Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Multiplatformní webová aplikace

Diplomová práce

Autor: Bc. Michal Sychra
Studijní obor: AI2-K

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové Duben 2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Michal Sychra

Poděkování:
Děkuji vedoucímu diplomové práce doc. Ing. Filipu Malému, Ph.D. za metodickou a odbornou pomoc a cenné připomínky, které mi při vypracování této práce pomohly. Rád bych poděkoval své rodině a svým přátelům za podporu, kterou projevili během zpracování diplomové práce.

Anotace

Hlavním cílem této diplomové práce je návrh a implementace multiplatformní webové aplikace. Tato aplikace je zaměřena na sdílení poznatků a to v oblasti pracovních pohovorů. Vedlejším cílem je seznámení čtenáře s vybranými technologiemi a postupy, které se uplatňují při vývoji softwaru.

Úvodní část práce je věnována rozboru motivace k vytvoření takové aplikace a průzkumu dostupných řešení. Dále se práce zabývá analýzou, návrhem aplikační logiky a výběrem vhodných implementačních technologií.

Následuje praktická část, ve které je popsána implementace pomocí dříve zvolených technologií. V závěru práce jsou shrnuty výsledky a možnosti dalšího rozvoje aplikace. Stávající podoba webové aplikace je k nalezení v přílohách.

Annotation

Title: Multiplatform web application

The main objective of this diploma thesis is the design and implementation of a multiplatform web application. This application is focused on knowledge sharing in the field of job interviews. The secondary objective is to familiarize reader with selected technologies and with procedures that are applied in the software development.

The first part of the thesis is devoted to the analysis of the motivation to create such an application and to research existing solutions. Furthermore, the thesis deals with analysis, design application logic and selection of suitable implementation technologies.

The following is a practical part describing implementation using previously selected technologies. At the end of the thesis are summarized the results and possibilities of further development of the application. The current form of web application can be found in the attachments.

Obsah

1 Úvod

Díky dostupnosti a rychlému rozvoji informačních a komunikačních technologií je dnes běžné před učiněním rozhodnutí shromáždit a zpracovat dostupné informace. Ne jinak tomu je v oblasti hledání zaměstnání. Často si lidé sjednávají pracovní pohovory skrze webové prezentace firem a získávají zde základní informace o společnostech.

Dále se v posledních letech na trhu objevilo hned několik portálů, které se zaměřují na hodnocení zaměstnavatelů. Mezi takové portály můžeme zařadit například Jobinsider.cz, Jobadvisor.cz, Vimvic.cz a několik dalších. Díky těmto portálům lze získat názory zaměstnanců a vytvořit si tak skutečnější představu o daném zaměstnavateli.

Vzhledem k období s historicky nejnižší mírou nezaměstnanosti [1], byli někteří zaměstnavatelé nuceni snížit své nároky na nově příchozí zaměstnance. Přesto zde zůstávají profese, u kterých je součástí přijímacího pohovoru určitá forma přezkoušení znalostí a vědomostí.

Výstupem této práce má být webová aplikace, která bude zaměřena na zlepšení přípravy uchazečů o zaměstnání s využitím informačních technologií.

Cíle práce

Hlavním cílem této práce je vytvoření multiplatformní webové aplikace, která by měla usnadnit uchazečům přípravu na pracovní pohovor u konkrétního zaměstnavatele či pro určitou pracovní pozici.

Vedlejším cílem je seznámení se zvolenými technologiemi a postupy, které se uplatňují v praxi při vývoji software.

Členění a metodika práce

Úvodní dvě kapitoly této práce se věnují teoretické části. První z nich se zabývá zhodnocením existujících řešení, druhá se poté zabývá analýzou požadavků na cílovou aplikaci, jejím návrhem a současně představením technologií, které budou použity v praktické části.

V praktické části jsou popsány implementační detaily za použití zvolených technologií. Obsah je věnován zejména konkrétním problémům a úlohám. Součástí práce jsou výňatky zdrojových kódů, které jsou doprovázeny vysvětlujícím textem.

Závěr práce autor věnuje zhodnocení dosažených výsledků, upozorňuje na možný budoucí rozvoj a detaily, které by mohli být řešeny jiným způsobem.

V práci se mohou vyskytovat technické termíny, které nejsou přeloženy do českého jazyka z důvodu možné změny původního významu. Vybrané termíny budou vysvětleny v poznámkách pod čarou.

2 Průzkum trhu

Tato kapitola je věnována průzkumu trhu. Autor zde zhodnocuje stávající dostupnost alternativ ke zde navrhované aplikaci a rovněž uvádí příklady úspěšných diskusních portálů, které pomáhají definovat budoucí požadavky na kvalitu a funkcionalitu aplikace.

2.1 Podobné aplikace

V posledních letech se na českém webu vyskytly portály, které se zaobírají hodnocením zaměstnavatelů. V úvodní kapitole byly příklady těchto webů uvedeny avšak ani jeden z nich se v době psaní této diplomové práce nevěnoval tématice pracovních pohovorů.

Při dalších pokusech o vyhledání relevantních informací, které se týkají pracovních pohovorů, bylo nalezeno několik výsledků v různých diskusních fórech a portálech inzerujících práci, avšak nebyly nalezeny žádné specializované české weby.

Ani v zahraničním prostředí není snadné najít aplikaci, která by se alespoň podobala cíli této práce. Portál Glassdoor.com má vyhrazenou oblast, která je věnována právě pracovním pohovorům. Nalezneme zde nejčastější otázky z pracovních pohovorů hned z několika oblastí, avšak chybí možnost filtrování či vyhledávání obsahu, zaměstnavatele apod. Stejně jako v českém prostředí lze spatřit nejčastější otázky pracovních pohovorů na serverech, které inzerují pracovní pozice a roztroušeně v různých diskusních fórech.

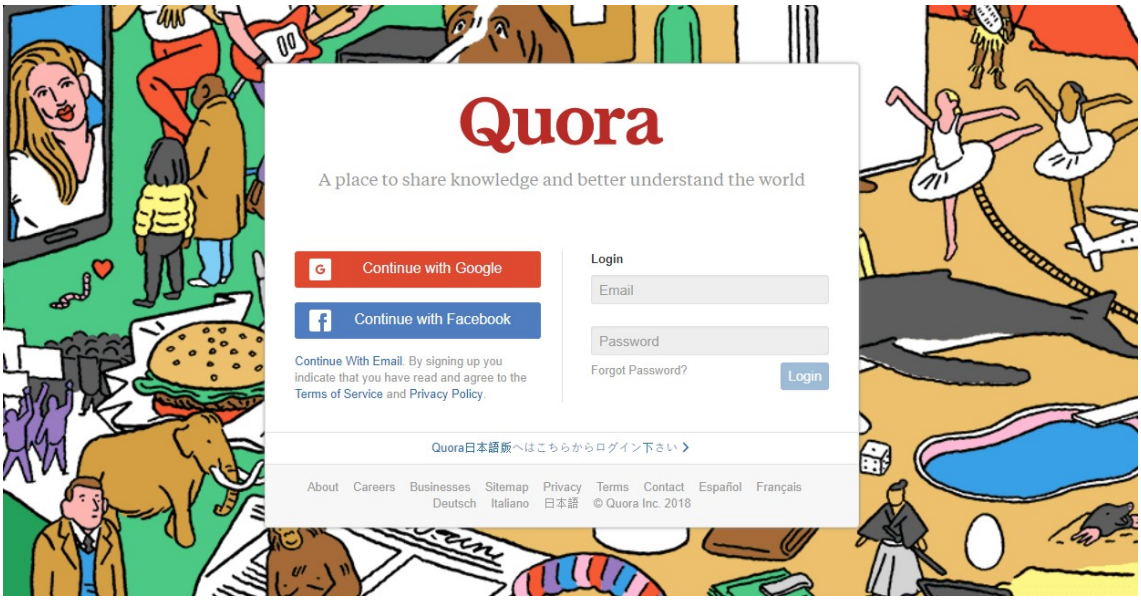
Celkově tedy lze dojít k závěru, že v době psaní této práce neexistuje žádná přímá alternativa.

2.2 Příklady úspěšných knowledge-sharing serverů

Quora

Jedná se o jeden ze známých internetových serverů, které se zabývají *knowledge sharing* (termín používaný pro sdílení znalostí). Pro neregistrované uživatele nabízí jednoduché rozhraní, přes které je možné vyhledávat již dříve položené otázky či témata, která jsou předmětem hledání a zakládat nové otázky.

Fulltextové vyhledávací pole při zadávání výrazu našeptává klíčové fráze, které jsou již na serveru zveřejněny. To značně urychluje získání výsledků. Výsledky lze poté filtrovat pomocí nabízených filtrů, jakými jsou například typ, téma, autor, čas zveřejnění.



Obrázek 1 Úvodní strana webu Quora.com

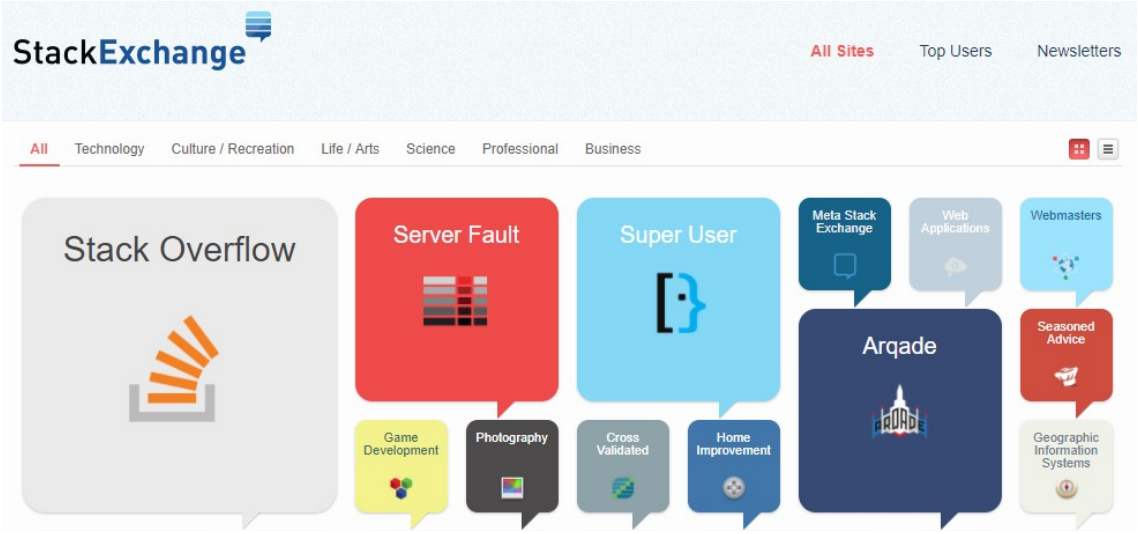
Zdroj: [2] Screenshot aplikace

Založit novou otázku lze jedním kliknutím na tlačítko. Poté se objeví modálové okno, které informuje o zásadách, které by měla otázka splňovat. Před uložením proběhne validace, a pokud je vše v pořádku, dochází ke zveřejnění otázky.

Uživatelé se smí rovněž přihlásit pomocí Google nebo Facebook, nebo se zaregistrovat pomocí emailu. Přihlášeným uživatelům Quora nabízí *feed* z oblastí, které si uživatelé mohou zvolit v průběhu aktivace účtu. Dále mají přihlášení uživatelé možnost na otázky odpovídat a hodnotit odpovědi ostatních.

Stack Overflow

Jde o jeden ze skupiny portálů Stack Exchange, mezi které dále patří například velmi populární Super User, Server Fault či Ask Ubuntu. Na rozdíl od Quory jsou tyto portály úzce zaměřeny na určitý druh témat, čímž se zvyšuje pravděpodobnost získání relevantních informací.



Obrázek 2 Část přehledu portálů, které spadají do skupiny StackExchange

Zdroj: [3] Screenshot aplikace

Pro nepřihlášené uživatele nabízí možnost vyhledávání v tématech. Vyhledávací pole oproti Quoře nenabízí našeptávání, je tedy vhodné zadávat pouze klíčová slova. Lze ovšem využít pokročilé možnosti vyhledávání, jako je hledání přesného výrazu, omezit výsledky na otázky či odpovědi, na počet odpovědí, autora, počet zhlédnutí a mnohé jiné. Výsledky vyhledávání mají

pouze jednoduchou filtraci – například podle relevance, nejnovějších příspěvků atd.

Zakládání otázek je povoleno pouze přihlášeným uživatelům. Po stisknutí tlačítka se objeví stránka, která upozorňuje uživatele, že by měl v první řadě zkusit vyhledávání. Pokud nejsou nalezeny žádné výsledky nebo není uživatel s výsledky spokojen, je zde určitá sada pravidel, které by měly být respektovány při zakládání otázky. Po odsouhlasení těchto pravidel se zobrazí jednoduchý formulář, kde je třeba vyplnit stručné znění otázky, její podrobnější popis a také otázku označit alespoň jedním štítkem.

Ostatní přihlášení uživatelé mohou na otázku reagovat formou vlastních odpovědí, odpovědi hodnotit nebo dále větvit diskuzi k jednotlivým příspěvkům. Zakladatel otázky může odpověď, kterou považuje za nejlepší označit a tím pomoci dalším uživatelům, kteří by se potýkali se stejným problémem, rychleji vyhledat řešení.

3 Analýza požadavků

Tato kapitola je zaměřena na definici funkčních a nefunkčních požadavků na aplikaci. Dále autor definuje aktéry systému a jejich aktivity, které jsou na konci kapitoly znázorněny v Use-Case diagramu.

3.1 Požadavky na aplikaci

Při návrhu každé nové aplikace je vhodné definovat a rozdělit požadavky na výsledný produkt. Tento krok je nutné provést před zahájením implementačních prací na projektu, jinak hrozí zbytečné komplikace, které mohou ohrozit výsledek. Aplikace by měla splňovat nejen požadavky kladené cílovou skupinou/zákazníkem, ale i požadavky systémové. Požadavek by měl určovat, co má aplikace dělat, nikoliv jakou cestou má k výsledku dojít. Rozeznáváme dvě kategorie požadavků.

3.1.1 Funkční požadavky

Definice funkčních požadavků je následující: *„Funkčním požadavkem je formulace toho, co by měl systém dělat – popisuje požadovanou funkci systému“*.[4] Při formulaci funkčních požadavků autor vycházel z vlastních zkušeností a oslovením potencionálních členů cílové skupiny. Aplikace by měla nabídnout následující aktivity:

- Návštěvníci budou mít možnost prohlížet zveřejněné příspěvky včetně celé historie diskuze.
- Návštěvníci budou mít možnost fulltextově prohledávat existující přípěvky a filtrovat je dle zdaných kritérií.
- Uživatelé budou moci příspěvky hodnotit dle jejich užitečnosti.
- Uživatelé budou mít možnost zakládat nové příspěvky.

- Správci obsahu budou moci blokovat uživatele, pokud opakovaně poruší všeobecná pravidla aplikace.
- Správci obsahu budou smět odstranit nevhodné příspěvky.
- Správci obsahu budou mít možnost uzamknout či odemknout uzamčený příspěvek.

3.1.2 Nefunkční požadavky

Druhou kategorii požadavků lze formulovat následující větou: *„Nefunkční požadavek je omezující podmínka uvalená na daný systém.“*.[4] Popisuje způsob implementace zvolených technologií. Často se jedná o omezení plynoucí ze zvolené cílové platformy a druhu zařízení, pro které je aplikace určena. Aplikace by měla dodržet následující nefunkční požadavky:

- Multiplatformita – aplikace by měla být bez omezení zobrazitelná na mobilních zařízeních.
- Modularita a možnost růstu – aplikace by měla být členěna do souvisejících prvků a umožňovat další využití těchto částí, popř. jejich rozšíření.
- Kompatibilita – aplikace musí být navržena tak, aby umožňovala bezproblémové zobrazení v prohlížeči Internet Explorer 11.
- Externí autentizace – aplikace by měla být připravena k využití externích forem autentizace.
- Spring Framework – návaznost na projekt Jobinsider.cz.

Multiplatformita

Jedním z vyjmenovaných funkčních požadavků je multiplatformita. Pod tímto pojmem se v této práci rozumí schopnost provozovat aplikaci nezávisle na hardwarovém a softwarovém vybavení serveru.

Zároveň musí tato aplikace být schopna zobrazit obsah nejen uživatelům notebooků či PC, ale rovněž na menších mobilních zařízeních jako jsou chytré telefony a tablety. Tohoto lze dosáhnout několika způsoby, které budou dále v práci podrobněji vysvětleny.

Modularita a možnost růstu

Systém by měl být navržen a implementován v souladu s principy objektově orientovaného programování. To znamená, že bude možno použít jednotlivé komponenty i v jiných systémech či v jiných případech použití a zároveň nebudou omezovat další rozšíření systému.

Kompatibilita

S ohledem na to, že je zde navrhovaná aplikace stavěná na zelené louce, není třeba dodržet žádnou zpětnou kompatibilitu s datovým modelem apod. Nicméně kompatibilitou z hlediska požadavků na systém se v této práci rozumí kompatibilita s uživatelskou databází webové aplikace JobInsider.cz a kompatibilita směrem k webovým prohlížečům [5].

S přihlédnutím na evoluci frontendových řešení je stanovena minimální podporovaná verze webového prohlížeče Internet Explorer na verzi 13. Dřívější verze si žádaly řadu optimalizačních zásahů, které navyšují realizační čas a kladou jistá omezení při vývoji.

Externí autentizace

Aplikace by měla být schopná ověřit identitu uživatele pomocí externích služeb, jakými jsou například Google či Facebook. Z tohoto důvodu bude zapotřebí připravit a implementovat mechanismy, které si externí autentizace vyžaduje. Předpokládá se využití Spring Social, který usnadňuje autentizaci pomocí sociálních sítí.

Spring Framework

Po dokončení této práce se předpokládá budoucí rozvoj projektu za účasti členů vývojového týmu Jobinsider.cz. Z tohoto důvodu je kladen požadavek na stejný webový aplikační framework.

3.1.3 Aktéři systému

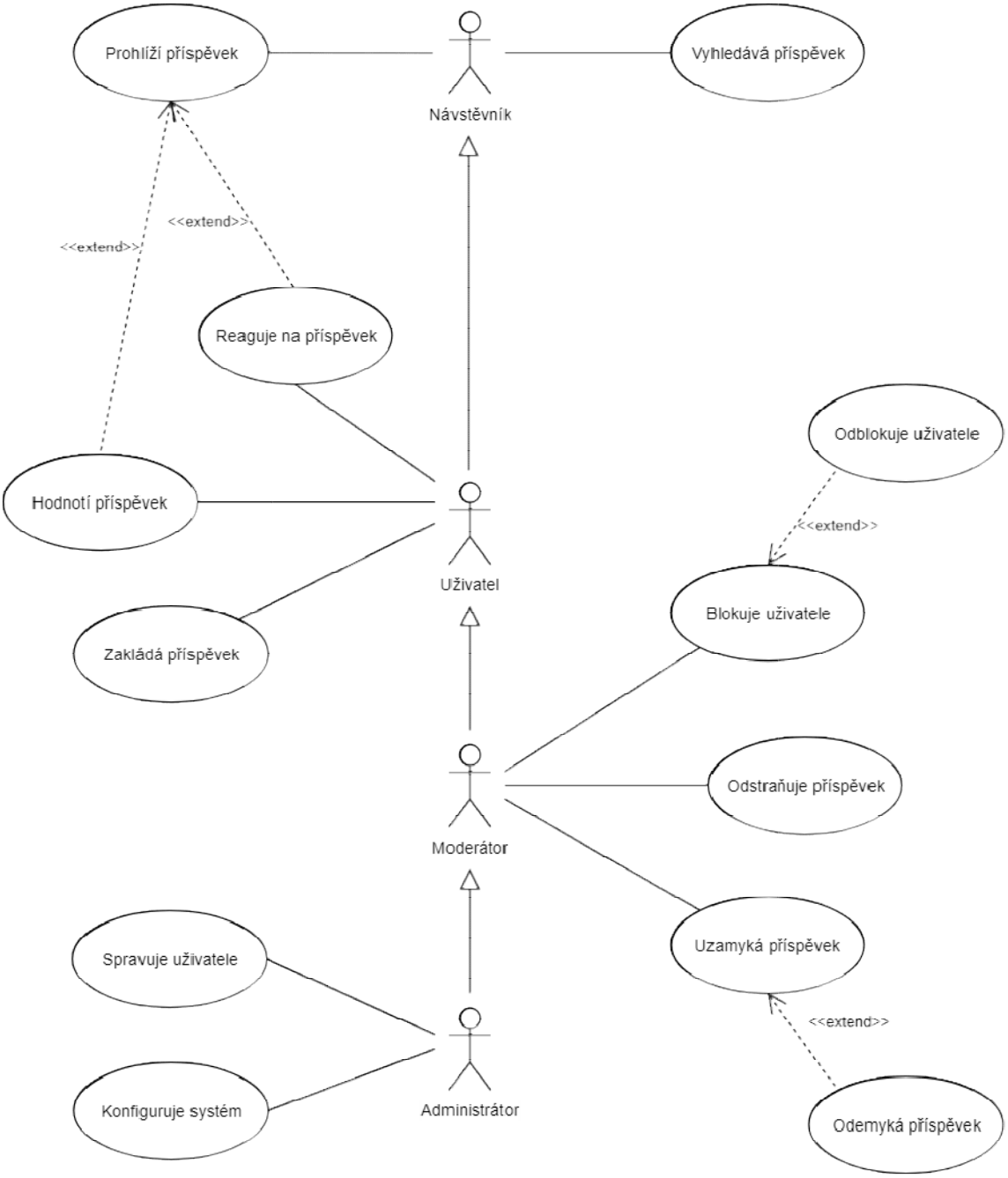
Navrhovaná aplikace je určena pro dvě skupiny uživatelů. První skupinou jsou uchazeči o zaměstnání a zástupci zaměstnavatelů, typicky členové HR oddělení firem. Druhou skupinou jsou moderátoři obsahu. Ve výchozím scénáři se nepředpokládá odlišnost mezi uchazeči a zástupci firem, proto sdílejí stejnou sadu aktivit.

Navrhovaná aplikace by měla podporovat alespoň následující uživatelské role:

- **Návštěvník (visitor / basic user)** – role reprezentující běžné návštěvníky webu, kteří si chtějí prohlédnout obsah
- **Uživatel (advanced user)** – role zastupující registrovaného uživatele webu. Takový uživatel může zakládat nové příspěvky, hodnotit cizí a reagovat na existující položky. Rovněž si může přizpůsobit svůj uživatelský profil, spravovat své osobní údaje apod.
- **Moderátor (moderator)** – uživatelé v této roli mají za úkol spravovat obsah webu. Kontrolují příspěvky zanechané uživateli, zda li jsou v souladu s pravidly webu
- **Administrátor (admin)** – tito uživatelé mají přístup k veškerým funkcionalitám systému, včetně konfigurace systému apod.

3.1.4 Use-Case diagram

Pro lepší znázornění aktivit uživatelů systému byl vytvořen Use-Case diagram v online nástroji dostupném na adrese http://draw.io .



Obrázek 3 Use-case diagram aplikace

Zdroj: autor

4 Návrh aplikace

Úvodní část této kapitoly je věnována výběru názvu projektu a jeho loga. Druhý celek je orientován na návrh architektury systému.

4.1 Jméno projektu a logo

Nedílnou součástí vývoje software je určení názvu a loga. Tyto dvě věci jsou základními prvky propagace, která pomáhá přilákat uživatele.

Název by měl přímo souviset se službou či zbožím, které produkt nabízí. Dle Kellera by měl název mít následující vlastnosti: „jednoduchost, snadná vyslovitelnost a napsatelnost, známost a smysluplnost, odlišnost, osobitost a neobvyklost“. [6]

Obrázek 4 Logo aplikace

Zdroj: autor

Vzhledem k tomu, že by výsledek této práce měl být dílčí částí projektu Jobinsider.cz byl zvolen název JobInview, který se drží nastavených konvencí a má podpořit dojem celistvosti a ctění tradice.

Název vychází ze spojení anglických slov:

- Job – v překladu „práce“
- Inview – doslovný překlad pro toto spojení neexistuje, ale je možné se na něj podívat jako na „in view“, což by mohlo být přeloženo jako „v pohledu“. Nebo je možné jej považovat za zkrácené slovo „interview“, které znamená „pohovor“ či „rozhovor“.

Logo projektu je složeno ze symbolu a názvu (obr. 4). Symbol představuje položenou otázku a významově tak ještě upřesňuje název.

4.2 Datový model

Následující podkapitoly jsou věnovány návrhu architektury aplikace a současně popisu datové struktury aplikace. Tato struktura vychází z požadavků kladených na systém z kapitoly 3.1.

4.2.1 Modelování systémů

V běžné praxi za tvorbu modelů zodpovídá osoba v pozici softwarového architekta či seniorního vývojáře. Po zhodnocení požadavků uvalených na systém se vytváří analytický model tříd. Tento krok spočívá v identifikaci tříd, základních atributů a operací zajišťujících tzv. business logiku. Dále se zabývá určením dědičnosti, zachycením vztahů mezi jednotlivými třídami a specifikováním multiplicit těchto relací.

Po dokončení analytického modelu se přechází k tvorbě návrhového modelu. Ten rozšiřuje analytický model o implementační třídy a další detaily. V této fázi se aplikují návrhové vzory, doplňují se chybějící relace a upřesňují se atributy a metody analytických tříd a rozhraní.

Cílem modelování je zachycení systému a jeho implementace ve standardizované podobě, což může být zajištěno použitím UML[1]. Model umožňuje rozdělit implementační práce na menší části, které jsou poté předány vývojářům k realizaci. V případě větších projektů umožňuje rozdělit vývoj do fází, kdy je po ukončení jednotlivých etap možné prezentovat dosažené výsledky či nasadit aplikaci do reálného provozu.

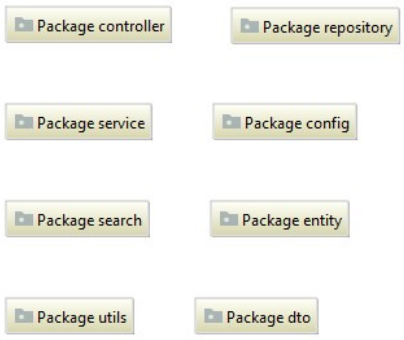
[1] **UML** – Unified Modeling Language – grafický jazyk sloužící k vizuálnímu zachycení systému.

Zároveň jsou modely často součástí dokumentace. To usnadňuje pochopení aplikace novými vývojáři, nebo při integracích s dalšími systémy apod.

4.2.2 Reverzní inženýrství

V rámci této práce se autor rozhodl pro méně tradiční formu reprezentace modelu systému a to s použitím tzv. reverzního inženýrství. Jedná se o přístup, který analyzuje hotové řešení, jeho dílčí součásti a jejich vzájemné propojení. Následně jsou analyzovaná data reprezentována například v grafické formě.

Tento postup je vhodný například pro dodatečnou tvorbu dokumentace, či pro kontrolní účely, zda li byl daný systém implementován dle původního návrhu. V následujících odstavcích jsou příklady diagramů, které lze získat pomocí integrovaných nástrojů v IntelliJ IDEA (kapitola 5.4).



Obrázek 5 Diagram balíčků

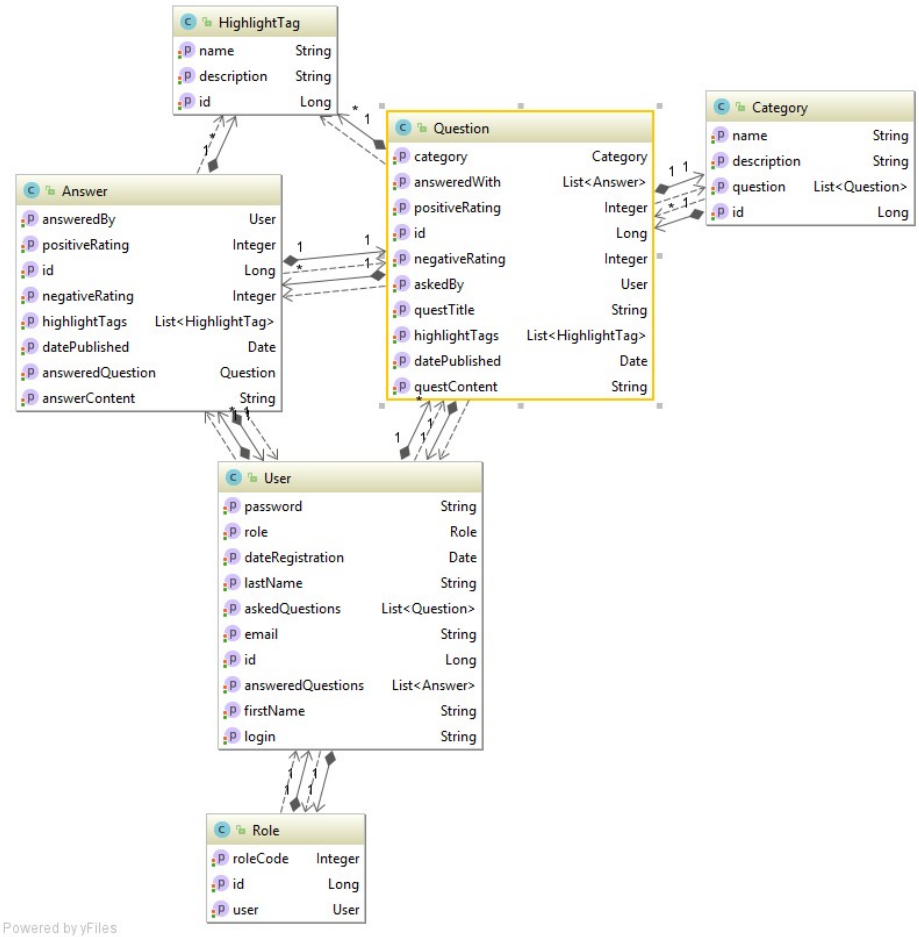
Zdroj: autor

Diagram balíčků

Na obrázku č. 5 je znázorněn diagram balíčků. Jednotlivé balíčky seskupují logicky závislé části aplikace. Z jejich názvu je patrný jejich účel, což přispívá k intuitivnímu třídění kódu.

Diagram tříd

Diagram na obrázku č. 6 představuje diagram tříd z balíčku *entity*. Jak je z obrázku patrné, nástroje pro reverzní inženýrství jsou schopny odvodit relace mezi třídami, atributy jednotlivých tříd a samozřejmě metody, které však autor z důvodu zachování čitelnosti vynechal.



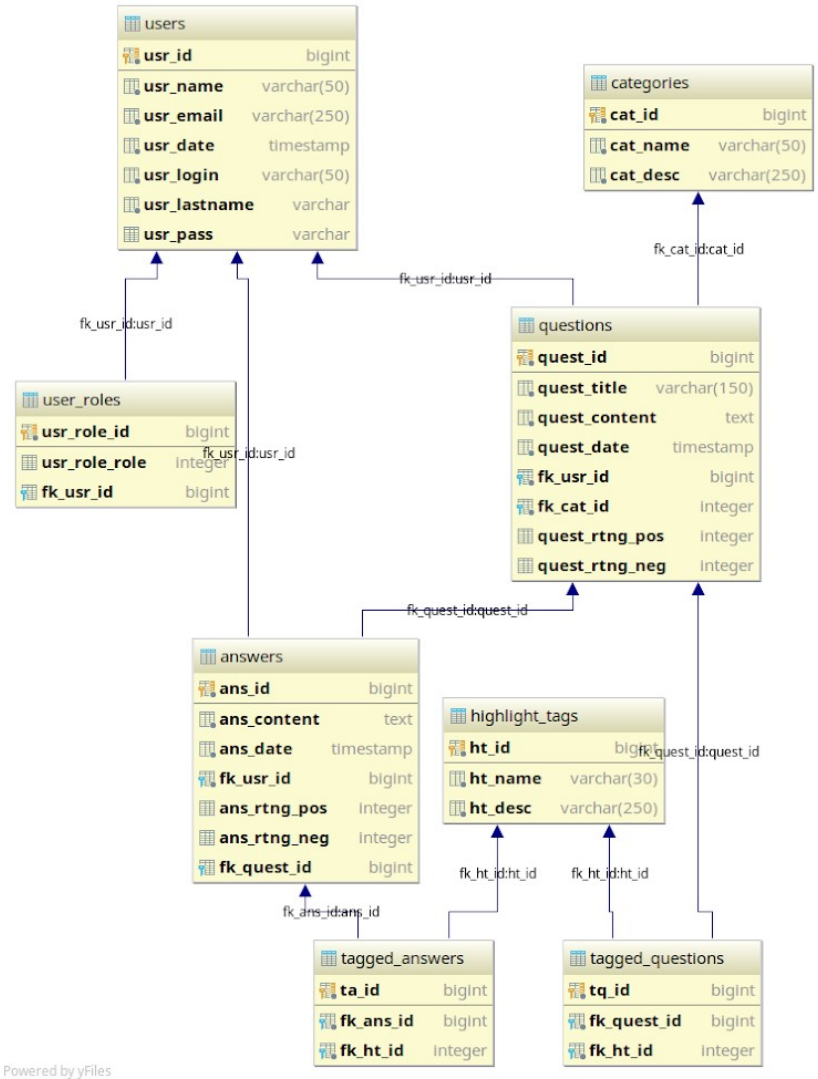
Obrázek 6 Diagram tříd v balíčku *entity*

Zdroj: autor

Vzhledem ke skutečnosti, že jsou tyto diagramy prostorově náročné, nebudou v rámci práce další diagramy tříd prezentovány.

Databázové schéma

I pro generování schématu databáze má IntelliJ IDEA nástroje. Za tvorbu databází často zodpovídají specialisté na pozici databázový vývojář. Nicméně i pro vývojáře aplikací je nutné znát schéma databáze, proto je často součástí zadání projektu.



Obrázek 7 Schéma databáze

Zdroj: autor

4.3 Struktura webu

Dobrá organizace obsahu je důležitou vlastností nejen webových aplikací. Rozhraní by mělo umožnit intuitivní ovládání a snadný přístup k datům. Při návrhu by měl být kladen důraz především na jednoduchost a přehlednost.

Zároveň s požadavkem na zobrazení napříč jednotlivými druhy zařízení je potřeba optimalizovat rozhraní pro použití v různých rozlišeních. K dosažení lze využít v podstatě dvě varianty.

Mobilní verze webu

První možností je vytvoření dvou oddělených verzí webu. Jedna z nich je určena pro zobrazení na klasických zařízeních, jakými jsou osobní počítače a notebooky. Ta druhá pak pro zařízení jako tablet či mobilní telefon. Tato možnost je řešena tzv. detekcí zařízení, což vede k přesměrování na správnou verzi webu. Často bývá možnost manuálně přepnout mezi těmito verzemi pomocí odkazu či tlačítka.

Výhodami tohoto řešení je omezení odesílaného množství dat, přizpůsobení jinému druhu ovládání či možnost poskytovat pouze část obsahu. Nevýhody plynou z nutnosti vytvořit dvě separátní verze, což vede ke zvýšení nákladů na vývoj apod. Dále může docházet ke špatné detekci zařízení, což vede k zobrazení špatné verze webu a tím diskomfortu uživatele.

Responsivní design

Druhou variantou je tzv. responsivní design. V tomto případě zůstává pouze jeden web, který nerozlišuje cílové zařízení. Nicméně jednotlivé díly jsou navrženy tak, že respektují velikost zobrazovací oblasti a tomu přizpůsobují vlastní velikost.

Výhodou je jednotný vývoj jak pro klasické tak i mobilní zařízení. Svým návrhem umožňuje pohodlně zobrazit obsah i při menší šířce okna prohlížeče. Jako nevýhodu lze považovat nemožnost optimalizace množství odesílaných dat pro mobilní zařízení.

Aplikace realizovaná v rámci této práce využívá kombinace responsivního designu a samostatné mobilní aplikace.

5 Výběr technologií

Před přechodem od návrhu do implementace musí být jednoznačně určeny technologie, které budou použity pro vývoj aplikace. Od tohoto kroku se odvíjí například alokace vývojářských kapacit, přesnější odhady časové náročnosti a nové cenové odhady. Výběr technologie rovněž stanoví možnosti dalšího rozvoje aplikace a integrace s jinými informačními systémy.

5.1 Backendové technologie

V následujících několika bodech jsou shrnuty použité backendové technologie, které mají za úkol poskytovat a uchovávat data, zajišťovat aplikační logiku a komunikaci s dalšími službami.

5.1.1 Spring Framework

V návaznosti na projekt JobInsider byl stanoven nefunkční požadavek na využití Spring frameworku, což má za cíl usnadnit vzájemnou integraci obou projektů.

Jedná se o open-source aplikační framework určený pro vývoj Java EE aplikací, který je charakteristický použitím návrhového vzoru *Inversion of Control (IoC)*[2]. Je navržen velice modulárně, díky čemuž najde uplatnění od malých projektů, až po ty velké, které naplno využijí možností tohoto frameworku. [7]

Lze jej kategorizovat jako kontejnerově založený framework. Velmi zjednodušeně kontejner spravuje veškeré „živoucí" objekty (tzv. Java Beans), ze

[2] **Inversion of Control (IoC)** – obecný návrhový vzor, který přesouvá odpovědnost za vytváření a propojování objektů z aplikace na framework pomocí tzv. Dependency Injection. Jsou známy tři způsoby vkládání objektů a to Setter injection, Constructor injection a Interface injection.

kterých je aplikace složena. O tyto objekty se stará po celý životní cyklus. Zajišťuje jejich vytváření, provázání a konfiguraci.

Konkrétně bude použit Spring Boot. Oficiální webové stránky jej ve volném překladu charakterizují následovně: *„Spring Boot umožňuje snadno vytvářet samostatně stojící aplikace založené na Springu v produkční kvalitě, které stačí jen spustit. Spojili jsme vybrané části Spring platformy a knihovny třetích stran, takže je snadné začít s minimem rozruchu. Většina Spring Boot aplikací vyžadují pouze malou Spring konfiguraci.“*. [8]

5.1.2 Výběr programovacího jazyka

Skutečnost, že Spring framework je určený především k vývoji pro Java platformu nijak neomezuje možnost použít k psaní kódu jiný programovací jazyk určený pro Java Virtual Machine (dále jen JVM). Mezi takové jazyky můžeme zařadit například Groovy, Scala či Kotlin.

Java 7-8-9

Není nutné dlouze popisovat tento programovací jazyk, jenž se za svoji více než dvacetiletou historii stal velmi populárním a hojně využívaným. Jedná se o objektově orientovaný jazyk, nezávislý na architektuře, se silnou typovou kontrolou.

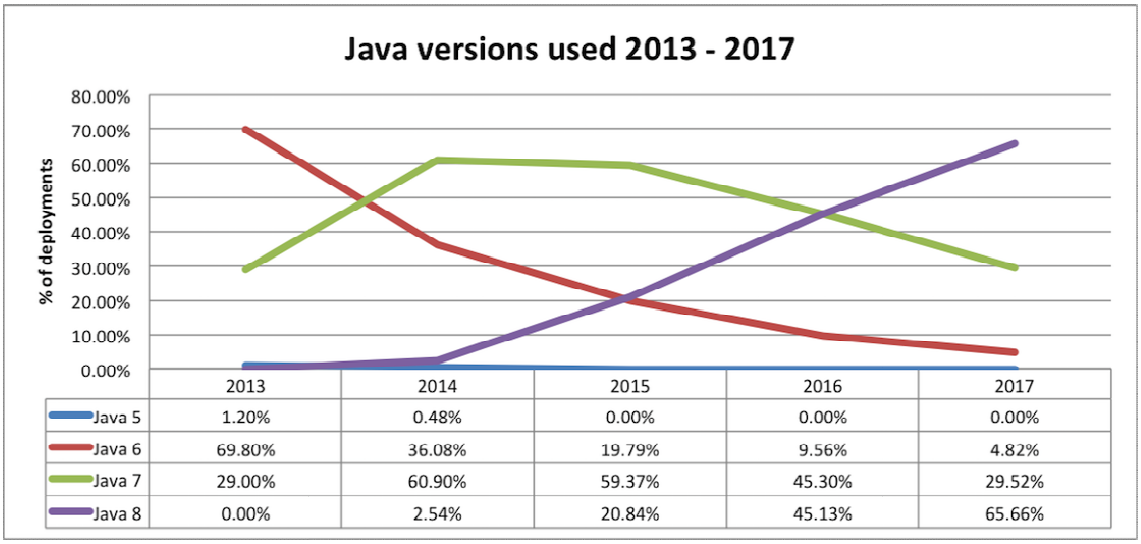
Java 7 byla publikována již v červenci roku 2011, a vzhledem k času, který uplynul ji lze považovat za standard, se kterým lze srovnávat ostatní jazyky.

V březnu 2014 následovala verze Java 8, která oproti předchůdci přinesla mnoho změn a vylepšení programovacího jazyka, mezi které je možné zařadit například: [9]

Lambda expressions – obecně jsou považovány za jednu z největších změn. Přináší do Javy funkcionální přístup k programování. Velmi zjednodušeně řečeno jsou lambda výrazy kusy kódu, které lze přiřadit do proměnné a dále s nimi zacházet jako s objektem. V mnoha případech pomáhají zkrátit a zpřehlednit kód (zejména pak v kombinaci se Stream API), vyhnout se anonymním třídám apod. Nicméně lambda výrazy mají v Javě jistá omezení – jedním z nich je například skutečnost, že proměnné použité v lambda výrazu musí být *final* nebo tzv. *effective final* a v některých případech je tak nutné použít *holder*.

- **Default methods** – jedná se o metody v rozhraních, které kromě hlavičky mají i implementaci. Toto rozšíření bylo v zásadě nutné pro podporu funkcionálního programování, jelikož díky rozšíření rozhraní o defaultní metodu již není nutné modifikovat veškeré implementační třídy tohoto rozhraní.
- **Method references** – umožňuje odkazovat na metody pomocí operátoru „::“. Lze odkazovat na statické metody, metody instance a konstruktory, pomocí klíčového slova „*new*“.
- **Stream API** – Kolekce v Javě byly rozšířeny o tzv. stream API. Jedná se o proudové zpracování dat, které velice snadno umožňuje řetězit operace. Na proudu lze provést hned několik druhů operací. Mezi nejběžnější patří *count, filter, forEach, map, sorted*.
- **Project Nashorn** – jedná se o nový JavaScriptový engine pro JVM od společnosti Oracle, který nahradil dosavadní Rhino engine, jenž se objevil v Javě verze 6. Zjednodušeně řečeno tento engine překládá JavaScriptový kód do Java bytecode. Oracle na svých stránkách prezentuje několikanásobné zvýšení výkonu oproti předchůdci [10].

Naproti tomu Java 9 uvedená v září 2017 dle dokumentace společnosti Oracle přináší jen několik drobných modifikací týkajících se jazyka. Kompletní seznam novinek je dostupný například na stránkách Oracle [11].



	2013	2014	2015	2016	2017
Java 5	1.20%	0.48%	0.00%	0.00%	0.00%
Java 6	69.80%	36.08%	19.79%	9.56%	4.82%
Java 7	29.00%	60.90%	59.37%	45.30%	29.52%
Java 8	0.00%	2.54%	20.84%	45.13%	65.66%

Obrázek 8 Graf využití jednotlivých verzí jazyka Java v rozmezí let 2013-2017

Zdroj: https://plumbr.io/blog/java/java-version-and-vendor-data-analyzed-2017-edition

Kotlin

Kotlin je velmi mladý programovací jazyk vyvinutý společností JetBrains, který je primárně, nikoliv však výhradně[3], určen pro JVM. Rozsáhlého uplatnění našel například v oblasti vývoje mobilních aplikací pro platformu Android. [12]

Jedná se o objektově orientovaný, staticky typovaný jazyk, který však uplatňuje některé principy funkcionálního programování. Syntaxe tohoto jazyka není kompatibilní se zápisem Javy, avšak Kotlin umožňuje využívat knihovny a třídy Javy. Tento přístup umožňuje poměrně snadný a nenásilný přechod od Javy ke Kotlinu. Společnost JetBrains navíc do svého IDE (Integrated Development Environment) IntelliJ IDEA zakomponovala konvertor Java kódu do Kotlinu.

[3] Kotlin je dále možné kompilovat pro ART (Android Runtime), do JavaScriptu, nebo pro LLVM (Kotlin Native)

Na první pohled se jedná o jazyk, který se snaží razantně omezit „upovídanost“ kódu. V porovnání s Javou je zápis syntaxe zkrácený, i přesto ale zůstává snadno srozumitelný a lehce čitelný i pro nováčky.

Další významnou charakteristikou tohoto jazyka je odvozování typů. Během psaní kódu není ve většině případů nutné specifikovat datový typ, ten je odvozen při kompilaci.

Kotlin oproti Javě nabízí mnoho dalšího „syntaktického cukru“:

- **Extension functions** – umožňují rozšiřovat třídy bez nutnosti využití dědičnosti či využití návrhového vzoru jakým je například Decorator. Lze je využít například k rozšíření funkcionality cizích knihoven.
- **Null Safety** – na rozdíl od Javy, Kotlin neumožňuje přiřadit do instancí *null* hodnotu. Tento přístup se snaží minimalizovat riziko vzniku NPE (Null Pointer Exception). V případě nutnosti je možné využít operátor „?“, který označí danou proměnnou jako *nullable*.
- **Smart Casts** – v mnoha případech není nutné používat explicitní přetypování, kompilátor dokáže odvodit typ v podmínkovém bloku s pomocí klíčového slova *is*.
- **String templates** – pomocí operátoru *$* lze přímo do řetězce vkládat hodnoty proměnných, není tedy nutné používat String format jako v případě Javy.
- **Data classes** – Kotlin zavádí konvenci pro vytváření tříd, které slouží k namapování dat. Zápis takové třídy je velice krátký a v maximální možné míře omezuje tzv. *boilerplate code*.

Předchozí body jsou jen zkrácený výčet možností, které Kotlin oproti Javě nabízí. Oficiální stránky tohoto jazyka nabízí kvalitně zpracovanou dokumentaci a rovněž možnost prohlédnout a vyzkoušet si příkladové implementace v online Kotlin konzoli.

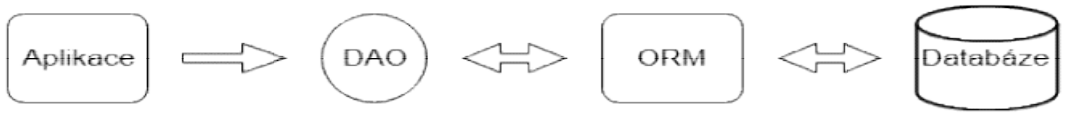
Zajímavá je také možnost použití tzv. inkrementální kompilace. Při klasické kompilaci dochází ke zkompilování všech zdrojových souborů v rámci projektu, nicméně inkrementální kompilace umožňuje zkompilovat pouze ty soubory, které byly změněny od posledního buildu a soubory, které jsou na nich závislé. To samozřejmě může ušetřit kompilační čas v rámci kompilování větších projektů, nicméně ze zkušenosti autora této práce může docházet k neočekávaným anomáliím v běhu aplikace, jelikož je Kotlin stále usilovně vyvíjen a laděn.

5.1.3 PostgreSQL

PostgreSQL je objektově-relační databázový systém, který se řadí mezi nejpokročilejší open-source řešení. Primárně je vyvíjen pro unixové systémy, ale existují varianty pro Windows, Mac OS X a Solaris. [13]

5.1.4 Hibernate

Hibernate ORM je objektově relační mapovací nástroj, jenž je jednou z implementací JPA[4]. V podstatě jde o abstraktní vrstvu nad databází, která umožňuje pracovat nad persistovanými daty v databázi jako s objekty aplikace. [14]



Obrázek 9 Diagram znázorňující komunikace mezi jednotlivými částmi aplikace

Zdroj: autor

[4] **JPA –** Java Persistance API - standart který popisuje rozhraní a chování ORM nástrojů. Mezi jeho implementace patří např. OpenJPA, Hibernate či EclipseLink.

5.1.5 Apache Lucene

Pro snadné vyhledávání obsahu v rámci projektu bylo nutné najít způsob, jak snadno, rychle a efektivně prohledávat založené příspěvky. Jedním z nejznámějším vyhledávácích enginů je Apache Lucene, knihovna která je napsaná v Javě.

Vzhledem k tomu, že získávání dat z databáze je zprostředkováno pomocí Hibernate, přímo se nabídlo řešení využít Hibernate Search [15], který Lucene využívá. Jediné co je zapotřebí udělat, je přidat několik dalších anotací k namapovaným entitním třídám. Hibernate se sám postará o údržbu indexů, aniž by bylo nutné provádět další kroky.

5.2 Frontendové technologie

Následující odstavce jsou věnovány technologiím použitým pro vytvoření grafického uživatelského rozhraní. Výpis není zcela kompletní, a to z toho důvodu, že některé technologie jsou plošně známé. Mezi takové technologie by bylo možné bezesporu zařadit značkovací jazyk HTML, kaskádové styly CSS.

5.2.1 React.js

V květnu 2013 se objevila nová open-source JavaScriptová knihovna zvaná React.js, která v tu dobu byla interně již nějakou dobu používána a vylepšována. Za jejím uvedením stojí Facebook, jenž v současnosti ve velké míře ovlivňuje trendy při vývoji webových aplikací.

Je určena výhradně pro tvorbu uživatelského rozhraní. V návrhové architektuře MVC by tedy zastávala pozici V – tedy zobrazovací vrstvu. V následujících bodech jsou zmíněny zásadní pojmy, týkající se React.js. [16]

Komponenta – React je komponentově založená knihovna. Jedná se o malé kousky kódu, ze kterých se skládá výsledné uživatelské rozhraní. Jedním z hlavních cílů komponent je jejich znovupoužitelnost. Jako příklad komponenty lze uvést například tabulku. Ta je ovšem rovněž složena z komponent a to sice z řádků. Je nutné mít na paměti, že dělení na subkomponenty je vhodné pouze za předpokladu, že nově vytvořená komponenta bude mít logické opodstatnění a zajistí integritu dat.

JSX – jedná se o rozšíření JavaScriptové syntaxe o možnost použití XML / HTML tagů pro vykreslování subkomponent. Nicméně tyto tagy nejsou používány napřímo. Ve skutečnosti se totiž používají JavaScriptové funkce, které jsou velmi podobné standardnímu HTML zápisu. Použití JSX v Reactu není nutné, avšak ve většině dostupných ukázek kódu jej používají, a proto je použit i v rámci této práce.

Virtuální DOM[5] - při zpracování dat nedochází k přímé změně DOMu prohlížeče. Místo toho React používá zjednodušenou reprezentaci, kterou si drží v paměti. Při změně stavu vyhledá změny, které se ve virtuálním DOMu odehrály a ty jsou pak aplikovány do DOMu prohlížeče. Výhoda spočívá v tom, že se nenahrazuje celý DOM ale pouze jeho pozměněné části, což má pozitivní vliv na výkon.

5.2.2 NPM

NPM je správce balíčků pro JavaScript, který spravuje tisíce balíčků. Jeho hlavní úlohou je automatická správa závislostí a balíčků. Podobně jako Maven (nástroj blíže popsaný v kapitole 5.4), má i NPM konfigurační soubor nazvaný package.json, do kterého se zapisují nezbytné závislosti. Poté, zadáním příkazu do CLI[6], proběhne instalace balíčků a vytvoření složky s těmito závislostmi. Mimo to umožňuje snadno sdílet svůj kód s ostatními vývojáři a podporuje myšlenku znovupoužitelnosti kódu.

Výhody tohoto řešení jsou zřejmé. Není zapotřebí ručně spravovat závislosti, stahovat balíčky a umisťovat je do projektu. Zároveň se redukuje velikost projektu, který je distribuován mezi jednotlivými vývojáři prostřednictvím verzovacích nástrojů. To ovšem platí pouze v případě úpravy konfigurace repositáře.

---

[5] **DOM** – Document Object Model – jedná se o objektovou reprezentaci HTML či XML dokumentu, která si klade za cíl sjednotit výslednou reprezentaci obsahu nezávisle na platformě či použitém programovacím jazyce.

[6] **CLI** – Command Line Interface - neboli příkazová řádka. Běžná součást operačních systémů (cmd.exe pro Windows, shell pro unixové systémy) sloužící k ovládání systému či programů skrze textové rozhraní. Často umožňuje provádět složitější operace, než grafické uživatelské rozhraní (GUI).

5.3 Mobilní technologie

Součástí této práce je mobilní aplikace určená pro platformu Android. Striktní omezení platformy je filozoficky v rozporu s požadavkem na platformní nezávislost, nicméně klade si za cíl zlepšit uživatelský komfort majitelů zařízení založených na platformě, která má většinový podíl na trhu (až 87,8% v druhém kvartálu roku 2017 [17]).

Pro ostatní zařízení a uživatele, kteří si nechtějí instalovat aplikaci do svého zařízení je frontend aplikace uzpůsoben tak, že dodržuje zásady responzivního designu.

5.3.1 Android OS

Jde o open-source operační systém založený na linuxovém jádře, který je určen pro mobilní zařízení, jakými jsou například chytré telefony, tablety, chytré hodinky a například multimediální zařízení v automobilech. Původně vznikal pod hlavičkou společnosti Android Inc.. Tato společnost byla v srpnu 2005 odkoupena společností Google. O necelé dva roky později, v listopadu 2007, Google poprvé představil testovací verzi tohoto systému.

První komerční verze Android 1.0 byla vypuštěna v září roku 2008 pod kódovým označením[7] Applebread. Naopak nejnovějším přírůstkem je Android 8.0 s označením Oreo, který byl uveden v srpnu 2017.

[7] U vydávaných verzí systému Android je ve zvyku přidělovat kódová označení v abecedním pořádku. Každé označení je pojmenováno podle názvu sladkostí.

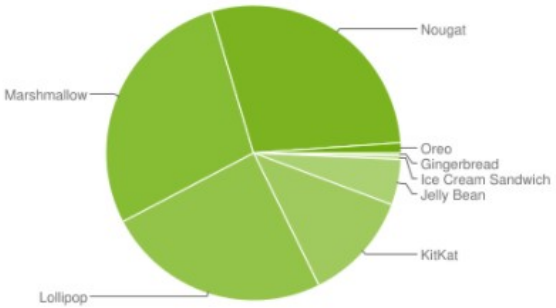
5.3.2 Podporované verze systému

Při vydávání nových verzí systému často dochází k omezení zpětné kompatibility se staršími zařízeními. Při návrhu nové aplikace je vždy nutné pečlivě zhodnotit situaci na trhu a zvolit minimální podporovanou verzi systému dostatečně vysokou, ovšem při zachování co největší části uživatelské základny.

Na obrázku č. 10 je možné shlédnout informace o zastoupení jednotlivých verzí operačního systému. Pro zde navrhovanou aplikaci byla zvolena minimální verze podporovaného systému na API level 23. Tato volba umožňuje oslovit více než polovinu uživatelů.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%

Data collected during a 7-day period ending on February 5, 2018.
Any versions with less than 0.1% distribution are not shown.



Obrázek 10 Procentuální zastoupení jednotlivých verzí OS Android zveřejněné 5. února 2018

Zdroj: https://developer.android.com/about/dashboards/index.html

5.3.3 Běhové prostředí

Do verze systému 4.3 používal Android virtuální stroj Dalvik. Důvodem pro vznik tohoto virtuálního stroje byly mimo jiné licenční podmínky JVM od Oracle, které omezují šiřitelnost knihoven a jazyku Java, který se primárně používá pro vývoj aplikací.

Od verze 4.4 se vedle Dalviku objevilo nové běhové prostředí ART – Android RunTime, které se stalo hlavním ve verzi systému 5.0. Hlavní změnou oproti Dalviku je jiný způsob kompilace zdrojového kódu. Dalvik využívá JIT[8] kompilace, což znamená, že při každém spuštění aplikace v Dalviku se přeloží Java bytecode na Dalvik bytecode, a ten je poté spuštěn v DVM[9].

ART převážně využívá AOT[10] kompilaci. Zdrojové soubory jsou rovněž zkompilovány do Java bytecode, jenž je přeložen na Dalvik bytecode ale během instalace aplikace je tento přeložen do finálního strojového kódu pro dané zařízení.

Mezi hlavní výhody AOT kompilace patří významně rychlejší běh aplikací a nižší energetická náročnost. Nevýhodou je zpomalení instalace, při které se provádí optimalizace pro dané zařízení a o něco vyšší nároky na úložiště, jelikož je potřeba uschovat zkompilovaný kód.

Jako další výhody ART jsou uváděny optimalizace Garbage collectoru a vylepšená podpora pro ladění programů, která zahrnuje podrobnější výpisy výjimek a reportování kolizí.

---

[8] **JIT** – Just-in-time kompilace – probíhá až v momentě, kdy se budou používat zdrojové soubory

[9] **DVM** – Dalvik Virtual Machine – virtuální stroj, ve kterém běží Dalvik bytecode

[10] **AOT** – Ahead-of-time kompilace – probíhá v předstihu před samotným používáním zdrojových souborů

5.3.4 Úvod do architektury systému

Na obrázku č. 11 je znázorněna architektura systému Android. Pro vývoj aplikací je nejdůležitější vrstva Java API Framework, která poskytuje přístup ke službám. Důležitý je fakt, že vývojáři mají přístup ke stejným API jako systémové aplikace.

Obrázek 11 Vyobrazení jednotlivých prvků architektury OS Android

Zdroj: https://developer.android.com/guide/platform/index.html

- **View System** – sada prvků, které jsou používány k vytvoření uživatelské rozhraní. Prakticky nahrazuje standardní Java knihovny pro tvorbu uživatelského rozhraní (AWT, Swing). Patří mezi ně mřížky rozložení, seznamy, textová pole, tlačítka, zaškrtávací pole a mnohé další.
- **Content Providers** – umožňuje přístup k obsahu jiných aplikací, například kontaktů apod. Rovněž slouží ke sdílení vlastních dat, které jsou tímto způsobem dosažitelné pro ostatní aplikace.
- **Resource Manager** – poskytuje přístup k nekódovým zdrojům aplikace. Mezi tyto zdroje patří lokalizační soubory, grafika a soubory grafického rozvržení.
- **Notification Manager** – s jeho pomocí lze zobrazovat vlastní upozornění na stavovém řádku. Pokud má zařízení k dispozici stavovou LED diodu, obsluhuje její činnost.
- **Activity Manager** – má na starosti životní cyklus aplikace. Umožňuje přecházet mezi aplikacemi v zásobníku.
- **Package Manager** – poskytuje rozhraní, kterým lze zjistit a popřípadě upravit informace o nainstalovaných balíčcích a jejich oprávněních. Některé části tohoto API jsou přístupné pouze systémovému uživateli.
- **Location Manager** – jeho prostřednictvím lze periodicky přistupovat k údajům o poloze zařízení. Také dokáže vyvolat událost, pokud se zařízení dostane do určité geografické oblasti. To vše lze poskytnout pouze za předpokladu, že daná aplikace má požadované oprávnění.
- **Window Manager** – tato služba je odpovědná za seřazení oken aplikací. Určuje viditelnost a umístění v rámci pracovní plochy. Rovněž se stará o přechody a animace v situacích, kdy se otevírají či zavírají aplikace, nebo pokud dojde k otočení zařízení.

- **Telephony Manager** – zprostředkovává informace o SIM kartě, o síti ke které je zařízení přihlášeno a o stavu zařízení vůči sítím (např. připojení k datové síti, vytáčení čísla apod.).

5.3.5 Android Virtual Device Manager

Součástí Android SDK[11] je AVD Manager. Tento nástroj umožňuje vytvářet, upravovat či mazat virtuální konfigurace mobilních zařízení. V základu nabízí předdefinované zařízení, není tedy nutné vytvářet vlastní. Poskytuje možnost upravovat softwarové i hardwarové vybavení, např. typ procesoru, rozlišení displeje, cílový API level apod.

Tyto virtuální zařízení lze využívat v emulátoru pro simulaci chování a testování aplikace. Emulované zařízení nemusí přesně odpovídat chování reálného zařízení, nicméně pro základní otestování je emulátor nenahraditelný.

5.3.6 Distribuce aplikací

Android jako otevřená platforma umožňuje distribuci aplikací více způsoby, které lze mezi sebou kombinovat. Nejobvyklejším distribučním kanálem je oficiální *marketplace* Google Play. Tento způsob šíření umožňuje získání aplikace všem uživatelům systému Android, jelikož aplikace Google Play je jednou ze základních systémových aplikací.

Dalšími benefity Google Play jsou možnost distribuování placených aplikací a omezení neautorizovaných instalací prostřednictvím licenční služby. Nicméně pro využití tohoto distribučního kanálu je nutné být registrovaným vývojářem a zaplatit malý poplatek.

---

[11] **SDK** – Software Development Kit – sada vývojových nástrojů, které umožňují tvorbu aplikací pro různé platformy, frameworky, operační systémy apod.

Alternativou ke službě Google Play je distribuce skrze email či vlastní webové stránky. V tomto případě je nutné ošetřit neautorizované šíření a v případě placených aplikací i způsob vybírání poplatků. Uživatelé, kteří si chtějí nainstalovat aplikaci z těchto neoficiálních distribučních kanálů, musí na svém zařízení povolit instalaci aplikací z neověřených zdrojů.

5.4 Ostatní použité technologie

Následující odstavce jsou věnovány technologiím, které jsou nezbytné pro vývoj zde navržené aplikace a které nebylo možné zařadit do předchozích kapitol.

Git

Verzování a správa zdrojového kódu je nedílnou součástí moderního vývoje aplikací. Git je jedním z verzovacích nástrojů, mezi další patří například Mercurial a SVN. Hlavní funkcí je správa zdrojových kódů a jejich historie, které jsou distribuovány ke každému vývojáři. V praxi to znamená, že každý vývojář má vlastní kopii obsahu repositáře a jeho kompletní historii.

Git umožňuje snadné sdílení kódu mezi vývojáři a podporu pro nelineární vývoj. To usnadňuje kooperaci více vývojářů nad jedním projektem a pomáhá řešit konflikty při práci nad stejnou částí zdrojového kódu. Díky uchovávání kompletní historie změn není problém se pohybovat napříč historií a snadno se vracet k poslední funkční verzi.

Rovněž Git nabízí možnost větvení a následné slučování kódu z jednotlivých větví a řešení konfliktů pomocí externích nástrojů, jakým je například Meld[12].

[12] Meld je open-source nástroj pro linuxové systémy, který umožňuje porovnávat soubory, adresáře a repositáře verzovacích nástrojů. Podporuje např. tyto verzovací nástroje - Git, Mercurial, SVN.

Zajímavá technika je tzv. *„cherry-picking“*, která umožňuje vybrat jen některé změny z jiné větve.

Git také umožňuje vyloučit určité soubory z hlídaného repositáře. Lze tak učinit jejich vyjmenováním v souboru .gitignore. Lze vyloučit jednotlivé soubory, či adresáře, ale i soubory odpovídající masce. Tato funkcionalita je vhodná pro vyloučení zkompilovaných souborů, logů a dalších nepotřebných souborů.

V současnosti je na internetu dostupné množství serverů, které nabízejí hosting repositáře zdarma. Mezi takové patří například BitBucket, CodeBase, GitHub, GitLab, atd. Pro účely projektu byl zvolen BitBucket, na kterém jsou hostovány i zdrojové kódy projektu JobInsider.

Apache Maven

Maven je nástroj pro automatickou správu závislostí a automatizaci procesu kompilace, který byl vyvinut organizací Apache Software Foundation. Jde o alternativu k nástrojům Ant a Gradle.

Mezi jeho další funkce patří například verzování buildů, což je důležité z hlediska testování a následného nasazování na produkční prostředí. Dále umožňuje použít různé profily, díky kterým lze z jednoho místa ovlivnit například místní konfiguraci pro vývojové a produkční prostředí.

IntelliJ IDEA

Jde o velice populární vývojové prostředí pro vývoj aplikací především v jazyce Java. Za jeho vývojem stojí společnost JetBrains, která již byla zmíněna v této práci v souvislosti s vývojem jazyka Kotlin. Mezi alternativy lze zařadit např. Eclipse, NetBeans, JDeveloper a BlueJ.

Tvůrci jej popisují jako produkt, který je ze všeho nejvíce zaměřen na maximální zvýšení produktivity vývojářů. Je výjimečný zejména kvůli silné

analýze kódu, širokým možnostem přizpůsobení a integraci užitečných nástrojů. Dalším bonusem je systém zásuvných modulů, díky kterým lze rozšířit možnosti tohoto nástroje. [18]

Prostředí má integrovaný terminál, který lze používat místo systémového CLI, bez opuštění vývojového prostředí. Dále jsou přítomny testovací nástroje, vestavěný dekompilátor Java kódu a jak již bylo zmíněno v kapitole 5.1.2, konvertor Java kódu do Kotlinu.

Toto prostředí lze používat na operačních systémech Windows, Linux a macOS. JetBrains jej distribuuje ve dvou variantách. První variantou je Community edice, která je volně šiřitelná. Integruje základní sadu nástrojů, mezi něž patří podpora programování v Javě, Kotlinu, Groovy a Scale. Dále umožňuje využívat vestavěné sestavovací nástroje jako Maven, Gradle a SBT a verzovací systémy jakými jsou Git, SVN apod.

Zajímavostí Community edice je fakt, že je nad ní postaven současný doporučovaný nástroj pro vývoj Android aplikací – Android Studio. Zároveň je možné ji používat i pro komerční účely.

Autor práce k vývoji použil Ultimate edici, která rozšiřuje sadu nástrojů o podporu nejběžnějších webových frameworků (Spring, Java EE, GWT, Vaadin, atd.), databázové nástroje, které jsou schopny pracovat s většinou běžně používaných databázových systémů (MySQL, PostgreSQL, H2, Oracle, atd.), pokročilé možnosti vývoje v JavaScriptu a šablonovacích enginech (FreeMarker, Velocity).

Uživatelskou přívětivost umocňuje možnost přizpůsobení klávesových zkratek pro provádění téměř veškerých operací s kódem. Registrovaní uživatelé mají možnost synchronizovat lokální nastavení a nainstalované zásuvné moduly se svým účtem. Je tak velmi jednoduché přecházet mezi jednotlivými pracovními stanicemi se zachováním stejné konfigurace prostředí.

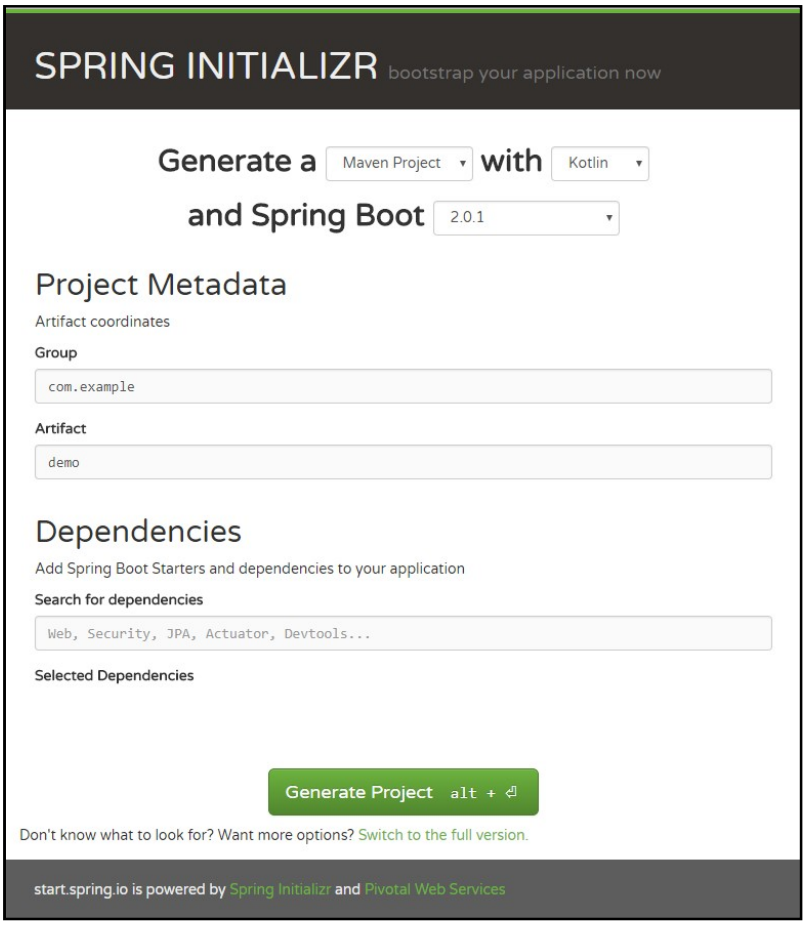
JetBrains rovněž nabízí možnost využít tzv. EAP (Early Access Program) edice. Jedná se o předprodukční sestavení. Výměnou za možnou nestabilitu prostředí mohou vývojáři bezplatně využívat Ultimate edici po omezenou dobu.

6 Implementace

Kapitola popisuje vybrané části vývoje aplikace. Velká část je věnována implementačním detailům, které jsou nezbytné pro realizaci aplikace dle předchozího návrhu.

6.1 Založení Spring Boot projektu

Založení projektu ve Spring Boot je velice jednoduché. Tvůrci totiž vytvořili podprojekt Spring Initializr, který dokáže sestavit kostru aplikace dle parametrů, které si uživatel zvolí. Na obrázku č. 12 je vidět základní webové rozhraní.



Obrázek 12 Webové rozhraní pro vygenerování Spring Boot projektu
Zdroj: https://start.spring.io/

Jak je z obrázku patrné, je možné určit typ projektu, a to konkrétně Maven či Gradle projekt. Dále je možné zvolit programovací jazyk, ve kterém bude aplikace vyvíjena. Výběr je možný mezi Javou, Kotlinem a Groovy.

Rozhraní rovněž umožňuje vybrat verzi Spring Bootu, nastavit výchozí Group ID a Artifact ID a vybrat si závislosti projektu, jakými mohou být například další součásti Spring frameworku, ORM nástroje atd.

Výhodou při vývoji v IntelliJ IDEA (kapitola 5.4) je možnost vygenerovat projekt přímo v rámci prostředí. Není tedy zapotřebí opouštět vývojové prostředí, což velmi zlepšuje komfort vývoje.

6.2 Entitní třídy

Entitní třídy reprezentují persistované objekty aplikace. V případě této aplikace se nacházejí ve společném balíčku entity. V současném stavu se aplikace skládá z šesti entit. Těmi jsou *Uživatel, Role, Otázka, Odpověď, Štítek a Kategorie*. Lze však předpokládat budoucí rozšíření.

Jak již bylo uvedeno dříve, pro vývoj této aplikace se používá programovací jazyk Kotlin. Pro demonstrační účely byla vybrána entitní třída *Kategorie*, na které budou vysvětleny základní odlišnosti mezi Javou a Kotlinem.

Na obrázku č. 13 můžeme vidět deklaraci entitní třídy bez bloku importů a označení balíčku. Celá třída je napsána na pouhých čtyřiceti řádcích a obsahuje několik anotací.

- **@Entity** - označuje třídu, která může být namapována na tabulku v databázi. Díky této anotaci JPA spravuje entitní třídy.
- **@Table** - upřesňuje název tabulky v databázi, pod kterou JPA bude hledat data. Tato anotace je však volitelná a v případě jejího vynechání se JPA pokusí název tabulky odvodit z názvu třídy.
- **@GenericGenerator** - Hibernate anotace, která popisuje způsob, jakým jsou generovány identifikátory.
- **@Id** - označuje sloupec s primárními klíči.
- **@GeneratedValue -** zde určujeme generátor.
- **@Column** - určuje sloupec v tabulce. Tato anotace je volitelná stejně jako **@Table**.
- **@JsonIgnore** - informuje knihovnu Jackson, aby ignorovala toto pole při serializaci či deserializaci.
- **@OneToMany –** označuje vazbu 1:N mezi entitami.

Hned za názvem třídy začíná primární konstruktor, který je definován mezi kulatými závorkami. Celkově obsahuje čtyři atributy, kde první atribut je

identifikátor objektu, díky kterému bude později možné objekt vyhledávat, mazat či odstraňovat z rozhraní webové aplikace.

Na obrázku č. 14 je třída *Kategorie* napsaná v Javě. Celková délka třídy je pak 84 řádků, což je oproti Kotlinu dvojnásobek. Úspornost je dána především automaticky generovanými gettery a settery, které v Javě zabírají mnoho místa.

```
@Entity
@Table(name = "categories")
data class Category(
    @GenericGenerator(
           name = "catSequenceGenerator",
           strategy =
"org.hibernate.id.enhanced.SequenceStyleGenerator",
           parameters = arrayOf(
                Parameter(name = "sequence_name", value = "cat_seq"),
                Parameter(name = "initial_value", value = "1000"),
                Parameter(name = "increment_size", value = "1")))
        @Id
        @GeneratedValue(generator = "catSequenceGenerator")
        @Column(name = "cat_id")
        var id: Long? = null,

        @Column(name = "cat_name")
        var name: String = "",

        @Column(name = "cat_desc")
        var description: String = "",

        @JsonIgnore
        @OneToMany(mappedBy = "category", cascade =
arrayOf(CascadeType.ALL))
        var questions: MutableList<Question> = mutableListOf()

) : Serializable
```

Obrázek 13 Ukázka kódu entitní třídy napsané v Kotlinu

Zdroj: autor

```java
@Entity
@Table(name = "categories")
public class Category implements Serializable {

   @GenericGenerator(
           name = "catSequenceGenerator",
           strategy =
"org.hibernate.id.enhanced.SequenceStyleGenerator",
           parameters = {
               @Parameter(name = "sequence_name", value = "cat_seq"),
               @Parameter(name = "initial_value", value = "1000"),
               @Parameter(name = "increment_size", value = "1")
           }
   )
   @Id
   @GeneratedValue(generator = "catSequenceGenerator")
   @Column(name = "cat_id")
   private Long id;

   @Column(name = "cat_name")
   private String name;

   @Column(name = "cat_desc")
   private String description;

   @OneToMany(mappedBy = "category", cascade = { CascadeType.ALL })
   private List<Question> question;

   public Category() {

   }

   public Category(String name, String description) {
      this.name = name;
      this.description = description;
   }

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   /* Ostatní gettery a settery */

}
```

Obrázek 14 Ukázka kódu entitní třídy napsané v Javě

Zdroj: autor

6.3 Využití JPA Repository

Repository je návrhový vzor, který popisuje základní operace s entitami. V rámci projektu bylo využito generického rozhraní *JpaRepository<T, ID>*. Toto rozhraní rozšiřuje několik dalších rozhraní, mezi které patří také *CrudRepository*, které definuje CRUD operace (Create, Read, Update, Delete).

Díky využití tohoto rozhraní není nutné implementovat jak CRUD operace, tak i vyhledávání a získávání entit z databáze. Přesto se však mohou vyskytnout situace, kdy je nutné vlastní funkcionalitu implementovat.

```
@Repository
interface UserRepository : JpaRepository<User, Long>,
UserRepositoryCustom

interface UserRepositoryCustom {
    fun findByLogin(login: String): User?
    fun findByEmail(email: String): User?
    fun isLoginOccupied(login: String): Boolean
    fun isEmailOccupied(email: String): Boolean
}
```

Obrázek 15 Ukázka kódu rozhraní, jenž rozšiřuje rozhraní JPA Repository

Zdroj: autor

Obrázek č. 15 znázorňuje deklaraci dvou rozhraní pro přístup k datům. Rozhraní *UserRepositoryCustom* popisuje metody, které budou v rámci aplikace implementovány ručně. Implementace rozhraní je pak na obrázku č. 16. *EntityManager* umožňuje vytvořit JPQL[13] dotaz, kterým získáme požadované výsledky.

[13] **JPQL** – Java Persistance Query Language – dotazovací jazyk, který je součástí JPA. Na rozdíl od SQL není určen pro dotazování nad databází ale nad entitami.

```
class UserRepositoryImpl : UserRepositoryCustom{

    @PersistenceContext
    lateinit var entityManager: EntityManager

    override fun findByLogin(login: String): User? {
        return entityManager.createQuery("SELECT obj FROM User obj
WHERE obj.login LIKE '$login'", User::class.java).singleResult
    }

    override fun findByEmail(email: String): User? {
        return entityManager.createQuery("SELECT obj FROM User obj
WHERE obj.email LIKE '$email'", User::class.java).singleResult
    }

    override fun isLoginOccupied(login: String): Boolean {
        val result = entityManager.createQuery("SELECT COUNT(usr) FROM
User usr WHERE usr.login LIKE '$login'").singleResult as Long
        return result != 0L
    }

    override fun isEmailOccupied(email: String): Boolean {
        val result = entityManager.createQuery("SELECT COUNT(usr) FROM
User usr WHERE usr.email LIKE '$email'").singleResult as Long
        return result != 0L
    }
}
```

Obrázek 16 Ukázka kódu rozhraní, které implementuje vlastní metody s využitím dotazovacího jazyka JPQL

Zdroj: autor

6.4 Propojení aplikace a databáze

Díky zvoleným technologiím je velmi snadné propojit aplikaci s databází. V konfiguračním souboru *application.properties* stačí specifikovat druh databáze a parametry spojení. Konfigurace v rámci projektu je na obrázku č. 17.

```
#JPA Config
spring.jpa.database=POSTGRESQL
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreS
QLDialect

#DataSource config
spring.datasource.url=jdbc:postgresql://localhost:5432/jobinview_serve
r
spring.datasource.username=postgres
spring.datasource.password=matrix
```

Obrázek 17 Ukázka konfigurace připojení k databázi

Zdroj: autor

6.5 Mapování požadavků na REST API

Aby mohl klient komunikovat se serverem, je nutné namapovat URL adresy k obslužným metodám REST rozhraní. Spring nabízí několik možností jak mapovat požadavky, a to například dle jmen controllerů a jejich metod.

V projektu se však používá tradiční způsob za pomocí anotací. V ukázce kódu na obrázku č. 18 lze vidět anotaci **@RequestMapping** hned na začátku deklarace třídy. Tato anotace zajistí, že kontroler bude přijímat požadavky, které začínají cestou *„/questions“*. Díky tomu jsou metody uvnitř kontroleru namapované na adresy, začínající touto cestou. Metody mají anotaci ve tvaru ***<Method>Mapping***, kde první část označuje druh HTTP metody, která bude pro odeslání požadavku použita.

```
@RestController
@RequestMapping("/questions")
class QuestionController {

    @Autowired
    lateinit var questionService: QuestionService

    @Autowired
    lateinit var searchService: SearchService

    /* Další atributy */

    @GetMapping
    fun listAllQuestions() = questionService.findAllQuestions()

    @GetMapping("/{id}")
    fun getQuestionById(@PathVariable id: Long) = questionService
        .findQuestionById(id)


    @PostMapping
    fun createNewQuestion(@RequestBody question: Question) {
        val auth = SecurityContextHolder
                    .getContext().authentication

        val currentUser = userService.findUserByLogin(auth.name)
        currentUser?.let {
            question.askedBy = it)
            questionService.saveQuestion(question)
        }
    }

    @PutMapping("/{id}")
    fun updateQuestionById(@PathVariable id: Long, @RequestBody
question: Question) = questionService.updateQuestion(question)

    @DeleteMapping("/{id}")
    fun deleteQuestionById(@PathVariable id: Long) = questionService
        .deleteQuestion(id)

    @GetMapping("/search")
    fun searchForQuestion(@RequestParam(value = "text", required =
true) text: String): MutableList<Question> {
        val results = searchService.search(text)
        return results.filter { it is Question }
                    .map { it as Question }.toMutableList()
    }
}
```

Obrázek 18 Ukázka kódu REST Controlleru

Zdroj: autor

6.6 Autowiring

Autowiring je jedním ze způsobů, jakým lze vkládat závislosti mezi jednotlivými beanami. V případě použití anotace **@Autowired** se Spring pokusí automaticky vložit závislosti.

V ukázce na obrázku č. 18 jsou dva atributy třídy označeny anotací **@Autowired**. V obou případech se jedná o rozhraní. V případě, že by existovalo více implementací těchto rozhraní, je nutné blíže identifikovat implementaci, kterou chceme použít.

6.7 Fulltextové vyhledávání

Projekt svou povahou vyžaduje implementaci vyhledávání obsahu. K tomu byla v rámci návrhu aplikace vybrána knihovna Apache Lucene, respektive Hibernate Search. Konfigurace je obsažena v souboru *application.properties* a je vidět na obrázku č. 19.

```
# SUPPORT OF HIBERNATE SEARCH
# Specify the DirectoryProvider to use (the Lucene Directory)
spring.jpa.properties.hibernate.search.default.directory_provider =
filesystem

# Using the filesystem DirectoryProvider you also have to specify the
default
# base directory for all indexes (make sure that the application have
write
# permissions on such directory)
spring.jpa.properties.hibernate.search.default.indexBase =
/home/michal/lucene/indexes/
#spring.jpa.properties.hibernate.search.lucene_version =
LUCENE_CURRENT
```

Obrázek 19 Ukázka konfigurace Hibernate Search

Zdroj: autor

Následně je zapotřebí přidat další anotace k entitám, které mají mít indexovaný obsah. K demonstraci poslouží ukázka kódu na obrázku č. 20. Prvním krokem je určení analyzéru a to anotací **@AnalyzerDef**. Samotné nastavení je převzato přímo z dokumentace Hibernate Search. Úprava spočívá ve změně filtru, kdy se pro analýzu obsahu používá *CzechStemFilterFactory*, jenž umožní vyhledávat i ve slovech podobných původnímu hledanému výrazu. Druhým krokem je označení atributů, které budou indexovány – k tomu slouží anotace **@Field.**

```
@Entity
@Indexed
@AnalyzerDef(name = "questAnalyzer", tokenizer = TokenizerDef(factory
= StandardTokenizerFactory::class),
        filters = [TokenFilterDef(factory =
LowerCaseFilterFactory::class),
                TokenFilterDef(factory =
CzechStemFilterFactory::class)])
@Table(name = "questions")
class Question(
        @GenericGenerator(name = "questSequenceGenerator",
        strategy = "org.hibernate.id.enhanced.SequenceStyleGenerator",
        parameters = arrayOf(
             Parameter(name = "sequence_name", value = "quest_seq"),
             Parameter(name = "initial_value", value = "1000"),
             Parameter(name = "increment_size", value = "1")))

        /* Další atributy */

        @Field
        @Analyzer(definition = "questAnalyzer")
        @Column(name = "quest_title")
        var questTitle: String = "",

        @Field
        @Analyzer(definition = "questAnalyzer")
        @Column(name = "quest_content")
        var questContent: String = "",
}
```

Obrázek 20 Ukázka konfigurace analyzéru slov

Zdroj: autor

Dále je vhodné zajistit indexování stávajícího obsahu. To je vyřešeno implementací *ApplicationListeneru*, který naslouchá na *ApplicationReadyEvent*. Ve chvíli, kdy listener zachytí tuto událost, využije *EntityManager* k získání přístupu k datům uloženým databázi a indexy uloží do umístění, které bylo definováno v konfiguraci. Samotná implementace je na obrázku č. 21.

V tuto chvíli je možné vytvořit servisní třídu, která se bude v případě požadavku provádět vyhledávání. Na obrázku č. 22 je deklarace třídy, ve které získáme instanci *FullTextEntityManager*. Po sestavení dotazu pomocí *QueryBuilderu* a jeho předání *FullTextEntityManageru* získáme instance, které obsahují hledaný text, nebo prázdnou kolekci.

```
@Component
@Transactional
open class BuildSearchIndex :
ApplicationListener<ApplicationReadyEvent> {

    @PersistenceContext
    lateinit var entityManager: EntityManager

    override fun onApplicationEvent(event: ApplicationReadyEvent) {
        try {
            val fullTextEntityManager =
Search.getFullTextEntityManager(entityManager)
            fullTextEntityManager.createIndexer().startAndWait()
        } catch (e: InterruptedException) {
            println("An error occurred trying to build the search
index: " + e.toString())
        }
    }
}
```

Obrázek 21 Ukázka třídy, která má za úkol indexaci stávajícího obsahu

Zdroj: autor

```
@Repository
@Transactional
open class SearchService {

    @PersistenceContext
    lateinit var entityManager: EntityManager

    open fun search(text: String): MutableList<Any?> {

        val fullTextEntityManager =
Search.getFullTextEntityManager(entityManager)
        val queryBuilder = fullTextEntityManager.searchFactory

.buildQueryBuilder().forEntity(Question::class.java).get()
        val query = queryBuilder
                .keyword()
                .onFields("questTitle", "questContent")
                .matching(text)
                .createQuery()

        val jpaQuery =
fullTextEntityManager.createFullTextQuery(query, Question::class.java)

        return jpaQuery.resultList ?: mutableListOf()
    }
}
```

Obrázek 22 Ukázka servisní třídy, která zajišťuje fulltextové vyhledávání

Zdroj: autor

6.8 Zabezpečení komunikace

Standartně aplikace vytvořená ve Spring Bootu komunikuje pomocí protokolu HTTP. Nicméně doporučuje se použití zabezpečeného protokolu HTTPS, který využívá certifikátu. Tento protokol minimalizuje riziko narušení komunikace mezi serverem a klientem.

V první řadě je nutné získat certifikát. Pro testovací účely autor vygeneroval tzv. self-signed certifikát pomocí nástroje *keytool*, který je součástí Java Runtime Environment.

```
server.port=8443
server.ssl.key-store=keystore.p12
server.ssl.key-store-password=heslo1234
server.ssl.keyStoreType=PKCS12
server.ssl.keyAlias=jobinview_new
```

Obrázek 23 Ukázka konfigurace zabezpečeného spojení HTTPS

Zdroj: autor

V tuto chvíli aplikace přijímá požadavky pouze ze zabezpečeného spojení. Pokud uživatel odešle požadavek nezabezpečeným protokolem, server vrátí chybový status se správou, že očekával šifrovaný obsah. Některé aplikační servery dokonce tyto nevalidní požadavky ignorují a poté může požadavek selhat na vypršení časového limitu pro odpověď.

Tuto komplikaci lze řešit přesměrováním požadavku na zabezpečené spojení. Existuje více možností jak toto vyřešit. V rámci projektu bylo využito možnosti přesměrovat provoz z nezabezpečeného portu na port zabezpečený. Kód, který toto zajišťuje je ke zhlédnutí na obrázku č. 24.

```
@Configuration
open class ConnectorConfig {

    @Bean
    open fun servletContainer(): ServletWebServerFactory {
        val tomcat = object : TomcatServletWebServerFactory() {
            override fun postProcessContext(context: Context) {
                val securityConstraint = SecurityConstraint()
                securityConstraint.userConstraint = "CONFIDENTIAL"
                val collection = SecurityCollection()
                collection.addPattern("/*")
                securityConstraint.addCollection(collection)
                context.addConstraint(securityConstraint)
            }
        }
        tomcat.addAdditionalTomcatConnectors(redirectConnector())
        return tomcat
    }

    private fun redirectConnector(): Connector {
        val connector =
Connector("org.apache.coyote.http11.Http11NioProtocol")
        connector.scheme = "http"
        connector.port = 8080
        connector.secure = false
        connector.redirectPort = 8443
        return connector
    }
}
```

Obrázek 24 Ukázka třídy, která přesměrovává požadavky z nezabezpečeného portu na zabezpečený

Zdroj: autor

6.9 Konfigurace ReactJS

Pro sestavení produkčního buildu je v projektu používán Maven plugin a upravený profil. Konfigurace pluginu a profilu se nachází v soubor *pom.xml.* Konfigurace je patrná z obrázku č. 25.

```
<!-- Ostatní konfigurace, závislosti apod. -->
<plugin>
   <groupId>com.github.eirslett</groupId>
   <artifactId>frontend-maven-plugin</artifactId>
   <version>1.5</version>
   <configuration>
      <nodeVersion>v6.11.1</nodeVersion>
      <workingDirectory>src/main/react</workingDirectory>
   </configuration>
</plugin>

<!-- Ostatní pluginy a profily -->

<profile>
   <!-- Run 'mvn clean install -Dcompile-react' -->
   <id>Install node and npm</id>
   <activation>
      <property>
         <name>compile-react</name>
      </property>
   </activation>
   <build>
      <plugins>
         <plugin>
            <groupId>com.github.eirslett</groupId>
            <artifactId>frontend-maven-plugin</artifactId>
            <executions>
               <!-- Execution bloky jednotlivých NPM goalů -->
            </executions>
         </plugin>
      </plugins>
   </build>
</profile>
```

Obrázek 25 Ukázka konfigurace Maven pluginu a profilu pro React JS

Zdroj: autor

Další konfigurace Reactu se nachází v souborech webpack.build.config.js a webpack.config.js. Tyto soubory upřesňují cesty ke zdrojovým souborům

6.10 Mobilní aplikace – komunikace s REST API

Aktivity mobilní aplikace využívají REST rozhraní webové aplikace JobInView. Je to standartní způsob, jakým mobilní komunikace komunikují se vzdálenými servery. V ukázce kódu na obrázku č. 26 je vidět aktivita, která načítá všechny otázky, které byly v systému založeny. Aktivita má vnitřní třídu *HttpsRequestClass*, která se připojuje zabezpečeným spojením ke vzdálenému serveru. Vzhledem k testovacímu prostředí je zde uvedena IP adresa lokálního stroje, na kterém běží webová aplikace. Rovněž se zde využívá Basic autentizace k ověření identity uživatele. Uživatelské údaje jsou získány prostřednictvím jiné aktivity, které jsou následně sdíleny v rámci aplikace.

```
class QuestionActivity : AppCompatActivity() {

    lateinit var questions: List<Question>
    lateinit var textView: TextView
    lateinit var listView: ListView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_question)
        textView = findViewById(R.id.textView)
        listView = findViewById(R.id.listView)

        val credentials = intent.getStringExtra("credentials")
        HttpsRequestClass(credentials).execute()
    }

    inner class HttpsRequestClass(private val authString: String) :
AsyncTask<Any?, Any?, String>() {

        override fun doInBackground(vararg params: Any?): String? {
            val connection =
URL("https://192.168.111.162:8443/questions/").openConnection() as
HttpsURLConnection
            connection.requestMethod = "GET"
            connection.addRequestProperty("Authorization", authString)
            connection.connect()

            val inputStream = connection.inputStream
            val reader = InputStreamReader(inputStream)

            return reader.readText()
        }

        override fun onPostExecute(json: String) {
            val gson = GsonBuilder().serializeNulls().create()
            val requestResult: List<Question> = gson.fromJson(json,
object : TypeToken<List<Question>>() {}.type)
            textView.text = requestResult.toString()
            questions = requestResult
        }
    }
}
```

Obrázek 26 Ukázka aktivity mobilní aplikace, jenž komunikuje se vzdáleným serverem přes REST rozhraní

Zdroj: autor

7 Shrnutí výsledků

V této práci vznikl návrh a také prototyp webové aplikace, která má za cíl zlepšit přípravu uchazečů o zaměstnání. Aplikace v nynější formě umožňuje zakládat nové příspěvky a rovněž umožňuje uživatelům na tyto příspěvky reagovat.

Aplikace postrádá některé funkcionality, které většina běžných webových aplikací nabízí. Mezi tyto funkcionality patří například ověření identity uživatele prostřednictvím emailu, resetování hesla apod. Tyto prvky si vyžadují zapojení většího kolektivu vývojářů, jelikož aplikace ve stávající podobě je výsledkem práce jednoho vývojáře a jejich vývoj je značně časově náročný.

S ohledem na vývoj s omezenými prostředky aplikace není zdaleka tak uživatelsky přívětivá, jak bylo ve fázi tvorby konceptu předpokládáno. Nicméně autor předpokládá výrazný posun v případě, že se na vývoji bude podílet tým, jenž má na starosti vývoj portálu Jobinsider.cz.

Vystavení aplikace na internet bylo po zvážení zamítnuto. Toto rozhodnutí bylo jednak ovlivněno stavem projektu, který by mohl v současnosti spíše ohrozit budoucí vnímání projektu. Rovněž se pak blíží termín uvedení nařízení k ochraně osobních údajů neboli GDPR (angl. General Data Protection Regulation), které vstoupí v účinnost 25. května 2018.

Toto nařízení si klade za cíl ochranu práv občanů EU proti neoprávněnému zacházení s jejich daty a to zejména osobních údajů. Týká se všech firem a institucí, jednotlivců a online služeb, které nějakým způsobem zpracovávají data uživatelů. Rovněž zavádí vysoké pokuty při porušení pravidel.

Další fáze vývoje by měli být zaměřeny na integraci s portálem Jobinsider.cz. V současné době tento portál prochází rozsáhlými změnami, jejichž dokončení výrazně zlepší možnosti kooperace s dalšími webovými službami a aplikacemi.

Autor práce se tedy rozhodl neimplementovat řešení, které by v budoucnu postrádalo opodstatnění a jehož odstranění by mohlo negativně ovlivnit integraci.

Rovněž by se měl vývoj soustředit na zlepšení uživatelského prostředí a rozšíření uživatelských funkcionalit, jakými jsou například odběry žádaného obsahu či pokročilá správa uživatelských profilů. Aplikace by měla rovněž nabízet určitou formu cílené avšak neobtěžující reklamy. To by mohlo být zajištěno například spoluprací se zaměstnavateli, kteří budou aplikaci rovněž využívat.

8 Závěr

Cílem této práce bylo vytvoření webové aplikace, která bude zaměřena na téma pracovních pohovů. Potencionálním uchazečům o zaměstnání by tato aplikace měla nabídnout možnost získat relevantní informace, které se týkají jednotlivých zaměstnavatelů či profesních pozic. Zaměstnavatelům by měla aplikace sloužit jako prostředek, kterým mohou oslovit případné zájemce a připravit je na budoucí kariéru.

Práce se zabývá popisem jednotlivých fází vývoje takové aplikace. Část práce je věnována analýze požadavků, které jsou kladeny na webovou aplikaci daného zaměření. Další sekce je věnována návrhu řešení a výběru technologií. Následuje implementace dle zvolených technologií, kdy jsou v práci zveřejněny části kódu zabývající se funkcionalitou, které autor považoval jako stěžejní.

Mimo to se práce věnuje popisu jednotlivých technologií. Zahrnuje popis základních myšlenek a představení klíčových bodů, které autora vedly k jejich využití. Část je věnována popisu vývojového prostředí IntelliJ IDEA, které se v době psaní práce řadí mezi špičky na trhu.

V rámci práce se podařilo vytvořit funkční prototyp, který chce autor nadále vyvíjet s podporou dalších vývojářů a jiných nezbytných profesí. Aplikace se klade za cíl zlepšit podmínky při volbě zaměstnání. V případě jejího úspěchu by mohla mít positivní dopad na pracovní trh v České republice.

9 Seznam zdrojů

1. Zaměstnanost, nezaměstnanost. *Český statistický úřad.* [Online] https://www.czso.cz/.

2. *Quora.* [Online] 2018. www.quora.com.

3. All sites. *StackExchange.* [Online] 2018. https://stackexchange.com/sites.

4. **Arlow, Jim.** *UML 2 a unifikovaný proces vývoje aplikací.* Praha : Computer press, 2007. ISBN 9788025115039.

5. **Sojka, Lukáš.** *Pracovní portál na bázi Spring framework.* Praha : Univerzita Hradec Králové, Fakulta informatiky a managementu, Katedra informatiky a kvantitativních metod, 2014.

6. **Keller, Kevin Lane.** *Strategické řízení značky.* Praha : Grada Publishing, 2007. ISBN: 9788024714813.

7. **Gutierrez, Felipe.** *Introducing Spring Framework: A Primer.* Berlín : Springer, 2014. ISBN: 1430265329.

8. Spring Boot. *Spring.* [Online] Pivotal Software, Inc., 2018. https://projects.spring.io/spring-boot/.

9. What's New in JDK 8. *Oracle.* [Online] Oracle Corporation, 2018. http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html.

10. **Author, Guest.** Nashorn Architecture and Performance Improvements in the Upcoming JDK 8u40 Release. *Oracle Blogs.* [Online] Oracle Corporation, 12. Prosinec 2014. https://blogs.oracle.com/nashorn/nashorn-architecture-and-performance-improvements-in-the-upcoming-jdk-8u40-release.

11. Java Platform, Standard Edition What's New in Oracle JDK 9. *Oracle Help center.* [Online] Oracle Corporation, Září 2017. https://docs.oracle.com/javase/9/whatsnew/toc.htm#JSNEW-GUID-C23AFD78-C777-460B-8ACE-58BE5EA681F6.

12. Statically typed programming language. *Kotlin.* [Online] JetBrains s.r.o., 2018. https://kotlinlang.org/.

13. PostgreSQL. *PostgreSQL.* [Online] The PostgreSQL Global Development Group, 2018. https://www.postgresql.org/.

14. Hibernate ORM. *Hibernate.* [Online] Red Hat, 2018. http://hibernate.org/orm/.

15. Hibernate Search. *Hibernate.* [Online] Red Hat, 2018. http://hibernate.org/search/.

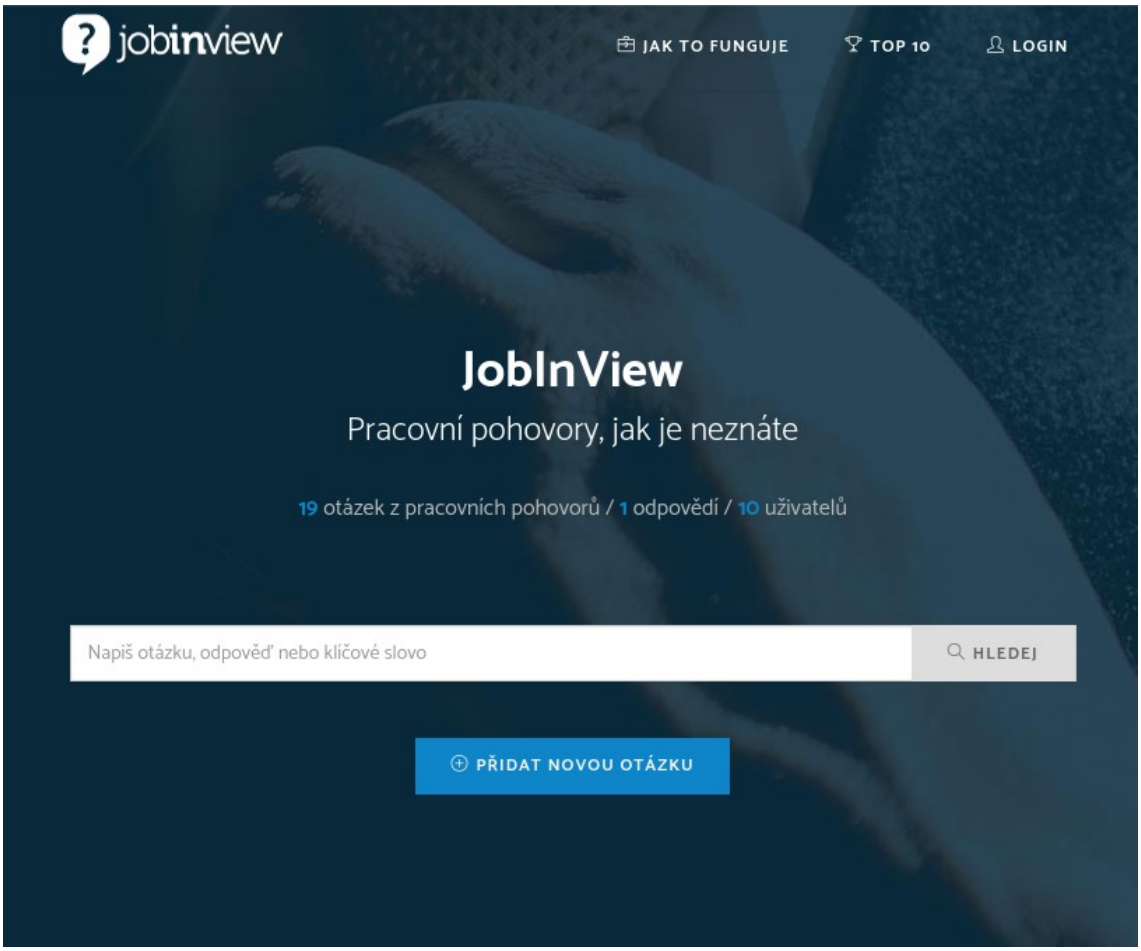
16. A JavaScript library for building user interfaces. *React.* [Online] Facebook Inc., 2018. https://reactjs.org/.

17. Statista. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017.* [Online] Srpen 2017. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

18. IntelliJ IDEA. *JetBrains.* [Online] Jetbrains s.r.o., 2018. https://www.jetbrains.com/idea/.

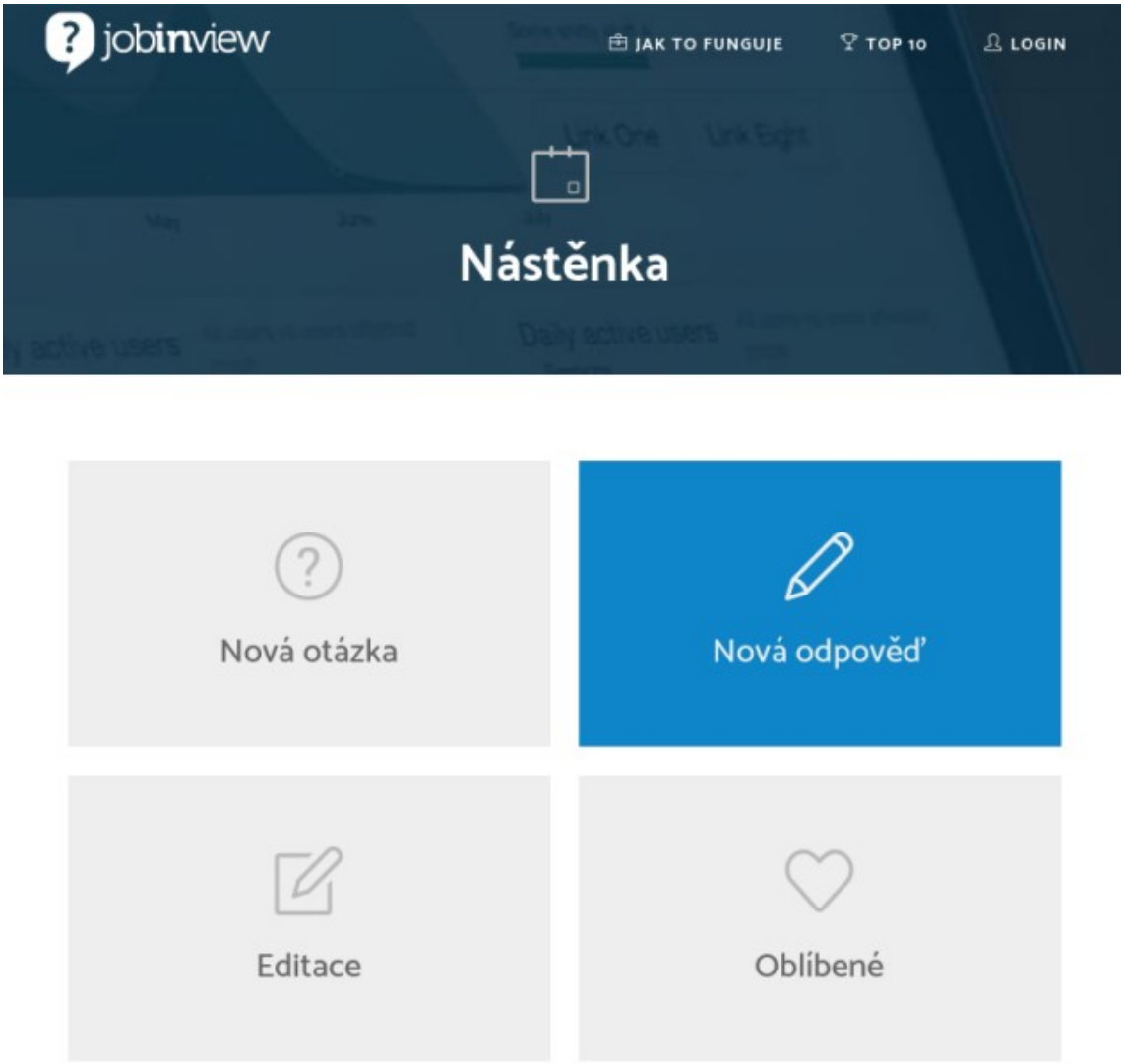
10 Seznam obrázků

11 Přílohy



Obrázek 27 Úvodní stránka aplikace

Zdroj: autor



Obrázek 28 Nástěnka uživatele

Zdroj: autor

Obrázek 29 Stránka vysvětlující základní ovládání aplikace

Zdroj: autor

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Sychra Michal	V Borku 293, Albrechtice nad Orlicí	I1600271

TÉMA ČESKY:

Multiplatformní webová aplikace

TÉMA ANGLICKY:

Multiplatform web application

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a implementovat multiplatformní webovou aplikaci. Dále pak prozkoumat možnosti SSO (Single Sign On), či jiné formy externí autorizace.

Osnova:
1) Úvod
2) Průzkum trhu
3) Návrh a analýza aplikace
4) Implementace
5) Shrnutí výsledků
6) Závěr a doporučení
7) Seznam zdrojů

SEZNAM DOPORUČENÉ LITERATURY:

Java 8 (Herbert Schildt), React.js Essentials (Artemij Fedosejev), PostgreSQL (Bruce Momjian), Bezpečný kód (David LeBlanc, Michael Howard)

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....