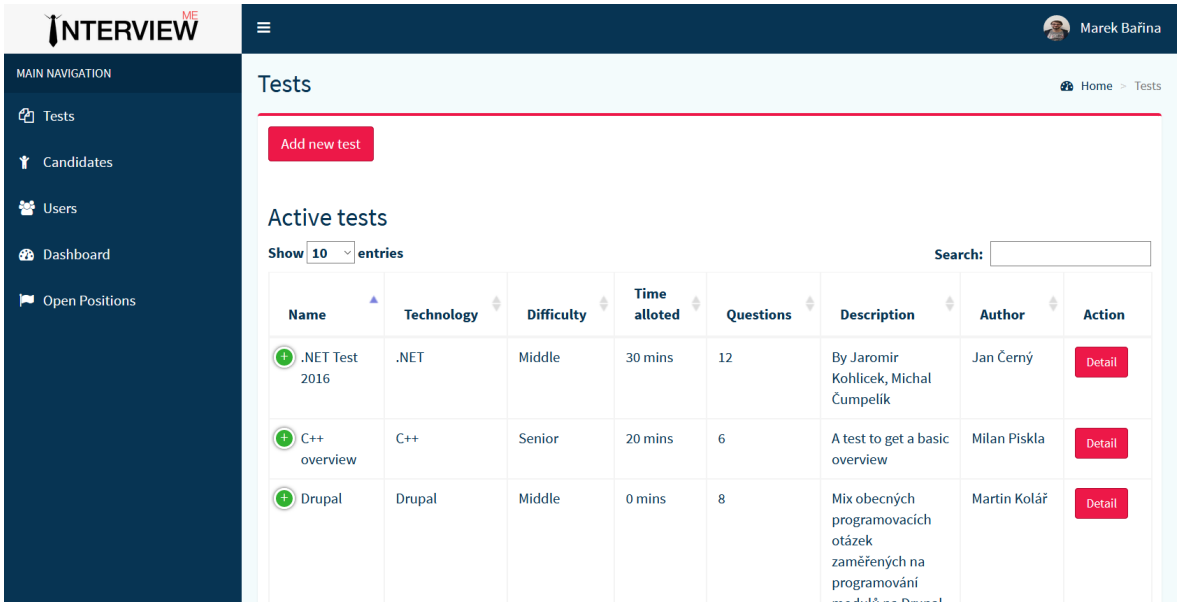
Obrázek 27 – ukázka rozšířených informací kandidáta



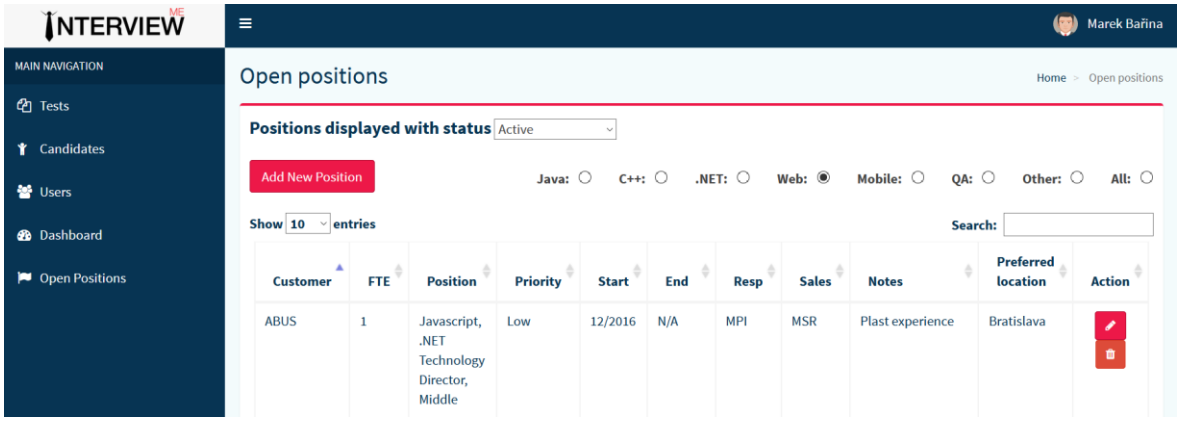
Obrázek 28 – ukázka chatu v detailu kandidáta



Obrázek 29 – ukázka příloh a přiřazených testů u kandidáta



Obrázek 30 – ukázka seznamu testů



Obrázek 31 – ukázka seznamu otevřených pozic

ZÁVĚR

V této práci jsem popsal nejdůležitější kroky začátku vývoje systému, žačínajících u analýzy požadavků, tvorby UML diagramů, návrhování uživatelského rozhraní a datových modelů aplikace. Dále jsem popsal a porovnal výhody a nevýhody nejpoužívanějších technologií pro tvorbu aplikací, se speciálním rozdělením na technologie pro tvorbu klientské části aplikace a serverové části aplikace. Z hlediska technologií je nejvíce na vzestupu pravděpodobně JavaScript a to už nějakou dobu i když přesná data moc nálezt nelze například počet aktivních repozitářu na GitHubu, je v dominován JavaScriptem ale to není až tak směrodatné. Nicméně poptávka po JavaScriptu na trhu práce už je. Bylo pro mě opravdu obtížne zvolit si z dostupných technologií, protože z těch popsaných je jen málo z nich horších než ty druhé. Nakonec jsem si vybral technologii klienta podle osobního zájmu a jak už jsem psal i kvůli rostoucí popularitě JavaScriptu a jeho knihoven vyvýjených velkými firmami jako Facebook (React) nebo Google (Angular). Technologi pro backend pak ovlivnil i fakt že .NET je mi bližší než Java nebo NodeJS ale i to, že podle mě je práce s databází při použití .NET, MS SQL a entity frameworku velmi dobrá. Nehledě na tyto preference je ASP.NET Web API je výborná a velmi jednoduchá knihovna pro opravdu rychlý a efektivní vývoj webové služby.

Takto nabité znalosti jsem pak využil při implementaci systému pro vyřizování pohovorů. Nejprve jsem sepsal a analyzoval funkční a nefunkční požadavky na aplikaci. Podle nich jsem vytvořil návrh řešení a ten použil jako podklad při samotné implementaci systému.

Aplikace interviewMe umožnuje zpravovat databázi kandidátů, otevřených pozic a testů pro budoucí účastníky pohovoru. Díky tomu může velmi napomoct HR oddělení a pomoci z-efektivnit celý náborový proces. Jelikož jsem tuto aplikaci psal pro zadávající firmu, ve které pracuji nejsou zdrojové kódy součástí této práce.

SEZNAM POUŽITÉ LITERATURY

[1] SOEGAARD, Mads a Rikke FRIIS DAM. *The Encyclopedia of Human-Computer Interaction, 2nd Ed* [online]. 2nd. The Interaction Design Foundation [cit. 2017-03-30]. ISBN 9788792964. Dostupné z: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed

[2] Analýza požadavků. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-30]. Dostupné z: https://cs.wikipedia.org/wiki/Anal%C3%BDza_po%C5%BEadavk%C5%AF

[3] Enterprise modelling. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-31]. Dostupné z: https://en.wikipedia.org/wiki/Enterprise_modelling

[4] ITNetwork. *UML* [online]. [cit. 2017-04-01]. Dostupné z: https://www.itnetwork.cz/navrhove-vzory/uml

[5] *Požadavky a jejich specifikace* [online]. [cit. 2017-04-6]. Dostupné z: http://www.fit.vutbr.cz/study/courses/PPS/public/pdf/5_2.pdf

[6] MCCONNELL, Steve. *Rapid development: taming wild software schedules*. Redmond, Wash.: Microsoft Press, c1996. [cit. 2017-04-06]. ISBN 15-561-5900-5.

[7] *Algomica: Vývoj a testování softwaru* [online]. [cit. 2017-04-06]. Dostupné z: http://www.algomica.cz/specifikace_softwaru.htm

[8] GORTON, Ian. *Essential software architecture*. 2nd ed. Heidelberg: Springer, 2011. ISBN 36-421-9175-4. [cit. 2017-04-06].

[9] Alistair Cockburn: Use cases, ten years later. *Alistair Cockburn* [online]. [cit. 2017-04-08]. Dostupné z: http://alistair.cockburn.us/Use+cases%2c+ten+years+later

[10] PHP vs. JavaScript. *UpWork* [online]. [cit. 2017-04-13]. Dostupné z: https://www.upwork.com/hiring/development/php-vs-javascript/

[11] PHP (88) - provoz ve Windows. *Linuxsoft* [online]. [cit. 2017-04-13]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=629

[12] *PHP: Hypertext Preprocessor* [online]. [cit. 2017-04-13]. Dostupné z: http://php.net/

[13] *JScripters* [online]. [cit. 2017-04-13]. Dostupné z: http://www.jscripters.com/javascript-advantages-and-disadvantages/

[14] What are the Advantages of Javascript? *Quora.com* [online]. 2016 [cit. 2017-04-13]. Dostupné z: https://www.quora.com/What-are-the-Advantages-of-Javascript

[15] Technologie pro tvorbu webových aplikací – 3. díl (PHP, PERL, ASP.NET). *Posterus.sk* [online]. 2012 [cit. 2017-04-13]. Dostupné z: http://www.posterus.sk/?p=13523

[16] ASP.NET overview. In: *Microsoft* [online]. [cit. 2017-04-13]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/overview

[17] React – Úvod. *Dzejes* [online]. [cit. 2017-04-13]. Dostupné z: https://www.dzejes.cz/react-uvod.html

[18] The HTML DOM (Document Object Model). *W3schools* [online]. [cit. 2017-04-20]. Dostupné z: https://www.w3schools.com/js/js_htmldom.asp

[19] React (JavaScript library). *Wikipedie: otevřená encyklopedie* [online]. [cit. 2017-04-20]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library)#One-way_data_flow

[20] What are the disadvantages of using React? *Quora.com* [online]. [cit. 2017-04-20]. Dostupné z: https://www.quora.com/What-are-the-disadvantages-of-using-React

[21] UHERČÍK, Maroš. *Multiplatformní aplikace s využitím TypeScript a AngularJS*. Brno, 2014. Dostupné také z: https://is.muni.cz/th/418134/fi_m/dp.pdf. Masarykova Univerzita, Fakulta Informatiky.

[22] Java Application Development. *Tricon Infotech* [online]. [cit. 2017-04-25]. Dostupné z: http://www.triconinfotech.com/blog/2015/09/01/java-application-development-advantages-and-disadvantages/

[23] Java. In: *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation [cit. 2017-04-25]. Dostupné z: https://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)

[24] Swing. In: *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation [cit. 2017-04-25]. Dostupné z: https://cs.wikipedia.org/wiki/Swing_(Java)

[25] SWT: The Standard Widget Toolkit. In: *Eclipse* [online]. [cit. 2017-04-25]. Dostupné z: http://www.eclipse.org/swt/

[26] Úvod do C# a .NET frameworku. *ITNetworks* [online]. [cit. 2017-04-25]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

[27] C Sharp Standard. In: *ECMA* [online]. [cit. 2017-04-25]. Dostupné z: http://www.ecma-international.org/publications/standards/Ecma-334.htm

[28] .NET Framework Versions and Dependencies. In: *MSDN* [online]. [cit. 2017-04-25]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb822049(v=vs.110).aspx

[29] Historie .NETu. *ITNetwork* [online]. [cit. 2017-04-26]. Dostupné z: https://www.itnetwork.cz/csharp/historie-dotnetu

[30] Úvod do WPF (Windows Presentation Foundation). *ITNetwork* [online]. [cit. 2017-04-26]. Dostupné z: https://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace

[31] Alex Handy. *Node.js pushes JavaScript to the server-side* [online]. [cit. 2015-04-26]. SDTimes. Dostupné také z: http://sdtimes.com/node-js-pushes-javascript-to-the-server-side/

[32] ČERVENÝ, Pavel. *WEBOVÉ APLIKACE NA PLATFORMĚ NODE.JS*. Brno, 2016. [cit.2017-04-26]. Dostupné také z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=132264. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

[33] Adam Gent. *How to choose between Jersey, Apache Wink and JBossRESTEasy?* [cit. 2016-04-26]. 2010. Dostupné také z: http://stackoverflow.com/a/10922135

[34] Project Jersey. In: *Wikipedia: the free encyclopedia* [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://en.wikipedia.org/wiki/Project_Jersey

[35] ASP.NET Web API. In: *Microsoft* [online]. [cit. 2017-04-26]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

[36] What is ASP.NET Web API ? Why to Choose ASP.NET Web API ? In: *.net-stuff* [online]. [cit. 2017-04-26]. Dostupné z: http://www.dotnet-stuff.com/tutorials/web-api/what-is-asp-net-web-api-why-to-choose-asp-net-web-api

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

RE Requirements engineering – analýza požadavků

UML Unified Modeling Language – Unifikovaný Modelovací Jazyk

MOE Measure of Effectiveness – míra efektivity

MOS Measure of Suitability – míra vhodnosti

API Application Interface – aplikační rozhraní

JSX JavaScript extension

PHP PHP: Hypertext Preprocessor – Skriptovací jazyk určený pro tvorbu webových aplikací.

HTML HyperText Markup Language – název značkovacího jazyka používaného pro tvorbu webových stránek

HTTP Hypertext Transfer Protocol – internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML

ASP Active Server Pages – Framework sloužící k vytváření webových aplikací.

XHP Je rozšíření PHP a Hack vyvinutý Facebookem umožňující syntaxi XML za účelem vytvoření vlastních a opakovaně použitelných prvků HTML

SPA Single Page Aplication – Aplikace používající jednu stránku, na které překresluje jednotlivé komponenty.

MVC Model View Controller – Architektonický vzor rozdělující aplikaci na modely, pohledy, a kontroléry.

REST Representational State Transfer – Architektonický styl pro tvorbu aplikací typu klient-server.

SQL Structured Query Language – Programovací jazyk specificky určený pro komunkaci s relační databází.

JSON JavaScript Object Notation – Formát pro přenos dat využívající strukturu klíč-hodnota.

IIS Internet Information Services – Je rozšiřitelný webový server vytvořený společností Microsoft

CRUD Create, Read, Update, Delete. Sada operací vytvoř, čti, edituj, odstraň.

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: ŽIVOTNÍ CYKLUS ZPRÁVY V ASP.NET WEB API II