VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

VÝVOJ WEBOVÉ APLIKACE K TESTOVÁNÍ ZNALOSTÍ UCHAZEČŮ

DEVELOPMENT OF WEB APPLICATION FOR TESTING CANDIDATE'S KNOWLEDGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Martin Janík

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Dydowicz, Ph.D.

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Martin Janík**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Informační management
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2017/18

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Vývoj webové aplikace k testování znalostí uchazečů

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Cílem diplomové práce je vylepšit firemní proces náboru nových zaměstnanců pomocí vývoje vlastní webové aplikace sloužící k testování znalostí uchazečů před pohovorem. Dílčím cílem je snížení nákladů vztahující se k tomuto procesu.

Základní literární prameny:

BASL, J. a R. BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Z. Automatizované informační systémy. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

MOLNÁR, Z. Efektivnost informačních systémů. Praha: Grada Publishing, 2000. 142 s. ISBN 80-716--410-X.

ŘEPA, V. Analýza a návrh informačních systémů. Praha: Ekopress, 1999. 403 s. ISBN 80-86119--3-0.

SODOMKA, P. a H. KLČOVÁ. Informační systémy v podnikové praxi. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2017/18

V Brně dne 28.2.2018

L. S.

doc. RNDr. Bedřich Půža, CSc. ředitel	doc. Ing. et Ing. Stanislav Škapa, Ph.D. děkan

Abstrakt

Diplomová práce se zabývá analýzou, návrhem a tvorbou modulu do informačního systému společnosti XYZ s. r. o. Na základě požadavků získaných od společnosti je navrhnuta architektura modulu, identifikovány objekty aplikace, a nakonec samotný vývoj modulu včetně ekonomického zhodnocení. Modul je vyvíjen v nové verzi JavaScript a využívá moderní technologie React, Redux, REST, Node.js a MongoDB.

Klíčová slova

informační systém, modul, nábor zaměstnanců, klient, server, JavaScript, React, Redux, Node.js, MongoDB

Abstract

Master thesis is focused on analysis, design and development of information system module for the company XYZ s. r. o. The design of the module's architecture, application objects identification and the development of the module including the economic evaluation are all based on the requirements acquired from the company. Modul is developed in new JavaScript version and uses modern technologies such as React, Redux, REST, Node.js and MongoDB.

Key worlds

information system, module, recruitment, client, server, JavaScript, React, Redux, Node.js, MongoDB

Bibliografická citace

JANÍK, M. *Vývoj webové aplikace k testování znalostí uchazečů.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2018. 86 s. Vedoucí diplomové práce Ing. Petr Dydowicz, Ph.D.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 15. května 2018 ……

podpis autora

Poděkování

Touhle cestou chci poděkovat vedoucímu práce, Ing. Petr Dydowicz PhD., za odborné vedení a cenné rady a zkušenosti, které mi velmi pomohli při tvorbě diplomové práce a aplikace. Dále společnosti za umožnění tvorby, konzultace a pomoc při vývoji webové aplikace. V neposlední řadě děkuji své rodině, za umožnění studia a podporu při něm.

OBSAH

ÚVOD

Informační systém je nedílnou součástí každé firmy, nehledě na to, zda se jedná o malý podnik nebo korporaci. Jedná se o nástroje na zefektivnění a automatizaci každodenních procesů, přístup k jednotné pravdě napříč společností a podporu v rozhodování především pro management. Společnosti bez informačního systému, který neplní některou z vyjmenovaných funkcí ztrácí konkurenční výhodu a dává náskok ostatním podnikům. Moderní informační systém se vyznačuje vysokou modularitou a škálovatelností s důrazem na specifické potřeby daného podniku, případně podniků ve stejném oboru.

Při uvažování o informačním systému je potřeba brát ohled i na jeho nákladovost. Je potřeba uvažovat, zda daný informační systém, nebo jeho modul podporuje společnost v zefektivnění a automatizaci procesů alespoň ve výšce nákladů vynaložených na něj. V některých případech je možné modul, nahradit modulem jiným, levnějším, případně i vlastním za vynaložení nižších nákladů a udržení dosavadní úrovně podpory procesů.

VYMEZENÍ PROBLÉMU A CÍLE PRÁCE

Hlavním cílem diplomové práce je návrh a vývoj webové aplikace pro firmu XYZ s. r. o. sloužící k otestování znalostí uchazečů před pohovorem. Práce je zaměřená na analýzu současného stavu procesu náboru zaměstnanců a návrhu nejenom finančně efektivního řešení, ale i řešení, které je přizpůsobené potřebám firmy. Z toho vyplývají dílčí cíle práce, kterými jsou zefektivnění a snížení nákladu procesu náboru nových zaměstnanců a odstranění poslední části informačního systému patřící třetí straně.

Návrh a tvorba nejvýhodnějšího řešení vyžadovala důkladnou analýzu aktuálního stavu společnosti, definování a rozbor firemních procesů, formulace požadavků, funkcionalit a vlastností vyvíjené webové aplikace. Pomocí dotazníkové metody byly identifikovány hlavní nedostatky stavu firmy XYZ s. r. o., která byla zaměřená na vnější a vnitřní prostředí firmy. Analýza informačního systému odkryla nedostatky v procesu náboru zaměstnanců. Tento proces byl poté znázorněn pomocí EPC diagramu a RACI matice. Pro potřeby správného využití a interpretaci získaných informací byly využity teoretická východiska uvedené v první části práce.

Při návrhu a vývoji webové aplikace bylo využito především aktuálně dostupných technologií, jejich dokumentace a nejlepších praktik v oboru. Na základě identifikace způsobů využití modulu bylo možné pokračovat k reálnému návrhu, a to pomocí návrhu architektury aplikace a datového a funkčního modelování.

1 TEORETICKÁ VÝCHODISKA PRÁCE

1.1 Základní pojmy

1.1.1 Informace

Zakladatel kybernetiky definoval, že informace je nehmotné povahy. Exaktní definice z hlediska moderní podnikové informatiky je nedostačující a pro takové účely existuje řada neexaktních definic kladoucí důraz na úrovně pohledu na informaci:

- syntaktický pohled se orientuje na vnitřní strukturu a souvislosti utvářecích znaků,
- sémantický pohled se orientuje na obsahový význam informace,
- pragmatický pohled se orientuje na praktické využití informace, klade důraz na význam informace pro příjemce [4].

1.1.2 Proces

Funkce procesu spočívá v přeměně vstupů na výstupy. Tuto funkci lze popsat jako soubor souvisejících nebo na sebe působících činností. Výstupem procesu je služba nebo produkt s přidanou hodnotou pro zákazníka procesu (interní, nebo externí). Definované podnikové procesy jsou měřeny klíčovými výkonnostními indikátory (KPI), které jsou stanovené na strategické úrovni. Pomocí KPI jsou procesy následně zlepšovány. Procesy lze rozdělit do tří kategorií:

- procesy řídící, (strategické) zabezpečující rozvoj a řízení výkonu společnosti,
- procesy hlavní, vytvářející hodnotu pro externího zákazníka a jsou součásti hodnototvorného řetězce,
- procesy podpůrné, zajišťující fungování ostatních procesů [4].

1.1.3 Informační systém

Informační systém lze definovat jako soubor prvků zabezpečující činnosti pro potřeby uživatelů. Mezi prvky IS se řadí především lidi, technické prostředky a metody (programy), které v daném IS působí. Činnosti IS jsou především sběr, zpracování, přenos a uchování dat a informací [5].

1.2 JavaScript

JavaScript (dále jen JS) je dynamický programovací jazyk působící na vyšších úrovních a slouží nejenom pro programování moderních webových aplikací. Jazyk je součástí triády webových technologií: HTML slouží na specifikování obsahu, CSS vytváří styl a JS určuje chování webových stránek [2].

```
document.getElementById("button").onclick = () => {
    alert('Hello word!');
};
```

Kód 1: Ukázka "Hello World!" v JS (Zdroj: Vlastní zpracování)

JS jako programovací jazyk vhodně charakterizuje zákon, který zformuloval programátor Jeff Atwood. Zákon pojednává o tom, že pokud je možné aplikaci napsat v JS, tak daná aplikace nakonec bude napsaná v JS [3].

1.2.1 Vývoj JS

ECMAScript je skriptovací jazyk, který vytváří základy pro JS. ECMAScript je standardizován organizací ECMA International ve specifikacích ECMA-262 a ECMA-402. Mezi nejdůležitější verze patří ECMA-262 5. edice vydaná v roce 2009, která se používá ještě dnes a je plně podporována všemi moderními prohlížeči, a ECMA-262 6. edice vydaná v roce 2015. Novější edice (7. a 8. edice) ECMA-262 jsou vydané v rocích 2016 a 2017. Dále existuje dynamické pojmenování právě vyvíjené verze ES.Next [6].

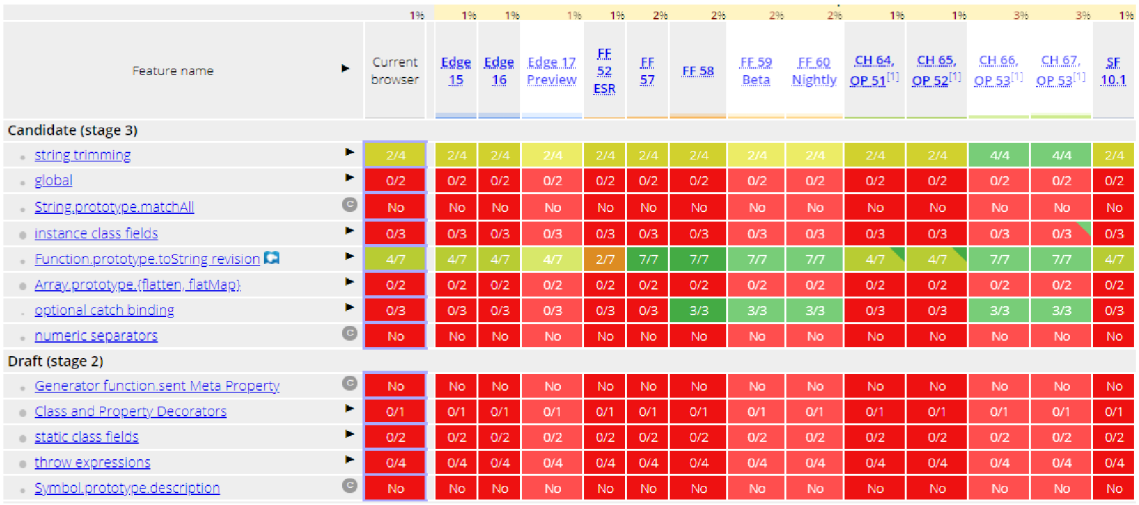
		100%	99%	96%	99%	100%	100%	100%	100%	100%	100%	100%	100%
Feature name	▸	Current browser	SF 10.1	SF 11	IE 11	Edge 15	Edge 16	Edge 17 Preview	FF 58	FF 59 Beta	CH 65, OP 52[1]	CH 66, OP 53[1]	CH 67, OP 53[1]
Object/array literal extensions	▸	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
Object static methods	▸	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13
Array methods	▸	12/12	12/12	11/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12
String properties and methods	▸	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
Date methods	▸	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
Function.prototype.bind		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JSON		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Immutable globals	▸	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
Miscellaneous	▸	8/8	7/8	7/8	7/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8
Strict mode	▸	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19

Obrázek 1: Podpora nových funkcionalit u vybraných prohlížečů, ECMA-262 5.edice (Zdroj: [7])

Feature name	Current browser	IE 11	Edge 15	Edge 16	Edge 17 Preview	FF 52 ESR	FF 57	FF 58	FF 59 Beta	CH 66, OP 53[1]	CH 67, OP 53[1]	SF 10.1	SF 11	SF 11.1
	97%	11%	96%	96%	96%	94%	97%	97%	97%	98%	98%	99%	99%	99%
Optimisation														
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2
Syntax														
default function parameters	7/7	0/7	7/7	7/7	7/7	6/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7
rest parameters	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
spread (...) operator	15/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
object literal extensions	6/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
for..of loops	9/9	0/9	9/9	9/9	9/9	7/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9
octal and binary literals	4/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
template literals	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
RegExp "y" and "u" flags	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
destructuring, declarations	22/22	0/22	22/22	22/22	22/22	21/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22
destructuring, assignment	24/24	0/24	24/24	24/24	24/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
destructuring, parameters	24/24	0/24	23/24	23/24	23/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
Unicode code point escapes	2/2	0/2	2/2	2/2	2/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
new.target	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2

Obrázek 2: Podpora části nových funkcionalit u vybraných prohlížečů, ECMA-262 6.edice

(Zdroj: [7])



Feature name	Current browser	Edge 15	Edge 16	Edge 17 Preview	FF 52 ESR	FF 57	FF 58	FF 59 Beta	FF 60 Nightly	CH 64, OP 51[1]	CH 65, OP 52[1]	CH 66, OP 53[1]	CH 67, OP 53[1]	SF 10.1
	1%	1%	1%	1%	1%	2%	2%	2%	2%	1%	1%	3%	3%	1%
Candidate (stage 3)														
string.trimming	2/4	2/4	2/4	2/4	2/4	2/4	2/4	2/4	2/4	2/4	2/4	4/4	4/4	2/4
global	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
String.prototype.matchAll	No	No	No	No	No	No	No	No	No	No	No	No	No	No
instance class fields	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3
Function.prototype.toString revision	4/7	4/7	4/7	4/7	2/7	7/7	7/7	7/7	7/7	4/7	4/7	7/7	7/7	4/7
Array.prototype.{flatten, flatMap}	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
optional catch binding	0/3	0/3	0/3	0/3	0/3	3/3	3/3	3/3	0/3	0/3	0/3	3/3	3/3	0/3
numeric separators	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Draft (stage 2)														
Generator function.sent Meta Property	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Class and Property Decorators	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
static class fields	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
throw expressions	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
Symbol.prototype.description	No	No	No	No	No	No	No	No	No	No	No	No	No	No

Obrázek 3: Podpora části nových funkcionalit u vybraných prohlížeči, ES.Next (Zdroj: [7])

Výše vyobrazené obrázky zobrazují výběr funkcí a internetových prohlížečů. ECMAScript 5. edice je momentálně nejrozšířenější, díky 99-100 % podpoře v prohlížečích, nové aplikace se již píší v ECMAScript 6. edice, právě díky téměř 100 % podpoře dostupných prohlížečů. ECMAScript Next je uveden pro zajímavost, protože nová funkcionalita ECMAScript se objevuje až se zpožděním v prohlížečích [7].

1.2.2 Výhody JS

JS je jedním z nejvyvinutějších expresivních programovacích jazyků. V následujících podkapitolách je shrnutí jeho výhod [3].

Výkon

Většina JS kódu v moderních prohlížečích je kompilována, vysoce optimalizována a vykonávána jako nativní kód. Jedná se o Just-in-time kompilaci a rychlost výkonu kódu se blíží k softwaru napsaném v C nebo C++. Aplikace, které využívají Node.js, jsou řízeny událostmi a bez blokací, což kompenzuje dynamické vázání a ostatní režijní náklady JS [3].

Objekty

JS obsahuje mnoho objektově orientovaných funkcí. JSON je využíván ve většině moderních webových aplikací pro komunikaci a uchovávání dat. Dále využívá model prototypového dědění, tzn. že místo tříd se využívá model objektových prototypů. Nové objekty automaticky dědí metody a atributy od svého nadřazeného objektu (rodiče) skrz prototypový řetěz. Kdykoli možné modifikovat prototyp objektu, co činní JS velmi flexibilním a dynamickým jazykem [3].

Objektová orientace využívající prototypy je mnohem flexibilnější než klasická dědičnost, takže je v JS možné napodobit objektovou orientaci založenou na třídách a modely dědičnosti, které jsou typická pro programovací jazyk Java. Toto je možné vykonat pro každou jednu funkcionalitu, a ve většině případech, dosáhnout zkrácení a vyšší přehlednost kódu [3].

Navzdory všeobecnému přesvědčení, JS podporuje funkce jako zapouzdření, polymorfismus, vícenásobnou dědičnost a kompozici [3].

Syntaxe

Syntaxe se podobá na jazyky jako C++, Java, C# apod. Avšak ovšem JS má pod povrchem svá specifika, kterými se od těchto jazyků liší. Syntaxe objektového literálu JS je velmi jednoduchá, flexibilní a stručná. Objektový literál se stal hlavním standardem pro komunikaci klient-server ve formě již zmíněného JSON, který je více kompaktní a flexibilní než XML, který JSON nahradil [3].

```
let object = {
    firstProperty: 'value of the first property',
    secondProperty: 2,
    thirdProperty: true,
    fourthPrperty: null,
    fifthProperty: []
};
```

Kód 2: Příklad objektového literálu JS (Zdroj: Vlastní zpracování)

Funkce

Téměř všechno v JS je objekt, včetně funkcí. Díky této funkcionalitě, funkce mohou být použity všude, kde by mohla být použita proměnná, včetně parametrů při volání funkce. Převážně se tomuto využívá na definování autonomních callback (zpětné volání) funkcí pro asynchronní operace, nebo k vytvoření funkcí vyššího řádu [3].

```
let array = [1, 2, 3];
let newArray = array.map(callback(currentVal));
console.log(newArray); // [2, 4, 6]

const callback = value => {
    return value * 2;
};
```

Kód 3: Příklad callback funkce (Zdroj: Vlastní zpracování)

1.3 Architektura moderní JS aplikace

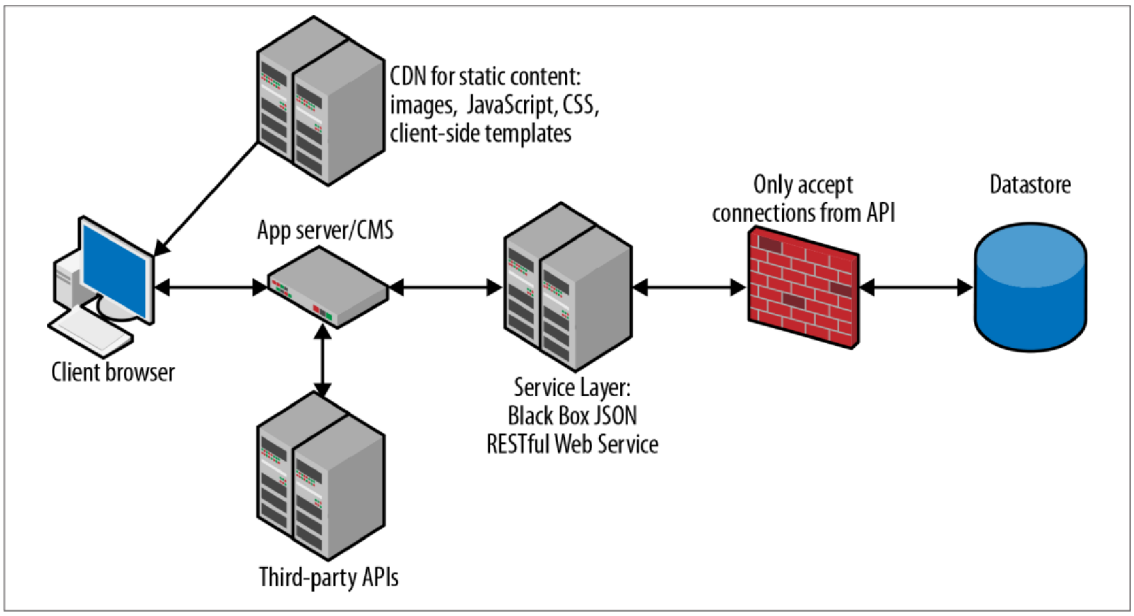
I když každá aplikace je jedinečná, většina sdílí některé zájmy, jako třeba hostingové infrastruktury, řízení zdrojů, prezentace a chování uživatelského rozhraní [3].

1.3.1 Infrastruktura

Infrastruktura se může velmi lišit a mít různé způsoby ukládání dat do mezipaměti. Vyobrazení, jak vyjmenované komponenty do sebe zapadají se nachází v následujícím obrázku. Infrastruktura se obecně skládá z následujícího (od serveru po uživatelské rozhraní):

- datové úložiště,
- VPN nebo firewall,
- vrstva webové služby s černou skříňkou (JSON RESTful),
- různé API třetích stran,

- aplikační server nebo CMS ke směrování požadavků a doručování stránek klientovi,
- statický CDN pro data v mezipaměti,
- webový klient (prohlížeč) [3].



Obrázek 4: Infrastruktura (Zdroj: [3])

1.3.2 Datové úložiště a komunikace

Datové úložiště je místo pro ukládání aplikačních dat. Běžně se jedná o systém řízení relačních databází (RDBMS) s jazykem SQL, ale popularita NoSQL řešení je na vzestupu. JSON je přístupný standard vytvořený Douglesem Crockfordem určující syntaxi objektového literálu JS pro účely prezentaci, komunikaci a ukládání dat [3].

Na následujícím obrázku je příklad JSON popisující kolekci knih. Jak lze vidět, formát je velmi podobný syntaxi objektového literálu JS s několika důležitými rozdíly:

- všechny jména atributů a textové řetězce jsou ohraničené dvojitým citačním znaménkem,
- JSON záznamy nemohou obsahovat kruhové odkazování,
- JSON nemůže obsahovat funkce [3].

```json
{
  "object": {
    "firstProperty": "value of the first property",
    "secondProperty": 2,
    "thirdProperty": true,
    "fourthPrperty": null,
    "fifthProperty": []
  }
}
```

Kód 4: Příklad JSON (Zdroj: Vlastní zpracování)

1.3.3 SQL a NoSQL databáze

Databáze je systém pro ukládání dat takovým způsobem, že uložené informace se dají zpětně získat [8].

Relační databáze SQL

V relační databázi se informace ukládají do tabulek s řádky a sloupci. K tabulce se přistupuje jako ke kolekci objektů stejného typu (řádky). Databázový systém se stará o to, jak jsou data uloženy, spravovány a získávány. Pro zajištění přesnosti a přístupnosti dat, relační tabulky mají určitá integritní pravidla. Prvním pravidlem je, že všechny řádky v tabulce by měly být odlišné. Druhým pravidlem je, že hodnoty ve sloupcích nesmí být opakující se skupiny nebo pole. Třetím integračním pravidlem je koncept hodnoty *null*, kdy se databáze postará o data, která nejsou dostupná. Prázdná hodnota je rovna prázdné hodnotě, ale dvě *null* hodnoty si nikdy nejsou rovny. K identifikaci řádků se používá unikátní sloupec, případně skupina sloupců nazývající se primární klíč [8].

Employees Table

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Obrázek 5: Příklad tabulky se zaměstnanci (Zdroj: [9])

SQL je jazyk navržený pro potřeby relačních databází. Základní SQL příkazem je SELECT, který využívá základní klauzule FROM a WHERE. Díky těmto dvěma vybraným slovům je možné z databáze získat potřebné informace. Například pro získání jména zaměstnanců, kteří mají v databázi uložené číslo aute, lze použít následující SQL dotaz [9].

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number IS NOT NULL
```

Kód 5: Ukázka SQL (Zdroj: [9])

Objektově orientované databáze NoSQL

NoSQL na rozdíl od relačních databází ukládají záznamy v dokumentech, nebo v úryvkách dokumentů bez využití tabulkově orientovaných úložišť. Dokumentově orientovaná úložiště obvykle ukládají data v XML formátu, na rozdíl od objektově orientovaných úložišť, která využívají JSON formát. Objektově orientované úložiště jsou vhodným formátem pro vývoj webových aplikací, protože jak již bylo řečeno, JSON je formát, který je v JS nativně využíván při komunikaci. Příkladem populárních databází ve formátu JSON jsou MongoDB a CouchDB [3].

V databázi vedené v MongoDB lze k vybrání dat použít metodu *find()*. Tento dotaz může vrátit veškeré dokumenty v kolekci anebo dokumenty, které jsou specifikovány v dotazu. V případě, že budeme mít databázi s objektem (kolekcí) *restarants*, můžeme využít dotaz *db.restaurants.find()*, který nám vrátí všechny dokumenty uložené v objektu. Pro potřeby filtrování, lze do závorek metody vložit objekt s podmínkami. Tento objekt má následující strukturu: *{field1: value1, field2: value2, ...}* a dotaz pro získání všech restaurací ve čtvrti Manhattan by vypadal následovně: *db.restaurants.find({ "borough": "Manhattan" })* [9].

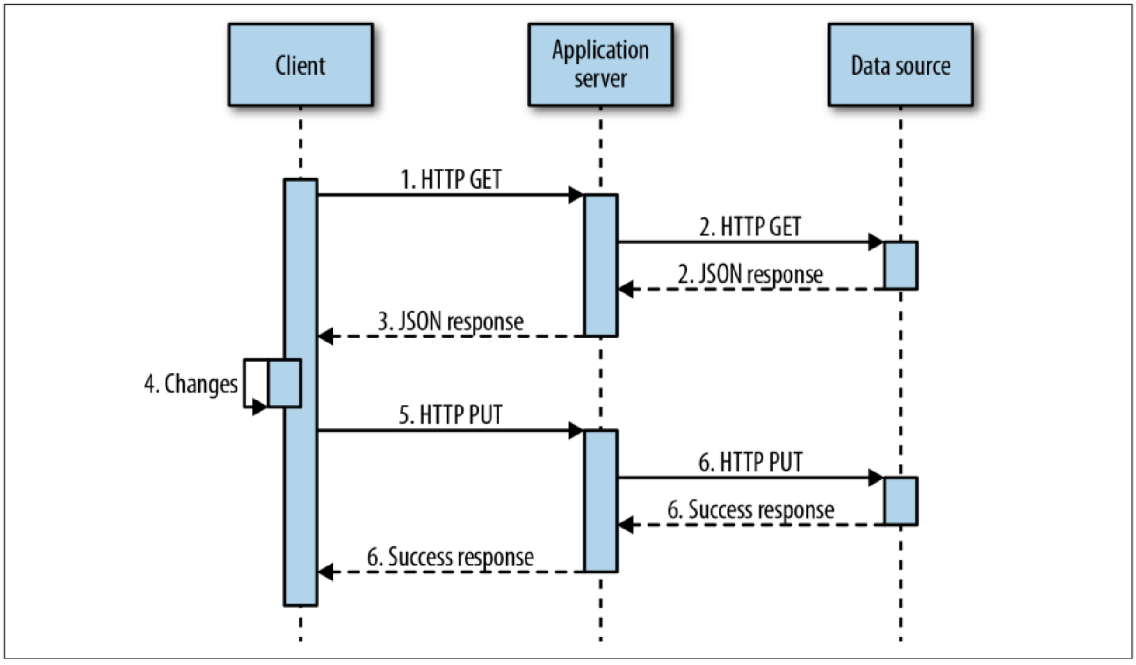
1.3.4 Webové služby RESTful JSON

RESTful (Representional State Transfer) je architektura na bázi klient-server komunikace, která odděluje zájmy mezi zdroji dat a uživatelským rozhraním (případně jinými konzumenty dat). Ke službám, které implementují REST se odkazuje jako RESTful. Server spravuje zdroje dat, neimplementuje uživatelské rozhraní. Klienti implementují uživatelské rozhraní v jakémkoli směru a jazyku. REST architektura se

nestará, jak je uživatelské rozhraní implementováno, soustředí se na udržování stavu aplikace mezi klientem a serverem [3].

Webové služby RESTful používají HTTP akce (slovesa) pro účely oznámení serveru, jakou akci klient zamýšlí. Podporované akce jsou následující:

- vytvoření nového záznamu v kolekci zdrojů: HTTP POST,
- získání zastoupení zdrojů: HTTP GET,
- nahrazení zdroje: HTTP PUT,
- smazání zdroje: HTTP DELETE [3].



Obrázek 6: REST sekvence (Zdroj: [3])

Jednotlivé body z obrázku jsou následující:

1. Klient zažádá server pomocí HTTP GET o zdroje pomocí URI (Uniform Resource Indicator). Každý zdroj na serveru má unikátní URI [3].
2. Server získá data (typicky z databáze nebo cache) a upraví je do vhodného zastoupení pro klienta [3].
3. Data jsou vráceny zpět ve formě dokumentu. Dokumenty jsou běžně textové řetězce obsahující JSON objekty, ale REST je agnostický ohledně úpravy dat. Je běžné se setkávat s RESTful službami založené na XML. Většina nových služeb používají data ve formátu JSON a mnoho z nich podporují oba typy formátování [3].

4. Klient manipuluje se zastoupením dat [3].
5. Klient vytvoří volání na stejný koncový uzel URI s HTTP PUT. K tomuto požadavku připojí upravená data [3].
6. Zdrojová data na serveru jsou nahrazená daty přijaté v HTTP PUT požadavku [3].

Je běžné nebýt si jistý, jestli použít PUT nebo POST na změnu zdroje. REST zjednodušuje toto zmatení. PUT je používán tehdy, kdy klient je schopen si vygenerovat vlastní bezpečná ID. V opačném případě je vždy vytvořen POST požadavek pro vytvoření nového zdroje. V tomto případě server vygeneruje ID a vrátí jej klientovi [3].

Například pro vytvoření nového uživatele je využitá cesta ke zdroji /users/ pomocí POST požadavku, kde server vygeneruje unikátní ID a je možno k němu přistupovat na */users/:userid* (kde :userid je unikátní ID). Server vrátí klientu zastoupení uživatele s unikátním URI. Nemělo by docházet k případům modifikování existujícího zdroje pomocí POST, je možné pouze přidat data na nižší úroveň (děti) [3].

Ke změně uživatelského jména se využije PUT kde cesta ke zdroji je */users/:userid* a k požadavku je přiložen modifikovaný záznam uživatele. Důležité je podotknout, že tato operace nahradí celý zdroj uživatele, takže je nutné k PUT přiložit i nezměněné údaje [3].

1.3.5 HTML

HTML (Hypertext Markup Language) je základní technologií používaná pro definování struktury webové stránky. Užívá se k specifikování, zda webový obsah je paragraf, seznam, odkaz, obrázek, multimediální přehrávač, formulář, nebo jeden z mnoha ostatních dostupných elementů, případně nově definovaných elementů [10].

HTML není programovací jazyk, jedná se o jazyk užívající značky (tagy). HTML se skládá ze série elementů, které ohraničují, obalují, nebo označují různé části obsahu webové stránky, za účelem odlišného zobrazení nebo chování. Ohraničující tagy mohou zobrazovat část obsahu jako odkaz na jinou webovou stránku, měnit text na tučný, a podobně [10].

```
<p>                                    <!-- Opening tag -->
    My cat is very grumpy              <!-- Content -->
</p>                                   <!-- Closing tag -->
```

Kód 6: Ukázka HTML (Zdroj: [10], upraveno)

Hlavní části výše uvedeného HTML elementu jsou:

- počáteční tag (opening tag) se skládá z názvu elementu (p) zabaleného v lomených závorkách (< >), počáteční tag určuje, kde element začíná a získává dané vlastnosti – v tomto případě značí, kde začíná paragraf,
- uzavírací tag (closing tag) je stejný jako počáteční tag, až na to, že obsahuje přední lomítko před názvem elementu, to značí, kde element končí,
- obsah (content) je v tomto případě text daného elementu,
- element je počáteční a uzavírací tag, včetně obsahu [10].

JSX

JSX je rozšíření syntaxe pro JS a slouží jako náhrada dokumentu HTML. Je doporučen pro využití spolu s knihovnou React a popisuje strukturu uživatelského rozhraní. Díky JSX je možné využívat HTML objekty spolu s využitím JS. Zápis JSX je velmi podobný HTML, ale existují výjimky, jako například atribut *class*, který v HTML slouží k identifikování elementu pro přiřazení CSS stylů. V JS je slovo *class* rezervováno pro tvorbu objektů, proto se v JSX namísto *class* využije *className* [11].

Uvnitř JSX je možné volat JS proměnné i funkce. V následujícím obrázku je příklad volání funkce přímo v JSX.

```
function formatName(user) {
    return user.firstName + ' ' + user.lastName;
}

const user = {
    firstName: 'Harper',
    lastName: 'Perez'
};

const element = ( <h1> Hello, { formatName(user) }</h1> );
```

Kód 7: Příklad JSX – pozdravení uživatele (Zdroj: [11], upraveno)

1.3.6 CSS

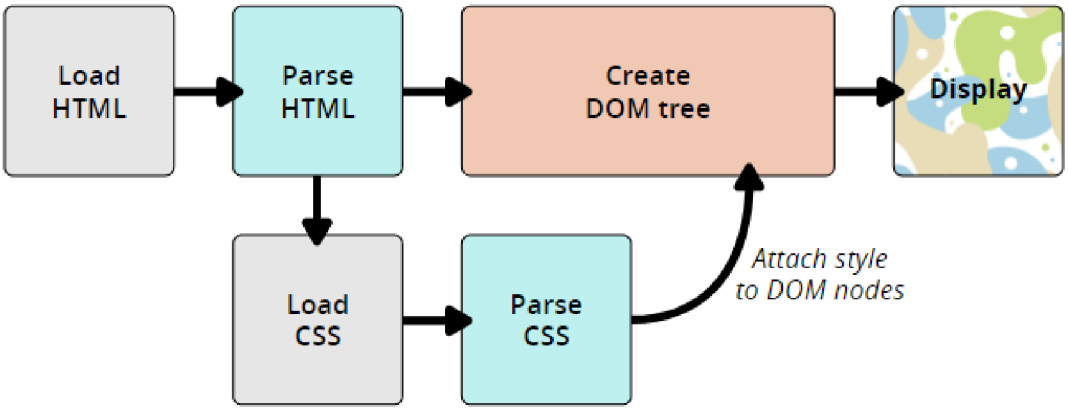
CSS (Cascading Stylesheets) je používáno pro stylování a úpravu (písmo, barva, velikost, animace atd.) HTML elementů. CSS je jazyk pro specifikování, jakým způsobem jsou dokumenty (HTML, XML atd.) prezentovány uživatelům. Webové prohlížeče použijí CSS pravidla pro úpravu elementů. CSS pravidla jsou formována z:

- soubor vlastností, které mají své hodnoty, díky kterým se HTML obsah upraví,
- selektor, kterým volíme elementy, pro které daný soubor vlastností platí [10].

```css
h1 {
    color: blue;
    background-color: yellow;
    border: 1px solid black;
}

p {
    color: red;
}
```

Kód 8: Příklad CSS (Zdroj: [10], upraveno)

Pro potřeby vykreslení webových stránek je CSS potřeba načíst, zpracovat a následně jej připojit k DOM stromu, jak je vyobrazené na následujícím obrázku [10].



Obrázek 7: Propojení HTML a CSS (Zdroj: [10])

DOM

DOM (Document Object Model) má stromovitou strukturu. Každý element, atribut, i část textu v HTML se stává ve stromové struktuře DOM uzlem. Tyto uzly jsou definovány vztahy k jiným uzlům, některé uzly jsou rodiče uzlů, které jsou děti a zároveň uzly děti mohou mít i sourozence. Například v následujícím obrázku uzel *p* je rodič a má čtyři děti. Tři z těchto dětí jsou zároveň rodiči, a to uzly *span*, kteří mají jako děti textový uzel [10].

```
<p>
    Let's use:
    <span>Cascading</span>
    <span>Style</span>
    <span>Sheets</span>
</p>
```

Kód 9: DOM struktura (Zdroj: [10], upraveno)

Sass

Sass (Syntactically Awesome Style Sheets) je rozšířením CSS přidávající sílu a eleganci základnímu jazyku. Povoluje využití proměnných, vnořených pravidel, importů a dalších. Sass napomáhá organizaci velkých stylových souborů a menším stylových souborů napomáhá optimalizací rychlosti Hlavní funkcionalita Sass je následující:

- plně kompatibilní s CSS,
- rozšíření jazyka, jako například proměnné, vnořování a mixins,
- mnoho funkcí pro práci s barvami a ostatními hodnoty,
- rozšířená funkcionalita jako například kontrolní směrnice pro knihovny,
- přehledně formátovaný a upravitelný výstup [12].

```
#main p {
  color: #00ff00;
  width: 97%;

  .redbox {
    background-color: #ff0000;
    color: #000000;
  }
}
```

Kód 10: Ukázka vnořování v Sass (Zdroj: [12], upraveno)

Bootstrap

Bootstrap je framework určený pro vývoj UI, který zjednodušuje a zrychluje vývoj webových stránek. Bootstrap obsahuje šablony založené na HTML a CSS pro formuláře, tlačítka, tabulky, a mnoho dalších, včetně JS dodatků. Boostrap také zlehčuje vývoj responsivního web designu, kterým se myslí automatické přizpůsobení velikosti stránky na základě zobrazovacího zařízení (velký monitor vs. smartphone). Dále Bootstrap obsahuje šablony pro Sass proměnné, které mohou být developerem využívané. Výhody Bootstrap jsou následující:

- jednoduchý na použití,
- responzivní vlastnosti,
- zaměření na mobilitu,
- kompatibilita s prohlížeči [13].

Bootstrap obsahuje mnoho šablon, například pro přidání různých druhů tlačítek lze využít třídu *btn* a *btn*-*. Na následujících obrázcích je vyobrazení všech tlačítek z Bootstrap [13].

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
```

Kód 11: Ukázka Bootstrap tříd (Zdroj: [13], upraveno)

Následující obrázek zobrazuje již vykreslená tlačítka, jak se vykreslí na stránce. Povšimněme si i faktu, že se délka tlačítek automaticky přizpůsobuje délce textu.

Obrázek 8: Bootstrap tlačítka (Zdroj: [13])

1.3.7 React

React je JS knihovna pro vytváření uživatelských rozhraní. React je určitý, vytvářející možnost jednoduše vytvořit interaktivní UI [11].

```
ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('root')
);
```

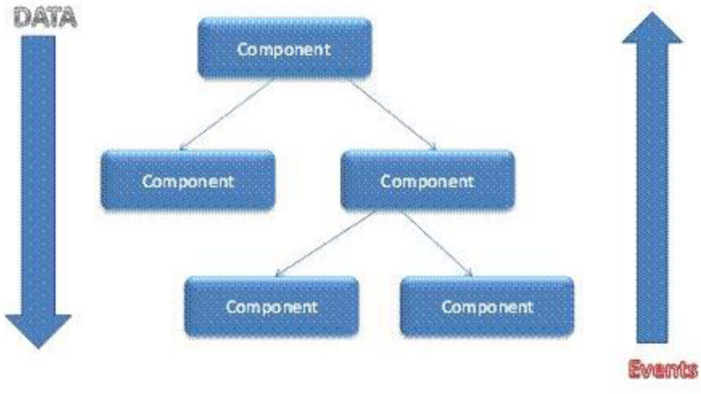
Kód 12: Ukázka React (Zdroj: [11], upraveno)

React je používán k vývoji takzvaných single page aplikací. Je využíván pro zobrazovací vrstvu pro webové a mobilní aplikace. React také umožňuje vytváření znovupoužitelných UI komponent. React vytvořil Jordan Walke, softwarový inženýr, který v té době pracoval pro Facebook a první nasazení části Facebooku se provedlo v roce 2011 [14].

React umožňuje vývojářům vytvářet velké webové aplikace, ve kterých se mění zobrazovaná data bez znovu načítání webových stránek. Hlavním účelem Reactu je být rychlý, škálovatelný a jednoduchý. To funguje pouze pro UI aplikace, což koresponduje s MVC (Model – View – Controller) šablonou [14].

Hlavní funkce, které charakterizují React jsou následující:

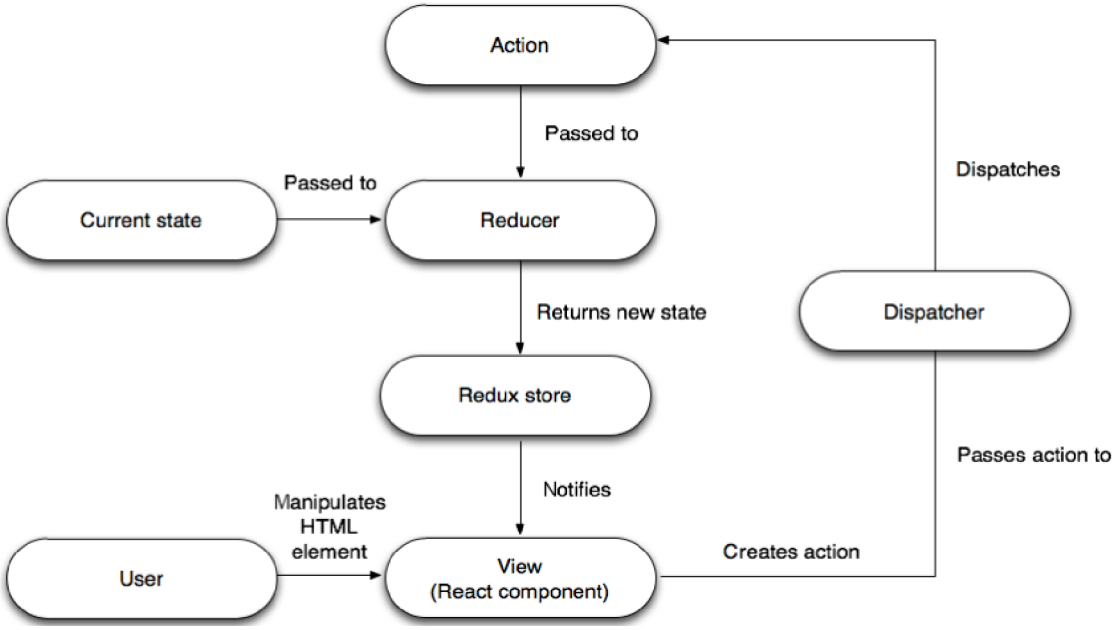
- JSX – jak již bylo zmíněno dříve, JSX umožňuje vytvářet HTML šablony pomocí JS,
- React Native – React má nativní knihovny, které byli oznámeny Facebookem v r. 2015, které poskytují React architekturu pro nativní aplikace pro operační systémy IOS, Android a UPD,
- data proudící jedním směrem – v Reactu je soubor nezměnitelných hodnot v HTML tazích předán komponentu, který hodnoty interpretuje; komponent tyto hodnoty nemůže přímo modifikovat, ale může předat callback funkci, která může hodnoty modifikovat; tento proces je obecně známý jako „vlastnosti proudí dolů, akce vzhůru“,
- virtuální DOM – React v mezipaměti vytváří strukturu dat, která vypočítává vytvořené změny a až poté se aktualizuje prohlížeč; to umožní vývojářům vytvářet funkce, jako by se celá webová stránka aktualizovala [14].



Obrázek 9: Vlastnosti proudí dolů, akce vzhůru (Zdroj: [14])

Redux

Redux je malá knihovna určená nejenom pro React a slouží jako předvídatelný stavový kontejner. Napomáhá vytváření aplikací, které se chovají konzistentně, fungují v různých prostředí (klient, server a nativní) a jsou jednoduché na testování [15].



Obrázek 10: Tok dat v knihovně Redux (Zdroj: [16])

Redux je užitečný nástroj pro organizování stavu aplikace, ale ne vždy to je potřeba. V následujících bodech jsou případy, kdy použití knihovny Redux dává smysl:

- v aplikaci se vyskytuje rozumné množství dat, která se v mezičase mění,
- je potřeba jednotný stav napříč aplikací,

- uchování stavu v nejvyšším komponentu již není dostatečné [15].

Redux zajišťuje tři hlavní principy, a to:

- jednotný zdroj pravdy,
- stav je pouze ke čtení,
- změny jsou vykonávány ryzími funkcemi [15].

1.3.8 Node.js

Node je událostmi řízený, asynchronní JS runtime a byl sestrojen za účelem tvorby škálovatelných síťových aplikací. V následujícím obrázku je příklad, kde je možné vyřizovat vícero požadavků současně. Při každém novém připojení je spuštěna callback funkce, pokud zde nejsou žádná připojení, Node se uspí [17].

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3090;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World\n');
});

server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Kód 13: Příklad Node.js (Zdroj: [17], upraveno)

Díky tomu se liší od dnes běžného modelu souběžnosti, kde jsou zaměstnána vlákna operačního systému. Síťové aplikace na bázi vláken jsou relativně neefektivní a velmi složité na užívání. Na rozdíl od toho, Node lze používat bez obav zablokování procesů, protože žádné blokace neobsahuje. Skoro žádné funkce neoperují přímo s I/O, takže se procesy nikdy neblokují a vzhledem k tomu, že se neblokují, je rozumné v Node vyvíjet škálovatelné systémy [17].

Node je designově podobný a ovlivněný systémy jako Event Machine (Ruby), nebo Twisted (Python). Node využívá systém řízení událostmi o něco více než již zmíněné systémy, vzhledem k tomu, že využívá smyčky událostí jako runtime namísto

knihovny. Node jednoduše vstoupí do smyčky událostí po vykonání vstupního skriptu a vystoupí ze smyčky, když už nejsou žádné callback funkce, které může spustit [17].

HTTP je subjekt, který podporuje veškeré operace globálně dostupné jiným subjektům, navržený s ohledem na tok dat a nízkou latenci. Díky tomu je Node vhodný pro webové knihovny, nebo framework [17].

Express

Express je rychlý, přizpůsobitelný a minimalistický webový framework pracující na serverové straně pro Node.js. Express je napsán v JS a chová se pouze jako tenká vrstva rysů webové aplikace. Volnost využití této knihovny propojená s rychlým prvotním nastavením a JS prostředí v Node dělá Express silného kandidáta pro rychlý vývoj a prototypování [21].

Node.js prostředí je klíčová část umožňující Express jednoduše kompilovat a nasadit. Node umožňuje vývojářům psát kód pro serverovou stranu (BE) pro webové aplikace v JS místo ostatním BE jazykům, jako například PHP, Python, Java atd. Používáním npm (Node Package Manager) lze jednoduše nainstalovat knihovny, včetně knihovny Express [21].

```
import express from 'express';

let app = express();
app.get('/', (req, res) => {
    res.send('<h1>Hello World!</h1>');
});

app.listen(3090);
```

Kód 14: Ukázka nastavení serveru pomocí Express (Zdroj: [21], upraveno)

Výše uvedená ukázka nastavení serveru pomocí Express je jednoduchý odsek JS (ES5) kódu, který vytváří server a poslouchá na portu 3090. První řádek ukázky importuje knihovnu Express a další řádek získává instanci. App.get zajišťuje, že pokud je vytvořen HTTP požadavek GET na localhost:3090, server odpoví jednoduchou odpovědí pomocí HTML. Poslední řádek zajišťuje, že server poslouchá na již zmíněném portu [21].

1.3.9 Babel

Babel je JS transpiler (překladač) především pro verzi ES6 (ECMA-262 6.edice), který je schopný tento kód přeložit do starších verzí, které jsou podporovány prohlížeči. Babel byl vytvořen Australským vývojářem jménem Sebastian McKenzie, který se staral o překlad nové syntaxe ES6 včetně JSX do starších verzí. Babel překládá i novější verze JS a to ES7 (ES2016). Díky tomu je již přístup k syntaxi, a tedy funkcionalitě například *async/await*, dvě vybraná slova, která již nyní mění to, jak developeři píší asynchronní kód [18].

Díky Babel knihovně je tedy následující JS kód ES6, který vytváří pole tří číslic a následně jej píše do konzole, kde každá číslice je inkrementovaná o 1. ES6 syntaxe si lze všimnout uvnitř volání funkce map [18].

```
const input = [1, 2, 3];
console.log(input.map(item => item + 1));
```

Kód 15: ES6 (Zdroj: [18], upraveno)

Kód ES6 je následně pomocí Babel přeložen do ES5, která má vysokou kompatibilitu se staršími prohlížeči. Uvnitř volání funkce map se přeložila přehlednější a kratší syntaxe ES6 do syntaxe ES5 [18].

```
var input = [1, 2, 3];
console.log(input.map(function (item) {
    return item + 1;
}));
```

Kód 16: ES5 výstup (Zdroj: [18], upraveno)

1.3.10 Swagger

Swagger je framework pro popisování API užitím běžně využívaných popisných frází. Swagger je možné přirovnat k technickým výkresům budov. Je možné využít jakékoli stavební materiály, ale je potřeba dodržet parametry technického výkresu [20].

Oproti ostatním dokumentačním nástrojům jako například RAML, APIBlueprint, nebo Summation, Swagger poskytuje další výhody, mezi které patří hlavně:

- je srozumitelný pro vývojáře, ale i pro ostatní členy týmu, jako například manažery, partnery, ale i potenciální klienty,

- čitelné rozhraní pro vývojáře, ale i stroje, čímž je myšleno, že dokumentace může být využita i k automatizaci procesů, které jsou na API závislé,
- jednoduše přizpůsobitelné, díky čemu lze jednodušeji testovat a odstraňovat chyby související s API [20].

Swagger UI slouží pro zobrazení a organizaci přístupných API. Veškerá dokumentace, která je uložená v JSON nebo YAML dokumentech slouží jako interaktivní dokumentace. Proto je jednoduché API testovat a jinak využívat, jelikož jedním kliknutím lze odeslat požadavek [20].

Obrázek 11: Příklad Swagger UI (Zdroj: [20])

1.4 GDPR

V této kapitole krátce popíši, co je GDPR a především jak nařízení ovlivňuje firmy a vlastníky údajů.

GDPR (General Data Protection Regulation) je obecné nařízení na ochranu osobních údajů je nová legislativa Evropské Unie a má za úkol zvýšit ochranu osobních dat občanů. Nařízení bylo v dubnu r. 2016 přijato a vstoupí v účinnost od 25. května 2018. Nařízení je namířeno na firmy i jednotlivce, kteří jakýmkoli způsobem nakládají s osobními údaji. Cílem nařízení GDPR je chránit digitální práva občanů EU [19].

GDPR přináší pro firmy novou povinnost, a to princip zodpovědnosti, mezi které paří zejména následující oblasti:

- implementace záměrné a nezbytné ochrany dat,
- vypracování DPIA,

- jmenování DPO,
- zavedení pseudonymizace osobních údajů,
- vedení záznamů o činnostech zpracování,
- konzultace s dozorovým orgánem před zpracováním osobních údajů [19].

Na druhou stranu, GDPR přináší pro vlastníky údajů (subjekty údajů) nová práva, o kterých musí být důkladně informování. Tyto práva jsou určitým způsobem omezena pouze tehdy, pokud existuje právní důvod pro jejich další zpracování. K těmto právům patří zejména:

- vznést námitku proti zpracování nebo přenositelnost osobních údajů,
- přístup k údajům, které jsou o něm shromažďován,
- právo být zapomenut [19].

GDPR přináší i rozšíření definic osobních údajů, díky kterým do kategorie osobních údajů nově spadají parametry jako e-mail, IP adresa nebo cookies. Dále je zavedená nová kategorie, která podléhá přísnějšímu dohledu, a to genetické a biometrické údaje [19].

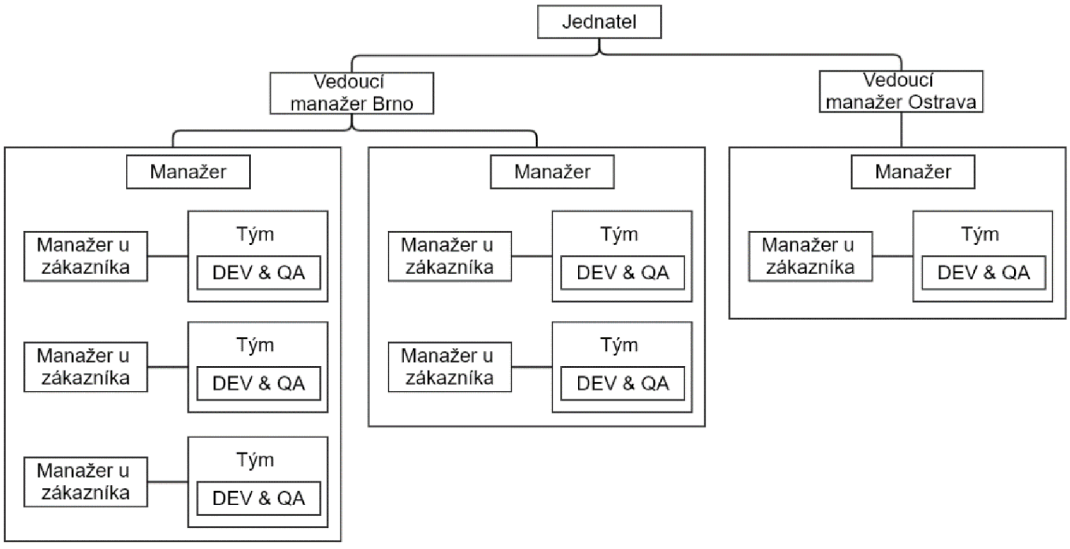
2 ANALÝZA PROBLÉMU A SOUČASNÉ SITUACE

2.1 Představení společnosti

Společnost XYZ s. r. o. byla založena v roce 1996 se sídlem v Brně. Jedná se o outsourcingovou společnost poskytující vývojářské a konzultační služby v oblasti vývoje a dodávání ICT služeb. Zákazníci společnosti jsou převážně menší a střední firmy se sídlem ve Spojených státech amerických. Vzhledem posílení české koruny vůči americkému dolaru za rok 2017 se společnost rozhodla zákazníky diverzifikovat tím, že služby nabízí na domácím, ale i evropském trhu.

2.2 Organizační struktura

Společnost zaměstnává přibližně 90 zaměstnanců, kteří pracují ve dvou pobočkách – Brno a Ostrava. Toto rozšíření společnosti proběhlo před 2 lety kvůli nedostatku softwarových vývojářů v Brně. Zaměstnanci pracují na pozicích softwarových vývojářů (DEV) a testerů (QA) pro 15 zákazníků v 2–10členných týmech. Tito zaměstnanci se zodpovídají manažerovi ve společnosti, ale také manažerovi u zákazníka, většinou manažeru odpovídající za produkt (PO). Jedná se tedy o maticovou organizační strukturu, kdy jsou zaměstnanci odpovědni většinou dvěma manažerům.



Obrázek 12: Organizační struktura společnosti (Zdroj: Vlastní zpracování)

Většina zákazníků při řízení projektů používá agilní metodiku Scrum, proto jsou týmy většinou složené ze zaměstnanců u zákazníka a zaměstnanců společnosti. Tým je typicky složený z PO, který působí u zákazníka, prezentuje požadavky na produkt a představuje mezičlánek mezi týmem a stakeholdery. Dále se v týmu nachází Scrum Master (SM), který aplikuje na celý tým metodiky Scrum, komunikuje a řeší nesrovnanosti mezi členy týmu ale i s PO. Nedílnou součástí týmu jsou DEV a QA, kteří samotný produkt vyvíjejí, starají se o jeho kvalitu, architekturu a funkčnost. Členové týmu SM, DEV a QA se nacházejí ve většině projektech jak ve společnosti, tak i u zákazníka a denně spolu komunikují na meetinzích, kde se probírají nesrovnalosti, blokace a náplň práce jednotlivých členů týmu.

2.3 Mise a vize

Mísí společností XYZ s.r.o. založené na poskytování služeb je vyvíjet a dodávat softwarová (SW) řešení, které podporují úspěšnost jejich zákazníků. Přestože se jedná o outsourcingovou společnost, nejedná se pouze o dodávání služeb, ale klade se důraz na výsledek spolupráce a vytvořenou hodnotu díky spolupráci obou stran, proto se společnost prezentuje jako co-sourcingová společnost.

Vize společnosti spočívá v ulehčení spolupráce mezi zákazníkem a společnosti při vytváření SW produktů, a to hlavně díky vysoce kvalifikovaným softwarovým inženýrům, kteří plynule hovoří v anglickém jazyce. Zde je kladen důraz na zaměstnávání odborníků v daných oborech, které zákazník požaduje.

2.4 Poskytované služby a technologie

Jak již bylo zmíněno výše, společnost nabízí služby spojené se SW vývojem a údržbou a to především:

- webové aplikace,
- mobilní aplikace,
- virtualizace prostředí a produktu,
- vysokorychlostní Linux/Unix procesy,
- návrh a tvorba databáze,
- automatizace testování.

Většina DEV a QA mají široký rozsah co se technologií týká, a tak se společnost může jednoduše přizpůsobit požadavkům zákazníka. Například u tvorby a správy webových aplikací se DEV, ale i QA se potkává s více technologiemi, jako na straně UI jsou JavaScript a knihovny k němu patřícím (např. Angular, React, Bootstrap), PHP, Ruby on Rails atd. ale i s BE technologiemi Java, .NET, C++, SQL a další.

Na základě rozsahu znalostí je společnost prezentována společenskou firmou XYZ, LLC, která sídlí v USA a stará se o navazování kontaktů s americkými společnosti.

2.5 Informační technologie

Vzhledem k tomu, že se jedná o společnost vyvíjející SW, hlavním aktivem spadající do informačních technologií jsou PC a notebooky. Každý zaměstnanec je vybaven PC nebo notebookem, dle své volby, na kterém provádí činnosti spojené s vývojem a údržbou SW. Tyto PC a notebooky jsou obměňovány dle aktuální potřeby, které pro zaměstnance schvaluje nadřízený manažer. Kromě PC a notebooků zaměstnanec disponuje většinou 2 monitory, myší, klávesnicí a sluchátky s mikrofonem pro komunikaci se zákazníky.

Dále se ve společnosti nachází místnosti vyhrazené pro komunikaci týmu a se zákazníky, které jsou vybaveny televizí/projektorem pro prezentování, PC, myší, klávesnicí, web kamerou a specializovaným komunikačním zařízení pro zajištění co nejvyšší kvality zvuku při komunikaci přes internet.

Pro potřeby virtuálních prostředí a úložišť, společnost disponuje serverovnou, kde se nachází ICT technologie pro zajištění komunikace s veřejnou sítí, DMZ, VPN a dále servery a úložiště pro potřeby DEV a QA.

Evidence informačních technologií je vedena IT oddělením, které se stará o správu, nastavení a obměnu veškerých zařízení ve společnosti. Nyní společnost disponuje HW s průměrným staří 3 roky, u PC a notebooků jsou to 2 roky.

2.6 Stručný přehled podnikových procesů

V následujících kapitolách popíši hlavní a vedlejší procesy vykonávané ve společnosti.

2.6.1 Hlavní procesy

Mezi hlavní procesy společnosti patří práce týmů na projektech za účelem uspokojení potřeb zákazníka. Vstupem do těchto procesů je především znalost a čas DEV a QA a jejich výstupem je částečná funkcionalita až hotová aplikace, kterou zákazník zhodnocuje. Ne každý proces je vykonávaný v každém týmu a bližší podrobnosti vycházejí z požadavků zákazníka.

Návrh aplikace nebo její části

Tento proces se vykonává vždy po specifikování požadavků PO a většinou u menších částí aplikace je vykonáván pouze DEV, případně i QA v týmu. Při návrhu větších částí, nebo celé aplikace jsou začleněni do procesu manažeři jako CTO, architekt apod. a spolupracují s týmem při výběru nejvhodnější technologie, návrhu aplikace a způsobu implementace.

Vývoj SW

Proces vývoj SW má za úkol DEV a v tomto procesu vzniká požadovaná služba. Po provedení návrhu se funkcionalita implementuje, spolu s implementací unit testů, provedení verifikaci kódu ostatními DEV (CR) a nasazení na vývojové, případně produkční prostředí. Na vývojovém prostředí mají k funkcionalitě přístup ostatní DEV, QA a i PO, který si na tomto prostředí ověřuje, zda bylo splněné zadání. Na produkční prostředí se funkcionalita dostává po otestování, kam má přístup i zákazník zákazníka.

Údržba SW

Údržba SW se v jistých ohledech podobá vývoji SW. Impulzy pro inicializaci údržby SW jsou především chyby nalezené QA, PO anebo zákazníkem zákazníka. V méně početných případech pak upgrade, či update technologií, na kterých je produkt postaven, nebo refaktoring, který často bývá spojován s upgrade, či update technologií.

Testování SW

Testování vyvinuté funkcionality je u některých zákazníků úkolem jejich zákazníků a z části DEV. Avšak u většiny zákazníků společnosti je pro tyto účely funkce QA. Jedná se o lidi v týmu, kteří zajišťují kvalitu dodávaného SW v průběhu celého cyklu vývoje. Při plánování má QA za úkol hledat potencionální problémy spojené s funkčností a po vývoji funkcionality má za úkol funkcionalitu manuálně otestovat. V některých týmech se využívá i automatizované testování a to tak, že QA pro danou funkcionalitu

napíše v programovacím jazyce test, který se spouští v intervalech na všech prostředí a zajišťuje kvalitu SW jako celku.

2.6.2 Podpůrné procesy

Podpůrné procesy zajišťují fungování hlavních procesů společnosti. Vstupem do těchto procesů je práce a znalost HR, managementu a společenské firmy za spolupráce zaměstnanců a zákazníků.

Nábor zaměstnanců

V procese náboru zaměstnanců mají největší roli HR, které podle požadavků společenské firmy a manažerů řídí nábor zaměstnanců. Na to, aby byl zaměstnanec přijat, musí projít následujícím procesem:

1. Absolvování online testu
2. Telefonický pohovor
3. Osobní pohovor

Aplikace pro online testování je zakoupena formou roční licence od třetí strany a výsledky z jednotlivých fází se ukládají do aplikace vytvořené a vlastněné společností. Ve fázi osobního pohovoru se potencionální zaměstnanec setkává s manažerem a HR.

Osobní růst zaměstnanců

Proces osobní růst zaměstnanců zajišťuje, aby zaměstnanci nestagnovali a postupně se stávali stále profesionálnějšími s rozsáhlou znalostí technologií. Vzhledem k tomu, že se sektor s informačními technologiemi neustále rozvijí, je tento proces nezbytný. Pro DEV i QA to znamená učení se novým technologiím a zdokonalování se v tom, co je pro zákazníka právě důležité. Z vize společnosti také vyplývá, že do tohoto procesu také spadá vzdělávání a zdokonalování se v anglickém jazyce.

Získávání zákazníků

Jak již bylo zmíněno výše, o získávání nových zákazníků se především stará společenská firma XYZ, LLC. V tomto procesu obchodník prezentuje potencionálním zákazníkům společnost, v jaké jsou její služby, zákazníci apod. V případě že má potencionální zákazník zájem, domluví se meeting s manažerem, DEV a QA, kteří mají s danými technologiemi zkušenosti a proberou se požadavky. Pokud je potencionální

zákazník spokojen, ubere se k podepsání smluv, mezi které patří i SLA, a formuje se tým, který následně zákazník zaškolí ve svém prostředí.

Spolupráce se zákazníkem

Tento proces zajišťuje neustálou spokojenost zákazníka s týmem a s poskytovanými službami. V tomto procesu vystupují především manažeři na obou stranách, kteří řeší případné neshody a tým přebírá odpovědnost. V případě, že zákazník navyšuje, nebo snižuje počet členů v týmu, společnost požadavkům vyhoví.

2.7 Dotazníková metoda

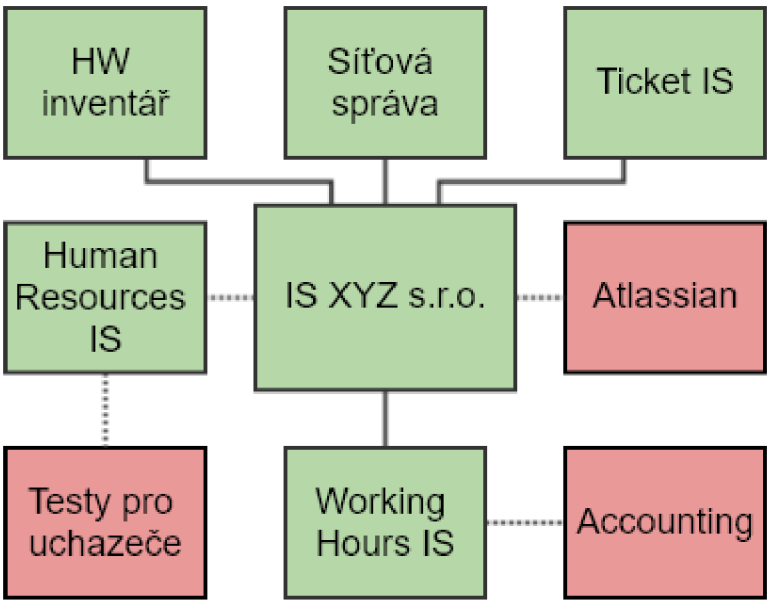
Cílem dotazníkového šetření bylo zjištění, jak lze posílit potenciál firmy. Sběr dat k problematice byl veden ústně krátkým pohovorem s manažery společnosti. Většina odpovědí souvisela se změnou kurzu amerického dolaru vůči české koruně. Výstup z dotazníkové metody lze shrnout do následujících bodů:

- snížení nákladů,
- osobní rozvoj stávajících zaměstnanců pomocí technický meetingů,
- získání zákazníků ze západu Evropy,
- nábor specialistů v oblasti IT pro nově příchozí zákazníky,
- analýza a rozvoj interních projektů,
- důraz na postupný růst stávajících projektů,
- spolupráce s univerzitami,
- propagace (zviditelnění) firmy.

Z pohovorů nejvíce vyplývalo, že společnost má potíže s vyššími náklady a rozvojem s čím souvisí i zviditelnění firmy. Pro účely diplomové práce jsem se rozhodl pro problém snížení nákladů a v následující části jej řešit analýzou informačního systému a identifikací části informačního systému, která není vlastněná společností a je zpoplatněná. Dále je relativně jednoduché tuto část nahradit pomocí interního projektu na kterém může pracovat zaměstnanec, případně zaměstnanci společnosti a po dokončení implementace a propojení části s informačním systémem se náklady na celý informační systém snižují.

2.8 Informační systém

Informační systém (IS) společnosti je z většiny vytvořen a vlastněn společností. V následujícím diagramu jsou zobrazené návaznosti na IS a odlišně zabarvené placené služby – červeně a vlastní služby – zeleně.



Obrázek 13.: Přehled IS (Zdroj: Vlastní zpracování)

Tento IS je spolu propojen a je nasazen pouze ve společnosti. Přístup do vlastních modulů IS je povolen pouze pro zaměstnance společnosti, a to z interní sítě, nebo pomocí VPN tunelu z veřejné sítě. Pronajímané moduly se nenacházejí v interní síti, a tak je umožněn přístup z veřejné sítě. Řízení přístupu do jednotlivých modulů propojené nepřerušovanou čarou je řízeno pomocí LDAP protokolu, do ostatních částí IS jsou zapotřebí přístupové údaje zvlášť.

Zaměstnanci, jako jsou DEV a QA mají pouze omezený přístup, a to do modulů následujících modulů:

- síťová správa – přidání/odebrání síťového zařízení,
- ticket IS – založení ticketu,
- Atlassian – zobrazení a správa dokumentace,
- working hours IS – sledování a úprava docházky.

Ostatní přístupy jsou uděleny podle pozice k přístupům zmíněným výše, a to následovně:

- oddělení HR má přístup do HR IS a aplikace Testů pro uchazeče,
- oddělení IT má plný přístup do HW inventáře, síťové správy a ticket IS,
- účetní oddělení má plný přístup do Working Hours IS a Accounting,
- manažeři a jednatel mají omezený přístup do všech modulů.

Z obrázku také vyplývá, že HR IS a Testy pro uchazeče jsou samostatné moduly nezávislé na IS společnosti XYZ s. r. o. Vzhledem k tomu, že společnost se snaží o vytvoření vlastního produktu, je možné modul Testy pro uchazeče vytvořit jako součást HR IS a tak mít potencionální vlastní produkt, který je možné nabízet ostatním firmám. Zároveň také vyplývá, že moduly Atlassian a Accounting nejsou vlastněné společností, ale vzhledem k rozsáhlosti daných modulů není výhodné tyto moduly vytvářet vlastní, pouze v případě, že by společnost chtěla podobný produkt vyrobit a prodávat což však není náplní problematiky práce.

2.9 Nábor zaměstnanců

Na proces náboru zaměstnanců je kladen velký důraz, jelikož dlouhodobě spolupracují se zákazníky a pro růst společnosti jsou důležité dobré reference. Cílovým stavem přijímacích řízení zaměstnanců je, že společnost přijme nadané a zajímavé lidi a pro ně vytvoří pozice. To se však při nedostatku DEV a QA na trhu práce, a i na projektech ne vždy upřednostňuje. Proces je tedy většinou inicializován novým zákazníkem, případně změnou u zákazníka. Po zjištění požadavků se zkontroluje, zda ve společnosti existuje takový zaměstnanec, který vyhovuje a je k dispozici. Pokud není, vystaví se inzerát na webových stránkách, sociálních sítí a zároveň vedení požádá zaměstnance společnosti o doporučení známých, kteří jsou vhodní a mají zájem.

Uchazeč nyní musí projít přijímacím řízením, které se skládá ze tří fází, jak již bylo zmíněno dříve. Pokud uchazeč vyhovuje, je přijat a inzerát na webových stránkách se odstraní. V případě, že uchazeč nevyhovuje, je mu tato skutečnost oznámena, ale stále je uložen v HR IS, odkud je možné s uchazečem navázat kontakt v budoucnu.

2.9.1 RACI matice

Na základě slovního popisu procesu lze sestrojit RACI matici, která definuje procesní role a činnosti včetně odpovědností k uvedeným činnostem.

Tabulka 1: RACI Matice (Zdroj: Vlastní zpracování)

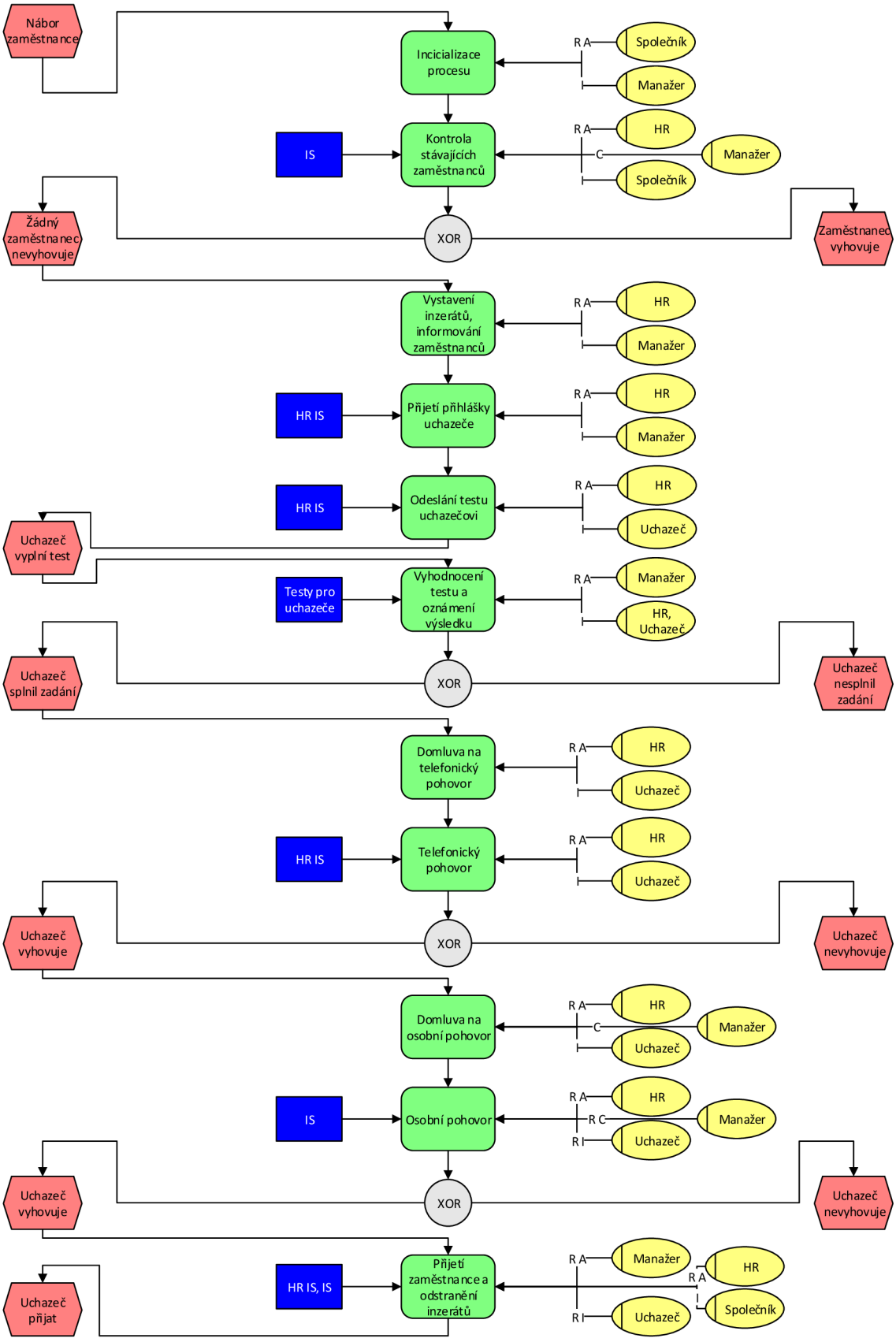
Popis aktivity	Procesní role			
	HR	Manažer	Společník	Uchazeč
Incicializace procesu	I	I	R A	
Kontrola vyhovujících zaměstnanců	R A	C	I	
Vystavení inzerátů, informování zaměstnanců	R A	C		
Přijetí přihlášky uchazeče	R A	I		
Odeslání testu uchazečovi	R A			I
Vyhodnocení testu a oznámení výsledku	I	R A		I
Domluva na telefonický pohovor	R A			I
Telefonický pohovor	R A			I
Domluva na osobní pohovor	R A	C		I
Osobní pohovor	R A	R		R
Přijetí zaměstnance	C	R A	I	I
Odstranění inzerátů	R A	C		

2.9.2 EPC diagram

Na následujícím obrázku je EPC diagram procesu nábor zaměstnanců. Proces je inicializován společníkem a má celkem 3 odlišné výstupy:

- zaměstnanec společnosti je k dispozici a vyhovuje požadavkům,
- uchazeč vše splnil a byl přijat,
- uchazeč nebyl přijat.

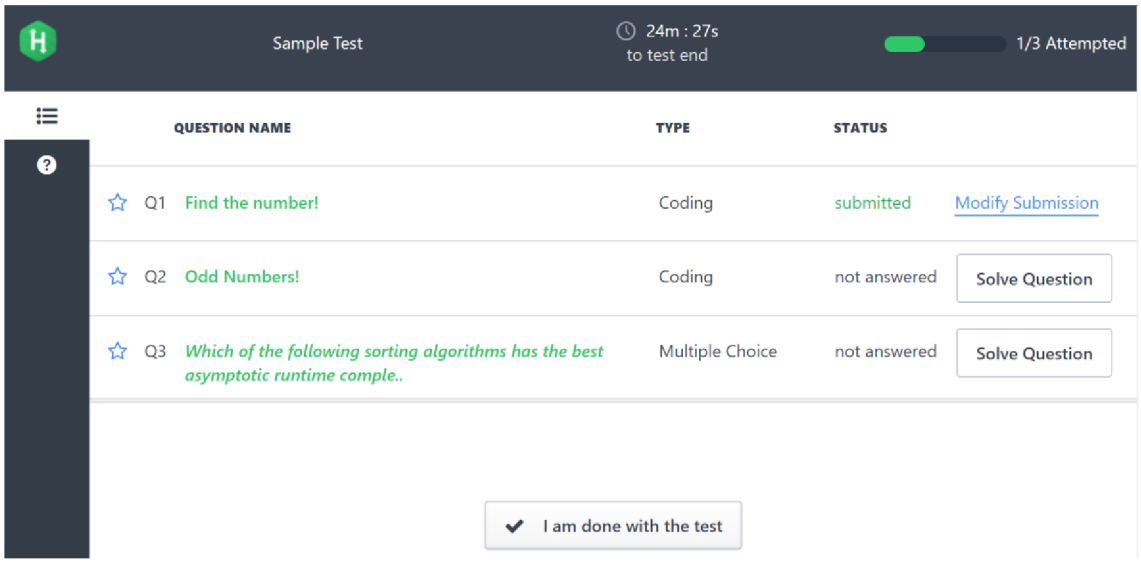
K případu, že uchazeč není přijat může dojít nesplněním testu, nebo odmítnutím uchazeče po telefonickém, případně osobním pohovoru. V takových případech je uchazeč stále uložen v HR IS a HR oddělení se k takovému uchazeči může vrátit a pokud má zájem, pozvat jej na pohovor.



Obrázek 14.: EPC Diagram náboru zaměstnance (Zdroj: Vlastní zpracování)

2.10 Testy pro uchazeče

Tato placená webová aplikace se využívá k testování znalostí uchazečů do společnosti XYZ s. r. o. Jedná se o produkt společnosti HackerRank, kde se provádí správa a oprava vyplněných testů. Produkt je využíván díky roční licenci, za kterou společnost platí 100 000Kč bez DPH.

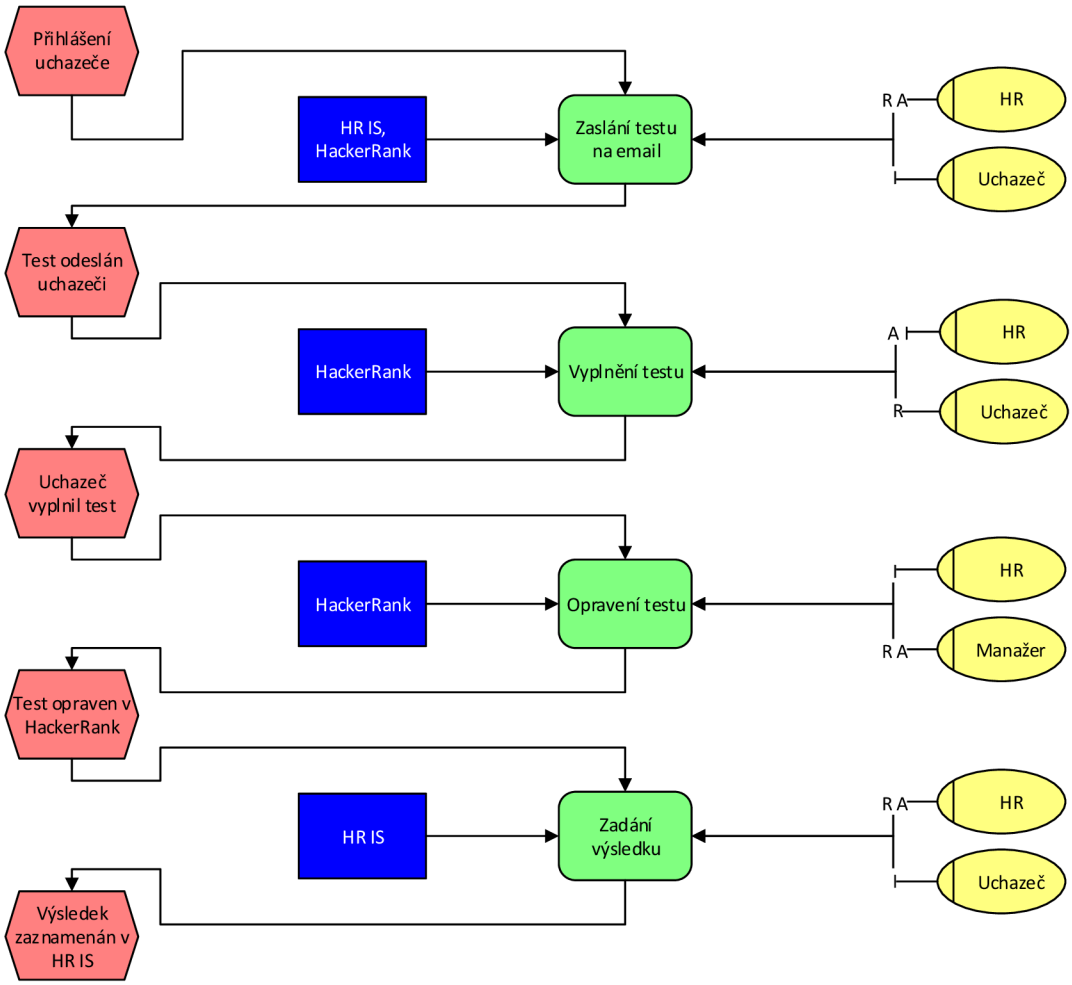


Obrázek 15.: Ukázkový test z HackerRank (Zdroj: [1])

Před prvním použitím bylo nutné nadefinovat testy pro uchazeče. Tyto definice testů jsou velmi jednoduché, obsahují 3-5 otázek a je definováno, že odpovědi jsou pouze v textovém formátu. Po definování různých testů pro DEV, QA, HR, manažery apod. bylo nutné nastavit API, aby bylo možné odkaz na test odesílat přímo z HR IS, včetně nastavení e-mailových upozornění, kdy uchazeč test vyplnil.

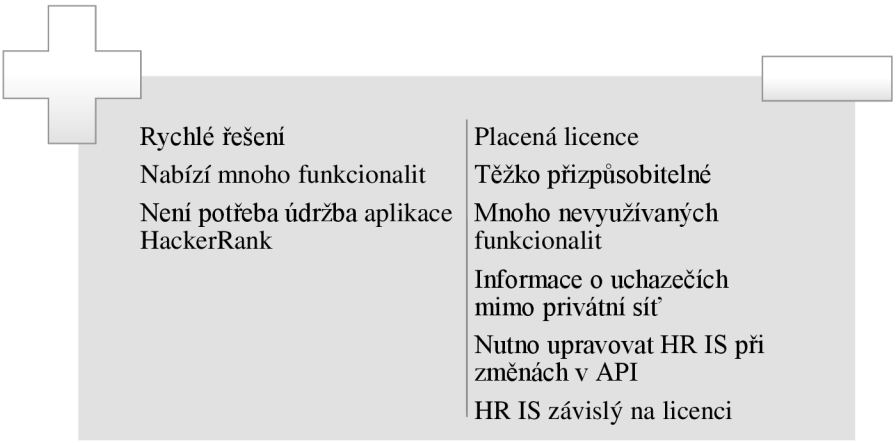
Po vyplnění testu aplikace HackerRank odešle HR oddělení e-mail s informací, že test byl uchazečem vyplněn a je připraven na opravu. Opravu provádí manažer, který v případě přijetí uchazeče, je jeho nadřízený. Tuto opravu provádí v prostředí HackerRank a po dokončení oprav uvědomí HR oddělení o provedení opravy a informuje je o další akci – zda pozvat uchazeče na telefonický pohovor, či nikoli. Tato informace se manuálně zaznamenává do HR IS.

V následujícím obrázku je EPC diagram znázorňující průběh odeslání, vyplnění a opravení testu.



Obrázek 16.: EPC diagram testování uchazeče (Zdroj: Vlastní zpracování)

Z EPC diagramu výše znázorněném vyplývá, že modul HackerRank je možné nahradit vlastním modulem, který sjednotí systém HR IS a tak bude nezávislý na produktu společnosti HackerRank. Modul HackerRank má následující silné a slabé stránky.



Obrázek 17.: Výhody a nedostatky používání HackerRank (Zdroj: Vlasntí zpracování)

2.11 Výstupy z analýz

V analytické části jsem představil společnost, její procesy a rozepsal problém, se kterým se společnost potýká. Pro potřeby práce jsem se rozhodl řešit problém snížení nákladů, a to vývojem vlastní části IS. Tento problém podporuje také fakt, že si společnost nepřeje, aby do systému HR IS vstupovali aplikace třetích stran. Díky této preferenci lze problém řešit pouze vývojem vlastní webové aplikace. V následující části práce jsou potřeby vývoje této aplikace jsou zpracovány veškeré požadavky a hlavní toky, které usnadňují chápání a potažmo i implementaci jednotlivých funkcionalit. Dále je zapotřebí webovou aplikaci navrhnout od databáze až po uživatelské rozhraní, následně webovou aplikaci implementovat ve zvolených technologiích a v poslední fázi se uskuteční testování a dvojitý provoz pro zajištění kontinuity náborového procesu.

3 VLASTNÍ NÁVRH ŘEŠENÍ

Následující část práce je zaměřená na milníky vývoje, architekturu a samotný vývoj webové aplikace.

3.1 Požadavky společnosti na produkt k testování uchazečů

Z analýzy a získaných informací vyplývá, že společnost není plně spokojená se současným stavem HR IS a hledá možnosti na zlepšení. Cílem společnosti je snížení nákladů na otestování znalostí uchazečů. Tento cíl je možno naplnit dvěma způsoby:

- naleznout levnější a vhodnější produkt,
- vyvinout vlastní aplikaci.

Dále společnost definuje požadavky společnosti, které základní vlastnosti aplikace musí obsahovat, aby bylo možné nahradit produkt od HackerRank:

- funkční aplikace prezentující firmu uchazečům,
- následující API rozhraní:
  - možnost vygenerovat test pro uchazeče,
  - status testu,
- následující požadavky k samotnému testu:
  - informace pro uživatele před začátkem testu,
  - časově omezený,
  - konfigurovatelný,
  - obsahuje pokyny k hodnocení,
  - více testů pro jednotlivé skupiny uchazečů (DEV, QA atd.)
  - odpovědi na otázku ve formátu volného textu nebo výběr z možností,
  - navázaný na odpovědi (pokud je test upraven, předchozí vyplněné testy zůstávají neměnné),
  - celkové hodnocení testu formou volného textu (čímž není myšleno „prošel“ a „neprošel“),
  - tisk vyplněného testu,
- požadavky společnosti týkající se GDPR:
  - export dat vyplněných uchazečem,
  - smazání dat vyplněných uchazečem,
  - souhlas o zpracování údajů před vyplněním testu.

Volitelné požadavky jsou následující:

- nastavení váhy u otázek,
- bodové ohodnocení otázek,
- časový limit na otázku,
- log událostí vykonaných uživatelem,
- zaslání upomínky o nevyplněném testu uchazeči a HR.

3.2 Toky aktivit vykonávaných v aplikaci

V následující sekci jsou popsané toky aktivit, rozdělené podle pozice. Jedná se o pozice manažera, HR a uchazeče. Tyto toky slouží pro upřesnění a jednoduššímu pochopení požadavků zmíněných v předchozí kapitole.

Tvorba testu

Za vytvoření a obsah šablony je odpovědný manažer. Je zapotřebí, aby šablony byly dostupné pro jednotlivé pozice a to: QA, DEV, SM, HR a Manažer. Po definování těchto 5 rozdílných testů se plánují pouze menší a občasné úpravy, případně přidání nového testu, proto není zapotřebí, aby definice testu byla vytvářená nebo upravována pomocí UI rozhraní.

Zaslání testu

Zaslání testu probíhá z HR IS po přijetí přihlášky z hlavních stránek společnosti. Pokud si HR přeje poslat e-mail uchazeči, je zapotřebí jej nalézt, pozvat jej vyplnění testu a zaslat mu e-mail z HR IS, kde se v šabloně již nachází unikátní URL odkaz na test. Odeslání e-mailu je poté viditelné v profilu uchazeče v HR IS.

Vyplnění testu

Uchazeči přijde přes e-mail URL odkaz na stránky testovací aplikace. Po kliknutí jsou zobrazeny informace o testu (např. obecné informace k vyplňování, časový limit apod.) včetně upozornění o zpracování osobních údajů. Pokud se uživatel rozhodne vyplnit test, spustí se časový limit a umožní se uživateli přístup k otázkám. V případě vypršení časového limitu se test uloží a uživateli se znemožní provádět další úpravy. V HR IS se u uchazeče zobrazí upozornění o tom, že test byl vyplněn.

Oprava testu

Manažer má po přihlášení možnost do testu nahlédnout a následně jej zhodnotit podle dostupných informací přímo v aplikaci. Po vyhodnocení se v HR IS zobrazí upozornění, že daný uchazeč již má test opravený a HR oddělení spolu s manažerem rozhodnou, zda uchazeče pozvou na telefonický rozhovor.

Tisk nebo export vyplněného testu

Manažer nebo zaměstnanci z HR oddělení mají možnost vyplněný test vytisknout nebo exportovat pro potřeby konzultace s uchazečem při osobním pohovoru, případně k poslání uchazeči, pokud si o to požádá.

Smazání vyplněného testu

Pro potřeby GDPR je nutné, aby manažer nebo zaměstnanci z oddělení HR měli možnost test smazat, pokud o to uchazeč požádá.

3.3 Posouzení možností implementace

Společnost zamítla způsob řešení obměnou modulu za jiný placený modul, protože si nepřeje, aby do HR IS vstupoval modul třetí strany, což znamená, že je potřeba tento modul naimplementovat a vlastníkem modulu po nasazení se stává společnost. V následující tabulce jsou porovnány dostupné možnosti toho, kdo může modul vytvořit s finančním odhadem. Data byly získány od společnosti XYZ s. r. o.

Tabulka 2: Možnosti implementace (Zdroj: Vlastní zpracování)

Zdroj práce	Odhad doby trvání	Cena za člověkoden	Celkem
Outsourcing	25	CZK 6,000	CZK 150,000
Zaměstnanec	25	CZK 4,000	CZK 100,000
Student	30	CZK 1,500	CZK 45,000
Student s podporou zaměstnance	30 + 3	CZK 1500 + 4000	CZK 57,000

Z výše uvedené tabulky vyplývá, že nejrychlejší a finančně výhodná možnost je implementace modulu zaměstnancem. Vzhledem k tomu, že společnost modul nepotřebuje co nejrychleji, nabízí se finančně nejvýhodnější možnost, která znamená implementaci modulu přenechat studentovi, který již není tak rychlý a zároveň je potřeba vyzdvihnout, že produkt již nemusí odpovídat požadované kvalitě. Proto byla vytvořená

alternativa, která byla i zvolaná a sice znamená vyšší náklady, kvalita je již odpovídající požadavkům. Poslední alternativa, a to implementaci přenechat studentovi s podporou zaměstnance, je tedy nejvýhodnější, co se týká poměru ceny a kvality provedení.

3.4 Milníky vývoje

Při vývoji webové aplikace je potřeba dodržet určitý postup, jelikož některé části na sebe navzájem navazují a je jednoduší vyvíjet například UI aplikace až po vytvoření BE. Hlavní milníky webové aplikace jsou následující:

- volba technologií a architektury aplikace,
- návrh databáze,
- návrh API,
- návrh UI,
- vytvoření databáze a modelů,
- vytvoření API,
- vytvoření UI a stylování.

3.5 Volba technologií a architektury aplikace

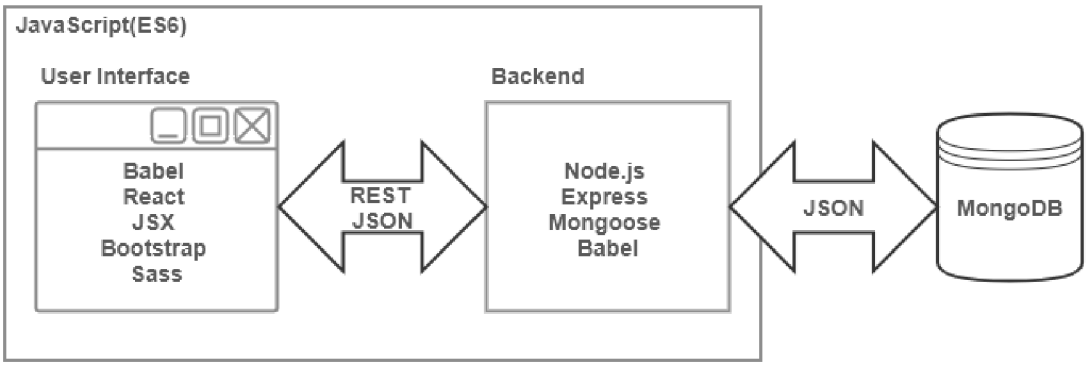
Jako hlavní stavební prvek pro tvorbu webové aplikace byl vybrán skriptovací jazyk JavaScript. JS byl vybrán především díky rozšířenosti a modularitě. Tento jazyk lze využít i pro tvorbu BE i UI a díky tomu se odstraňuje nutnost znát a integrovat vícero programovacích jazyků pro tvorbu webové aplikace.

Díky existenci možnosti využívat JS při tvorbě BE lze propojení s databází zjednodušit a využít NoSQL databáze využívající JSON – MongoDB. V JS lze využít notaci JSON a v případech ukládání, nebo získávání dat z databáze se data a jejich struktura nemusí nijak upravovat a lze ji přímo využít v JS. Při tvorbě samotné BE části je možnost využít Node.js a především knihovnu Express, která vytváří rozhraní pro daný server a je možné s tímto serverem komunikovat pomocí architektury rozhraní REST a komunikace zde bude probíhat pomocí protokolu HTTP. Komunikace mezi BE a UI bude probíhat opět pomocí JSON.

Pro tvorbu UI rozhraní lze využít mnoho knihoven anebo i čistý JS. Pro zjednodušení implementace lze využít knihovnu React, pomocí které lze vytvořit UI rozhraní.

Při samotné implementaci bude nutno využít více knihoven než pouze Express a React. Jedná se především o knihovny pro propojení serveru s databází, knihovny Babel pro překlad ES6 do ES5 pro zajištění kompatibility se staršími webovými prohlížeči apod.

V poslední části je nutné webovou aplikaci stylovat. V tomto případě bude využita knihovna Bootstrap, která nabízí širokou škálu stylu webové aplikace a zároveň při správném využití Boostrap je aplikace responzivní. Bootstrap ovšem neobsahuje plně modulární nastavení, a tak lze využít CSS. Samotné CSS je velmi omezené a při tvorbě se nedají využívat proměnné (například pro barvy), proto je vhodné využít rozšíření CSS a to Sass.



Obrázek 18: Technologie webové aplikace (Zdroj: Vlastní zpracování)

Z výše uvedeného obrázku lze vidět, jaké části webové aplikace spolu budou komunikovat a jaké hlavní knihovny budou využity v daných částech. User Interface je část webové aplikace, která bude běžet na klientovi a skrz toto uživatelské rozhraní se budou dostávat požadavky na Backend (server). Server slouží pro zpracování požadavků a jako brána pro přístup do databáze. Jedná se i o kritickou část z pohledu pravdy – testy pro kandidáty jsou časově omezené a toto omezení je na serveru i UI, pokud by bylo pouze na UI, zkušený uživatel může toto časové omezení odstranit a tím naruší integritu dat. MongoDB je databáze, ve které budou uloženy testy, šablony a informace o uživatelích aplikace. Způsob využití zmíněných knihoven a další knihovny jsou popsány v příslušných částech.

3.6 Datové a funkční modelování

Před návrhem samotné aplikace je potřeba vytvořit návrh datové základny čili databáze. V této části jsou identifikovány datové modely, jejich metody a závislosti. Jak již bylo zmíněno dříve, jedná se o objektově orientovaný datový model využívající JSON. Nejdříve je potřeba vytvořit diagramy tříd, jejich vzájemné vztahy a kardinalitu. Poté následuje funkční modelování, kde probíhá analýza a projekce činností v aplikaci.

3.6.1 Datové modelování

V datovém modelování je potřeba identifikovat datové modely – diagramy tříd a jejich vzájemné vztahy.

Diagramy tříd

Z analýzy vyplývají následující datové objekty: pozvánka (invitation), šablona (template) a otázky (questions), test a odpovědi (answers) a administrátor (manager). V následující tabulce jsou popsány jednotlivé objekty, jejich atributy a typ atributu.

Tabulka 3: Objekty (Zdroj: Vlastní zpracování)

Objekt	Atribut	Typ	Objekt	Atribut	Typ
Invitation			**Manager**		
	email	*String*		email	*String*
	name	*String*		name	*String*
	expiration	*Number*		password	*String*
	testCategory	*String*			
Template			**Test**		
	name	*String*		name	*String*
	author	*String*		email	*String*
	category	*String*		agreedToTerms	*Boolean*
	description	*String*		started	*Number*
	terms	*String*		finishUntil	*Number*
	timeLimit	*Number*		templateId	*String*
	created	*Number*		finished	*Number*
	active	*Boolean*		revised	*Number*
	questions	*Array*		auditor	*String*
				comment	*String*
				answers	*Array*
Questions			**Answers**		
	sequence	*Number*		sequence	*Number*
	header	*String*		answer	*String*
	question	*String*		comment	*String*
	expectedAnswer	*String*			

Pro účely sestavení UML diagramu je nutné si stanovit, jaké operace budou prováděny s objekty. Tyto operace jsou shrnuty v následující tabulce.

Tabulka 4: Operace (Zdroj: Vlastní zpracování)

Objekt	Operace	Objekt	Operace
Invitation		**Manager**	
	invite() getInvitationWithTemplate() deleteInvitation() getInvitations()		login()
Template		**Test**	
	createTemplate() getTemplates() getTemplate() deleteTemplate()		createTest() getTest() saveTest()
Questions		**Answers**	

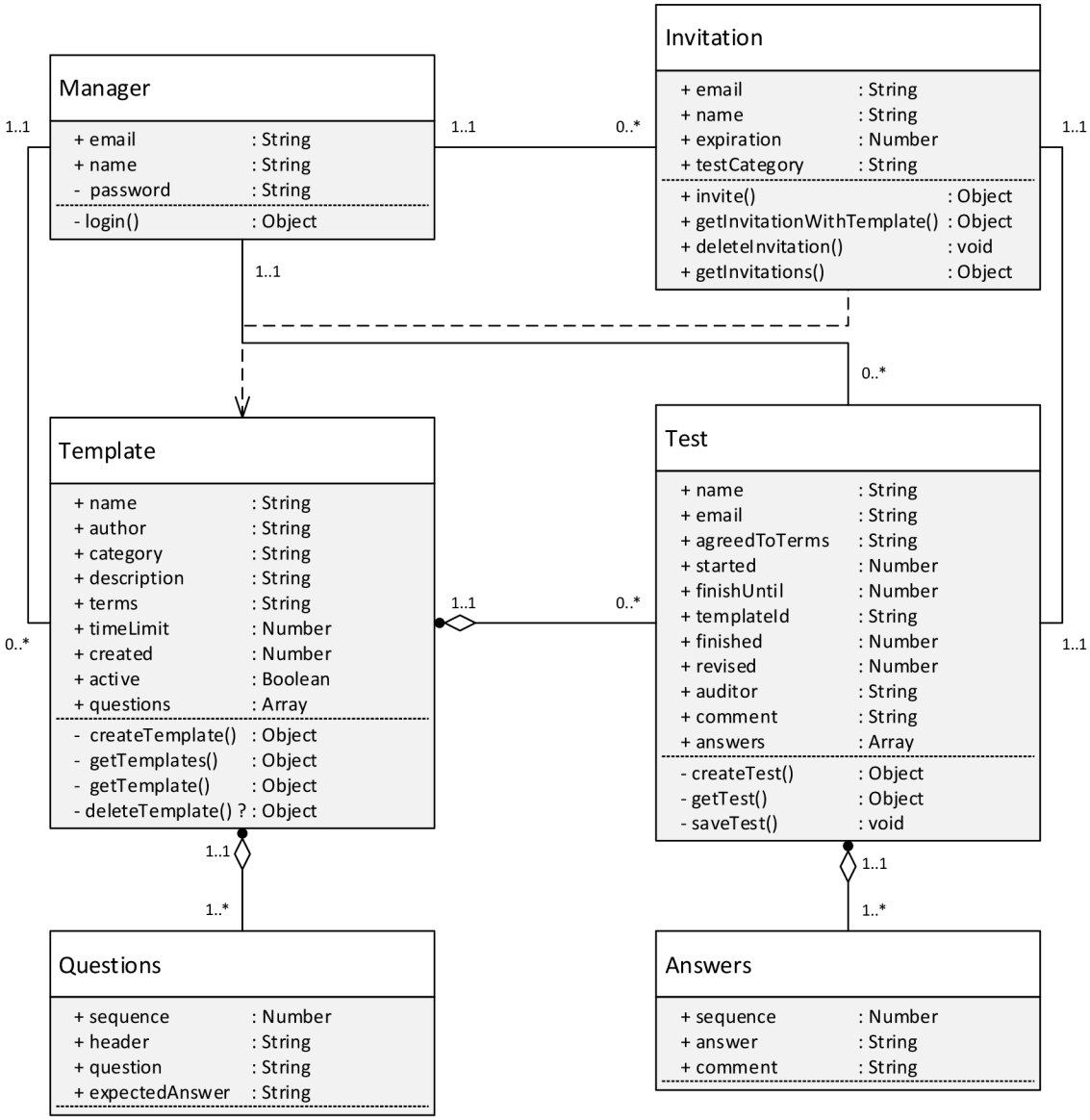
Z výše uvedené tabulky vyplývá, že objekty Questions a Answers nemají na sebe navázané žádné operace, a tak k těmto objektům bude přistupováno nepřímo, a to pomocí objektů Template a Test.

Tabulka 5: Vztahy (Zdroj: Vlastní zpracování)

Objekt	Multiplicita	Vztah	Multiplicita	Objekt
Manager	1..1	spravuje	0..*	Invitation
Manager	1..1	spravuje	0..*	Template
Manager	1..1	spravuje	0..*	Test
Invitation	1..1	vytváří	1..1	Test
Test	1..1	má	1..*	Answers
Template	1..1	popisuje	0..*	Test
Template	1..1	má	1..*	Questions

Z tabulky uvedené výše lze vyčíst jednotlivé vztahy objektů a jejich kardinalitu. Tabulku lze slovně číst jako: „právě jeden manažer spravuje (vytváří, modifikuje, maže) 0 nebo více“. Upřesnění, kandidát existuje v objektu Invitation do vyplnění testu a v objektu Test při vyplňování a po vyplnění testu.

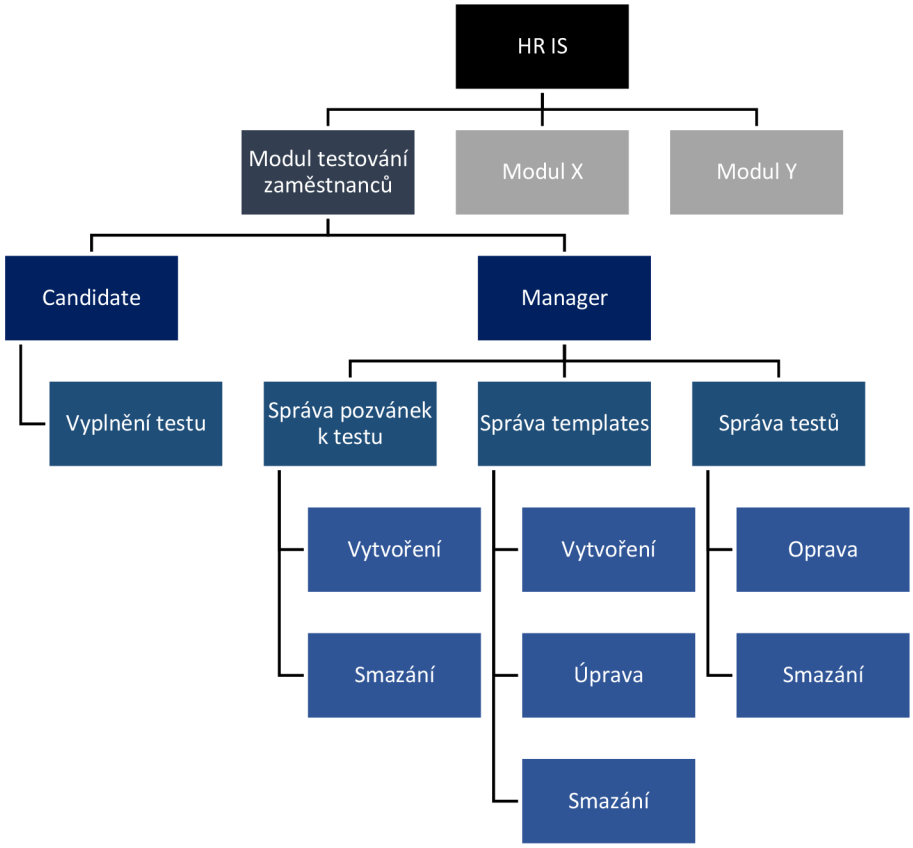
Na následující straně je uveden UML diagram, kde jsou vyobrazeny třídy, operace a vztahy mezi třídami.



Obrázek 19: UML diagram databáze (Zdroj: Vlastní zpracování)

3.6.2 Funkční modelování

Ve funkčním modelování je kladen důraz na to, jaké činnosti v aplikaci probíhají a jak se zpracovávají data. V části popisů činností a toků je kladen důraz na využití diagramů pro větší přehlednost a jasnost.

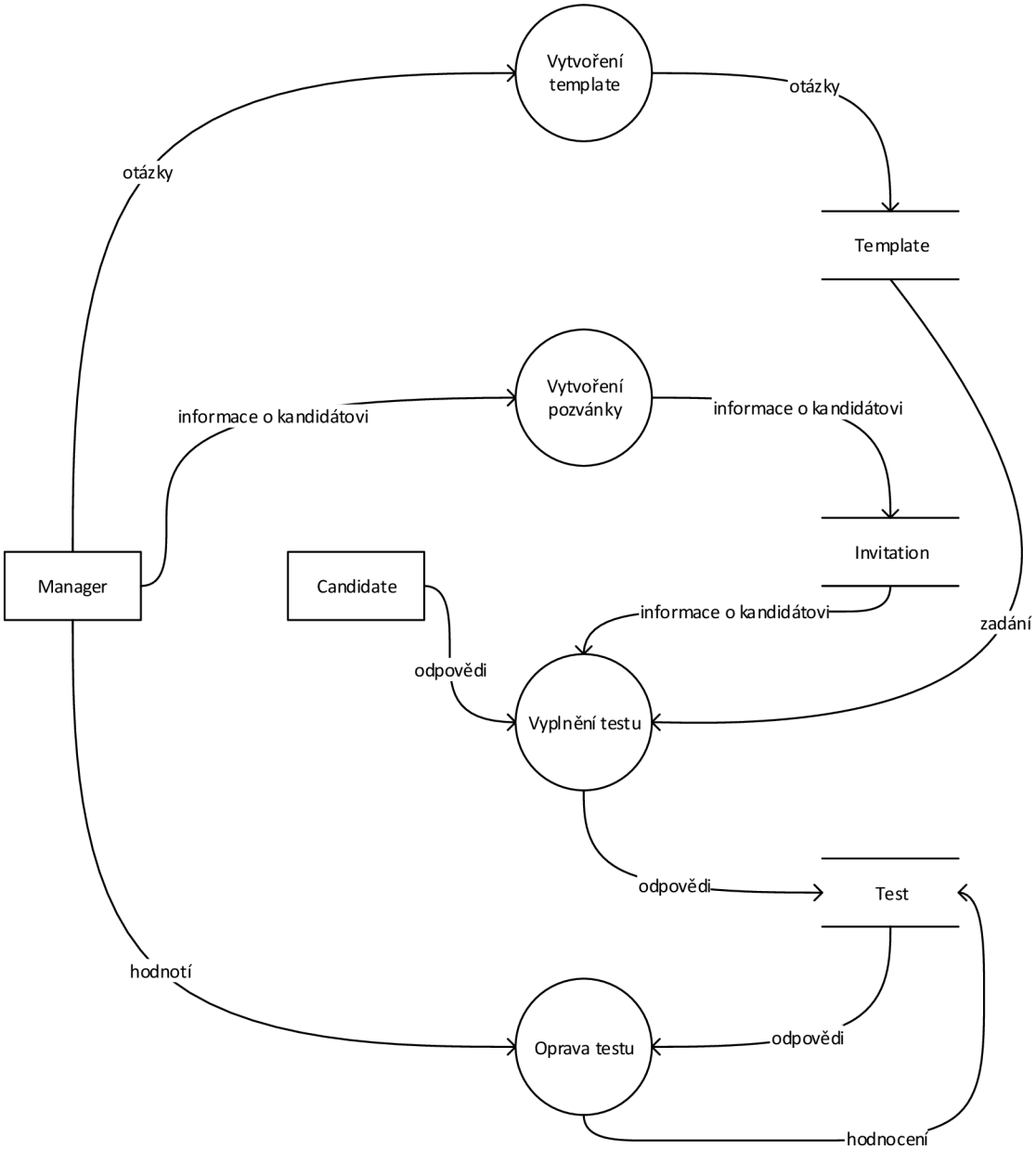


Obrázek 20: Dekompozice funkčního modelu (Zdroj: Vlastní zpracování)

Jednotlivé toky v aplikaci jsou nastíněny dříve. Díky dekompozici funkčního modelu zjišťujeme, kdo bude v aplikaci vykonávat dané toky. Kandidát (uchazeč) se bude starat a bude vidět pouze část plnění testu a manažer bude administrátor aplikace, který bude spravovat pozvánky – pozívání nových uchazečů a mazání existujících; spravovat šablony – tvorba nových šablon, úprava existujících (v tomto případě se vytváří nová šablona, pokud na existující šablonu je navázaný test) a mazání šablon (pouze v případě, že šablona není propojena s žádným testem). Jako poslední část má na starosti správu testů a tím je myšlena oprava vyplněných testů a případné smazání.

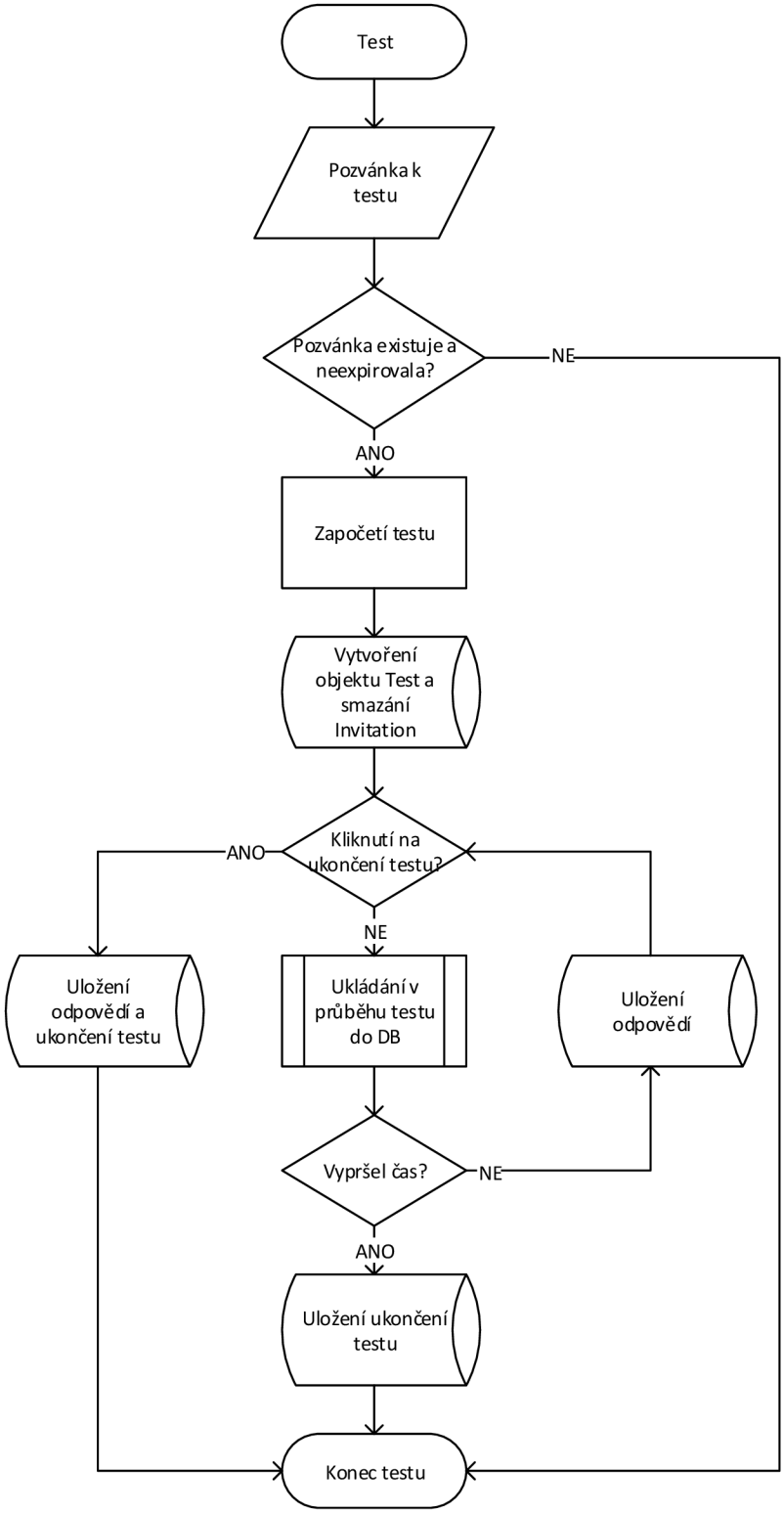
Pro upřesnění toku dat je uveden na další straně DFD diagram, ze kterého lze zjistit zdroje dat a hlavní činnosti v aplikaci, a to vytvoření šablony, pozvání k testu, vyplnění testu a následná oprava testu.

V DFD diagramu jsou uvedeny zdroje dat, a to manažer a kandidát. Manažer a kandidát jsou tedy jediní uživatelé aplikace a tedy jediní, kteří manipulují s daty. K objektům (třídám) Template, Invitation a Test se přistupuje pomocí již zmíněných funkcí v Tabulka 4.



Obrázek 21: DFD Diagram 0. úrovně (Zdroj: Vlastní zpracování)

Prvotním tokem dat v aplikaci musí být přidání šablony manažerem, pouze tak může následně kandidáta pozvat. Pokud je pozvánka využita, data v ní uložené jsou uloženy do objektu Test a samotná pozvánka je smazána. K Testu jsou dále připojeny odpovědi kandidáta a po vyplnění a odeslání k testu přistupuje manažer. Ten provádí kontrolu odpovědí zadaných kandidátem a komentáře k odpovědím jsou uloženy do objektu Test a případně i komentáře ke každé otázce. Je třeba zdůraznit, že kandidát nemá přístup k očekávaným odpovědím testu a zároveň manažer nemá možnost modifikovat odpovědi uložené kandidátem – pouze k nim může přidávat komentář.

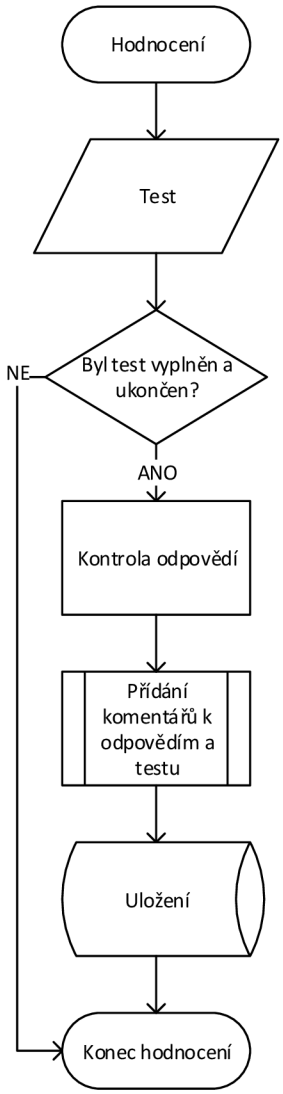


Obrázek 22: Vývojový diagram vyplnění testu (Zdroj: Vlastní zpracování)

Vývojový diagram se týká samotného vyplnění testu kandidátem. Poté co kandidát obdrží a použije URL adresu pro vyplnění testu, která obsahuje platný id pozvánky, dostává se na stránku se základními informacemi o testu, podmínkách a době

trvání. Po odsouhlasení podmínek se uloží souhlas a informace o kandidátovi do objektu Test a pozvánka se smaže, aby se zamezilo druhému použití. V průběhu plnění testu se odpovědi automaticky ukládají a zároveň se provádí kontrola, zda již nevypršel čas pro plnění testu – pokud ano, odpovědi se již neukládají a test se ukončí. V případě, že kandidát ukončí čas před vypršením limitu, je test uložen s veškerými změnami a ukončen pro zamezení dalších změn. Po ukončení testu následuje kontrola odpovědí manažerem.

Kontrola testu manažerem je jednoduchý proces. Před kontrolou testu systém ověří, zda test již byl vyplněn a pokud ano, manažer získává přístup k odpovědím a může přidávat komentáře. Vzhledem k požadavkům, zde není možnost test hodnotit jednoznačně prospěl/neprospěl – takto se hodnotí až v jiném modulu HR IS.



Obrázek 23: Vývojový diagram kontroly testu (Zdroj: Vlastní zpracování)

3.7 Backend

Jedná se o jádro aplikace, které propojuje uživatelské rozhraní s databází, řídí přístupy a poskytuje různou funkcionalitu.

Nejdříve je nutno inicializovat samotný server, poté vytvořit databázové modely podle již zmíněných tříd, dále vytvořit API s autorizací, přes které se bude přistupovat k datovým zdrojům a jako poslední je potřeba vytvořit funkce, které se budou volat, pokud je poslán požadavek na danou cestu.

3.7.1 Tvorba serveru

Příkazem *npm init* vytvoříme soubor *package.json*, kde se uchovávají informace o projektu, skripty a jeho hlavní závislosti na knihovnách. Tyto knihovny se nainstalují pomocí příkazu *npm install –save <knihovny>* a v projektu jsou využívané následující knihovny:

- *babel-cli a babel-present-env* – pro překlad ES6 na ES5,
- *body-parser* automaticky vytváří json objekty,
- *express* vytváří server,
- *jwt-simple, passport a passport-jwt* slouží pro autentizaci a autorizaci,
- *mongoose* zabezpečuje komunikaci s databází,
- *nodemon* je pomůcka při vývoji, která restartuje aplikaci při změně kódu,
- *swagger-ui-express* slouží pro zjednodušení dokumentace API.

```
{
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon index.js",
    "start": "nodemon --exec npm run babel-node -- ./index.js",
    "babel-node": "babel-node"
  },
  "dependencies": {
    "babel-cli": "^6.26.0",
    "express": "^4.16.3"
  }
}
```

Kód 17: Ukázka package.json (Zdroj: Vlastní zpracování)

Nastavení knihovny express je popsáno v teoretické části práci, proto jsou zde uvedeny ostatní nastavení, jako připojení k databázi MongoDB, kterou je potřeba mít nainstalovanou a nastavenou, aby poslouchala na výchozím portu 27017. Propojení databáze se serverem lze učinit jednoduchým vytvořením *promise* (příslib) a připojením v tomto případě k lokální databázi s názvem *candidates*. Vytvoření *promise* je důležité, vzhledem k tomu, že dotazování se na databázi probíhá asynchronně.

```
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/candidates');
```

Kód 18: Připojení databáze (Zdroj: Vlastní zpracování)

Swagger slouží k dokumentaci API, proto je vhodné v tento moment propojit Swagger knihovnu s aplikací a dokumentaci psát souběžně s postupným vytvářením API. Tato dokumentace se nachází v externím souboru, v tomto případě ve stejné složce s názvem *swagger.json* k uživatelskému rozhraní dokumentace lze přistupovat z webového prohlížeče na adrese *localhost:3090/swagger*.

```
import swaggerDocument from "./swagger.json";
app.use('/swagger', serve, setup(swaggerDocument));
```

Kód 19: Swagger nastavení (Zdroj: Vlastní zpracování)

3.7.2 Datové modely

Popis datových modelů je uložen přímo v repositáři se serverem, vzhledem k tomu, že u objektově orientované databáze se v každém objektu je i definice dat. Za tímto účelem vzniká v repositáři 5 souborů – modelů – *category.js, invitation.js, manager.js, template.js* a *test.js*.

Na další stránce je příklad invitation.js. První dva řádky jsou importy, *mongoose* je knihovna, a *testCategoryEnum* je *enum*, kde jsou obsáhnuté možné kategorie testů. Dále je definovaná proměnná *invitationSchema* – schéma, které popisuje strukturu a požadavky ukládaných dat. Toto schéma je využito v následující proměnné *ModelClass* – třída, která vytváří model s názvem *invitation* a připojuje k ní již zmíněnou proměnnou. Jako poslední řádek je export default *ModelClass*. Díky tomuto řádku budeme moci z importovaného souboru invitation.js přistupovat k modelu a díky němu vytvářet dotazy na MongoDB. Podobně jako na další straně jsou vytvořené ostatní modely.

```
import mongoose, { Schema } from 'mongoose';
import { testCategoryEnum } from '../props';

const invitationSchema = new Schema({
    email: {
        type: String,
        unique: true,
        lowercase: true
    },
    name: String,
    expiration: Number,
    testCategory: {
        type: String,
        enum: testCategoryEnum
    }
});

const ModelClass = mongoose.model('invitation', invitationSchema);
export default ModelClass;
```

Kód 20: Ukázka modelu invitation.js (Zdroj: Vlastní zpracování)

3.7.3 API

Komunikace nejenom s UI, ale i s ostatními moduly aplikace bude probíhat pomocí REST požadavků a URI ke zdrojům jsou uvedeny v následující tabulce.

Z tabulky vyplývá, že kandidát přistupuje k datům pomocí GET metody, která využívá operaci *getInvitationWithTemplate()* a právě díky tomuto API se uživateli následně v UI zobrazují informace o testu. Dále má přístupnou API POST nad cestou /test. Tato akce nastane, pokud uživatel odsouhlasí podmínky a ještě než začne s vyplňováním testu, zavolá se operace *createTest()*, která vytváří objekt Test a maže Invitation z databáze. Kandidát následně získává token, kterým je platný pouze po dobu vyplňování testu a skládá se z Test ID. Pomocí tohoto tokenu se posílají požadavky PUT na stejnou cestu a z tokenu se na serveru získává Test ID a zároveň se ujišťuje, že je kandidát ověřený. Po uplynutí časového limitu, nebo v případě odeslání testu se do databáze ukládá čas ukončení a následné změny již nejsou možné. Tímto krokem končí tento proces pro kandidáta.

Tabulka 6: API (Zdroj: Vlastní zpracování)

zdroj		auth	metoda	cesta	popis
Candidate	**Test**				
			GET	/invitation/:id	získání pozvánky podle id
			POST	/test	vytvoření testu, smazání pozvánky
		x	PUT	/test	úprava testu (token složen z testId)
Manager	**Login**				
			POST	/manager	přihlášení manažera
	Invitations				
		x	GET	/manager/invitation	získání všech pozvánek
		x	POST	/manager/invitation	vytovření povánky
		x	DELETE	/manager/:id	smazání pozvánky podle id
	Templates				
		x	GET	/manager/template	získání všech šablon
		x	POST	/manager/template	vytovření šablony
		x	GET	/manager/template/:id	získání šablony podle id s otázkami
		x	DELETE	/manager/template/:id	smazání šablony podle id
	Tests				
		x	GET	/manager/test/:id	získání testu podle id
		x	PUT	/manager/test/:id	oprava testu podle id
		x	DELETE	/manager/test/:id	smazání testu podle id
		x	GET	/manager/test	získání všech testů

Po takto zdokumentovaném API lze přidat soubor *router.js*, který bude obsahovat cesty, autentizační middleware i volání funkce, která zpracuje požadavek a vhodně odpoví. Vzhledem k tomu, že ve webové aplikaci jsou 2 uživatelé (Candidate a Manager), budou vytvořeny routery 2 s odlišnou autentizací – kandidát se autorizuje pomocí Test ID, který získává z pozvánky a manažer se autorizuje emailem a heslem.

Na následující stránce je ukázka routeru candidates.js. Jako první dva importy jsou knihovna *passport* a služba, middleware zajišťující autentizaci. Další importy jsou funkce, které sice existují, ale neobsahují žádnou funkcionalitu. Tyto funkce vycházejí z Tabulka 4 z Datového modelování. Ostatní funkce jsou využity v routeru manager.js. Konstanta *reqAuth* je již zmíněný middleware povolující přístup pouze autorizovaným. Objekt sloužící k nastavení knihovny *passport*, kde je uvedeno *session: false* značí, že autorizace nebude vykonávána pomocí session, ale pomocí tokenu. V poslední části je proveden hromadný export funkcí, které se starají o samotné routování.

```
import passport from 'passport';
import passportService from '../services/passport';
import { getInvitationWithTemplate } from '../controllers/invitation';
import { createTest, saveTest } from '../controllers/test';

const reqAuth = passport.authenticate('candidate', { session: false });

export default (app) => {
    app.get('/invitation/:id', getInvitationWithTemplate);
    app.post('/test', createTest);
    app.put('/test', reqAuth, saveTest);
}
```

Kód 21: Router candidate.js (Zdroj: Vlastní zpracování)

Z příkladu výše si lze všimnout, že jediné nastavení knihovny *passport* nespočívá v uvedení, zdali používat session, či nikoli. Proto je na následující ukázce uveden příklad autentizace kandidáta. Import *passport* jsem již zmínil a knihovna *passport-jwt* (*json web token*) slouží pro extrakci autorizační části (tokenu) v hlavičce zprávy. Další import – *secret*, je tajemství uložené na serveru, kterým se de/šifrují data z tokenu. Z dalších dvou importů lze vidět, že je využíván jeden soubor (passport.js) pro uložení rozdílných pravidel pro Manager i Candidate, jelikož jsou do tohoto souboru importovány datové modely Manager a Test. V ukázce na další straně je pouze Candidate. Konstanta *jwtOptions*, jak již název napovídá, je nastavení, kde má middleware hledat token a jaké je tajemství pro dešifrování. V poslední části ukázky je již samotné využití knihovny *passport*, kde se middleware již postaral o dešifrování tokenu a ověřuje se, zda daný test v databázi existuje (funkce *Test.findById()*). Pokud nastane chyba mezi serverem a databází, funkce vrací chybu. Pokud byl nalezen daný test, je předán dále do funkce, která vyžadovala autentizaci, v opačném případě funkce vrací *false*, což znamená, že test s daným ID nebyl naleze a je vrácen HTTP status *401 Unauthorized*.

Autorizace a autentizace manažera je na podobný způsob s tím, že jediné API, kde není požadován token k přístupu je API sloužící k přihlášení manažera. Token se dá jednoduše využít i pro požadavky mimo aplikaci – v systému HR IS lze uložit přihlašovací údaje manažera a v případě potřeby nové pozvánky by stačilo pouze zadat informace o kandidátovi a pomocí API by byl vygenerován token a následně samotný Invitation ID.

```
import passport from 'passport';
import { ExtractJwt, Strategy } from 'passport-jwt';

import { secret } from '../props';
import Test from '../models/test';
import Manager from '../models/manager';

const jwtOptions = {
    jwtFromRequest: ExtractJwt.fromHeader('authorization'),
    secretOrKey: secret
};

passport.use('candidate', new Strategy(jwtOptions, (payload, done) => {
    Test.findById(payload.sub, (err, test) => {
        if (err) { return done(err, false); }

        if (test) {
            done(null, test);
        } else {
            done(null, false);
        }
    });
}));
```

Kód 22: Služba passport.js (Zdroj: Vlastní zpracování)

3.7.4 Kontrolory

Kontrolory jsou soubory s funkcemi a jejich definicemi obsluhující router. V tomto případě se jedná o 4 kontrolory, a to stejné jako datové modely. V každém souboru jsou funkce, které byly identifikovány při datovém modelování a zajišťují logiku serveru. Jako první příklad je uvedená jednoduchá funkce *getInvitations()*, která má za úkol vyhledat v databázi veškeré pozvánky a poslat je zpět v odpovědi. Tuto funkci lze využít pouze po autorizaci a je určena pouze pro manažera, což vyplývá z Tabulka 4.

```
export function getInvitations(req, res, next) {
    Invitation.find({}, (err, invitations) => {
        if (err) { return next(err); }

        if (invitations.length) {
            return res.json(invitations);
        } else {
            return res.status(204).send();
        }
    });
}
```

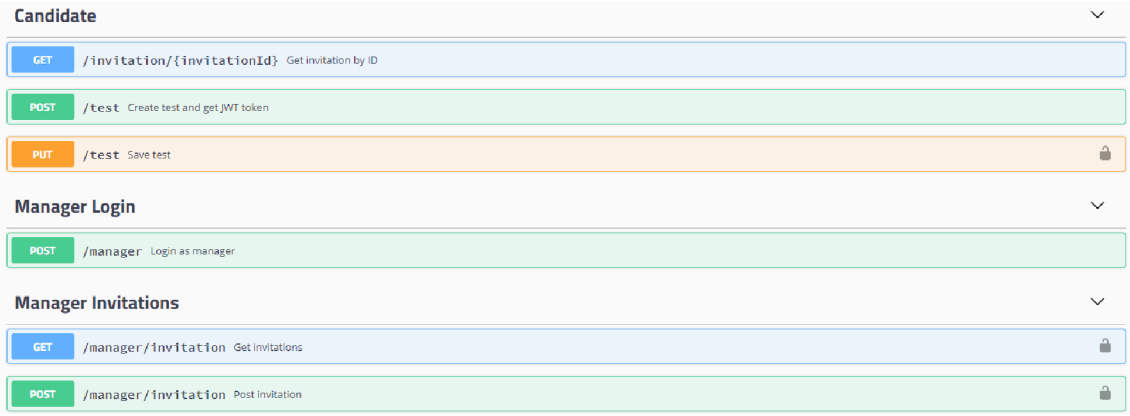
Kód 23: Funkce getInvitations() (Zdroj: Vlastní zpracování)

Následující ukázka funkce *saveTest()* ukládá test vyplněný kandidátem. Jak již bylo zmíněno při autentizaci (Kód 22), při autorizovaném požadavku jsme již získali objekt test z databáze (*req.user*). Dále jsou prováděny kontroly, pokud test již nebyl ukončen, případně pokud již vypršel čas. V případě, že čas vypršel, je dobré si všimnout, že ukládání do databáze (*test.save()*) je stále asynchronní volání, ale díky vybraným slovům *async, await* se asynchronní volání stává synchronním. Pokud nebyly nalezeny problémy, test je uložen spolu s novými odpověďmi a je zaslán HTTP status *204 No content*.

```
export async function saveTest(req, res, next) {
    const test = req.user;
    const { answers, finished } = req.body;
    const timestamp = Date.now();

    if (test.finished) {
        return res.status(403).send({ error: "Already finished." });
    }

    if (timestamp >= test.finishUntil) {
        test.finished = timestamp;
        let newTest = await test.save(err => {
            if (err) { return next(err); }
        });

        return res.status(403).send({ error: "Expired." });
    }

    if (finished) {
        test.finished = timestamp;
    }

    test.answers = answers;
    test.save((err, test) => {
        if (err) { return next(err); }

        return res.status(204).send();
    });
}
```

Kód 24: Funkce saveTest() (Zdroj: Vlastní zpracování)

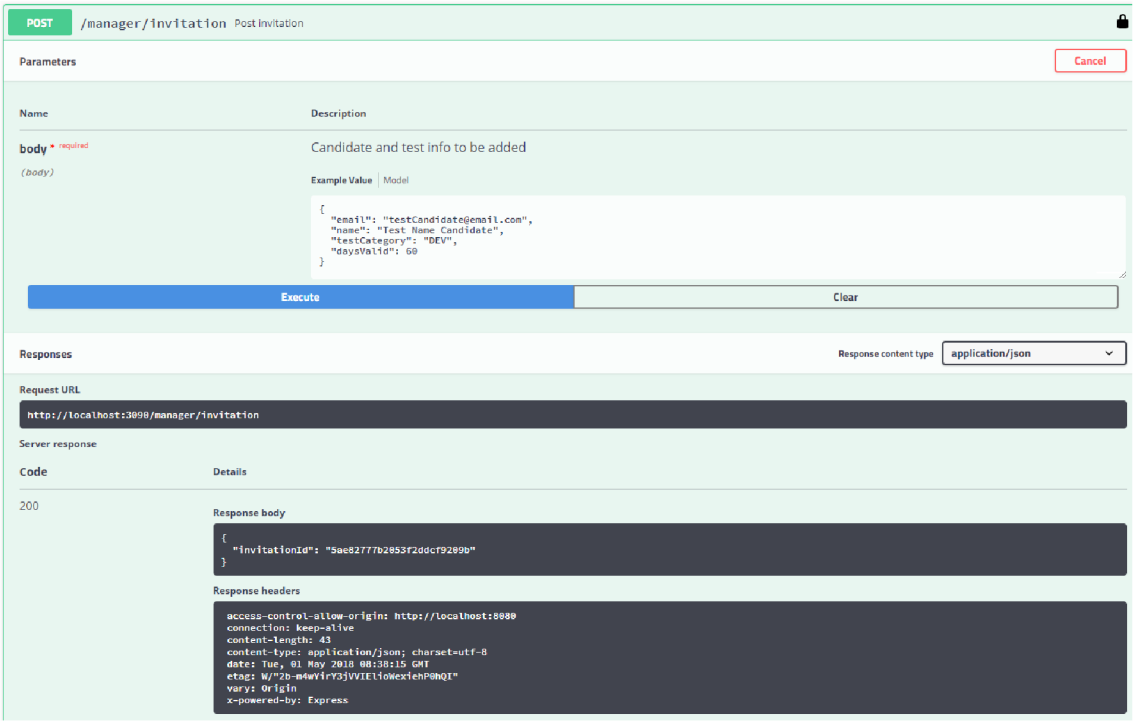
Jak již bylo zmíněno, je potřeba vytvářet k API dokumentaci – ke snazšímu propojení jiných aplikací a pro případné změny. Tato dokumentace je popsána v souboru *swagger.json*. Po popsání jednotlivých API a přejití ve webovém prohlížeči na Swagger, lze vidět uživatelské rozhraní, kde jsou zdokumentované jednotlivé API.



Obrázek 24: Swagger UI (Zdroj: Vlastní zpracování)

Po rozkliknutí jakéhokoli API dostáváme možnost požadavek reálně vyzkoušet. Níže je uveden požadavek na vytvoření pozvánky pro kandidáta. Pokud tento testovací kandidát využije Invitation ID do 60 dnů od pozvání, může vyplnit test.

Veškerá tato dokumentace na produkčních serverech bývá k dispozici pouze uvnitř firemní sítě, což uvádím i jako doporučení při nasazení aplikace na server. Stejné doporučení dávám i pro API týkající se manažera. I přes to, že se manažer musí do aplikace přihlásit před vykonání jakékoli akce, je bezpečnější tuto část povolit pouze v uvnitř firemní sítě.



Obrázek 25: Swagger požadavek (Zdroj: Vlastní zpracování)

3.8 UI a stylování aplikace

Uživatelské rozhraní aplikace je ta část, kterou vidí uživatel ve svém prohlížeči. Jedná se o HTML, který je základním stavebním prvkem UI, dále JavaScript, který přidává funkcionalitu a CSS stylujíc UI.

Při vývoji uživatelského rozhraní je potřeba si opět nastavit úložiště – v tomto případě dvě úložiště, protože UI pro kandidáta má velmi málo společných prvků s UI pro manažera. Takto lze jednodušeji přidávat změny, bez ovlivnění druhé strany. Dále je postup při vývoji určen podle jednotlivých toku aktivit v aplikaci – je nutné, aby se manažer přihlásil, než bude vytvářet šablony atd. Adresář webové aplikace má následující strukturu.

Tabulka 7: Struktura webové aplikace (Zdroj: Vlastní zpracování)

složka / soubor	popis
node_modules/	nainstalované knihovny
src/actions/	akce zabývající se získáním a posíláním dat do úložiště
src/components/	jednotlivé komponenty aplikace
src/reducers/	úložiště stavu
src/index.js	centrální konfigurační soubor
style/	styly
.babelrc	nastavení knihovny babel
index.html	hlavní html dokument pro render komponent
package-lock.json	veškeré závislosti
package.json	obecné nastavení a závislosti
webpack.config.js	nastavení knihovny webpack

Podobně jako u serveru, je nutné vytvořit *package.json* a následně nainstalovat následující knihovny:

- *babel-core*,
- *babel-present-env*,
- *axios* slouží pro posílání požadavků na server,
- *react* je základním prvkem pro tvorbu UI,
- *react-dom* pro práci s HTML DOM,
- *react-redux* slouží k propojení funkcionalit z knihoven *react* a *redux*,
- *react-router* je využit při navigování mezi cestami stránky,
- *redux* úložiště stavu,

- *redux-form* pro tvorbu formulářů,
- *redux-thunk* je middleware umožňující u vytváření akcí předávání funkce namísto akcí,
- *webpack, webpack-dev-server* umožňuje vytvoření jednoho souboru JS souboru (např. *bundle.js*), který je využit pro běh aplikace v prohlížeči.

3.8.1 Obecná nastavení

Nejdříve je nutné uvést obecná nastavení webové aplikace.

Pro správnou funkčnost je potřeba knihovnu *webpack* správně propojit s naší budoucí aplikací. Je zapotřebí vytvořit soubor *webpack.config.js*, kde jsou uložena nastavení. Toto nastavení je přímočaré, *entry* udává, kde je základní blok naší aplikace, který vše propojuje, output určuje, jak má vypadat již vytvořený balíček našeho JS kódu. *Module.loaders* definuje využívaný kompilátor – *babel* a dále soubory, které se kompilovat nemají – *node_modules* je složka, kde jsou uložené veškeré knihovny a jejich závislosti. Jako poslední nastavení je *resolve*, které udává soubory, s jakou příponou se mají kompilovat.

```
module.exports = {
  entry: [
    './src/index.js'
  ],
  output: {
    path: __dirname,
    publicPath: '/',
    filename: 'bundle.js'
  },
  module: {
    loaders: [{
      exclude: /node_modules/,
      loader: 'babel',
      query: {
        presets: ['react', 'env']
      }
    }]
  },
  resolve: {
    extensions: ['', '.js', '.jsx']
  }
};
```

Kód 25: Ukázka webpack.config.js (Zdroj: Vlastní zpracování)

Již zmíněný *./src/index.js* obsahuje middleware a sdružuje všechny ostatní soubory. Jedná se o podobný *index.js*, jako u serveru, s tím, že odkazování na funkce bylo prováděno v samostatném souboru. První sekce importů jsou knihovny, které jsem již zmínil na začátku sekce. Druhou částí importů jsou vlastní soubory, které tvoří samotnou aplikaci. V *index.js* se propojuje i middleware s aplikací – *redux-thunk* a v poslední řadě je propojení všech komponent do cest – například pokud se manažer naviguje na cestu /login, zobrazí se mu komponenta (třída) Login, kterou si rozebereme v další části. Poslední řádek, a to *document.querySelector('.container')* znamená, že vše co je výše definováno bude dostupné podle aktuální cesty v elementu, který je v *index.html* a má třídu *container*. Všeobecně lze tedy říct, že *index.js* je centrálním souborem webové aplikace, na který je napojena všechna logika.

```
export const history = createBrowserHistory();

const createStoreWithMiddleware =
applyMiddleware(reduxThunk)(createStore);
const store = createStoreWithMiddleware(reducers);
const token = localStorage.getItem('token');

if (token) {
  store.dispatch({ type: AUTH_USER });
}

ReactDOM.render(
  <Provider store={store}>
    <Router history={history}>
      <App>
        <Route path="/login" component={Login} />
        <Route path="/logout" component={Logout} />
        <Route path="/invites" component={RequireAuth(Invitations)} />
        <Route path="/temps" component={RequireAuth(Templates)} />
        <Route path="/temps/:id" component={RequireAuth(Template)} />
        <Route path="/tests" component={RequireAuth(Tests)} />
        <Route path="/tests/:id" component={RequireAuth(Test)} />
      </App>
    </Router>
  </Provider>
  , document.querySelector('.container'));
```

Kód 26: Ukázka index.js (Zdroj: Vlastní zpracování)

3.8.2 Components (komponenty)

Komponenty se chovají jako JS funkce. Obecně přijímají vstupy – *props* a vrací elementy popisující co se má zobrazit. Ve webové aplikaci jsou hlavní komponenty popsány níže v tabulce. Většina těchto komponent má i subkomponenty, které jsou využity v různých částí aplikace.

Tabulka 8: Hlavní komponenty aplikace (Zdroj: Vlastní zpracování)

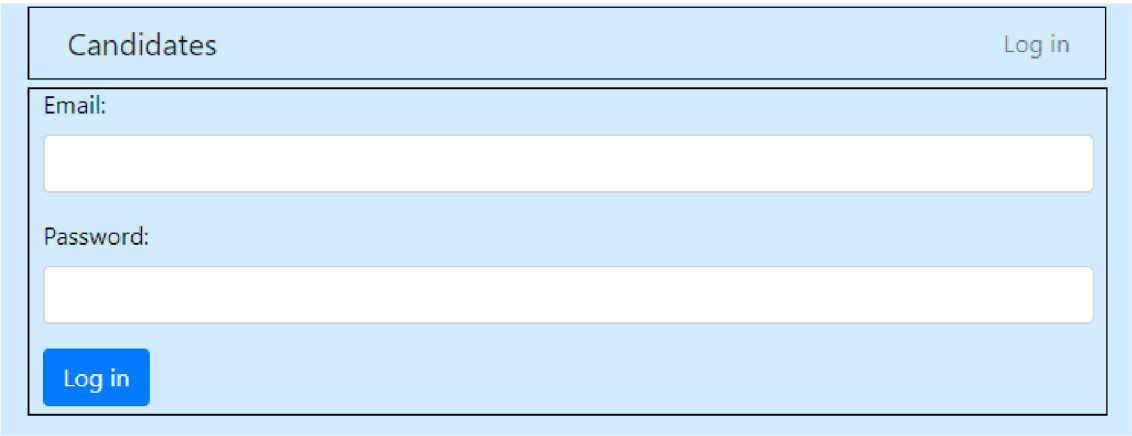
Hlavní komponenty		Popis
Candidate	landing	Před testem, informace a poučení
	test	V průběhu testu, subkomponenty - otázky
	navigation	V průběhu testu, navigace mezi otázky, zbývající čas
Manager	header	Navigace
	login	Přihlášení
	logout	Odhlášení
	invitations	Správa pozvánek - přidání, smazání
	templates	Správa šablon - přidání, úprava, smazání
	tests	Správa testů - oprava, export, smazání

Pro představu je na další straně uveden kód z komponentu login pro manažera. Import knihovny *react-redux* – funkce *connect* propojuje *state* (stav) a *actions* (akce) aplikace se třídou Login. U tohoto propojení si lze všimnou také funkce *reduxForm* z knihovny *redux-form*. Tato knihovna obsahuje funkce a třídy pro zjednodušenou tvorbu formulářů v prostředí v případě, že je využívána knihovna *redux*. Třída Login obsahuje 2 funkce – *handleFormSubmit()* a *render()*. Funkce *handleFormSubmit()* je volána při kliknutí na tlačítko Log in a tato funkce volá funkci ze složky *login* z *actions*. V další kapitole vysvětlím, co daná funkce dělá.

Druhá funkce – *render()* je povinná funkce tříd v *React* prostředí a vrací JSX. Díky tomu, že se jedná o JSX a ne HTML, je k dispozici mnoho funkcionalit. Jestliže chceme využít JS uvnitř JSX, je zapotřebí tento kód obalit do složených závorek. *{this.renderField('email')}* volá funkci *renderField*, kterou jsem pro vyšší přehlednost v příkladu neuvedl. Jedná se o funkci, která rovněž vrací JSX a díky této znovu použitelné funkci se renderují pole *email* a *password*. Funkce *mapStateToProps* má za úkol přiřadit hodnoty z reduktoru do *props* v dané komponentě.

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { Field, reduxForm } from 'redux-form';
import * as actions from '../../actions';

class Login extends Component {
  handleFormSubmit({ email, password }) {
    this.props.login({ email, password });
  }

  render() {
    const { handleSubmit } = this.props;

    return (
      <form onSubmit={handleSubmit(this.handleFormSubmit.bind(this))}>
        {this.renderField('email')}
        {this.renderField('password')}
        {this.renderAlert()}
        <button action="submit" className="btn btn-primary">
          Log in
        </button>
      </form>
    );
  }
}

function mapStateToProps(state) {
  return { errorMessage: state.auth.error };
}

export default reduxForm({
  form: 'login',
})(
  connect(mapStateToProps, actions)(Login)
);
```

Kód 27: Ukázka login.js component (Zdroj: Vlastní zpracování)



Obrázek 26: Komponenty Header a Login (Zdroj: Vlastní zpracování)

Na předchozí straně je uvedena část kódu, díky kterému se renderuje UI rozhraní, které je zobrazeno na Obrázek 26. Tento obrázek obsahuje 2 komponenty – v prvním obdélníku se nachází komponenta *header* a v druhém obdélníku již zmíněná komponenta *login*. Ač se UI rozhraní může tvářit, že je implementace kompletní, je tomu právě naopak. Dále je potřeba implementovat *actions* a *reducers*.

3.8.3 Actions (akce)

Akce obstarávají komunikaci se serverem a díky middleware knihovně *redux-thunk* nakládají s odpověďmi. Každý požadavek na server musí mít svoji akci, u popisování komponent jsem zmínil, že funkce *handleSubmit()* volá funkci *login* z *actions*.

V úryvku kódu je tedy zobrazena funkce login, která pomocí knihovny *axios* zajišťuje komunikaci mezi klientem a serverem. V daném případě se jedná o metodu POST a jsou zasílány přihlašovací data – *email* a *password*. Poté se volání dostává do *promis*, kde mohou nastat 2 situace.

```
export function login({ email, password }) {

  return function (dispatch) {
    axios.post(`${ROOT_URL}/manager`, { email, password })

      .then(response => {
        dispatch({ type: AUTH_USER });
        localStorage.setItem('token', response.data.token);
        history.push('/invitations');
      })

      .catch(error => {
        if (error.response.status === 404) {
          dispatch(authError('Bad Credentials'));
        } else {
          dispatch(authError('Unavailable'))
        }
      });
  }
}

export function authError(error) {
  return {
    type: AUTH_ERROR,
    payload: error
  };
}
```

Kód 28: Ukázka login action (Zdroj: Vlastní zpracování)

První ze situací nastává, když je email a heslo na serveru úspěšně ověřeno a je vrácen token. Tento token se zachytává v *.then* a pomocí něj se klient dále bude autorizovat, a proto je nutné jej uložit – nejvhodnější místo k uložení je *localStorage* (úložiště prohlížeče). Pomocí tohoto tokenu se bude klient autorizovat serveru. Ještě před uložením tokenu do *localStorage* se volá *dispatch* funkce s typem *AUTH_USER* – tato funkce je dále zpracována v reduktoru. V poslední řadě je klient přesměrován z cesty /login na cestu /invitations, kde je renderován jiný komponent.

V případě, že uživatel zadal nesprávnou kombinaci emailu a hesla, je vše zachyceno pomocí *.catch,* kde je volaná opět *dispatch* funkce, ale již s typem *AUTH_ERROR* a chybovou hláškou.

3.8.4 Reducers (reduktory)

Reduktory určují, jak se mění stav ve webové aplikaci na základě akcí poslaných na úložiště. Akce popisují, co se stalo a reduktory určují, jak se změnil stav v aplikaci.

V předchozí podkapitole akce volala funkci *dispatch()* s určitým typem. Tento typ se v reduktoru překládá a ukládá do stavu. V našem případě se typ *AUTH_USER* ukládá do stavu aplikace jako část objektu *state*, s klíčem a hodnotou *authenticated: true*. V opačném případě, kdy uživatel zadal nesprávnou kombinaci emailu a hesla typ *AUTH_ERROR* již ukládá jinou informaci do stavu, a to chybovou hlášku, ke které lze přistupovat z jakékoli komponenty – v tomto případě postačí z komponenty *Login*.

```
import { AUTH_USER, UNAUTH_USER, AUTH_ERROR } from '../actions/types';

export default function(state = {}, action) {
    switch(action.type) {
        case AUTH_USER:
            return { ...state, authenticated: true };
        case UNAUTH_USER:
            return { ...state, authenticated: false };
        case AUTH_ERROR:
            return { ...state, error: action.payload };
    }

    return state;
}
```

Kód 29: Ukázka auth_reducer.js (Zdroj: Vlastní zpracování)

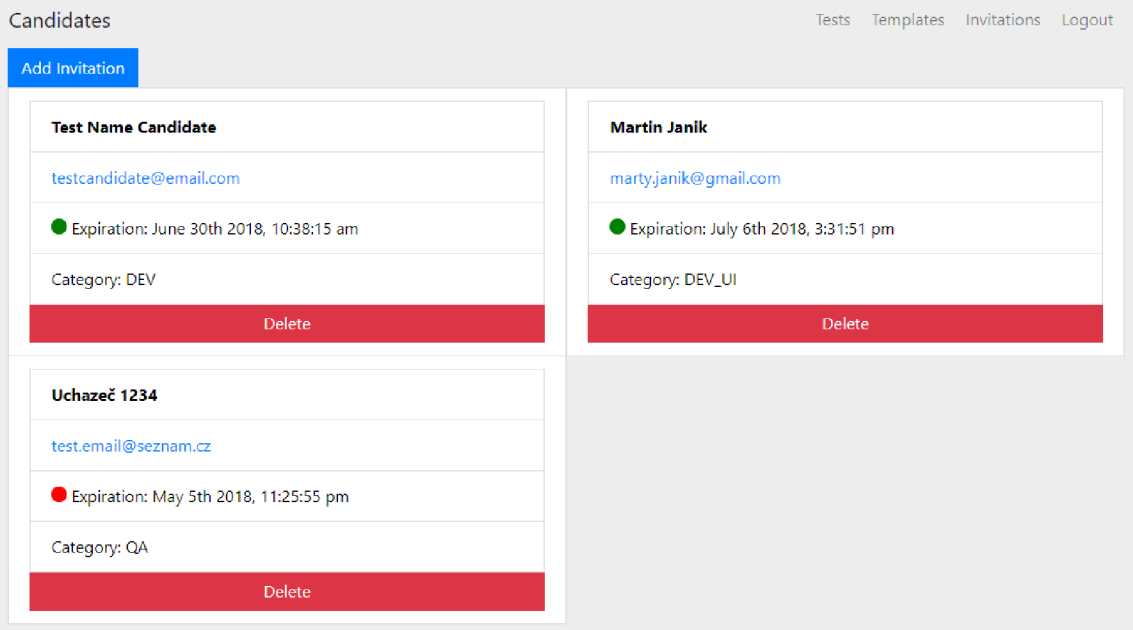
3.8.5 Styl

Pro stylování aplikace byly vybrány volně dostupné technologie Bootstrap a Sass. Díky knihovně Bootstrap je Sass využit pouze minimálně, ale v případě potřeb rozsáhlejšího stylování aplikace lze knihovnu Sass dále využít. V následujícím příkladu je *render* funkce *invitation* komponenty.

```
render() {
  if (!this.props.invitations) {
    return <Link className="btn btn-primary" to="/invitation">
      Add Invitation
    </Link>
  }

  const invitations = this.props.invitations.map(invitation => {
    const { _id, email, name, expiration, testCategory } = invitation;

    const expiredColor = Date.now() > expiration
      ? { backgroundColor: 'red' }
      : { backgroundColor: 'green' };
    const expirationDate = moment(expiration).format('MMMM Do YYYY, h:mm:ss a');

    return (
      <li className="col-md-6 list-group-item" key={_id}>
        <ul className="list-group">
          <li className="list-group-item"><strong>{name}</strong></li>
          <li className="list-group-item link">
            <a href={`mailto:${email}`}>{email}</a>
          </li>
          <li className="list-group-item">
            <span className="circle" style={expiredColor}></span>
            {' '}Expiration: {expirationDate}
          </li>
          <li className="list-group-item">Category: {testCategory}</li>
          <button
            className="btn btn-danger"
            onClick={this.handleDelete.bind(this)}
          >Delete</button>
        </ul>
      </li>
    );
  });

  return (
    <div className="container">
      <Link className="btn btn-primary" to="/invitation">Add Invitation</Link>
      <ul className="list-group d-flex flex-row flex-wrap">
        {invitations}
      </ul>
    </div>
  );
}
```

Kód 30: Příklad render metody invitation.js (Zdroj: Vlastní zpracování)

Komponent *invitation* se spolu s komponentem *header* vykreslují díky Bootstrap třídám a jedné vlastní třídě jako na obrázku na další straně.



Kód 31: Ukázka invitation a header UI (Zdroj: Vlastní zpracování)

3.9 Toky dat v aplikaci

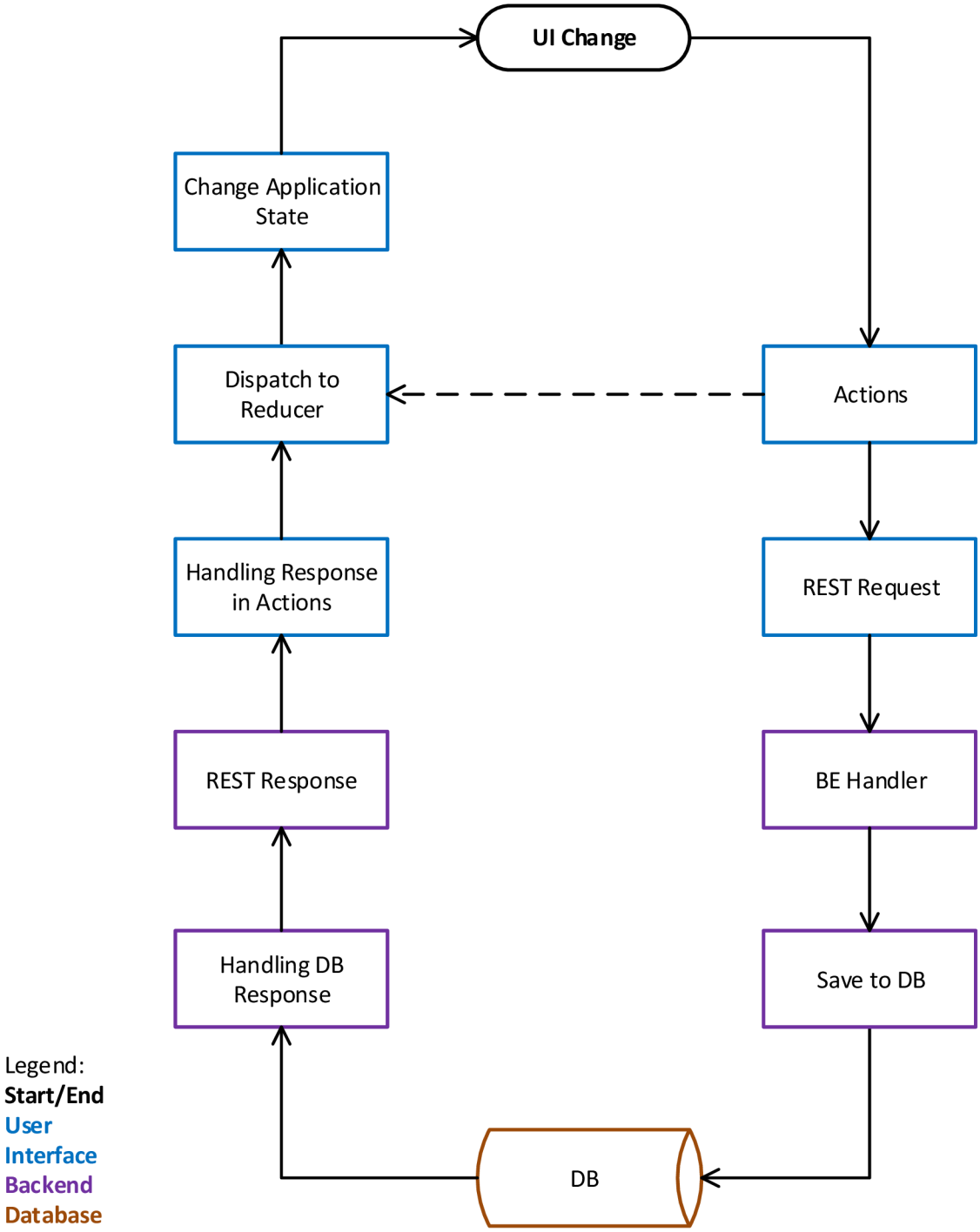
V podkapitole je uveden tok dat v celé aplikaci – od UI až po DB a zpět. Diagram toků dat je uveden na následující straně.

Tok začíná i končí změnou v UI. Uživatel na klientovi vyvolává změnu, například odesláním formuláře. Tato změna se zpracovává v *Actions*, kde pomocí externí knihovny je vytvořen REST požadavek, v našem příkladu se jedná o HTTP POST.

Tento požadavek se dále zpracovává na serveru, kde probíhá také hlavní validace dat. Po úspěšné validaci se data ukládají do objektově-orientované databáze. Při úspěšném uložení se vytváří odpověď, která se odesílá na klienta.

Klient stále v *Actions* odpověď přijímá a zpracovává. Při zpracování volá funkci *dispatch*, která zajišťuje předání stavu do *Reducer*. V *Reducer* se následně mění stav celé aplikace a díky této změně se mění i UI, tedy to, co uživatel vidí.

Pokud by se nejednalo o formulář, ale například pouze o odhlášení uživatele, vynechává se požadavek na server, vzhledem k tomu, že odhlášení uživatele spočívá pouze ve smazání tokenu z *localStorage* a přenastavení stavu aplikace.



Obrázek 27: Diagram toku dat aplikací (Zdroj: Vlastní zpracování)

3.10 Nasazení modulu

Po dokončení vývoje lze nasadit modul na server dedikovaný a nastavený IT oddělením společnosti pro tento modul. Je doporučeno oddělit server od klienta, který pro manažerské potřeby – pozvání, úprava šablon a opravy testů, bude přístupný pouze v interní síti, do které se lze dostat z vnější sítě pouze pomocí firemní VPN a klient pro kandidáta mít v DMZ, kde by se měl nacházet vysoce zabezpečený server, který bude komunikovat se serverem umístěným ve vnitřní síti podniku. Po nasazení vytvořeného modulu HR IS lze již modul využívat, ale je doporučeno dedikovat QA, který pomocí regresního testovaní ověří veškerou funkcionalitu a bezproblémový chod modulu. Vzhledem k rozsahu modulu, časový odhad pro testování nepřesahuje jeden pracovní den, a tak lze využít QA pracujícího pro zákazníka, který se na den omluví a otestuje nový modul HR IS.

Po provedení případného otestování nastává ostrý provoz modulu. Vzhledem k tomu, že se jedná o samostatný modul, který nemusí (ale může) být propojen přímo s HR IS, nezabraňuje nic běhu obou modulů, tedy HackerRank a vlastního od běhu těchto aplikací současně. V tomto případě je doporučen dvojí provoz, kdy bude uchazeč pozván k vyplnění testu ve vlastním modulu a v případě potíží lze využít produkt od HackerRank, pro který v době dvojího nasazení ještě nevypršela licence.

Za bezproblémový chod aplikace lze považovat stav, kdy alespoň tři uchazeči projdou všemi stavy v aplikaci – pozván, vyplněný test a následně opravený test. V případě vzniku potíží při průchodu uchazeče vlastním modulem, lze tyto problémy opravit a mezitím danému uchazeči poslat e-mail s pozvánkou na test od HackerRank. V opačném případě lze produkt od HackerRank již nepoužívat a používat modul vlastní.

Tento modul lze následně pomocí API propojit přímo s HR IS, ze kterého lze jednoduše generovat pozvánky a kontrolovat jejich status, případně také kontrolovat, zda byl test vyplněn a opraven. V modulu lze spravovat pozvánky, včetně zvaní nových uchazečů, tvorbu a úpravu šablon pro testy, které pro již vyplněné testy zůstávají neměnné a v poslední řadě lze provádět kontrolu a opravu již vyplněných testů, jejich export, tisk i mazání.

3.11 Přínosy práce

V rámci návrhové části bylo stanoveno, že vývoj aplikace bude prováděn studentem s podporou zaměstnance (senior developera) společnosti. Reálná doba trvání byla nižší a nebylo potřeba mnoho konzultací se senior developerem.

Tabulka 9: Porovnání odhadu a reálné doby trvání (Zdroj: Vlastní zpracování)

	Doba trvání	Cena za člověkoden	Celkem
Odhad	30 + 3	CZK 1500 + 4000	CZK 57,000
Reálná doba trvání	27 + 1	CZK 1500 + 4000	CZK 44,500

Dále je potřeba uvést tabulku, která porovnává řešení od HackerRank a nové řešení. Roční náklady na běh vlastní aplikace na vlastních serverech jsou minimální, vzhledem k tomu, že společnost má vlastní serverovnu a potřebný hardware k běhu aplikace. Jedná se pouze o náklady na elektřinu a opotřebení hardware, vzhledem k tomu, že v rámci vývoje nebylo potřeba pořizovat licence. V nákladech na provoz nejsou uvedeny náklady, které mohou vzniknout potřebou přidání nových funkcionalit webové aplikace.

Tabulka 10: Porovnání stávajícího a navrhovaného řešení (Zdroj: Vlastní zpracování)

Roční náklady	Licence	Provoz	Celkem
Vlastní webová aplikace	CZK 0	CZK 4,000	CZK 4,000
Licence HackerRank	CZK 100,000	CZK 0	CZK 100,000

Z tabulek vyplývá, že vývoj vlastní aplikace se vyplácí již v prvním roku používání. Existuje ovšem riziko, že mohou vzniknout neočekávané náklady spojené s provozem aplikace, které lze snížit přetestováním a sledováním webové aplikace. Tyto náklady nebyly zahrnuty v ekonomickém hodnocení.

Mezi ostatní přínosy diplomové práce patří podpora procesu náboru zaměstnanců pomocí vlastní aplikace, kterou lze jednodušeji přizpůsobovat aktuálním potřebám a integrovat do HR IS. Tato aplikace je součástí HR IS a nahrazuje produkt společnosti HackerRank, z čehož vyplývá, že HR IS je již nezávislý na licencích a celý informační systém je vlastněný společností.

3.12 Vize do budoucna

V aplikaci byly použity nejnovější technologie, díky kterým lze jednodušeji škálovat a rozvíjet daný modul. Při implementaci byly využité funkcionality z ES 7, které se pomocí knihovny Babel automaticky překládají do starší verze ES 5, který podporuje většina moderních prohlížečů. Vzhledem k tomu, že vývoj v tomto odvětví je rychlý, je možné, že do 2–3 let již nebude potřeba knihovna Babel a moderní prohlížeče již budou podporovat syntaxi ES 7. V případě dalšího rozvoje lze knihovnu ponechat a překládat syntaxi JavaScript na aktuálně podporované verze a díky tomu využívat nově implementovanou syntaxi, které ještě nebude podporována webovými prohlížeči. Iniciátorem změn může například upravení, případně zavedení nové funkcionality, nebo i samotný prodej modulu, či HR IS.

Modul pro testování zaměstnanců byl posledním prvkem, ke kterému byla potřeba licence, a tak se stal celý HR IS nezávislý na licencích. Díky tomuto faktu, lze tento HR IS upravit do podoby, kdy by bylo možné jej nabízet ostatním společnostem působícím v oboru. Je potřeba si však uvědomit, že před uskutečněním těchto kroků je zapotřebí provést analýzy, zda je pro takovýto produkt na trhu místo a dále analýzu využívaných technologií, zda je možné knihovny a použité technologie bezplatně distribuovat a prodávat.

ZÁVĚR

Cílem diplomové práce byl návrh a vývoj webové aplikace pro firmu XYZ s. r. o. sloužící k testování znalostí uchazečů před pohovorem. Z dotazníkové metody vyplynulo, že se firma snažila najít možnosti snížení nákladů. Pro potřeby práce a snížení nákladů bylo vybráno odstranění placených licencí v informačním systému a tento modul nahradit modulem vlastním. V analýze je popsána současná situace a propojenost placeného modulu s HR IS.

Ve stěžejní části práce – vlastní návrh řešení, je postup návrhu a vývoje aplikace. Nejdříve bylo nutné rozhodnutí, kdo a za jaké náklady bude modul vytvářet, z čehož vzešla nejvýhodněji varianta student s pomocí zaměstnance společnosti. Po návrhu architektury aplikace se bylo možné přesunout k datovému a funkčnímu modelování, kde byly identifikovány třídy, objekty a jejich propojení. V další části je již popsána samotná implementace systému, od vytvoření datových modelů až po konečnou úpravu uživatelského rozhraní.

V post implementační části je uvedené doporučení při nastavení serveru, který však bude nastavovat IT oddělení společnosti podle vnitřních pravidel. Dále jsou uvedeny přínosy práce, kterými jsou především snížení nákladů na provoz modulu pro testování znalostí uchazečů a odstranění poslední licencované části v HR IS.

SEZNAM POUŽITÉ LITERATURY

[1] *HackerRank: Sample Test* [online]. [cit. 2018-01-28]. Dostupné z: https://www.hackerrank.com/tests/sample

[2] FLANAGAN, David. *JavaScript: The Definitive Guide, Sixth Edition*. 6th ed. O'Reilly, 2011. ISBN 0596805527.

[3] ELLIOT, Eric. *Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries*. O'Reilly, 2014. ISBN 1491950293.

[4] SODOMKA, Petr a Hana KLČOVÁ. *Informační systémy v podnikové praxi*. 2. aktualiz. a rozš. vyd. Brno: Computer Press, 2010, 501 s. : il., grafy, tab. ISBN 978-80-251-2878-7.

[5] MOLNÁR, Zdeněk. *Efektivnost informačních systémů*. Praha: Grada Publishing, 2000, 142 s. ISBN 80-7169-410-X.

[6] *JavaScript | MDN: JavaScript language resources* [online]. 2018 [cit. 2018-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[7] *ECMAScript: compatibility table* [online]. 2018 [cit. 2018-03-11]. Dostupné z: http://kangax.github.io/compat-table/

[8] *Oracle: A Relational Database Overview* [online]. 2017 [cit. 2018-03-11]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[9] *MongoDB | Documentation: Find or Query Data with the mongo Shell* [online]. 2018 [cit. 2018-03-11]. Dostupné z: https://docs.mongodb.com/getting-started/shell/query/

[10] *MDN web docs: Learn web development* [online]. 2017 [cit. 2018-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn

[11] *React* [online]. 2018 [cit. 2018-03-11]. Dostupné z: https://reactjs.org/docs/

[12] *SASS_REFERENCE* [online]. 2018 [cit. 2018-03-17]. Dostupné z: https://sass-lang.com/documentation/file.SASS_REFERENCE.html

[13] *W3schools: Bootstrap Get Started* [online]. 2018 [cit. 2018-03-17]. Dostupné z: https://www.w3schools.com/bootstrap/bootstrap_get_started.asp

[14] *What Is ReactJS and Why Should We Use It?* [online]. Nitin Pandit, 2017 [cit. 2018-03-17]. Dostupné z: https://www.c-sharpcorner.com/article/what-and-why-reactjs/

[15] *Redux: Read Me* [online]. 2018 [cit. 2018-03-17]. Dostupné z: https://redux.js.org/

[16] *Introducing Redux* [online]. 2017 [cit. 2018-05-12]. Dostupné z: https://www.ibm.com/developerworks/library/wa-manage-state-with-redux-p1-david-geary/index.html

[17] *About Node.js®* [online]. 2018 [cit. 2018-05-12]. Dostupné z: https://nodejs.org/en/about/

[18] *Why Babel Matters* [online]. 2015 [cit. 2018-05-12]. Dostupné z: https://codemix.com/blog/why-babel-matters

[19] *Co je GDPR?* [online]. 2018 [cit. 2018-05-12]. Dostupné z: https://www.gdpr.cz/gdpr/

[20] *What is Swagger and Why it Matters* [online]. 2016 [cit. 2018-05-12]. Dostupné z: https://blog.readme.io/what-is-swagger-and-why-it-matters/

[21] *What is Express.js and Why Does It Matter?* [online]. 2017 [cit. 2018-05-12]. Dostupné z: https://www.programmableweb.com/news/what-expressjs-and-why-does-it-matter/analysis/2017/05/05

SEZNAM OBRÁZKŮ

SEZNAM UKÁZEK KÓDU

SEZNAM TABULEK

SEZNAM ZKRATEK

API	–	Application Programming Interface
BE	–	Backend
CMS	–	Content Management System
CR	–	Code Review
CSS	–	Cascading Stylesheets
CTO	–	Chief Technician Officer
DEV	–	Developer
DMZ	–	Demilitarized Zone
DOM	–	Document Object Model
DPIA	–	Data Protection Impact Assessment
DPO	–	Data Protection Officer
EPC	–	Event-driven Process Chain
ES	–	ECMAScript
GDPR	–	General Data Protection Regulation
HR	–	Human Resources
HTML	–	Hypertext Markup Language
HW	–	Hardware
IS	–	Information System
JS	–	JavaScript
JSON	–	JavaScript Object Notation
LDAP	–	Lightweight Directory Access Protocol
MVC	–	Model – View – Controller
NoSQL	–	Non-SQL
npm	–	Node Package Manager
PO	–	Product Owner
QA	–	Quality Assurance
RDBMS	–	Relational Database Management System
REST	–	Representational State Transfer
Sass	–	Syntactically Awesome Style Sheets
SLA	–	Service-Level Agreement
SM	–	Scrum Master
SQL	–	Structured Query Language
SW	–	Software
UI	–	User Interface
URI	–	Uniform Resource Indicator
VPN	–	Virtual Private Network
XML	–	Extensible Markup Language