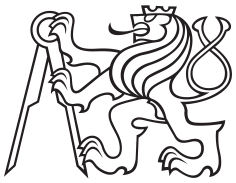
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Systém pro pohovory v IT a jejich přípravu

Adilbek Utemissov

Vedoucí: Ing. Jiří Šebek
Obor: Softwarové inženýrství a technologie
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Utemissov** Jméno: **Adilbek** Osobní číslo: **445375**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro pohovory v IT a jejich přípravu

Název bakalářské práce anglicky:

System for IT interviews and their preparation

Pokyny pro vypracování:

1. Prostudujte existující nástroje pro podporu pohovorů, testů a jejich přípravy v IT (např. AlgoExpert [3])
2. Prostudujte hlavní oblasti, které by uchazeč v IT měl vědet (BE developer ,FE developer , procesní analytik, datový analytik, systémový architekt a jiné)
3. Seznamte se s strukturou pohovorů a zvolte co nejlepší způsob provedení v aplikaci
4. Navrhněte a naimplementujte systém, který:
1. má možnost připravit uchazeče na danou pozici
2. vybere testovací otázky a strukturu při pohovoru a pomůže zkousejícímu (usetri cas, energii)
2. Systém vhodně otestujte (uživatelské testy)
3. Vyhodnoťte testy, výhody a možná omezení řešení

Seznam doporučené literatury:

1. ian sommerville: Software engineering, Global edition, Pearson Higher Ed, 2016, isbn 1292096144
2. SURYOTRISONGKO, Hatma; JAYANTO, Dedy Puji; TJAHYANTO, Aris. Design and development of backend application for public complaint systems using microservice spring boot. Procedia Computer Science, 2017, 124: 736-743.
3. AlgoExpert [online]. 2020. Dostupné z: https:// www.algoexpert.io/product

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych především poděkoval vedoucí práce, Ing. Jiří Šebek, za jeho ochotu a pomoc po celou dobu psaní. Dále bych rád poděkoval celé své rodině za obrovskou podporu, kamarádům, kteří se podíleli na návrhu nebo testování mé aplikace. Díky, že jste mi věnovali svůj čas a poskytli mi zpětnou vazbu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s metodickým návodem o dodržování principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21 května 2021

Abstrakt

Bakalářská práce se zabývá návrhem a implementací systému pro podporu programátorů. Systém je webovou aplikací, která je určena pro provedení technického pohovoru IT společností a připravu k němu programátory. Nabízí firmam flexibilní variantu provedení pohovoru a tím šetřit vlastní čás. Programátoři mají příležitost pro zlepšování přípravy před pohovorem a zároveň zvyšovat své programátorské dovědnosti. Tato práce obsahuje analýzu pohovorů a trhu podobných aplikací, návrh a implementaci systému a nakonec testování dosažených výsledku.

Klíčová slova: Systém pro pohovory v IT, Technická část pohovoru, Příprava k technickému pohovoru, Programátor, Webová aplikace

Vedoucí: Ing. Jiří Šebek

Abstract

The bachelor thesis deals with design and implementation of a system for supporting developers. The system is a web application, which is intended for conducting technical interview in IT company and preparing programmers for it. It offers companies a flexible variant of conducting an interview and thus, saving their own time. Programmers have the opportunity to improve preparation for interview and at the same time increase their programming skills. This work includes analysis of interviews and the market of similar applications, design and implementation of the system and finally, testing the achieved results.

Keywords: System for IT interviews, Technical part of interview, Preparation for a technical interview, Programmer, Web application

Title translation: System for IT interviews and their preparation

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Tato bakalářská práce se zabývá analýzou, návrhem, technickou specifikací a následným vývojem webové aplikace pro podporu programátorů. Její součástí je dokumentace samotné aplikace, která bude sloužit jako souhrn funkcionality, a technická část jako samotná aplikace. Aplikace bude sloužit jako výukový systém(moodle[1]), kde využití najdou hlavně programátoři a zkoušející. Cílem této aplikace je doplnění zkouškových otázek, úloh, pohovorů a příprava programátorů k technickému pohovoru do IT společnosti prostřednictvím imitovaných pohovorů a sbírky teoretických a praktických úloh z technické stránky pohovoru.

1.1 Motivace

V současné době každý člověk používá velké množství různých webových aplikací pro různé účely. Hlavní motivací k tvorbě této práce byla snaha vytvořit užitečný nástroj, který bude v budoucnosti vhodný zejména pro ty, kteří nemají možnost platit za online kurzy programování. Zaprvé bude aplikace využita jako spolehlivý zdroj informací, zadruhé jako vhodný asistent programátorům pro rychlou a komfortní přípravu k technickému pohovoru a společnostem pro provedení tohoto pohovoru. Osobní motivací bylo v první řadě poskytnout užitek lidem a zjednodušit jim život, ve druhé řadě naučit se nové technologie a udělat takovou práci, na kterou bude možné dále navázat.

1.2 Cíl

Prvním cílem je prostudování celého procesu technické části pohovoru a určení klíčových kroků na základě svých vlastních zkušeností. Druhým cílem je analýza již existujících systémů v této oblasti. Třetím cílem je na základě provedené softwarové analýzy navrhnout a implementovat aplikaci.

Kapitola 2
Analýza

V této kapitole je podrobně popsána programátorská oblast. Konkrétně si klade za cíl zjistit, jak probíhá pohovor v IT společnosti, jaké teoretické, praktické otázky a úlohy se objevují na technickém pohovoru u programátorů a obecně poznat, proč je taková aplikace vhodná.

Z výše uvedeného kontextu je pak jednodušší pochopit potřeby, které s tím přicházejí a které je možné správným řešením uspokojit. Navržené řešení je ale nutné si ověřit a prvním krokem po seznámení s problémem je analýza existujících řešení, zabývající se stejnou nebo aspoň podobnou problematikou. Nakonec bude vybráno optimální řešení, tento výběr bude podpořen relevantními argumenty a bude provedena analýza technologií pro vývoj finální verze aplikace.

2.1 Seznámení s problémem

Každý zaměstnanec prochází pohovorem před nástupem do jakékoliv firmy. V oblasti programování se pohovory v IT firmách obvykle skládají ze dvou částí - přijímací a technické. Většinou je technická část nezbytnou součástí pohovoru, kde důležitou roli kromě vlastních kvalit hrají programátorské dovednosti [2][3].

V přijímací části programátor představuje, co studoval nebo nyní studuje, jestli měl předchozí zaměstnání v oblasti IT, co tam dělal a jak všechno probíhalo [4][5]. Povídají si s představitelem o firmě a po ukončení této části firma se rozhoduje o druhém, technickém kole.

V technické části sledují a kontrolují úroveň znalostí programátora v oblasti programování. To může probíhat v různé podobě, buď teoretickými otázkami nebo psaním vlastního kódu dle zadání [6][7]. V nejvíce případech jde programátor na technickou část buď nepřipravený a ztrácí více času, nebo se připravuje na různých webových stránkách na různá témata. Taky může být ještě jeden případ, když např. programátor se učí samostatně mimo vysoké školy nebo pomocí kurzů a během samostatného studia chce upevnit své praktické znalosti a zkontrolovat, jestli správně pochopil probrané témata.

2.2 Analýza existujících aplikací

V tomto bodě je popsány analogické aplikaci, které by mohly být dobrým zdrojem informací vhodných pro vývoj vlastní aplikace. Analogických aplikací je na trhu dost a každá má svá specifika, výhody a nevýhody. Některé aplikace zde budou uvedeny. Aplikace jsou vybrané podle následujících kritérií:

- Existence programátorských úloh pro různé IT pozice
- Nabízení imitovaného pohovoru
- Nabízení výukových programátorských kurzů na různá témata

2.2.1 AlgoExpert

Je to webová aplikace, ve které splněna většina cílů a požadavků pro tento účel a i něco navíc [8]. Bezplatná verze dává k dispozici omezené možnosti, jako například v kategorii AlgoExpert je několik přístupných praktických úloh s plným kompletem funkcí a několik video kurzů. V kategorii SystemsExpert je dostupný jeden teoretický úkol. Obecně, tato aplikace se pokouší nabídnout rozšířenou funkcionalitu programátorům a nese velký počet výhod v rámci účelů tohoto projektu, ale existuje i opačná strana. Jedna z výhod je velký počet programátorských jazyků pro řešení stejných úloh. Dále aplikace nabízí seznam video příruček na různá témata, kde se vysvětluje určité téma. Ze seznamu nevýhod jsou hlavní balíčková funkcionalita a nemožnost přidávat vlastní úlohy v roli zkoušejícího. V základní bezplatné verzi uživatele mají pro použití jen několik příkladů a videonávodů. Příklady otázek z pohovoru, zbytek praktických úloh, videonávodů, procházení imitovaným pohovorem a další jsou až v placené verzi [9]. Ještě jednou nevýhodou této aplikace je absence mobilní aplikace. Jak bylo uvedeno výše, využívání mobilních zařízení pořád roste a mít mobilní aplikace by bylo pro uživatele přívětivé.

Výhody

- Velká sada funkcionalit
- Uspokojí požadavky

Nevýhody

- Placená verze - 99 USD/rok
- Není možnost psaní vlastních úloh
- Pouze verze v angličtině
- Není mobilní aplikace
- Není k dispozici fórum pro diskuze

2.2.2 Leetcode

Je to webová aplikace, která je podobná AlgoExpertu, ale má svou charakteristiku [10]. Tato aplikace také má placenou a bezplatnou verzi, kde bezplatná dává k dispozici stejné omezené možnosti, a navíc k tomu několik imitovaných pohovorů, které a misto video tutorialu detailní, stručný popis témat k úlohám. Hlavní výhody jsou mobilní verze aplikace, příklady procházení imitovanými pohovory, rozdělení otázek a praktických úloh dle velkých firem jako Apple, Microsoft, Facebook a podobné. Taky k výhodám patří účast v soutěži na rychlost a správnost řešení úloh s jinými uživateli, větší počet možností v bezplatné verze než v AlgoExpertu pro prohlédnutí, možnost diskutovat s jinými uživateli o úloze a jejím řešení, ale pořad zůstává největší nevýhoda, placená verze [11].

Výhody

- Velká sada funkcionalit
- Uspokojí požadavky
- Mobilní aplikace
- Příklady pohovorů určitých firem

Nevýhody

- Placená verze - 35 USD/měsíc, 159 USD/rok
- Není možnost psaní vlastních úloh
- Pouze verze v angličtině

2.2.3 Coderbyte

Je taky webová aplikace, která má víc strukturovanější obsah než předchozí příklady [12]. Poskytuje bezplatnou a placenou verzi, kde uživatel má k dispozici taky omezené možnosti, např. několik příkladů praktických úloh a k nim video s řešením dané úlohy, přístup k prohlédnutí řešení jiných uživatelů a k diskuzi. Taky jsou pro používání aplikace video kurzy na různá témata a sady pro pohovory. Ale zůstávají hlavní stejné nevýhody jež jsou placená verze a absence mobilní aplikace [13].

Výhody

- Velká sada funkcionalit
- Uspokojí požadavky
- Příklady pohovorů určitých firem

Nevýhody

- Placená verze - 29 USD/14 dnů, 79 USD/90 dnů, 35 USD/měsíc, 150 USD/rok
- Není možnost psaní vlastních úloh
- Pouze verze v angličtině
- Není mobilní aplikace

2.2.4 HackerRank

HackerRank je webová aplikace, která má trochu jiný směr než výše uvedené aplikace [14]. Obecně nabízí nejrozšířenou funkcionalitu uživatelům. Tato aplikace nese spousta výhod programátorům ve směru tohoto projektu kromě už výše určených, jako jsou praktické úlohy různé složitosti na různá témata, přístup k diskuzi na různá témata, videonávody s vysvětlením určitých úloh a napsaným popisem k nim, sbírání bodů za správná řešení a tímto soutěžit s ostatními uživateli v hodnocení. Další výhody jsou procházení zaměřenými testy v různých směrech a získání tím certifikátu a účast v různých konkurzech. Na rozdíl od ostatních aplikací HackerRank nabízí druhý typ uživatele, který nemá žádná jiná z tohoto seznamu, a to je firma. Kvůli tomu, že nemáme možnost se přihlásit do systému jako firma, dle informací psaných pro firmy můžeme předpokládat, že firmy mají možnost přidávat vlastní úlohy pro programátory, což je velká výhoda. Ale kvůli svému směru, který je zaměřený víc na praxi programátorů než na přípravu k technickému pohovoru. Aplikace má i své nevýhody, jako je absence teoretických otázek a neflexibilní úlohy, kde nelze používat různé programátorské jazyky pro řešení určitého zadaní.

Výhody

- Velká sada funkcionalit
- Uspokojí požadavky
- Bezplatné použití
- Mobilní aplikace

Nevýhody

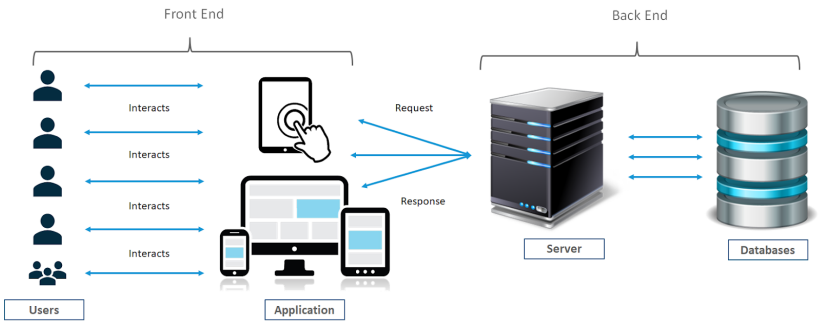
- Pouze verze v angličtině
- Absence teoretických otázek
- Není možnost se zaregistrovat a přihlásit se jako firma
- Nelze používat různé programátorské jazyky pro řešení určité úlohy
- Neexistuje návod nebo textový popis tématu
- Aplikace není zaměřená na přípravu programátorů k pohovoru

2.3 Sběr požadavků

V předchozích podkapitolách byly vysvětleny procesy technických pohovorů a jejich problémy. Z toho vyplynuly základní požadavky na řešení daných problémů. Požadavky je potřeba konkretizovat a rozšířit tak, aby splnily očekávání všech cílových uživatelů řešení. Pro správnou definici požadavků je nutné počítat s každou skupinou, které byly uvedené v úvodu práce, a to jsou programátoři a firmy v roli zkoušejících. Svou zkušenost z pohovorů využívám ke splnění všech svých potřeb v roli programátora a také sbírám informace, co se týče IT pohovoru od svých kamarádů a kolegů. Tím se požadavky stále upřesňují. Veškeré funkční a nefunkční požadavky jsou uvedeny na základě vlastního předpokladu v návrhové kapitole bakalářské práce.

2.4 Analýza technologií

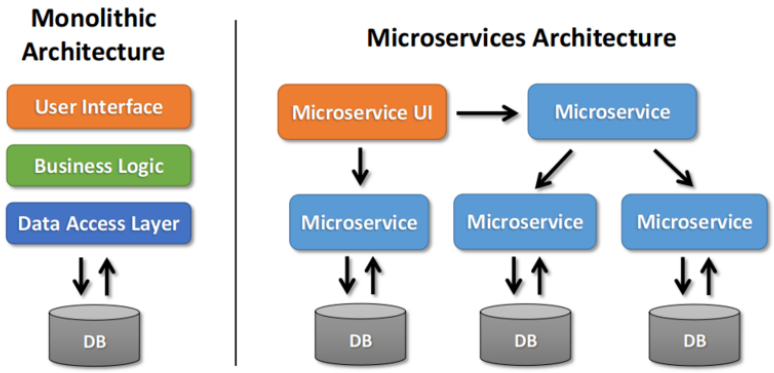
V této podkapitole se práce zabývá technologiemi a nástroji, které se během implementace aplikace používají. Uvádí se, jakou mají roli, a stručně se popisuje, proč jim byla dána přednost před jinými nástroji a technologiemi. Aplikace se skládá ze tří spolu komunikujících částí: Databáze, Backend a Frontend. *Obrázek 2.1* ukazuje infrastrukturu celého systému.



Obrázek 2.1: Infrastruktura webové aplikace [15]

2.4.1 Architektura

V roli architektury aplikace jsem zvolil Monolit, protože lze lehce implementovat a pro účely tohoto projektu bohatě stačí [16][17]. Nejčastěji v aplikacích používají 2 druhy architektury, to je "Monolit" nebo "Microservices"(*Obrázek 2.2*) [18]. Rozdíl mezi nimi je v tom, že v architektuře monolitu "Using interface", "Business logic", "Data Access Layer" a "API REST" se budou řídit uvnitř jednoho modulu. Velká výhoda monolitu je v tom, že má lehčí implementaci, ale má současně zpětnou stranu nevýhod, a tou je složitost buď doplnění nové funkčnosti nebo oprava chyb v modulu kvůli tomu, že je potřeba měnit všechny části systému.



Obrázek 2.2: Rozdíl mezi Monolit a Microservices architekturami [19]

K popularním architekturám taky patří "Layered or N-tier architecture" a "Event-driven architecture" [20]. Hlavním důvodem, proč byla vybrána jiná architektura, je nedostatečná zkušenost a znalost detailů těchto architektur.

Ve srovnání monolit a microservice architektur jak je popsáno v *tabulce 2.1*, monolit je postaven jako jeden velký systém a obvykle je to jeden kódový základ. Monolitická aplikace je pevně spojená a zapletená, jak se aplikace vyvíjí, takže je obtížné izolovat služby pro účely, jako je nezávislé škálování nebo udržovatelnost kódu.

Microservices architektura je postavena jako malý nezávislý modul založený na podnikových funkcích. V aplikaci mikroslužeb jsou všechny projekty a služby navzájem nezávislé na úrovni kódu.

Na závěr lze shrnout výše uvedené tak, že Microservices na rozdíl od Monolit je určený pro velké aplikace (víc než 100 uživatelů) a na začátku vyžaduje komplikované konfigurace jednotlivých komponent a messaging systému [21].

Rozdíl	Monolit	Microservice
Základní	Postaven jako jeden velký systém a je obvykle jedinou základnou kódu.	Postaven jako malý nezávislý modul založený na podnikových funkcích
Rozsah	Není snadné škálovat na základě poptávky	Je snadné škálovat na základě poptávky
Databáze	Má sdílenou databázi	Každý projekt a modul má svou vlastní databázi
Nasazení	Velká základna kódu zpomaluje IDE a zvyšuje čas sestavení.	Každý projekt je nezávislý a má malou velikost. Celková doba sestavení a vývoje se tak zkracuje.
Úzce spřažené a volně spřažené	Je těžké změnit technologii, jazyk nebo strukturu, protože vše spolu úzce souvisí a je na nich závislé.	Technologie nebo struktura se snadno mění, protože každý modul a projekt jsou nezávislé

Tabulka 2.1: Základní rozdíl mezi Monolit a Microservices architekturami

2.4.2 Databáze

V roli databáze byl vybrán PostgreSQL [22]. Mezi nejpopulárnější patří Mysql, Oracle a PostgreSQL [23]. Obecné důvody vlastní přednosti PostgreSQL jsou popularita databáze, dostupnost velkého počtu zdrojů a open source. Ve srovnání s Oracle byl důvod v bezplatném používání a vlastní zkušenosti [24]. Ve srovnání s MySQL je PostgreSQL bezplatná s open-source a vydaná pod license PostgreSQL podobná MIT. MySQL patří korporaci Oracle a nabízí různé placené verze. PostgreSQL je nejvíce kompatibilní s SQL, protože odpovídá 160 ze 179 základních funkcí standardu SQL a řadě dalších funkcí. Na druhou stranu je MySQL částečně kompatibilní s SQL, protože neimplementuje celý standard SQL. Ale co se týče výkonu, MySQL je lehčí, protože větší počet webových aplikací jako vlastní aplikace potřebují databázi jenom pro jednoduché datové transakce a PostgreSQL se používá pro složité dotazy. Taky MySQL je více bezpečný a obsahuje hodně bezpečnostních funkcí, z nichž některé jsou poměrně pokročilé. Implementuje bezpečnostní protokoly založené na seznamech řízení přístupu (ACL) pro uživatelské operace, jako jsou připojení a požadavky. PostgreSQL nabízí integrovanou podporu SLL pro připojení k šifrování komunikace klient/server (*Tabulka 2.2*).

Na závěr byla konečná přednost udělena PostgreSQL, protože je pokročilý objektově-relační systém s otevřeným zdrojovým kódem, který používá jazyk SQL. Postgres vám umožňuje bezpečně ukládat velká a složitá data. Pomůže vytvořit složitější aplikace v případě následujícího rozšíření aplikace, provádět administrativní úkoly a vytvářet soudržná prostředí [25].

Rozdíl	PostgreSQL	MySQL
Řízení	Plná bezplatná verze s open-source	Částečně bezplatná verze s open-source
Soulad SQL	Velká část kompatibilní s SQL a splňuje téměř všechny základní funkce standardu SQL	Částečně kompatibilní s SQL a neimplementuje celý standard SQL.
Výkon	Vyžaduje složité dotazy	Široce se používá pro webové projekty, které potřebují databázi pouze pro datové transakce.
Bezpečnost	Nabízí integrovanou podporu SLL pro šifrované připojení	Vysoce zabezpečené

Tabulka 2.2: Základní rozdíl mezi PostgreSQL a MySQL databáze

2.4.3 Backend

Pro backendovou čast byla vybrána Java, konkrétně framework Spring Boot [26]. Existuje velký počet frameworků pro vývoj webových aplikací [27]. Jaký vybrat a jaký z nich je lepší nebo horší pro vývoj konkrétní aplikace je to otázka osobní preference, protože aplikaci lze implementovat různými způsoby a každý má své výhody a nevýhody. Z populárních technologií je Python, C Sharp, Spring Boot, PHP a NodeJS.

Spring Boot může pomoct rychle vytvářet libovolné aplikace a není potřeba se starat o bezpečnost a správné konfigurace [29]. Spring Boot má velkou komunitu uživatelů, což znamená, že lze jednoduše nalézt dokumentaci a materiál už řešených problémů [28].

Další výhody Spring Boot jsou :

- Pomáhá automaticky konfigurovat všechny komponenty aplikace.
- Vyhýbá se ruční práci při psaní standardního kodu, anotací a složitých konfigurací XML.
- Dodává se s integrovanými servery HTTP, jako jsou Jetty a Tomcat, pro testování webových aplikací.
- Umožňuje vám snadné připojení k databázím a frontovým službám, jako jsou Oracle, PostgreSQL, MySQL, MongoDB, Redis a další.

Konečná přednost byla dána Spring Boot z důvodů výše uvedených výhod, vlastní zkušenosti s ním, velké popularity a počtu zdrojů.

V aplikace taky byla použita technologie web services. Web services je sada otevřených protokolů a standardů používaných pro výměnu dat mezi aplikacemi nebo systémy. Nejlepší výběr je REST ze seznamu protokolů realizace web service ve kterém jsou: SOAP, REST a GraphQL [30]. REST je ideální pro načítání dat. REST se často používá, když vystavujeme veřejné nebo partnerské API přes internet. Pro tento účel je ideální, protože poskytuje izolovaný a kontrolovaný přístup k pojmenovaným zdrojům. Oproti tomu SOAP zobrazuje skutečnou logiku aplikace prostřednictvím jiné sady rozhraní, což je ideální pro provádění operací. SOAP je technicky starší protokol API, ale REST už není novým dítětem v bloku. REST výrazně zjednodušené rozhraní s webovými službami pomocí HTTP.

Ve srovnání s GraphQL je načítání dat často označováno jako největší vylepšení GraphQL oproti REST. Při použití rozhraní REST API získáte k načtení informací vždy kompletní datovou sadu. Pokud byste například chtěli požadovat informace ze dvou objektů, musíte provést dva požadavky REST API. Výhodou rozhraní REST API je jeho jednoduchost - máte jeden koncový bod, který plní jeden úkol, takže je snadné jej pochopit a manipulovat s ním. Naopak, pokud byste chtěli shromáždit nějaké informace z konkrétního koncového bodu, vždy získáte kompletní sadu dat, která vrací rozhraní REST API. GraphQL používá svůj dotazovací jazyk k přizpůsobení požadavku přesně na to, co potřebujete, od více objektů až po konkrétní pole v každé entitě [31].

Jak bylo uvedeno výše, každý z protokolů web service má své výhody/-nevýhody, ale konečná přednost byla dána REST z důvodů svých vlastních zkušeností, velké popularity a počtu zdrojů.

2.4.4 Frontend

Frontendová strána bude stát na frameworku React [32]. Existuje velký počet volně otevřených frameworků pro vývoj klientských aplikací. Momentálně

nejpopulárnější z javascriptových jsou React, Angular a Vue [33]. Nelze říct, jaký z nich je lepší nebo horší pro vývoj konkrétní aplikace.

React jako Spring Boot má stejně velkou komunitu uživatelů, což znamená, že lze jednoduše nalézt dokumentaci a materiál už řešených problémů [34] [35]. React má taky výhody pro používání a těmi jsou:

- Používá JSX - užitečné při vytváření komponent, eliminuje překlepy ve velkých stromových strukturách a usnadňuje převod z rozložení HTML na stromy ReactElement.
- Poskytuje rychlejší vykreslování.
- Zajišťuje stabilní kód - aby bylo zajištěno, že i malé změny, které nastanou v podřízených strukturách, nebudou mít vliv na jejich rodiče. Při změně objektu vývojáři pouze upraví jeho stav, provádí změny a poté budou aktualizovány pouze jednotlivé komponenty. Tato struktura datové vazby zajišťuje stabilitu kódu a nepřetržitou práci aplikace.
- Dodává se s užitečnou sadou vývojářských nástrojů - např. React Developer Tools, to je rozšíření prohlížeče dostupné pro Chrome i Firefox. To umožňuje vývojářům sledovat hierarchii reaktivních komponent, objevovat podřízené a nadřazené komponenty a kontrolovat jejich aktuální stav a vlastnosti.
- Pro vývoj mobilních aplikací existuje React Native.

React byl vybrán z důvodů vlastních zkušeností, velké popularity a počtu zdrojů, jednoduchému a rychlému prototypování funkčních komponent. Pro snadnou a rychlou práci se stylizací aplikace byl používán React Bootstrap [36].

2.5 Argumentace vlastního řešení

Tady zdůrazňuji , proč moje řešení je nejvhodnější, ačkoliv výše uvedené systémy jsou dobré varianty pro finální řešení (*Tabulka 2.3*).

Hlavním důvodem, proč není vybrána žádná z těchto aplikacích, jsou finance, protože nejvíc z výše uvedených aplikací je placené a jak bylo zmíněno výše nejvíc uživatelů používají bezplatné aplikace. Dalším důvodem preference vlastní aplikace ve srovnání s podobnými aplikacemi, je možnost firem přidávat vlastní úkoly programátorům a dál z těchto úkolů vytvářet buď tréninkové nebo oficiální “Mock Interview”. Aplikace se tak stane víc flexibilní, protože uživatelé nebudou potřebovat čekat na administrátora, až přidá nějaké nové úlohy nebo “Mock Interview”, ale můžou to udělat samostatně. Např. jestli firma má k dispozici své vlastní úkoly pro technickou část pohovoru a chce je někde opublikovat, aby se programátoři před pohovorem do této firmy lépe připravili a věděli, co na pohovoru očekávat (jaký typ úkolů, jaké složitosti, jaké témata a podobně).

Pro komfortní použití aplikace uživatelům z různých zemí je potřeba mít podporu různých jazyků. Ze zde uvedených aplikací žádná nepodporuje vícejazyčnou verzi. Tuto funkci může mít v budoucnu vlastní řešení aplikace v rámci rozšíření projektu. K budoucímu rozšíření projektu taky může patřit mobilní verze aplikace, kterou má jenom část z uvedených aplikací.

Výhody / Nevýhody	AlgoExpert	Leetcode	Coderbyte	HackerRank	Vlastní řešení
Cena	placená	placená	placená	zdarma	zdarma
Doplnění vlastních úloh	nelze	nelze	nelze	nejde se dozvědět	lze
Doplnění vlastních imitovaných pohovorů	nejde se dozvědět	nejde se dozvědět	nejde se dozvědět	nejde se dozvědět	lze
Vícejazyčná verze	není	není	není	není	není (možné v budoucím vývoji)
Mobilní verze	není	je	není	je	není (možné v budoucím vývoji)

Tabulka 2.3: Hlavní rozdíly mezi existujícími aplikacemi

Kapitola 3
Návrh řešení

V této kapitole uvádí component a use case diagramy a popisuje, jak probíhala specifikace požadavků, jaké typy úloh použity v aplikaci a navrhuje se vzor finální verze řešení.

3.1 Specifikace požadavků

Jak již bylo v analytické kapitole zmíněno, požadavky pro systém se sbíraly od dvou typu uživatelů - programátor a zkoušející. Funkční požadavky jsou rozdělené na ty, které budou uspokojené v rámci bakalářské práce a na ty, které bude moct doplnit k dalšímu vývoji a rozšíření pro budoucí rozvoj, proto bude aplikace škálovatelná. Dále jsou uvedeny nefunkční požadavky na systém. Před uvedením požadavků je potřeba si ujasnit, jaké role budou existovat v systému.

3.1.1 Role v systému

V systému bude existovat tři roli:

1. Administrator
2. Programátor
3. Zkoušející

Administrator bude mít celá práva v systému. Programátor má k dispozici přehled veřejných úloh, psaní vlastního řešení, kontrolu tohoto řešení a bude moct procházet imitovaným pohovorem. Zkoušející bude mít možnost tvořit programátorům vlastní úlohy, testy k nim a z těchto úloh vytvářet *MockInterview*.

3.1.2 Funkční požadavky

V tomto bodě jsou uvedené prostřednictvím seznamu veškeré funkční požadavky.

Neregistrovaný uživatel:

- Registrovat se do systému
- Vstup do systému bez přihlašování
- Zobrazit celý seznam veřejných teoretických nebo praktických úloh
- Hledat veřejné úlohy v seznamu dle názvu, tématu, pozice nebo složitosti
- Vyplňovat veřejné teoretické, praktické úlohy a kontrolovat vlastní řešení
- Vyplňovat veřejné tréninkové *MockInterview*

Registrovaný uživatel:

- Přihlásit se do systému
- Zobrazit celý seznam veřejných teoretických nebo praktických úloh
- Hledat veřejné úlohy v seznamu dle názvu, tématu, pozice nebo složitosti
- Upravit/Smazat uživatelský účet
- Vyplňovat veřejné teoretické, praktické úlohy a kontrolovat vlastní řešení
- Vyplňovat veřejné tréninkové *MockInterview*

Programátor bude mít možnost po přihlášení do systému:

- Procházení soukromým oficiálním *MockInterview* společnosti, pokud má přístup

Zkoušející bude mít možnost po přihlášení do systému:

- Vytvořit novou veřejnou nebo soukromou úlohu
- Upravit/Smazat vlastní privátní úlohu
- Vytvořit nový veřejný nebo soukromý pohovor
- Upravit/Smazat libovolné vlastní pohovory
- Přidávat uživatele do soukromých pohovorů
- Prohlížet výsledky uživatelů soukromých pohovorů

Admin bude mít možnost:

- Vytvořit novou veřejnou úlohu
- Vytvořit nový veřejný pohovor
- Zobrazit detail libovolné úlohy
- Zobrazit detail libovolného pohovoru

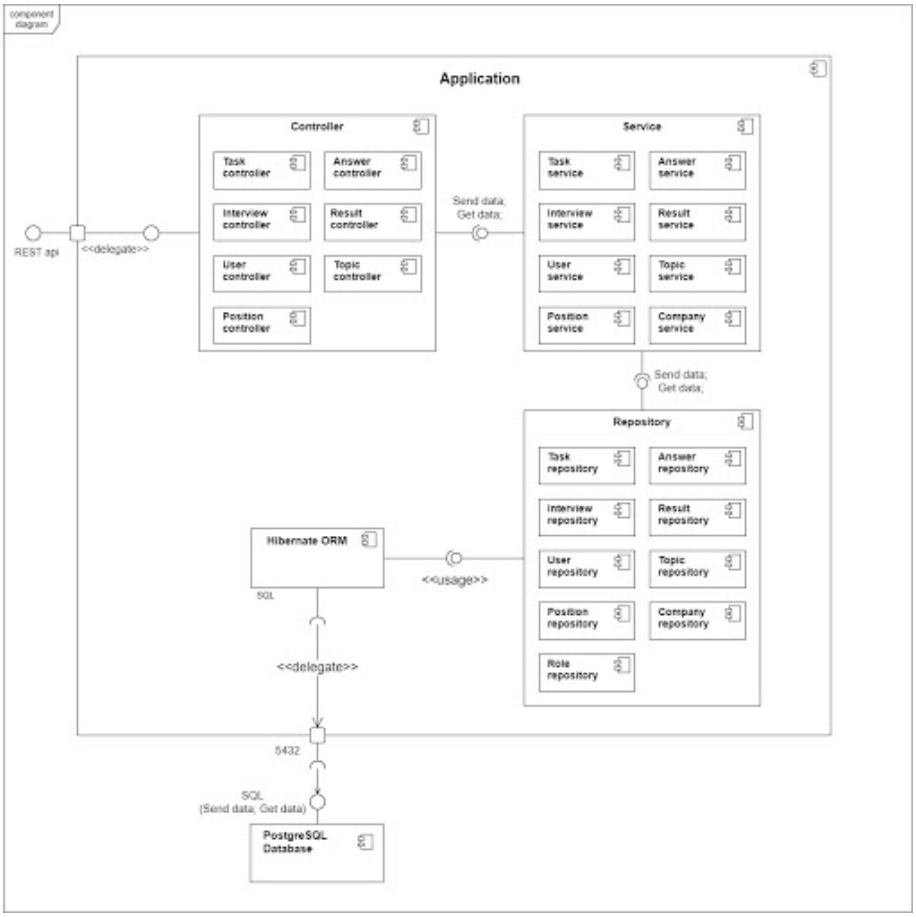
- Upravit/Smazat libovolnou úlohu
- Upravit/Smazat libovolný pohovor
- Zobrazit detail libovolného uživatele
- Smazat libovolného uživatele

3.1.3 Nefunkční požadavky

- Systém bude obsahovat český jazyk
- Systém bude možné v budoucnu jednoduše upravit nebo rozšířit
- Systém umožní pracovat s citlivými údaji pouze osobě, která k tomu má příslušná oprávnění

3.1.4 Component diagram

Component diagram určený na *obrázku 3.1* popisuje jednotlivé komponenty systému, jak jsou mezi sebou propojeny a jak obecně fungují.



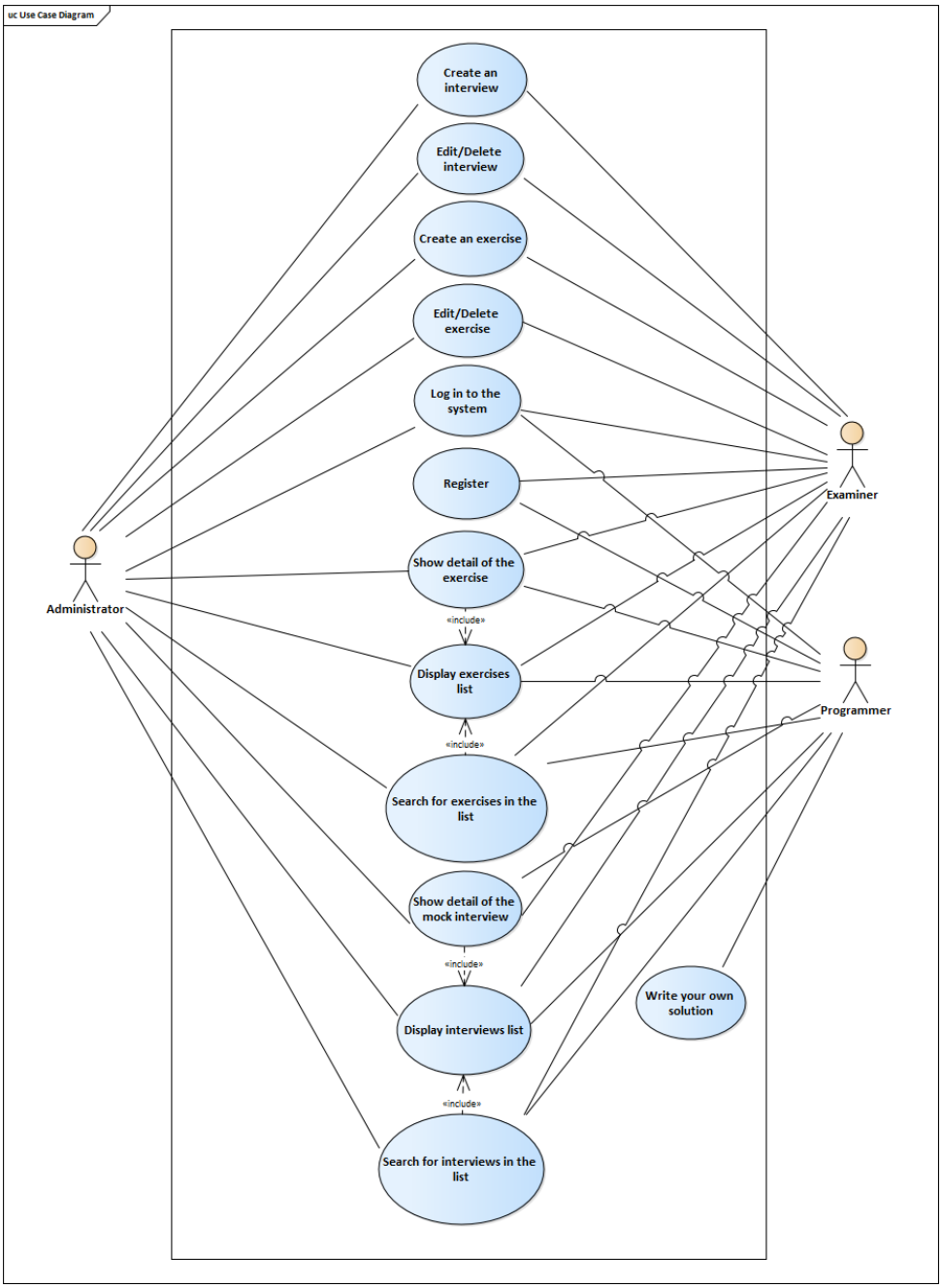
Obrázek 3.1: Component diagram

Systém se skládá ze tří hlavních komponent:

- *Controller komponenta* - je zodpovědná za zpracování příchozích požadavků z klientské strany. Po přijetí požadavků volá odpovídající metodu v Service.
- *Service komponenta* - je centrum byznys logiky celé aplikace. Service vrstva se nachází nad vrstvou repository a stará se o byznys požadavky. Navíc je zodpovědná za vypracování a vyhodnocení kódu uživatele.
- *Repository komponenta* - je mechanismus pro zapouzdření chování při ukládání, vytažení, vyhledávání a mazání dat v databázi pomocí Objektově-relačního modelu (ORM).

3.1.5 Use cases

Use case diagram na *obrázku 3.2* popisuje scénáře spolupráce uživatelů a systému.



Obrázek 3.2: Use case: Admin, programátor a zkoušející

Na *obrázku 3.2* jsou zobrazeny aktivity registrovaných a neregistrovaných uživatelů: admin, programátor a zkoušející. Neregistrovaný uživatel může vstoupit okamžitě bez přihlášení do veřejné stránky se seznamem úloh, kde bude k prohlédnutí otevřeno několik příkladů. Registrovaný návštěvník se na hlavní stránce může přihlásit do vlastního účtu a po přihlášení přejde také do veřejné stránky se seznamem úloh, kde už budou otevřeny všechny úlohy s celým přístupem.

Admin má úplná práva v systému vytvářet úlohy, imitované pohovory a také upravovat/odstraňovat úlohy, imitované pohovory a účty existujících uživatelů. Jakmile zkoušející bude registrovaný a přihlášený, bude moct vytvářet úlohy. Po vytvoření úlohy budou mít přihlášení uživatelé na stránce k dispozici seznam úloh, kde lze vyhledat potřebnou úlohu a otevřít její detaily na vlastní stránce. V úloze budou mít možnost psát vlastní řešení zadání.

Zkoušející jako ostatní uživatelé mohou na stránce seznamu úloh vyhledat potřebnou úlohu a otevřít její detaily, kde budou mít možnost upravovat data nebo úplně smazat úlohu. Pro pohodlnou práci, aby nebyla potřeba vyhledávat svoje úlohy v seznamu u zkoušejícího, bude možnost zobrazit jenom vlastní úlohy prostřednictvím samostatné stránky.

Programátoři budou mít jenom možnosti otevřít seznam úloh, vyhledat potřebné zadání a otevírat jeho detaily, kam bude možnost psát vlastní kód. K dispozici budou mít k tomu imitovaný pohovor ve formě testu s určitým časem pro vyplnění, ve kterém budou jak teoretické otázky, tak i praktická zadání.

3.1.6 Typy úloh

Tady jsou uvedeny typy teoretických a praktických úloh na různých technických pohovorech na základě analýzy a sběru informací [37]. Programátoři se setkávají na různých pohovoru s různými typy úloh v závislosti na pozici, o kterou programátor požádal. Kromě odborných typů jako úlohy na znalost programátorských jazyků, jsou nejpopulárnější témata algoritmy a datové struktury [38][39]. V aplikaci se uživatel setká s následujícím seznamem témat úloh (*Tabulka 3.2*) a technologických znalostí (*Tabulka 3.3*) a jejich složitostí(*Tabulka 3.1*):

Složitost	Lehké	Střední	Těžké

Tabulka 3.1: Složitost úloh

Téma	Algoritmy	Datové struktury	Znalosti databázových systémů	Znalosti v Back-end vývoje

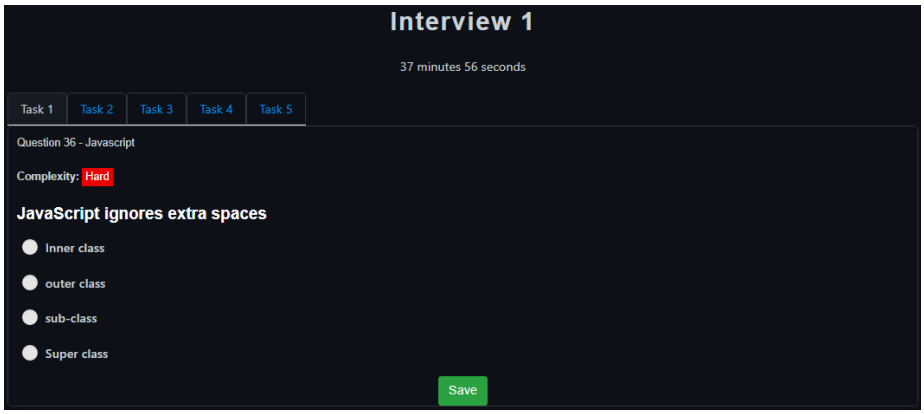
Tabulka 3.2: Seznam témat úloh

Technologie	Java	Spring	Javascript	ReactJS	NodeJS	Typescript

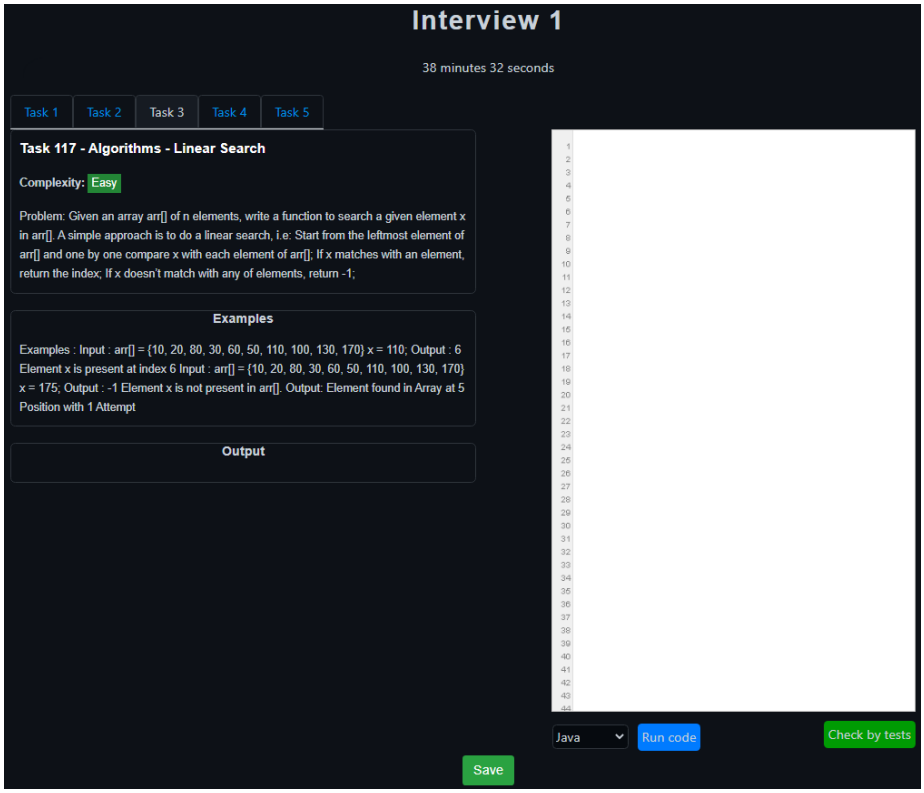
Tabulka 3.3: Seznam technologií úloh

Praktické úlohy je možné řešit v jazycích Java a Javascript. Část úloh lze řešit v obou jazycích a některé jen v jednom. Pro kontrolu řešení uživatele v Javě, aplikace nabízí testy, které přidává autor při vytváření úlohy.

Mock Interview se bude skládat ze seznamu existujících teoretických (*Obrázek 3.3*) a praktických úloh (*Obrázek 3.4*) na různá témata a různé složitosti [40][41]. Tréninkové pohovory se budou generovat náhodně a oficiální pohovory budou vytvářet firma pro reálný pohovor s programátorem.



Obrázek 3.3: Teoretická úloha Mock Interview



Obrázek 3.4: Praktická úloha Mock Interview

3.1.7 Finální řešení

Na základě problémů a příležitostí uvedených v předchozích podkapitolách vzniká několik cílů a obecných podmínek finálního řešení, kterými jsou:

- Nabídnout programátorům jednotnou sbírku teoretických a praktických příkladů na různá témata, pozice a složitosti.
- Nabídnout zkoušejícím zakládat vlastní libovolné programátorské úlohy a pohovory.
- Zkoušející budou mít možnost provádět oficiální privátní pohovory.
- Programátoři budou mít možnost procházet soukromým oficiálním pohovorem společnosti, pokud mají přístup.

Splnění podmínek finálního řešení je základním předpokladem, aby tento systém splňoval výše uvedené cíle. Nutnou podmínkou finálního řešení je, aby bylo použitelné zdarma bez žádného poplatku. Uživatelé nechtějí používat placenou aplikaci, když mohou najít bezplatnou. Další podmínkou je, aby aplikace byla dostupná odkudkoliv a kdykoliv. Webové aplikace jsou oblíbené tím, že fungují na všech platformách připojených k internetu a to i mobilních zařízeních, jejichž využívání stále roste. Hlavní podmínkou aplikace je největší pokrytí témat podle technologií a IT pozic pro zisk lepší pomoci, příležitosti se připravit a absolvovat technickou část pohovoru. Jak bylo uvedeno výše, k dispozici bude programátorům určitý seznam úloh různé složitosti, na různá témata a IT pozice. Ve finálním řešení aplikace v rámci bakalářské práce programátoři budou mít k úlohám “Mock Interview”. Zaměstnanci firem v roli zkoušejících budou moci přidávat programátorům tréninkové veřejné MockInterview a vytvářet vlastní oficiální interview pro provedení reálného pohovoru do své firmy. Tím byly určeny cíle a podmínky finálního řešení.

Kapitola 4

Implementace

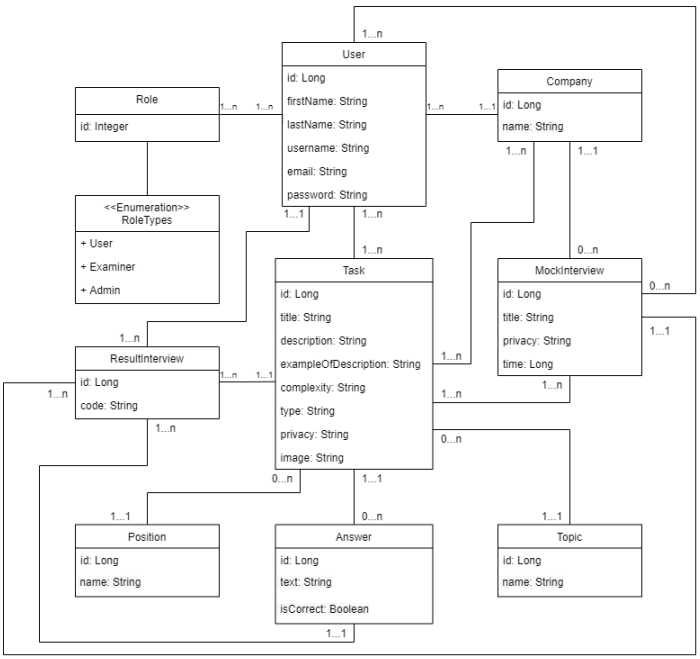
Tato kapitola se zabývá technickou specifikací rozpracované webové aplikace. Uvedeny jsou příklady diagramů, popisy, jak probíhala implementace aplikace, způsob instalace a spuštění aplikace a nakonec obsah uživatelské příručky pro lehkou orientaci v interface aplikace.

4.1 Návrh datového modelu

V tomto bodě jsou uvedeny diagramy tříd, aktivit, sekvenční diagramy.

4.1.1 Class diagram

Class diagram na *obrázku 4.1* popisuje systémové entity a veškeré vztahy mezi nimi. Vzhledem k rozsahu aplikace je diagram k nahlédnutí a níže se uvádějí klíčové entity a vazby.

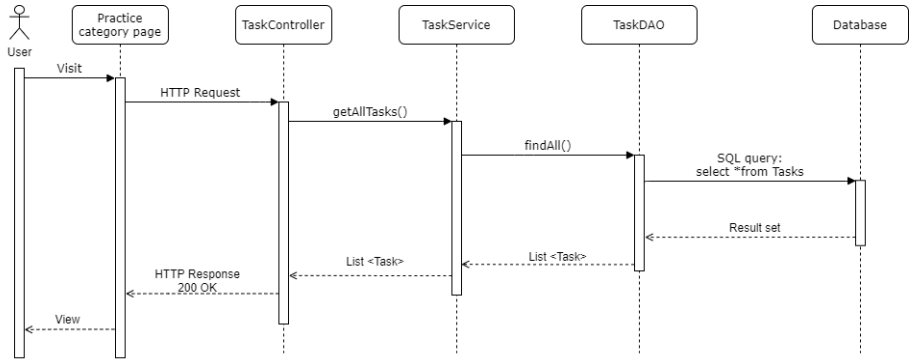


Obrázek 4.1: Class diagram

Entita User se dělí do systémových rolí uživatelů: admin, programátor a zkoušející. Admin řídí aplikaci a kontroluje, jestli funguje jak to má. Zkoušející jako zaměstnanec firmy, má možnost vytvářet neomezený počet úloh(Exercise) a k nim MockInterivew. Programátor může prohlížet detaily úloh a řešit je, také může řešit imitované pohovory(MockInterivew), které se skládájí z těchto úloh a po ukončení se tohoto pohovoru úloží jeho výsledek uživatele do databáze, pro následující review zkoušejícím. Každá úloha má patřit do určitého Topic a Position.

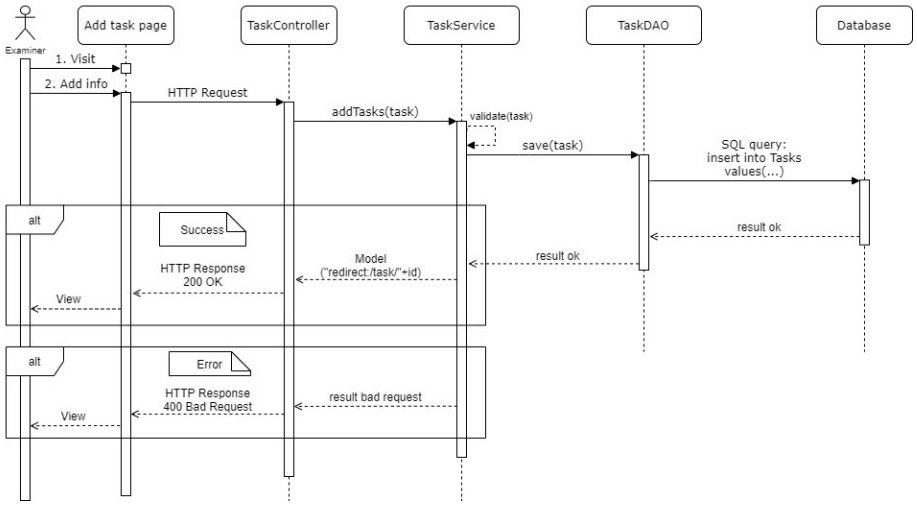
4.1.2 Sequence diagram

Sekvenční diagram na *obrázku 4.2* a obrázku *obrázku 4.3* zobrazuje chování a spolupráci jednotlivých objektů v rámci jednoho případu užití. Níže jsou uvedeny objekty a jejich spolupráce v případě, když uživatel se dostává celý seznam úkolů a případ vytváření zaměstnancem firmy nového úkolu.



Obrázek 4.2: Sequence diagram: Získat všechny úlohy

Entita User na *obrázku 4.2* po otevření hlavní stránky může přejít na stránku praktických úloh, kde se zobrazí seznam buď podle složitosti, tématu nebo IT pozice.

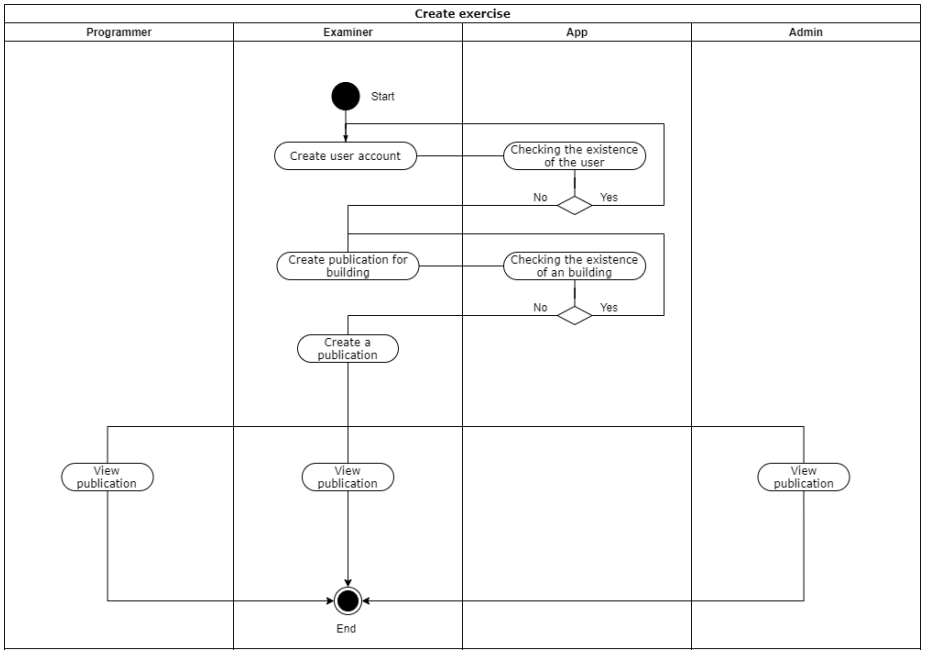


Obrázek 4.3: Sequence diagram: Přidat novou úlohu

Entita Examiner na *obrázku 4.3* po otevření hlavní stránky a přihlášení do systému může přejít na stránku formuláře nového tasku, kde bude mít možnost vytvořit nový úkol, který bude přístupný pro programátora a bude buď tréninkový nebo oficiální pro Mock Interview své firmy.

4.1.3 Activity diagram

Diagram aktivit na *obrázku 4.4* popisuje, jak probíhá proces tvorby úlohy v systému.



Obrázek 4.4: Activity diagram: Vytvoření nové úlohy zkoušejícím

Na *obrázku 4.4* je ukázán postup tvorby úlohy v systému uživatelem v roli zkoušejícího. Před vytvořením úkolu se uživatel potřebuje registrovat do systému v roli Examiner. Při registraci systém zkontroluje, jestli takový uživatel existuje a v případě existence odmítne registraci, v opačném případě zaregistruje uživatele. Dále zkoušející může poslat požadavek do serveru při tvorbě úlohy a systém zkontroluje, jestli taková úloha už existuje a udělá stejný postup jako při registraci uživatele. Po vytvoření úlohy se každý z uživatelů může podívat do detailu úlohy.

4.2 Procesy implementace

V tomto bodě je uveden popis hlavních procesů aplikace a jak tyto procesy byly implementovány . Také jsou zde uvedena oprávnění uživatelů a jaké problémy vznikly během vývoje projektu.

4.2.1 Registrace běžného uživatele

Jestli se člověk chce zaregistrovat jako nový uživatel, tak potřebuje použít požadavek typu (POST s URL /auth/signup), tělo požadavku by mělo obsahovat jméno, příjmení, uživatelské jméno, email, heslo a automaticky se do těla přidá Role jako Examiner a tělo bude nakonec mít JSON format(*Listing 4.1*)

Listing 4.1: Příklad těla požadavku na registrace zkoušejícího

```
{
  "username": "test_user",
  "first_name": "user",
  "last_name": "test",
  "password": "test"
  "email": "test@fel.cvut.cz",
  "company": {
    "name": "company_1"
  },
  "role": ["examiner"]
}
```

V případě registrace programátorem není potřeba vyplňovat pole “Company” a tělo požadavku v takovém případě bude mít formát(*Listing 4.2*),

Listing 4.2: Příklad těla požadavku na registraci programátora

```
{
  "username": "test_user",
  "first_name": "user",
  "last_name": "test",
  "password": "test"
  "email": "test@fel.cvut.cz",
  "role": ["user"]
}
```

Uživatel se může stát administrátorem jenom v případě ručního přidání do databáze běžným administrátorem. To bylo určeno kvůli bezpečnosti, v případě pokusu samostatně poslat požadavek s určitými daty a v poli Role ukázat hodnotu admin, uživatel se přidá do databáze v roli User a nebude mít přístupné možnosti administrátora.

Jestli uživatel špatně vyplní nebo neuvede nějakou položku kromě nepovinné "Company" v těle požadavku, tak server jeho požadavek odmítne a vypíše zprávu, obsahující informace o tom, co je špatně. V případě, že všechna data jsou uvedena ve správném formátu a požadavek je v pořádku, dostane uživatel odpověď s kódem 201 Created, což bude znamenat, že nový uživatel byl vytvořen a uložen v databázi.

4.2.2 Přihlášení

Klient, který se chce přihlásit, musí mít účet v systému a potřebuje poslat POST požadavek s URL /auth/signin, v těle kterého je uvedené uživatelské jméno a heslo (*Listing 4.3*).

Listing 4.3: Příklad těla požadavku na přihlášení

```
{
  "username": "test_user",
  "password": "test"
}
```

Server při zpracování požadavku kontroluje, jestli takový uživatel existuje v databázi a uvedl-li správné heslo. Pokud kontrola byla úspěšná, generuje se JSON Web Token, který je zabezpečen symetrickým algoritmem HS512 s tajným klíčem a obsahuje zakódovaná data uživatele. Tento token je odpovědí na požadavek klienta. Při jakémkoli dalším požadavku se bude používat token klienta v hlavičce požadavku.

Jestli uživatel špatně vyplní nebo neuvede nějakou položku v těle požadavku, tak server jeho požadavek odmítne a vypíše zprávu obsahující informace o tom, co je špatně.

4.2.3 Vytváření nové úlohy

Pro vytvoření nové úlohy uživatel musí být přihlášen do systému v roli admin nebo examiner a k tomu potřebuje požadavek typu POST s URL /tasks/create. Pro tyto uživatele na základě bezpečnosti ProtectedRoute bude k dispozici stránka pro vytvoření úlohy. ProtectedRoute vždycky kontroluje při přesměrování na zadanou stránku roli uživatele pomoci Bearer JSON Web Tokenu, který se ukládá po přihlášení do Local Storage prohlížeče , a v případě otevření stránky nepřihlášeným uživatelem nebo uživatelem s rolí User stránka nebude přístupná a uživatel se přesměruje na hlavní stránku.

Jestli běžný uživatel je admin data companies, která znamená k jaké firmě bude patřit úloha, automaticky zůstane prázdná v těle požadavku. V případě

zkoušejícího data company se automaticky nastaví na firmu běžného uživatele v těle požadavku (*Listing 4.4*).

Listing 4.4: Příklad těla požadavku na vytvoření nové úlohy

```
{
  "title": "First Task",
  "description": "First task description",
  "exampleOfDescription": "First task example of description",
  "complexity": "Hard",
  "type": "Theory",
  "privacy": "Public",
  "image": "http://localhost:8080/tasks/files/TkFirstTask",
  "topic": {
    "name": "My topic 1"
  },
  "position": {
    "name": "My position 1"
  },
  "companies": [{
    "name": "test_company"
  }],
  "answers": [
    {
        "text": "answer text 1",
        "isCorrect": true
    },
    {
        "text": "answer text 2",
        "isCorrect": false
    }
  ]
}
```

V těle požadavku, jak je ukázáno v *Listing 4.4*, se ještě odeberou objekty Topic a Position, které budou buď odebrané z už existujících nebo vytvoří se nové v databázi.

Jestli uživatel špatně vyplní nebo neuvede nějakou položku kromě nepovinných (Position, Image, Example of Description) v těle požadavku, tak server jeho požadavek odmítne a vypíše zprávu obsahující informace o tom, co je špatně. Taky server kontroluje při požadavku roli uživatele v tokenu, a jestli on nemá povolení k tomuto požadavku, server jeho požadavek odmítne a vypíše zprávu (*Listing 4.5*).

Listing 4.5: Příklad kontroly uživatele při vytvoření úlohy

```
@PostMapping("")
@ResponseStatus(HttpStatus.CREATED)
@PreAuthorize("hasRole('EXAMINER') or hasRole('ADMIN')")
public void createTask(@RequestBody TaskDto taskDto) throws
    IOException {
    taskService.createTask(taskDto);
```

```
}
```

V případě, že všechna data jsou uvedena ve správném formátu a požadavek je v pořádku, dostane uživatel odpověď s kódem 201 Created, což bude znamenat, že nová úloha byla vytvořena uložena v databázi.

4.2.4 Zobrazení úloh nebo interview

Pokud se klient v roli zkoušejícího potřebuje dozvědět nějakou informaci o úlohách nebo o interview, které patří do jeho firmy, může použít GET požadavek /tasks/allPublic pro získání veřejných úloh nebo /interview/allPublic pro veřejné interview jeho firmy a /tasks/allPrivate požadavek pro získání soukromých úloh, které se patří do interview nebo /interview/allPrivate pro pro získání soukromých interview. Server při zisku požadavku kontroluje na základě ID firmy uživatele v tokenu (*Listing 4.6*), jestli uživatel má přístup k datům úloh této firmy a potom, jestli takové úlohy existují, dostane odpověď, která bude obsahovat data o úlohách (*Listing 4.7*).

Listing 4.6: Příklad kontroly zkoušejícího při získání úloh firmy

```
@GetMapping("/allPublic")
@PreAuthorize("hasRole('EXAMINER')")
public Collection<TaskDto> getAllPublicTasksByCompanyId() {
    UserDetailsImpl userDetails = (UserDetailsImpl)
        SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    return
        taskService.getAllPublicTasksByCompanyId(userDetails.getCompany().getId());
}

@GetMapping("/allPrivate")
@PreAuthorize("hasRole('EXAMINER')")
public Collection<TaskDto> getAllPrivateTasksByCompanyId() {
    UserDetailsImpl userDetails = (UserDetailsImpl)
        SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    return
        taskService.getAllPrivateTasksByCompanyId(userDetails.getCompany().getId());
}
```

Listing 4.7: Příklad návratné hodnoty při získání úloh firmy

```
[
    {
        "id": 1,
        "title": "First interview",
        "privacy": "Public",
        "time": 45,
        "company": {
            "id": 1,
            "name": "Dateio"
        },
        "taskTitles": null,
```

```
        "users": null,
        "taskDtos": [
            {
                "id": 1,
                "title": "Question 1 - Analysis of Algorithms",
                "description": "What is recurrence for worst case of
                    QuickSort and what is the time complexity in
                    Worst case?",
                "exampleOfDescription": null,
                "complexity": "Easy",
                "type": "Theory",
                "privacy": "Public",
                "image": null,
                "topic": {
                    "id": 1,
                    "name": "Algorithms"
                },
                "companies": [
                    {
                        "id": 1,
                        "name": "Dateio"
                    }
                ],
                "position": null,
                "answers": null,
                "getAnswerDtos": [
                    {
                        "id": 1,
                        "text": "Recurrence is T(n) = T(n-2) + O(n)
                            and time complexity is O(n^2)"
                    },
                    {
                        "id": 2,
                        "text": "Recurrence is T(n) = T(n-1) + O(n)
                            and time complexity is O(n^2)"
                    }
                ],
                "interviews": null,
                "practiceTaskDtos": null,
                "code": null,
                "value": null
            }
        ],
        "userDtos": []
    }
]
```


4.2.5 Zpracování praktického řešení uživatele

Jak bylo uvedeno výše, úlohy se budou dělit na teoretické a praktické . Uživatelé budou mít možnost kontrolovat vlastní řešení obou typů úloh.

Teoretické úkoly formou vybrané odpovědí "True" nebo "False". V praktických úlohách budou možnosti kompilace napsaného uživatelem řešení pro kontrolu syntaktických chyb a spuštění předem vytvořených zaměstnancem firmy testů.

Při ukončení psaní řešení uživatelem bude možnost kompilace vlastního kódu, který se ze strany klienta se odebere podle POST požadavku s URL tasks/checkSolutionByCompile/id, ve kterém tělo bude mít kód uživatele jako jeden string, žádaný programovací jazyk (*Listing 4.8*). Na straně serveru request bude zpracovaný, pro kód uživatele se vytvoří nový java soubor s názvem a id úlohy v kořenovém adresáři a do něho se zachová kód a poté se spustí, přejde kontrolu syntaxe a na konce se smaže soubory v kořenovém adresáři a vrátí chybu v případě špatné syntaxe nebo návratovou hodnotu vlastního řešení.

Listing 4.8: Příklad těla požadavku zpracování praktického řešení

```
{
  "code": "public class TkTask118AlgorithmsSelectionSort { public
      static void sort(int[] arr) { int n = arr.length; for (int i =
      0; i < n - 1; i++) { int min_idx = i; for (int j = i + 1; j <
      n; j++) if (arr[j] < arr[min_idx]) min_idx = j; int temp =
      arr[min_idx]; arr[min_idx] = arr[i]; arr[i] = temp; } } }",
  "value": "java"
}
```

Druhá možnost kontroly vlastního řešení na správnost je spuštění firmou předem vytvořených testů na tento úkol. Proces zpracování řešení uživatele je stejný jako při kompilaci a liší se pouze URL adresa požadavku tasks/-checkSolutionByTest/id, kromě tohoto ještě se souborem úlohy do kořenového adresáře se zkopíruje testový soubor této úlohy z serverové složky testů. Dále soubor úlohy a testovací soubor se zkompiluje pomocí ručně stažených do serverové složky .jar knihoven "junit-4.13.2" a "hamcrest-core.1.3", které pomáhají kompilovat a spouštět Unit testy těchto úloh. Nakonec po spuštění testovacích souborů se celý výsledek úspěšných a neúspěšných testů včetně chyb vrátí uživateli jako jeden celý string, jak lze vidět na *obrázku 4.5*.

```
1  JUnit version 4.13.2
2  Sorted array
3  [11, 12, 22, 25, 64]
4
5  Time: 0,012
6
7  OK (1 test)
```

Obrázek 4.5: Příklad návratné hodnoty testu

Jak je ukázáno v *Listing 4.9*, v metodě runProcess probíhá spuštění příkazu pro kompilace .java soubory a spuštění .class souboru pomocí metody Runtime.getRuntime().exec(command).

Listing 4.9: Příklad ověření řešení podle testu

```
public String checkSolutionByTest(String title, String task, String
    language) {
    try {
        outOfTask = "";
        if(language.equals("java")) {
            FileWriter myTaskWriter = new
                FileWriter(System.getProperty("user.dir") + "/Tk" +
                title +".java");
            myTaskWriter.write(task);
            myTaskWriter.close();
            File oldFile = new File(System.getProperty("user.dir") +
                "/server/src/test/java/TsTk" + title +".java");
            File newFile = new File(System.getProperty("user.dir") +
                "/TsTk" + title +".java");
            Files.copy(oldFile.toPath(), newFile.toPath());
            runProcess("javac -cp junit-4.13.2.jar;. TsTk" + title
                +".java "+"Tk" + title +".java");
            runProcess("java -cp
                junit-4.13.2.jar;hamcrest-core-1.3.jar;.
                org.junit.runner.JUnitCore TsTk" + title);
            Path pathClassTest =
                Paths.get(System.getProperty("user.dir") + "/TsTk" +
                title +".class");
            Path pathClassTask =
                Paths.get(System.getProperty("user.dir") + "/Tk" +
                title +".class");
            Path pathTest = Paths.get(System.getProperty("user.dir")
                + "/TsTk" + title +".java");
            Path pathTask = Paths.get(System.getProperty("user.dir")
                + "/Tk" + title +".java");
            try {
                Files.deleteIfExists(pathClassTest);
                Files.deleteIfExists(pathClassTask);
                Files.deleteIfExists(pathTest);
                Files.deleteIfExists(pathTask);
            } catch (IOException e) {
                e.printStackTrace();
            }
            return outOfTask;
        } else if (language.equals("javascript")) {
            File file = new File(System.getProperty("user.dir") +
                "/server/src/test/javascript/.js");
            Process process = new ProcessBuilder("npm.cmd test",
                "update").directory(file).start();
            process.waitFor();
            return outOfTask;
        }
        System.out.println("Successfully run the java file.");
    } catch (Exception e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
```

```
    }
    return "OK";
}
```

4.2.6 Vytváření nového interview

Při vytváření nového interview je stránka s formulářem pro vytvoření zabezpečena stejně, jak bylo popsáno v Bodě 4.2.3, a bude se lišit URL adresa požadavku /interview/create.

Jestli běžný uživatel je zkoušející data companies stejně automaticky se nastaví na firmu uživatele v těle požadavku. Zkoušející při vytváření veřejného ("Public") interview mají k dispozici "Randomizer" veřejných úloh pro náhodnou sadu úloh a proto je potřeba jen poslat požadavek typu GET s URL (tasks/randomGenerate?Parametr1andParametr2) kde:

- Parametr1 - theoryCount=číslo (počet teoretických otázek)
- Parametr2 - practiceCount=číslo (počet praktických úloh)

Po odeslání požadavku uživatel dostane výsledek se seznamem názvů úloh a automaticky se přiřazuje do těla požadavku interview. Uživatel má také druhou možnost pomocí ručního výběru a přidání úloh do těla požadavku. Nakonec finální POST požadavek interview bude mít následující format včetně možného času interview(*Listing 4.10*)

Listing 4.10: Příklad těla požadavku na vytvoření interview

```
{
  "title": "Interview 1",
  "privacy": "Private",
  "time": 60,
  "taskTitles": [
    "First task",
    "Second task",
    "Third task"
  ]
}
```

Jestli uživatel špatně vyplní nebo neuvede nějakou položku kromě nepovinného Time v těle požadavku, tak server jeho požadavek odmítne a vypíše zprávu, obsahující informace o tom, co je špatně. V případě, že všechna data jsou uvedena ve správném formátu a požadavek je v pořádku, dostane uživatel odpověď s kódem 200 OK, což bude znamenat, že nový interview byl vytvořen a uložen v databázi.

4.2.7 Přidělení přístupu k interview

Jestli zkoušející bude potřebovat přidat přístup k private interview uživatele, se kterým má domluvený technický pohovor, tak potřebuje použít požadavek

typu (PUT s URL /interviews/addUser/id), tělo požadavku by mělo obsahovat seznam username v případě několika uživatelů a id interview a bude mít JSON format jak je v *Listing 4.11* :

Listing 4.11: Příklad těla požadavku na přidělení přístupu

```
{
  "users": [
    "user1",
    "user2"
  ]
}
```

Server při zpracování požadavku nejprve zkontroluje přístup běžného uživatele k uvedené úloze stejně jak bylo popsáno v Bodě 4.2.4, poté vyhledá uživatele podle username z těla požadavku a přidělí uživateli aktuální interview. V případě, že všechna data jsou uvedena ve správném formátu a požadavek je v pořádku, dostane uživatel odpověď s kódem 200 OK, což bude znamenat, že uvedený uživatel byl přidělen k uvedenému interview a uložen v databázi.

4.2.8 Uložení výsledku interview

Během pohovoru, když programátor bude chtít ukončit interview nebo vyprší čas z klientské strany, se odebere POST požadavek s URL /results, který bude obsahovat seznam objektů úloh. Každý objekt se skládá z id interview, id úlohy tohoto interview, id odpovědi v případě teoretické otázky nebo kód jako řešení praktické úlohy a id uživatele, který vyplnil tento interview (*Listing 4.12*).

Listing 4.12: Příklad těla požadavku na uložení výsledku interview

```
{
    {
        "interviewId": 2,
        "userId": 3,
        "taskId": 49,
        "answerId": 169

    },
    {
        "interviewId": 2,
        "userId": 3,
        "taskId": 49,
        "code": "public class TkTask118AlgorithmsSelectionSort {
            public static void sort(int[] arr) { int n = arr.length;
            for (int i = 0; i < n - 1; i++) { int min_idx = i; for
            (int j = i + 1; j < n; j++) if (arr[j] < arr[min_idx])
            min_idx = j; int temp = arr[min_idx]; arr[min_idx] =
            arr[i]; arr[i] = temp; } } }"
    }
}
```

Server při zpracování požadavku kontroluje v databázi existenci uživatele, interview, úlohy a odpověď v případě teoretické otázky. Nakonec kontroly dat server smaže přístup uživatele k uvedenému interview a uloží výsledek do databáze. Pokud kontrola byla úspěšná a požadavek je v pořádku, dostane uživatel odpověď s kódem 200 OK, což bude znamenat, že výsledek byl vytvořen a uložen v databázi.

4.3 Instalace projektu

V tomto bodě jsou uvedena připravená základní data z databáze, instrukce a ukázky instalace se spuštěním aplikace.

Přihlašovací údaje:

- V roli administratora:
  - username: admin
  - heslo: admin
- V roli zaměstnance IT firmy:
  - username: examiner
  - heslo: examiner
- V roli programatora:
  - username: user
  - heslo: user

Instrukce:

- Projekt je potřeba otevřit v IDE jako celou složku "Impelementation", aby vevnitř projektu po otevření byli Frontendová(client) a Serverová(server) strány dohromady. Po otevření potřeba udělat "import server xml"
- Databáze lze nastavit a spustit pomocí IDE a tohoto pořadí kroků:
  1. Otevřít okno databáze a vybrat “new -> Data Source -> PostgreSQL”
  2. V okně nastavení uvést následující data a kliknout “OK”:
     - Host: localhost
     - Port: 5432
     - Uživatel: postgres
     - Heslo: admin
     - Databáze: bachelor
  3. Po automatickém nastavení otevřít Query Console

4. Do otevřené konzole vložit data ze souboru “Implementation/ server/src/main/resources/database.sql” a provést tento skript pomocí Run

5. Lokální databáze s připravenými daty je nastavená a běží na jdbc: postgresql://localhost:5432/postgres

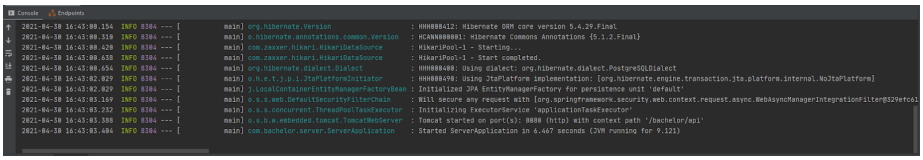
- Serverovou část - Backend (Spring Boot) lze nastavit a spustit dvěma způsoby:

  - Pomocí IDE:

    1. Zkompilovat a spustit projekt spuštěním hlavní třídy *ServerApplication*
    2. Strana serveru webové aplikace je přístupná přes localhost: 8080

  - Pomocí příkazové řádky:

    1. Zkompilovat projekt použitím příkazu *mvn clean install*
    2. Spustit použitím příkazu *mvn spring-boot:run*
    3. Strana serveru webové aplikace je přístupná přes localhost: 8080 a musí mít stejný výpis z konzole jak je na (*obrázku 4.6*)

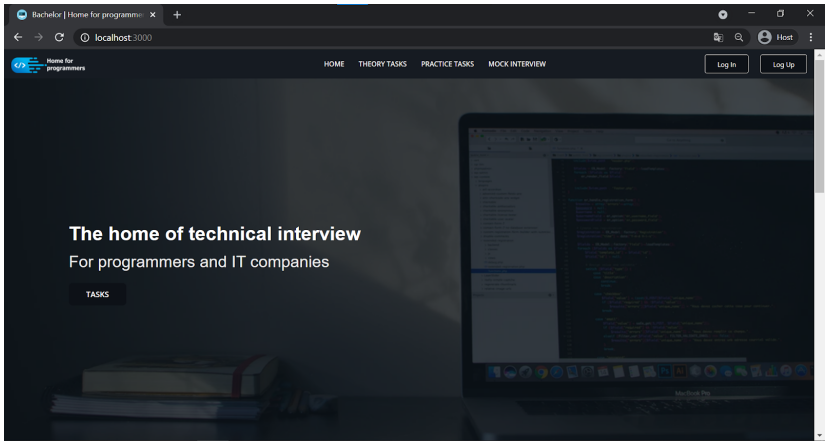


Obrázek 4.6: Správně spuštěná serverová část

- Klientskou část - Frontend (ReactJS) lze nastavit a spustit způsobem:

  - Pomocí příkazové řádky:

    1. Stáhnout knihovny použitím příkazu *yarn install*
    2. Spustit použitím příkazu *yarn start*
    3. Strana klienta webové aplikace je přístupná přes localhost: 3000 a musí se otevřít automaticky stránka, jak je na (*obrázku 4.7*)



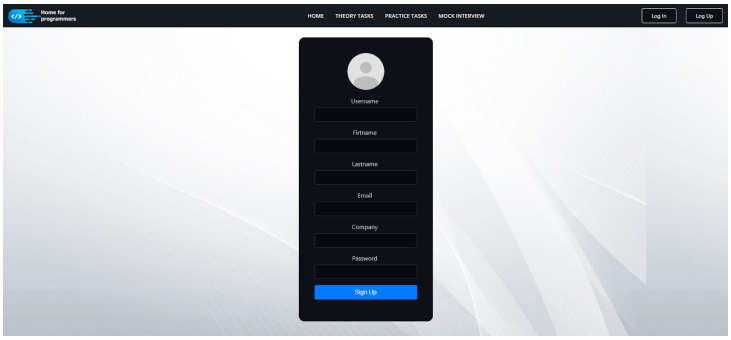
Obrázek 4.7: Správně spuštěná klientská část

4.4 Uživatelská příručka

V tomto bodě je popsán návod na používání aplikace uživatelem. Jaké existují stránky, jak se k nim dostat a jak v nich pracovat.

4.4.1 Registrační stránka

Jestli se člověk chce zaregistrovat jako nový uživatel, tak musí otevřít stránku s formulářem pro registrace (*Obrázek 4.8*) po kliknutí na tlačítko “Log Up”.

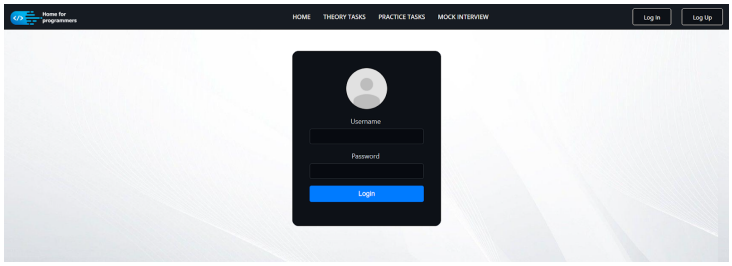


Obrázek 4.8: Registrační stránka

Dále je potřeba vyplnit celý formulář včetně pole “Company”, jestli je uživatel zaměstnanec firmy, která chce používat tuto aplikaci během vlastních pohovorů . Jestli je uživatel programátor, pole “Company” nechává prázdné.

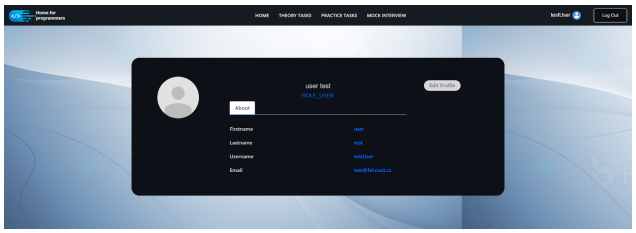
4.4.2 Přihlašovací stránka

Klient, který se chce přihlásit, tak musí otevřít stránku s formulářem pro přihlášení (*Obrázek 4.9*) po kliknutí na tlačítko “Log In”.



Obrázek 4.9: Přihlašovací stránka

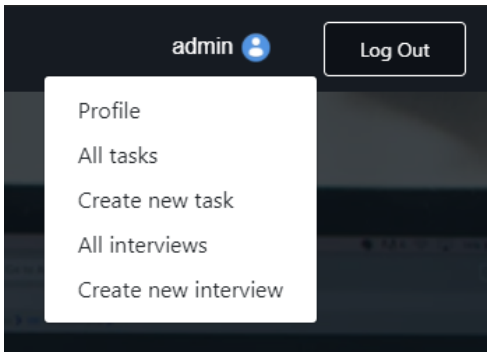
Dál vyplňuje své přihlašovací údaje a to jsou: uživatelské jméno a heslo. Po úspěšném přihlášení se uživatel přesměruje na profilovou stránku s vlastními daty klienta(*Obrázek 4.10*).



Obrázek 4.10: Profilová stránka

4.4.3 Uživatelské menu

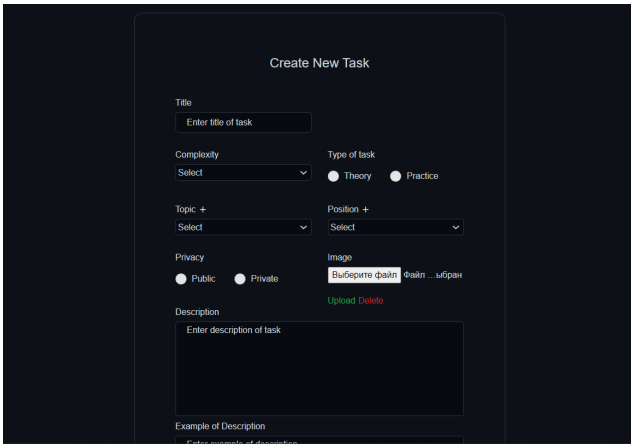
Přihlášený uživatel do systému v roli admin nebo examiner má po kliknutí na username vpravo nahoře k dispozici menu uživatele pro snadnou navigaci(*Obrázek 4.11*).



Obrázek 4.11: Uživatelské menu

4.4.4 Stránka tvorby úlohy

Pro vytvoření nové úlohy přihlášený uživatel v roli admin nebo examiner musí otevřít stránku s formulářem pro tvorbu (*Obrázek 4.12*) podle linku v uživatelském menu.

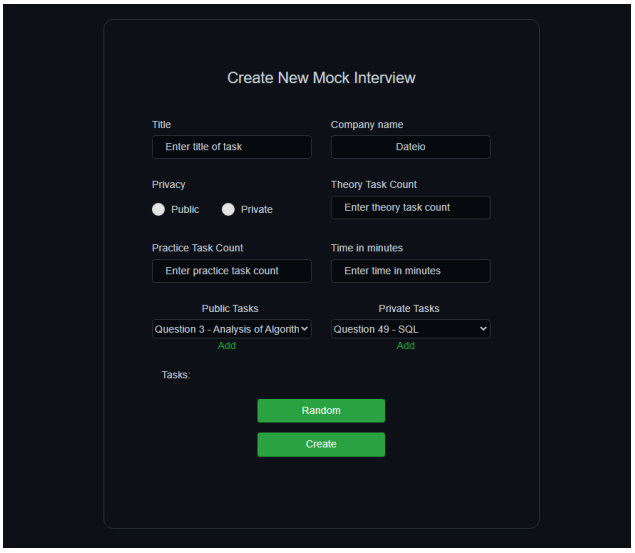


Obrázek 4.12: Formulář vytvoření nové úlohy

Při vyplnění formuláře v poli Topic nebo Position uživatel má možnost buď vybrat jednu ze seznamu existujících nebo vytvořit novou úlohu, pro tuto možnost je potřeba kliknout + . Při výběru typu "Theory" se objeví dynamická sada polí pro ukázání libovolného počtu odpovědi. Jestli vybrat "Practice" typu se objeví pole "Task template" a "Test template", kde uživatel napíše šablonu testu a úlohy, která se bude zobrazovat v detailu praktických úloh před uvedením řešení. Taky pro detailní obsah úlohy lze přiřadit obrázek, který je potřeba uložit před ukončení formuláře. V případě náhodného uložení, stačí kliknout na delete a obnovit stránku. Jak bylo uvedeno výše, Public soukromí patří do skupiny úloh, které mají veřejný přístup libovolného uživatele a Private bude souviset jen se zkoušejícím, protože ho bude potřebovat pro Interview.

4.4.5 Stránka tvorby interview

Pro vytvoření nového pohovoru přihlášený uživatel v roli admin nebo examiner musí otevřít stránku s formulářem pro tvorbu (*Obrázek 4.13*) podle linku v uživatelském menu.



Obrázek 4.13: Formulář vytvoření nového interview

Ve formuláři tvorby interview uživatel může pro veřejné pohovory "public" vybrat veřejné úlohy pomocí randomizeru. Proto je potřeba do polí "Theory Task Count" a "Practice Task Count" napsat, kolik teoretických a praktických úloh chce uživatel přidat. Taky při doplnění úloh uživatel může ručně vybrat veřejné nebo vlastní privátní úkoly v polích "Public Tasks" a "Private Tasks". Všechny přidané úlohy se zobrazí dole v seznamu "Tasks".

Kapitola 5

Testování

Pro vytvoření kvalitní aplikace bez chyb je nutné ji otestovat. Testování bylo provedeno během vývoje aplikace způsobem uživatelských testů. V této kapitole je popsáno , jak probíhalo testování aplikace uživateli.

5.1 Výkaz uživatelů

V této podkapitole se popisuje, jak uživatelé reagovali při použití aplikace.
Testovací scénáře v roli neregistrovaného uživatele:

- Řešit jednu veřejnou teoretickou a jednu praktickou úlohu a zkontrolovat získané odpovědi k úlohám
- Vyplnit jeden veřejný tréninkový MockInterview

Testovací scénáře v roli zkoušejícího:

- Registrovat se jako zkoušející s uvedením "Google" IT firmy, přihlásit se a zkontrolovat roli uživatele a úspěšné registraci s přihlášením
- Vytvořit jednu tréninkovou teoretickou a jednu praktickou úlohu a zkontrolovat přidání na veřejných stránkách seznamu úloh
- Vytvořit jednu privátní teoretickou a jednu praktickou úlohy firmy a zkontrolovat přidání na stránce seznamu vlastních úloh
- Opravit jednu vlastní privátní úlohu a zkontrolovat obnovená data
- Smazat jednu vlastní privátní úlohu
- Vytvořit jeden tréninkový pohovor z ručně vybraných tří veřejných teoretických a ze dvou praktických úloh a zkontrolovat přidání na veřejné stránce seznamu pohovorů
- Vytvořit jeden tréninkový pohovor z tří vybraných veřejných teoretických a dvou praktických úloh pomocí randomizeru a zkontrolovat přidání na stránce seznamu vlastních pohovorů

- Vytvořit jeden privátní pohovor z ručně tří vybraných vlastních teoretických a dvou praktických úloh a zkontrolovat přidání na veřejné stránce seznamu pohovorů
- Přidat přístup do privátního pohovoru programátoru podle username "user"
- Opravit jeden vlastní pohovor a zkontrolovat obnovená data
- Smazat jeden vlastní pohovor

Testovací scénáře v roli programátora:

- Registrovat se jako programátor bez uvedení firmy, přihlásit se a zkontrolovat úspěšnost registrace a přihlášení
- Opravit vlastní uživatelská data a zkontrolovat obnovená data
- Smazat vlastní uživatelský účet

Testovací scénáře v roli administrátora:

- Smazat uživatele

5.1.1 Makhambet Ismukhambetov (ČVUT, FEL, SIT student 3. ročníku)

Zpětná vazba:

- Seznamy veřejných úloh a pohovorů: Filtrování seznamu úloh a pohovorů je zvláštní. Není žádná nápověda, podle čeho lze hledat úlohy nebo pohovory. Bylo by lepší, kdyby seznam vypadal stejně jako seznam uživatelů pro administrátora.
- Detail praktických úloh: Při ověření řešení testů není vidět, podle jakých testů se kontroluje správnost řešení. Bylo by lepší, kdyby uživatel měl možnost vidět, jaké testy kontrolují jeho řešení.

5.1.2 Nazariy Shukatka (ČVUT, FEL, SIT student 3. ročníku)

Zpětná vazba:

- Teoretické otázky: Při rozkliknutí teoretickeho tasku, který patří k nějaké firmy není "label" firmy, a proto je těžké poznat, co je ukázáno. Pro lehčí pochopení je potřeba přidat nadpis "companies".
- Formulář úloh a pohovorů: Při vytvoření úloh nebo pohovorů ve formuláři byly některé problémy s kontrolou polí a se zobrazením informací o tom, jaké pole potřeba vyplnit, jsem zjistil až po pokusu odeslání. Bylo by lepší vidět okamžitě při vyplnění formuláře, jaké pole je povinné.

5.1.3 Ali Akhmadov (ČVUT, FEL, SIT student 3. ročníku)

Zpětná vazba:

- Interface praktických úloh: Pole pro psaní řešení praktické úlohy vypadá jinak než ostatní komponenty interface (např. sekce popisu úlohy nebo příkladu). Potřeba dodržovat jednu stylizaci.
- Seznamy úloh a pohovorů pro administrátora: Když si admin prohlíží v tabulce seznam všech úloh a pohovorů, nemá sloupec “firmy”. Pro zjištění, k jaké firmě patří úloha nebo pohovor, potřeba přejít do detailu. Bylo by lepší mít informaci o firmě v tabulce vedle ostatních dat.

5.1.4 Problémy během vývoje

Během implementace vznikly následující problémy:

- Spuštění java testů ve serverové složce (src/test/java)
- Spuštění javascript testů pro kontrolu správnosti řešení uživatele

První problém šlo řešit způsobem uložení a kopírování .java souborů a k nim určených testů do kořenového adresáře projektu (složka Implementation). Druhý problém se nepodařilo vyřešit. Zkoušel jsem to řešit instalaci knihoven "node modules"do složky s javascript testy a dalším spustěním. Další způsob byl spuštění testu pomocí třídy ProcessBuilder.

Dalé ukážu problémy které se objevili při testování aplikace a které bohužel nestihnul opravit:

- Při odeslání GET požadavku pro získání dat praktické javascript úlohy, vzniká vlastní Exception. On se objevuje v metodě "readStringOfFile"při získu "Template"úlohy pro řešení. V mětodě hledá se pouze Java soubory.
- Odeslání javascript "Task template"najednou s Java souborem při vytváření nové praktické úlohy. Na frontendu není možnost najednou výtvářet javascript a java soubory. Lze výtvářet pouze Java soubor, ale na backendu to funguje správně, jestli poslat POST požadavek s obou soubory jako array.
- CSS některých komponent
- Malý seznam praktických úloh
- View výsledku uživatele pro zkoušejícího
- Obnovení odpovědi teoretických otázek při obnovení úlohy
- Automatické odstranění možných obrázků a souborů při odstranění úlohy

Kapitola 6

Budoucí vývoj

V této kapitole jsou sepsány nápady, které byly vymyšleny, ale nejsou implementovány v rámci bakalářské práce a patří do budoucího rozšíření aplikace.

- Rozšiřování aplikace do vícejazyčné verze
- Vývoj mobilní verze
- Zobrazení v detailu praktické úlohy správného řešení
- Přidat více jazyků, které je možné na řešení praktických úloh
- Přidat diskusní fórum
- Doplnění teoretických a praktických úloh na znalosti ve Front-end vývoje
- Doplnění teoretických otázky na znalosti v Data analytics, Machine learning

Kapitola 7

Závěr

Cílem této práce bylo navržení a vytvoření webové aplikace pro podporu programátorů a IT společností, která slouží k zefektivnění a usnadnění provedení technického pohovoru.

Chtěl bych říct, že tato práce dosáhla svých cílů. Analytická část se věnovala definic problemů, přiváděla motivace a cíle navrhované aplikace včetně specifikace cílových uživatelů. Dále analyzovala existující aplikace věnující se podobným problémům a technologie pro vývoj tohoto systému.

Návrhová část uvedla veškeré funkční a nefunkční požadavky. Vzhledem k rozsahu práce byly požadavky rozdělené na ty, které jsou implementované v rámci bakalářské práce a na ty, které jsou příležitostí pro vylepšení a rozšíření systému. V rámci dalšího rozšíření a vývoje, priorita bude dana opravě označených chyb. Implementační část popsala proces implementace, jaké částí systému jakým způsobem byli implementovany.Testovací část ukázala, jak aplikace byla otestována pomocí uživatelských testů.

Systém je napsaný tak, aby ho bylo možné rozšiřovat o další funkcionalitu, která je uvedená v kapitole budoucího vývoje. Já bych rád tímto pomohl programátorům k lepší přípravě na pohovor.

Příloha A

Literatura

[1] *Moodle platforma* [online] [cit. 11.04.2021]. Dostupné z: `https://moodle.fel.cvut.cz`

[2] DOYLE, Alison. *Typy otázek a úkolů na technickém pohovoru* [online] [cit. 16.11.2020]. Dostupné z: `https://www.thebalancecareers.com/information-technology-it-job-interview-questions-2061206`

[3] GALLAGHER, James. *Plný návod technického pohovoru* [online] [cit. 16.11.2020]. Dostupné z: `https://careerkarma.com/blog/technical-interviews/`

[4] DOYLE, Alison. *Popis pracovního pohovoru* [online] [cit. 16.05.2021]. Dostupné z: `https://www.thebalancecareers.com/job-interview-questions-and-answers-2061204`

[5] BIKA, Nikoletta. *Seznam otázek na HR pohovoru* [online] [cit. 16.05.2021]. Dostupné z: `https://resources.workable.com/tutorial/hr-interview-questions`

[6] DOYLE, Alison. *Nejčastější technické otázky na IT pohovoru* [online] [cit. 16.11.2020]. Dostupné z: `https://www.thebalancecareers.com/top-technical-interview-questions-2061227`

[7] *Seznam běžných technických otázek* [online] [cit. 08.11.2020]. Dostupné z: `https://www.indeed.com/career-advice/interviewing/common-technical-interview-questions-and-answers`

[8] *AlgoExpert Homepage* [online] 2020. Dostupné z: `https://www.algoexpert.io/product`

[9] *AlgoExpert Subscribe Pricing* [online] 2020. Dostupné z: `https://www.algoexpert.io/purchase`

[10] *Leetcode Homepage* [online] 2020. Dostupné z: `https://leetcode.com/`

[11] *Leetcode Subscribe Pricing* [online] 2020. Dostupné z: `https://leetcode.com/subscribe/?ref=lp_pl`

[12] *Coderbyte Homepage* [online] 2020. Dostupné z: https://coderbyte.com/

[13] *Coderbyte Subscribe Pricing* [online] 2020. Dostupné z: https://coderbyte.com/member

[14] *HackerRank Homepage* [online] 2020. Dostupné z: http://www.hackerrank.com

[15] *Obrázek 2.1.* Dostupné z: https://www.quora.com/What-is-node-js-Is-it-client-side-language-or-server-side

[16] SIRAJ, Ul Haq. *Úvod do Monolit architektury* [online] [cit. 21.12.2020]. Dostupné z: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

[17] KANJILAL, Joydip. *Výhody Monolit architektury* [online] [cit. 21.12.2020]. Dostupné z: https://searchapparchitecture.techtarget.com/tip/Advantages-of-monolithic-architecture-that-prove-it-isnt-dead

[18] *Microservices vs Monolithic architektury* [online] [cit. 21.12.2020]. Dostupné z: http://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

[19] *Obrázek 2.2.* Dostupné z: https://www.mindinventory.com/blog/implement-microservices-architecture-in-mobile-app-development/

[20] *Seznam architektur* [online] 2020. Dostupné z: https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture

[21] PARAHAR, Mahesh. *Rozdíl mezi Microservices a Monolithic* [online] [cit. 16.11.2020]. Dostupné z: https://www.tutorialspoint.com/difference-between-monolithic-and-microservices-architecture

[22] *PostgreSQL* [PDF online] [cit. 08.11.2020]. Dostupné z: https://www.postgresql.org/files/documentation/pdf/13/postgresql-13-A4.pdf

[23] KAMARUZZAMAN, Md. *Top list nejpopulárnějších databází* [online] [cit. 28.04.2021]. Dostupné z: https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba

[24] *Oracle Pricing* [PDF online] [cit. 28.11.2020]. Dostupné z: https://www.oracle.com/assets/technology-price-list-070617.pdf

[25] *PostgreSQL vs MySQL: Všechno, co potřebujeme vědět* [online] [cit. 16.11.2020]. Dostupné z: https://hackr.io/blog/postgresql-vs-mysql

[26] *Spring Boot* [online] 2020. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[27] CLARK, Jessica. *Top list nejpopulárnějších backend frameworků* [online] [cit. 08.11.2020]. Dostupné z: https://blog.back4app.com/backend-frameworks/

[28] *Výhody Spring Boot* [online] [cit. 13.05.2021]. Dostupné z: https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/

[29] WALLS, Craig. *Spring Boot kniha*. Dostupné z: https://www.nitinagrawal.com/uploads/2/1/3/6/21361954/spring_boot_in_action.pdf

[30] LOGAN, Paul. *Rozdíl mezi web services* [online] [cit. 16.11.2020]. Dostupné z: https://medium.com/better-practices/rest-soap-graphql-gesundheit-6544053f65cf

[31] RUSSELL, Drew. *GraphQL vs REST* [online] [cit. 17.05.2021]. Dostupné z: https://www.rubrik.com/en/blog/technology/19/11/graphql-vs-rest-apis

[32] *ReactJS* [online] 2020. Dostupné z: https://reactjs.org/

[33] DAITYARI, Shaumik. *Jaký použít frontend framework* [online] [cit. 08.11.2020]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

[34] *Výhody ReactJS* [online] [cit. 13.05.2021]. Dostupné z: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try

[35] GACKENHEIMER, Cory. *ReactJS kniha*. Dostupné z: https://books.google.cz/books?hl=ru&lr=&id=NZCKCgAAQBAJ&oi=fnd&pg=PR6&dq#v=onepage&q&f=false

[36] *React Bootstrap* [online] 2021. Dostupné z: https://react-bootstrap.github.io/

[37] *Zdroj praktických a teoretických úloh №1* [online] 2021. Dostupné z: https://www.geeksforgeeks.org/technical-scripter-event-2020-by-geeksforgeeks/?ref=lbp

[38] *Zdroj teoretických otázek №2* [online] 2021. Dostupné z: https://quizizz.com/admin

[39] *Zdroj teoretických otázek №3* [online] 2021. Dostupné z: http://www.allindiaexams.in/engineering/cse

[40] *Zdroj teoretických otázek №4* [online] 2021. Dostupné z: https://www.tutorialspoint.com/index.html

[41] *Zdroj teoretických otázek №5* [online] 2021. Dostupné z: https://www.javatpoint.com/

[42] *Zdroj praktických javascript úloh* [online] 2021. Dostupné z: `https://www.w3resource.com/`