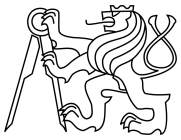
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Mobilní aplikace Důchody
Student:	Bc. Martin Greger
Vedoucí:	Ing. Josef Gattermayer
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je vyvinout aplikaci, která usnadní uživatelům zjišťování informací o důchodovém systému v ČR.

Pokyny:
1) Proveďte analýzu existujících mobilních aplikací věnujících se důchodovému systému.
2) Proveďte analýzu zdrojů otevřených dat důchodového systému v ČR.
3) Vyberte relevantní zdroje, nad kterými je možné postavit mobilní aplikaci.
4) Navrhněte vhodnou funkcionalitu mobilní aplikace na základě výstupů z 1) a 2).
5) Zpracujte návrh uživatelského rozhraní mobilní aplikace ve formě drátěného modelu.
6) Proveďte uživatelský průzkum a iterujte body 4-5.
7) Sepište požadavky na serverovou část, která bude vytěžovat data ze zdrojů otevřených dat a poskytovat je přes standardizované aplikační rozhraní mobilní aplikaci.
8) Proveďte analýzu technických řešení pro serverovou část.
9) Implementujte serverovou část.
10) Implementujte mobilní aplikaci pro OS Android.
11) Otestujte serverovou část a mobilní aplikaci.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 26. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Mobilní aplikace Důchody

Bc. Martin Greger

Vedoucí práce: Ing. Josef Gattermayer

23. června 2017

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Josefovi Gattermayerovi za vedení práce a své rodině za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 23. června 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Greger, Martin. *Mobilní aplikace Důchody.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá tvorbou mobilní aplikace a serverové části aplikace pro usnadnění zjišťování informací o důchodovém systému v České republice. Práce popisuje analýzu zdrojů otevřených dat o důchodovém systému, návrh, realizaci a testování mobilní aplikace a serverové části. Výstupem této práce je mobilní aplikace pro operační systém Android a serverová část aplikace v jazyce Node.js.

Klíčová slova Důchodový systém, otevřená data, mobilní aplikace, Android, serverová aplikace, Node.js

Abstract

The thesis deals with the creation of a mobile application and a server part of the application to facilitate the detection of information about the pension system in the Czech Republic. The thesis describes the analysis of sources of open data about pension system, design, implementation and testing of the mobile application and the server part. The result of this thesis is the mobile application for the operating system Android and the server part of the application in Node.js.

Keywords Pension system, open data, mobile application, Android, server application, Node.js

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době je na internetu dostupné poměrně velké množství otevřených dat. Tyto data jsou veřejně přístupná a snadno strojově zpracovatelná. Z tohoto důvodu lze nad těmito daty budovat různé aplikace. Aplikace pak mohou poskytovat zajímavou funkcionalitu, neboť ve zpracování otevřených dat není vývojář téměř vůbec omezen a tyto data lze libovolně kombinovat. Vzhledem ke stále se zvětšujícímu počtu mobilních zařízení, je vhodné cílit tyto aplikace právě na mobilní platformu.

Velká část otevřených dat pochází od orgánů státní správy. Mezi poskytovatele otevřených dat patří i Česká správa sociálního zabezpečení, která poskytuje otevřená data popisující důchodový systém v ČR. Použití těchto dat v kombinaci s ostatními zdroji dat nabízí příležitost k vytvoření mobilní aplikace usnadňující zjišťování informací o důchodovém systému v ČR. Jelikož tyto data popisují mnoho různých aspektů důchodového systému, je třeba vybrat pouze ty aspekty, které budoucí uživatelé mobilní aplikace nejvíce využijí.

Tato diplomová práce popisuje proces vývoje mobilní aplikace, která usnadňuje zjišťování informací o důchodovém systému v ČR, a s ní spojené serverové části.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit mobilní aplikaci pro operační systém Android sloužící k usnadnění zjišťování informací o důchodovém systému v ČR. Mobilní aplikace bude čerpat data od serverové části, která bude v práci rovněž implementována.

Cíle práce jsou do detailu popsány v následujícím bodech:

1. provést analýzu existujících mobilních aplikací věnujících se důchodovému systému
2. provést analýzu zdrojů otevřených dat důchodového systému v ČR
3. vybrat relevantní zdroje, nad kterými je možné postavit mobilní aplikaci a navrhnout vhodnou funkcionalitu mobilní aplikace na základě analýz uvedených v bodech 1 a 2
4. provést analýzu technických řešení pro serverovou část
5. definovat požadavky na serverovou část, která bude vytěžovat data ze zdrojů otevřených dat a poskytovat je přes standardizované aplikační rozhraní mobilní aplikaci
6. vybrat nejvhodnější technologii pro implementaci serverové části na základě analýzy z bodu 4 a požadavků z bodu 5
7. pomocí iterací (agilní metodiky vývoje) vyvinout mobilní aplikaci pro operační systém Android a serverovou část (v každé iteraci provést uživatelský průzkum za účelem výběru nejvíce preferované funkce a doladění návrhu funkce, zpracovat návrh uživatelského rozhraní funkce ve formě drátěného modelu a implementovat vybranou funkci)
8. otestovat mobilní aplikaci a serverovou část

Kapitola 2

Analýza

Tato kapitola je věnována analýze zdrojů dat, analýze existujících řešení se zaměřením na mobilní aplikace, analýze technických řešení pro serverovou část, analýze databázových technologií a analýze operačního systému Android zhlediska vývoje.

2.1 Analýza zdrojů dat

V této sekci jsou popsány zdroje dat a algoritmů, ze kterých mobilní aplikace a serverová část vychází.

2.1.1 Otevřená data

Otevřená data (open data) lze obecně charakterizovat jako data zveřejněná na internetu způsobem, který neomezuje žádné uživatele ve způsobu jejich použití (technicky ani legislativně) a opravňuje všechny uživatele k jejich dalšímu šíření, pokud při tomto využití a šíření bude uveden autor dat a pokud i ostatní uživatelé budou mít stejná oprávnění s dále šířenými daty nakládat (tj. šířením nedojde k omezení těchto práv například tím, že by uživatel dále šířící otevřená data omezil jejich užití pouze na nekomerční účely) [5].

Na základě 10 principů formulovaných Sunlight Foundation [6] jsou podrobně vymezeny podmínky, které musí data veřejné správy splňovat, aby je bylo možné považovat za otevřená. Data veřejné správy jsou otevřená, pokud jsou [5]:

- úplná (data jsou zveřejněna v maximálním možném rozsahu)
- snadno dostupná (data jsou dostupná na internetu a dohledatelná běžnými ICT nástroji a prostředky)

- strojově čitelná (data jsou ve formátu, který je strukturovaný takovým způsobem, že pomocí programové aplikace z nich lze získat žádané údaje)
- používající standardy s volně dostupnou specifikací (data musí být ve formátu, který je volně dostupný pro libovolné použití nebo do takovéhoto formátu převoditelný volně dostupnou aplikací)
- zpřístupněna za jasně definovaných podmínek užití dat s minimem omezení (podmínky musí být jasně a zřetelně definovány a zveřejněny)
- dostupná uživatelům při vynaložení minima možných nákladů na jejich získání (poskytovatelé jsou v souvislosti s poskytováním dat oprávněni žádat úhradu maximálně ve výši, která nesmí přesáhnout náklady spojené s jejich zpřístupněním uživateli; poskytovatel dat může jednorázově vyžádat i úhradu za mimořádně náročné pořízení dat, pokud si uživatel zpřístupnění těchto dat vyžádá)

2.1.1.1 Česká správa sociálního zabezpečení

Česká správa sociálního zabezpečení (dále jen ČSSZ) poskytuje otevřená data ve formě datových sad poskytujících informace o různých aspektech důchodového systému v ČR. Datové sady jsou dostupné ve formátech RDF a CSV. Datové sady jsou dostupné z [7] (včetně popisu struktury datových sad).

Jako možné zdroje dat pro mobilní aplikaci byly identifikovány datové sady popsané v tabulce 2.1. Výběr konkrétních datových sad je řešen až při návrhu funkcí mobilní aplikace (sekce 3.1).

2.1.1.2 Český statistický úřad

Dalším poskytovatelem otevřených dat je Český statistický úřad. Jako vhodný zdroj dat pro implementované řešení byla identifikována datová sada *Výsledky sčítání lidu, domů a bytů 2011 (SLDB 2011)* (dostupná z [8]). Z datové sady je možné získat data popisující rozdělení obyvatelstva ČR (např. počet ekonomicky aktivních obyvatel).

2.1.2 Ostatní zdroje

Dalším zdrojem informací o důchodovém systému v ČR jsou webové stránky ČSSZ.

2.1.2.1 Česká správa sociálního zabezpečení – webové stránky

Na webových stránkách ČSSZ je mimo jiné dostupný podrobný popis výpočtu důchodového věku [4] a popis výpočtu předpokládané výše důchodu [3]. Největší problém těchto dat je, že jsou špatně strojově zpracovatelná. V případě

Tabulka 2.1: Vybrané datové sady

Název	Popis
Výdaje na důchody	Výdaje na důchody podle roku a druhu důchodu
Přehled o počtu zaměstnavatelů, pojištěnců a pojistných vztahů v České republice	Datová sada obsahuje údaje o počtu komunikujících zaměstnavatelů, pojištěnců a o počtu pojistných vztahů ke konci daného období v České republice
Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle druhu důchodu	Celkový počet důchodců, průměrná výše důchodu a průměrný věk důchodců podle roku, druhu důchodu a pohlaví za ČR, kraje a okresy
Průměrná výše důchodů v Kč u nově přiznaných důchodů v České republice	Průměrná výše důchodů v Kč u nově přiznaných důchodů v České republice podle roku, druhu důchodu a pohlaví
Průměrná délka pobírání starobního důchodu	Průměrná délka pobírání starobního důchodu podle roku zániku a pohlaví
Počet vyplacených důchodů v České republice dle měsíční výše důchodu	Počet vyplacených důchodů v České republice podle roku, druhu důchodu, měsíční výše důchodu a pohlaví
Počet zaniklých důchodů v České republice	Počet zaniklých důchodů v České republice podle roku, druhu důchodu, statistického důvodu zániku a pohlaví
Počet nově přiznaných důchodů v České republice dle měsíční výše důchodu	Počet nově přiznaných důchodů v České republice podle roku, druhu důchodu, měsíční výše důchodu a pohlaví

výpočtu důchodového věku se jedná o data uvedená pomocí HTML stránky, v případě výpočtu předpokládané výše důchodu je k dispozici interaktivní XLS dokument.

2.2 Analýza existujících řešení

Jelikož je cílovou platformou operační systém Android, hlavním zdrojem existujících mobilních aplikací je služba Google Play. Google Play je oficiální distribuční služba obsahu pro tuto mobilní platformu. Primární zaměření této služby je distribuce aplikací, avšak poskytuje i další obsah ve formě filmů, hudby, knih atd. Aplikace z neoficiálních distribučních služeb v analýze nebyly zohledněny.

Jelikož aplikací o důchodovém systému není mnoho, analýza existujících řešení je rozšířena i o webové aplikace.

2.2.1 Mobilní aplikace

Aplikace týkající se přímo důchodového systému v ČR nebyla nalezena žádná. Z tohoto důvodu analýza zahrnuje i aplikace, které se oblasti důchodového systému týkají jen okrajově. Některé z těchto aplikací totiž poskytují zajímavou funkcionalitu, kterou by mohla poskytovat i implementovaná aplikace. Většinou se jedná o různé důchodové kalkulačky (kalkulačka předpokládané výše důchodu a kalkulačka důchodového věku). Dále byly do analýzy zahrnuty i zahraniční aplikace.

2.2.1.1 Finanční kalkulačka [9]

Aplikace nabízí kalkulaci různých finančních produktů. Mimo jiné také obsahuje možnost výpočtu výše důchodu. Avšak algoritmus výpočtu počítá výši důchodu z naspořené částky a nebere v úvahu úlohu státu. Myšlenka důchodové kalkulačky je pro implementovanou aplikaci vhodná, avšak použitý algoritmus musí odpovídat algoritmu pro státem vyměřený důchod v ČR.

2.2.1.2 My Pension [10]

Aplikace nabízí možnost výpočtu předpokládané výše důchodu z naspořené částky a funkci odpočet času do dosažení důchodového věku. Funkci odpočet času do důchodového věku by bylo vhodné použít v implementované aplikaci.

2.2.1.3 Track My Pension [11]

Aplikace nabízí podobné funkce jako aplikace My Pension (2.2.1.2). Navíc aplikace vizualizuje prezentovaná data ve formě grafů. V implementované aplikaci by bylo vhodné použití grafů k prezentaci vybraných statistik o důchodovém systému v ČR.

2.2.2 Webové aplikace

Webových aplikací týkajících se důchodového systému je dostatek. Většinou se jedná o různé důchodové kalkulačky (kalkulačka předpokládané výše důchodu a kalkulačka důchodového věku). Zástupcem tohoto typu aplikací je např. aplikace Výpočet.cz.

2.2.2.1 Výpočet.cz [12]

Aplikace nabízí velké množství praktických kalkulaček (včetně kalkulačky předpokládané výše důchodu a kalkulačky důchodového věku). Kalkulačka předpokládané výše důchodu v této aplikaci je uživatelsky přívětivější (vyžaduje méně vstupních dat) než kalkulačka předpokládané výše důchodu poskytovaná na webu ČSSZ (2.1.2.1) a v případě implementace této funkce by bylo vhodné se touto webovou aplikací inspirovat.

2.3 Analýza technických řešení pro serverovou část

Technologií pro tvorbu serverových aplikací existuje velké množství. Z tohoto důvodu se analýza zaměřuje pouze na následující technologie:

- Java
- PHP
- Node.js
- Ruby

2.3.1 Java

Java je jedním z nejpopulárnějších objektově orientovaných programovacích jazyků.

Jazyk Java se nachází na pomezí mezi kompilovanými a interpretovanými programovacími jazyky. Kompilací zdrojových kódů aplikace vzniká tzv. bytekód, který je následně interpretován pomocí JVM (Java virtual machine). Java je oproti klasickým interpretovaným jazykům rychlejší a je multiplatformní. Pro běh aplikace je potřeba mít nainstalované běhové prostředí Javy (JVM).

Obecně je Java velmi rozsáhlá technologie poskytující obrovské množství funkcí. Díky tomu je vývoj serverových aplikací složitější a časově náročnější než ve většině interpretovaných jazyků. Jistou míru zjednodušení vývoje je použití některého z existujících frameworků založených na Javě. Jako příklad lze uvést framework Spring. Pro běh serverových aplikací v tomto jazyce je potřeba aplikačního serveru, což může oproti jiným jazykům představovat mírnou nevýhodu. Za další nevýhodu lze považovat větší náročnost na systémové prostředky.

Pro správu použitých knihoven (závislostí) v technologii Java je možné použít několik nástrojů (např. Ant, Maven, Gradle). Nejpoužívanějším nástrojem však bývá nástroj Maven.

Použití Javy je vhodné zejména pro tvorbu rozsáhlých informačních systémů s důrazem na robustnost a bezpečnost celého řešení.

Java podporuje většinu existujících databází.

2.3.2 PHP

PHP je dynamicky typovaný interpretovaný jazyk pro tvorbu webových aplikací a dynamických webových stránek. PHP je multiplatformní, avšak nejčastěji se používá v kombinaci s operačním systémem Linux, databází MySQL a webovým serverem Apache (zkratka LAMP).

Za nevýhodu jazyka PHP lze považovat jeho nízkou modularitu (s rostoucím množstvím zdrojového kódu se aplikace stává hůře udržovatelnou). Díky tomu je PHP vhodné spíše pro menší aplikace. Naopak mezi výhody patří

velké množství rozšiřujících knihoven. Nad PHP je také postaveno mnoho frameworků.

PHP podporuje jak SQL i NoSQL databáze, avšak nejčastěji se v kombinaci s PHP používá SQL databáze MySQL.

2.3.3 Node.js

Node.js je moderní javascriptová technologie pro tvorbu serverových aplikací. Aplikace se v této technologii píší v jazyce Javascript, který je interpretovaným programovacím jazykem. Pro běh používá běhové prostředí založené na javascriptovém enginu internetového prohlížeče Google Chrome. Aplikaci vytvořenou v technologii Node.js je možné provozovat na různých platformách. Podmínkou je pouze nutnost mít nainstalované běhové prostředí pro Node.js.

Pro zpracování požadavků Node.js používá událostně řízený neblokující asynchronní model běžící v jednom vlákně. Příchozí požadavek je zařazen do fronty, asynchronně zpracován pomocí tzv. event loop (nekonečné smyčky) a po dokončení výpočtu je výsledek navrácen klientovi. Zpracování dlouho trvajících požadavků tedy neblokuje prostředky pro zpracování dalších požadavků. Aplikace vytvořené v technologii Node.js jsou dobře škálovatelné.

V současné době Node.js poskytuje velké množství existujících knihoven. Pro správu použitých knihoven (závislostí) se v technologii Node.js používá nástroj npm (Node package manager), se kterým je správa závislostí velmi jednoduchá (nástroj je integrován do běhové prostředí Node.js).

Node.js podporuje jak SQL, tak NoSQL databáze. Vzhledem k faktu, že se jedná od javascriptovou technologii, je vhodné použít dokumentovou databázi.

2.3.4 Ruby

Ruby je dynamický typovaný interpretovaný jazyk. Ruby je multiplatformní jazyk, avšak v kombinaci s operačním systémem Windows některé knihovny nemusí fungovat správně. Pro běh aplikace v tomto jazyce je potřeba pouze nainstalované běhové prostředí pro Ruby.

Pro tvorbu serverových aplikací v Ruby slouží framework Ruby on Rails. Tento framework se nejčastěji používá pro tvorbu webových aplikací, ale umožňuje i tvorbu čistě serverových aplikací. Na webové aplikace vytvořené v tomto frameworku je kladen striktní požadavek na dodržení architektury model-view-controller (v případě aplikací čistě serverových odpadá část view).

Pro správu použitých knihoven je v Ruby již integrovaný nástroj. V kontextu Ruby se o knihovnách hovoří jako o tzv. gemech.

Za nevýhodu Ruby lze považovat nižší výkon oproti ostatním jazykům [13] a omezení cílové platformy. Výhodami Ruby jsou modularita a jednoduchá syntaxe.

Ruby podporuje jak SQL i NoSQL databáze.

Tabulka 2.2: Přehled základních normálních forem v relačních databázích [14]

Název	Popis
1. normální forma	tabulka je v 1. normální formě, pokud jsou hodnoty všech atributů atomické
2. normální forma	tabulka je v 2. normální formě, pokud splňuje podmínky 1. normální formy a každý neklíčový atribut plně zavisí na každém klíčovém atributu
3. normální forma	tabulka je v 3. normální formě, pokud splňuje podmínky 2. normální formy a každý neklíčový atribut není tranzitivně závislý žádném klíčovém atributu

2.4 Analýza databázových technologií

Současné databázové technologie lze rozdělit do dvou kategorií:

- relační (SQL) databáze
- nerelační (NoSQL) databáze

2.4.1 Relační (SQL) databáze

Pro ukládání dat s v relačních databázích používají tabulky (relace). Každá tabulka má předem definované schéma (seznam sloupců/atributů včetně datových typů, integritních omezení a definicí primárních a cizích klíčů). Přesný výčet datových typů závisí na konkrétní implementaci relační databáze, avšak snad všechny relační databáze podporují následující datové typy:

- text (varchar)
- číslo (integer, float/double)
- datum (date)
- boolean

Jednotlivé záznamy (uložená data) jsou reprezentovány řádky tabulky (n-ticemi atributů dle schématu tabulky). Tabulky mohou mít mezi sebou vztahy.

S oblastí relačních databází souvisí pojem normalizace. Normalizace je proces úpravy (relačního) schématu databáze za účelem zjednodušení správy uložených dat (eliminace redundance a aktualizačních anomálií, zajištění konzistence). Normalizace se provádí dle stanovených pravidel (normálních forem). Základní normální formy jsou popsány v tabulce 2.2.

Transakce v relačních databázích splňují vlastnosti ACID [15]. Vlastnosti ACID jsou popsány v následujícím seznamu.

- **atomicita (atomicity)**
  všechny operace v transakci budou dokončeny nebo nebude provedena žádná
- **konzistence (consistency)**
  před provedením a po provedení transakce bude databáze v konzistentním stavu (nebude porušeno žádné integritní omezení)
- **izolovanost (isolation)**
  změny provedené v rámci nedokončené transakce nejsou viditelné pro ostatní transakce
- **trvalost (durability)**
  změny po úspěšném dokončení transakce budou trvale uloženy (nemohou být ztraceny)

Oproti nerelačním databázím relační databáze kladou důraz zejména na konzistenci dat. Rovněž jsou oproti nerelačním databázím hůře škálovatelné.

Pro usnadnění práce s relačními databázemi nabízí většina programovacích jazyků ORM (object-relational mapping) frameworky. ORM slouží k převodu objektů použitých v aplikaci na databázové záznamy a naopak.

2.4.2 Nerelační (NoSQL) databáze

Nerelační databáze používají pro ukládání dat jiný než relační model. Druhů nerelačních databází je poměrně velké množství. Jako příklad lze uvést např. dokumentové databáze, grafové databáze, objektové databáze či databáze ukládající data formou klíč-hodnota (key-value databáze).

Oproti relačním databázím nerelační databáze poskytují jednodušší přístup k ukládání dat (např. tzv. schemaless databáze, které umožňují dynamicky vytvářet databázové struktury, vkládat data s různými atributy do databázové struktury či ukládat záznamy bez ohledu na datové typy atributů). Rovněž na datové struktury v nerelační databázi není kladen tak striktní požadavek na normalizaci jako u relačních databází a jednotlivé atributy v datové struktuře nejsou omezeny datovým typem (např. atributem datové struktury může být jiná datová struktura). Za nevýhodu nerelačních databází lze považovat možnou nekonzistenci a redundanci dat plynoucí z uložení dat v denormalizované formě a modelu řízení konzistence BASE, který transakce nerelační databáze používají. Model BASE klade důraz více na dostupnost dat a povoluje jistou míru nekonzistence [15].

Použití nerelační databáze je vhodné v případech, kdy:

- jsou ukládána data nejasné struktury
- je kladen požadavek na rychlou odezvu na dotaz popisující složitější datovou strukturu (v důsledku denormalizovaného uložení dat je odezva rychlejší než výpočetně náročnější spojování tabulek v relační databázi)

- je kladen požadavek na vysokou škálovatelnost databázového systému
- jsou ukládány velké objemy dat (big data)

2.5 Analýza operačního systému Android z hlediska vývoje

Aplikace pro operační systém Android jsou vyvíjeny v jazyce Java. Aplikace lze vyvíjet v různých vývojových prostředích, avšak vhodným přístupem je použití oficiálního vývojového prostředí Android Studio poskytovaného přímo společností Google. V tomto vývojovém prostředí jsou již k dispozici téměř všechny nástroje pro vývoj a není nutná instalace pluginů podporujících vývoj jako u ostatních vývojových prostředí. Pro usnadnění vývoje aplikací pro Android je společností Google poskytován emulátor umožňující provoz tohoto operačního systému i na jiných platformách.

2.5.1 Aktivity a fragmenty

Aktivity a fragmenty jsou základními kameny aplikace pro operační systém Android. Aktivita (třída `android.app.Activity` a její podtřídy) slouží k zobrazení uživatelského rozhraní a k obsluze interakce s uživatelem. Fragment (třída `android.app.Fragment` a její podtřídy) slouží k rozdělení uživatelského rozhraní či interakce s uživatelem do menších částí uvnitř aktivity. Toto rozdělení zvyšuje modularitu implementované aplikace a umožňuje opakované použití již implementovaných komponent ve více aktivitách. Typickým přístupem je definice základního rámce aplikace pomocí aktivit a konkrétních částí aplikace pomocí fragmentů.

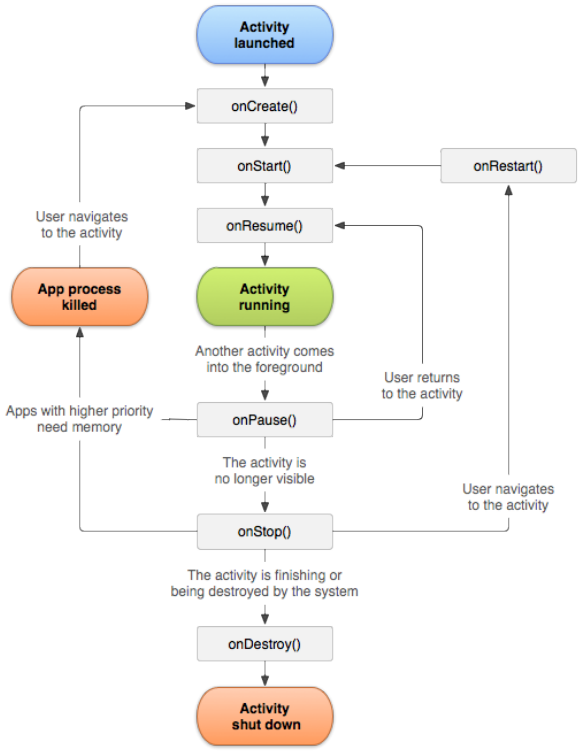
Stav aktivit a fragmentů je řízen pomocí životního cyklu. Životní cyklus aktivity je pro ilustraci zobrazen na obrázku 2.1.

2.5.2 Manifest

Manifestu (soubor `AndroidManifest.xml`) poskytuje základní informace o aplikaci, které operační systém musí mít před spuštěním libovolného kódu aplikace [1]. Jako příklad informací, které manifest poskytuje, lze uvést název jméno balíčku (package) aplikace, seznam aktivit, definici minimální podporované verze operačního systému či definici oprávnění požadovaných po operačním systému.

2.5.3 Zdroje (resources)

V kontextu projektu mobilní aplikace slouží zdroje (složka `res`) k definici statických částí aplikace. Mimo jiné lze ve zdrojích nalézt XML soubory popisující



Obrázek 2.1: Životní cyklus aktivity [1]

.

statické části uživatelského rozhraní (podsložka `layout`), soubor s textací aplikace (soubor `string.xml` v podsložce `values`) či definici ikony pro spouštěč aplikace ve formě obrázků v různých rozlišeních (v podsložce `mipmap`).

O vývoji pro operační systém Android by se dalo napsat mnohem více, avšak to není předmětem práce. Z tohoto důvodu jsou zde zmíněny pouze nejdůležitější aspekty.

Kapitola 3

Návrh

V této kapitole je popsán návrh mobilní aplikace, serverové části (včetně výběru databáze), komunikačního rozhraní a funkcí, které mobilní aplikace může poskytovat.

3.1 Funkce

Tato sekce popisuje návrh funkcí, které může mobilní aplikace poskytovat. Navrhované funkce vycházejí z analýzy zdrojů dat (sekce 2.1) a z analýzy existujících řešení (sekce 2.2). Návrh možných funkcí je vstupem pro proces vývoje mobilní aplikace a serverové části popsaný v sekci 4.1.

Možné funkce aplikace lze rozdělit do dvou kategorií:

- uživatelské funkce
- statistické funkce

Uživatelské fukce potřebují pro výpočet vstupní data (údaje o uživateli), nad kterými následně probíhá výpočet. Oproti tomu statistické funkce mají čistě informativní charakter a pouze vhodnou formu prezentují statistická data popisující určitý aspekt důchodového systému v ČR.

3.1.1 Uživatelské funkce

Tato podsekce popisuje uživatelské funkce a algoritmy potřebné k jejich výpočtu.

3.1.1.1 Kalkulačka předpokládané výše důchodu

Funkce *Kalkulačka předpokládané výše důchodu* vychází z algoritmu důchodové kalkulačky poskytované ČSSZ ve formě XLS dokumentu a z informací o důchodovém pojištění z webové stránky o starobních důchodech na webu ČSSZ (dostupné z [3]).

Vstupy výpočtu předpokládané výše důchodu jsou:

- průměrná měsíční mzda od roku 1986 do současnosti
- počet let účasti na sociálním pojištění (počet odpracovaných let)
- počet měsíců předčasného důchodu
- počet měsíců přesluhování

Aby používání této funkce bylo pro uživatele pohodlnější, výpočet bude schopen zpracovat i průměrné měsíční mzdy za určité období. Funkce bude počítat výši důchodu k aktuálnímu roku.

Výpočet předpokládané výše důchodu se skládá z následujících kroků:

1. Prvním krokem výpočtu je výpočet úhrnu ročních vyměřovacích základů. Pro tento výpočet je potřeba znát tzv. koeficienty nárůstu vyměřovacích základů pro každý rok, které jsou dostupné v důchodové kalkulačce poskytované ČSSZ. Úhrn ročních vyměřovacích základů se poté vypočítá následně:

   Nechť $koeficienty$ je pole koeficientů nárůstu vyměřovacích základů, $mzdy$ pole průměrných měsíčních mezd pro příslušné roky a $aktualni\_rok$ je aktuální rok. Úhrn ročních vyměřovacích základů je roven vzorci:

   $$\sum_{\forall rok \in <1986, aktualni\_rok>} koeficienty[rok] \cdot mzdy[rok] \cdot 12$$

2. Druhým krokem výpočtu je výpočet osobního vyměřovacího základu.

   Nechť $rozdil\_dnu$ je počet dní mezi 1. lednem 1986 a 1. lednem aktuálního roku a $uhrn\_rocnich\_vymerovacich\_zakladu$ je úhrn ročních vyměřovacích základů. Osobní vyměřovací základ je poté roven vzorci:

   $$30,4167 \cdot \frac{uhrn\_rocnich\_vymerovacich\_zakladu}{rozdil\_dnu}$$

3. Třetím krokem výpočtu je výpočet výpočtového základu, který se vypočte následujícím způsobem:

   Nechť ovz je osobní vyměřovací základ. Výpočtový základ je roven vzorci:

   $$\left\{ \begin{array}{cc} 12423 & pokud\ ovz \leq 12423 \\ 12423 + (ovz - 12423) \cdot 0,26 & pokud\ ovz \in (12423, 112928) \\ 38555 & pokud\ ovz \geq 112928 \end{array} \right\}$$

Tabulka 3.1: Minimální počet let účasti na sociálním pojištění dle roku dosažení důchodového věku [3]

Rok dosažení důchodového věku	Minimální počet let účasti na sociálním pojištění
do roku 2010	25
2010	26
2011	27
2012	28
2013	29
2014	30
2015	31
2016	32
2017	33
2018	34
po roce 2018	35

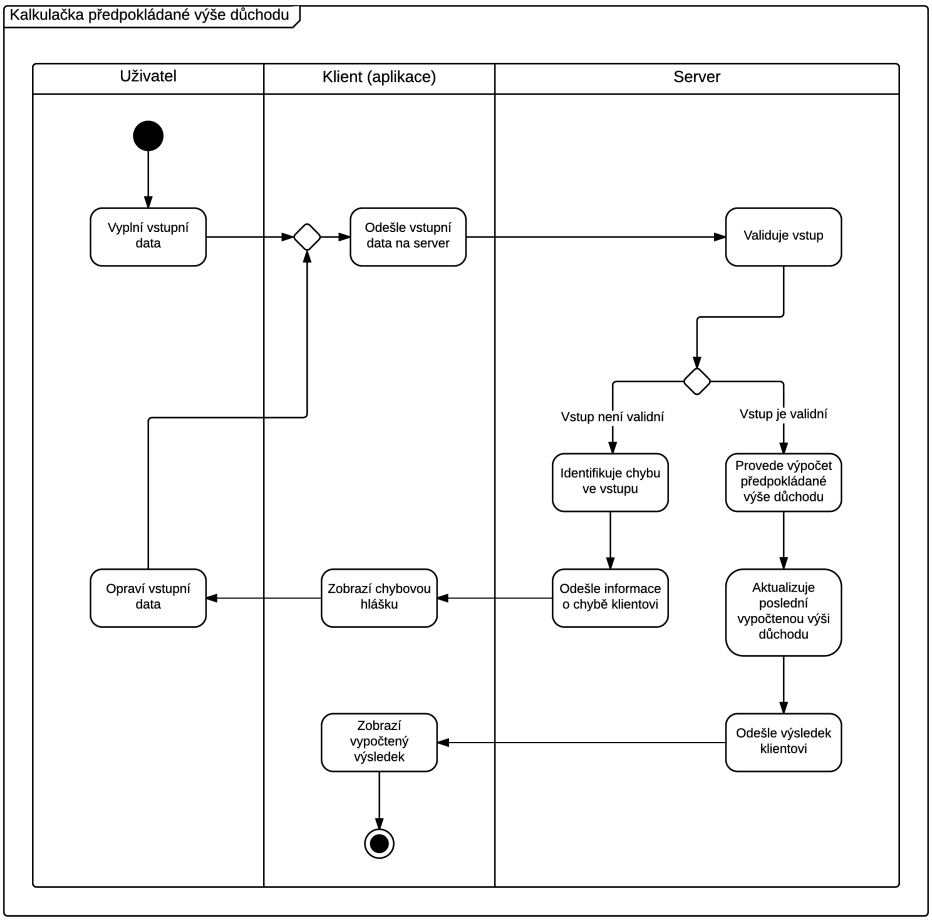
4. Čtvrtým krokem výpočtu je výpočet procentní výměry důchodu.

   Nechť *pol* je počet let účasti na sociálním pojištění (počet odpracovaných let), *pmpd* počet měsíců předčasného důchodu, *pmp* počet měsíců přesluhování a *vz* výpočtový základ. Procentuální výměra důchodu je rovna vzorci:

$$min\left\{770, vz \cdot \frac{(pol \cdot 1,5 - \left\lceil \frac{pmpd}{3} \right\rceil \cdot 0,9 + \left\lfloor \frac{pmp}{3} \right\rfloor \cdot 1,5)}{100}\right\}$$

5. V posledním kroku se k procentní výměře přičte pevná část důchodu a výpočet je dokončen. Pevná část důchodu je pro rok 2017 stanovena na 2550 Kč.

Vhodným rozšířením této funkce je možnost uložení poslední vypočtené výše důchodu, aby uživatel nemusel výpočet při každém použití aplikace opakovat. Dále připadá v úvahu možnost porovnání vypočtené předpokládané výše důchodu s průměrnou výší důchodu (průměrnou výši důchodu lze získat z datové sady *Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle druhu důchodu*). V případě, že mobilní aplikace bude poskytovat i funkci *Kalkulačka důchodového věku* (funkce je popsána v 3.1.1.2), lze tuto funkci rozšířit i o informaci o minimální době účasti na sociálním pojištění (minimální počet odpracovaných let) pro dosažení nároku na starobní důchod. Minimální počet let účasti na sociálním pojištění dle roku dosažení důchodového věku je popsán v tabulce 3.1.



Obrázek 3.1: Diagram aktivit pro funkci *Kalkulačka předpokládané výše důchodu*

Funkce *Kalkulačka předpokládané výše důchodu* je detailně popsána diagramem aktivit na obrázku 3.1.

3.1.1.2 Kalkulačka důchodového věku

Funkce *Kalkulačka důchodového věku* vychází z algoritmu popsaném na webové stránce o starobních důchodech na webu ČSSZ (dostupné z [4]). Vstupem výpočtu důchodového věku je datum narození uživatele, pohlaví a počet dětí (v případě ženského pohlaví).

Tabulka 3.2: Tabulka důchodového věku pro lidi narozené v roce 1936 až 1977 [4]

Rok narození	Důchodový věk činí u					
	mužů	žen s počtem vychovaných dětí				
		0	1	2	3 - 4	5 a více
1936	60r+2m	57r	56r	55r	54r	53r
1937	60r+4m	57r	56r	55r	54r	53r
1938	60r+6m	57r	56r	55r	54r	53r
1939	60r+8m	57r+4m	56r	55r	54r	53r
1940	60r+10m	57r+8m	56r+4m	55r	54r	53r
1941	61r	58r	56r+8m	55r+4m	54r	53r
1942	61r+2m	58r+4m	57r	55r+8m	54r+4m	53r
1943	61r+4m	58r+8m	57r+4m	56r	54r+8m	53r+4m
1944	61r+6m	59r	57r+8m	56r+4m	55r	53r+8m
1945	61r+8m	59r+4m	58r	56r+8m	55r+4m	54r
1946	61r+10m	59r+8m	58r+4m	57r	55r+8m	54r+4m
1947	62r	60r	58r+8m	57r+4m	56r	54r+8m
1948	62r+2m	60r+4m	59r	57r+8m	56r+4m	55r
1949	62r+4m	60r+8m	59r+4m	58r	56r+8m	55r+4m
1950	62r+6m	61r	59r+8m	58r+4m	57r	55r+8m
1951	62r+8m	61r+4m	60r	58r+8m	57r+4m	56r
1952	62r+10m	61r+8m	60r+4m	59r	57r+8m	56r+4m
1953	63r	62r	60r+8m	59r+4m	58r	56r+8m
1954	63r+2m	62r+4m	61r	59r+8m	58r+4m	57r
1955	63r+4m	62r+8m	61r+4m	60r	58r+8m	57r+4m
1956	63r+6m	63r+2m	61r+8m	60r+4m	59r	57r+8m
1957	63r+8m	63r+8m	62r+2m	60r+8m	59r+4m	58r
1958	63r+10m	63r+10m	62r+8m	61r+2m	59r+8m	58r+4m
1959	64r	64r	63r+2m	61r+8m	60r+2m	58r+8m
1960	64r+2m	64r+2m	63r+8m	62r+2m	60r+8m	59r+2m
1961	64r+4m	64r+4m	64r+2m	62r+8m	61r+2m	59r+8m
1962	64r+6m	64r+6m	64r+6m	63r+2m	61r+8m	60r+2m
1963	64r+8m	64r+8m	64r+8m	63r+8m	62r+2m	60r+8m
1964	64r+10m	64r+10m	64r+10m	64r+2m	62r+8m	61r+2m
1965	65r	65r	65r	64r+8m	63r+2m	61r+8m
1966	65r+2m	65r+2m	65r+2m	65r+2m	63r+8m	62r+2m
1967	65r+4m	65r+4m	65r+4m	65r+4m	64r+2m	62r+8m
1968	65r+6m	65r+6m	65r+6m	65r+6m	64r+8m	63r+2m
1969	65r+8m	65r+8m	65r+8m	65r+8m	65r+2m	63r+8m
1970	65r+10m	65r+10m	65r+10m	65r+10m	65r+8m	64r+2m
1971	66r	66r	66r	66r	66r	64r+8m

Pokračování tabulky na další stránce

Tabulka 3.2: Tabulka důchodového věku pro lidi narozené v roce 1936 až 1977 – pokračování [4]

Rok narození	Důchodový věk činí u					
	mužů	žen s počtem vychovaných dětí				
		0	1	2	3 - 4	5 a více
1972	66r+2m	66r+2m	66r+2m	66r+2m	66r+2m	65r+2m
1973	66r+4m	66r+4m	66r+4m	66r+4m	66r+4m	65r+8m
1974	66r+6m	66r+6m	66r+6m	66r+6m	66r+6m	66r+2m
1975	66r+8m	66r+8m	66r+8m	66r+8m	66r+8m	66r+8m
1976	66r+10m	66r+10m	66r+10m	66r+10m	66r+10m	66r+10m
1977	67r	67r	67r	67r	67r	67r

Důchodový věk pro lidi narozené v období 1936 až 1977 se vypočte dle tabulky 3.2. U lidí narozených po roce 1977 se důchodový věk stanoví tak, že se k věku 67 let přičte takový počet kalendářních měsíců, který odpovídá dvojnásobku rozdílu mezi rokem narození a rokem 1977 [4].

Minimální rok narození vstupující do výpočtu důchodového věku je rok 1936. Není předpoklad, že určitá část cílové skupiny uživatelů bude narozena před rokem 1936, neboť lidé narozeni před rokem 1936 již důchodového věku dosáhli.

Prezentace funkce uživateli je možné provést formou odpočtu dnů do důchodu nebo zobrazením důchodového věku uživatele aplikace.

3.1.2 Statistické funkce

Tato podsekce popisuje statistické funkce a algoritmy potřebné k jejich výpočtu.

Pojmem důchod (důchodce) je v kontextu statistických funkcí myšleno starobní důchod (důchodce). Detailní podmínky pro získání nároku na starobní důchod jsou dostupné z [4].

3.1.2.1 Porovnání počtu pracujících a důchodců

Pro výpočet této funkce je nutné z datových sad získat tyto údaje:

- počet důchodců
- počet pracujících

Počet důchodců je snadno získatelný z datové sady *Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle druhu důchodu*. Údaje o přibližném počtu pracujících je možné získat dvěma způsoby:

1. jako počet ekonomicky aktivních obyvatel z datové sady *Výsledky sčítání lidu, domů a bytů 2011 (SLDB 2011)*

2. jako součet počtu OSVČ (osob samostatně výdělečně činných) z datové sady *Přehled o celkovém počtu OSVČ v České republice* a počtu pojištěnců z datové sady *Přehled o počtu zaměstnavatelů, pojištěnců a pojistných vztahů v České republice* (dle kurátora otevřených dat ČSSZ pana Ing. Jiřího Šunky je pro zjištění počtu zaměstnanců relevantní použít ukazatel počet pojištěnců z této datové sady)

Jako vhodnějším způsobem získání počtu pracujících se ukázal druhý způsob, neboť je možné získat počet pracujících pro každý kalendářní rok zvlášť a výpočet je přesnější (do ekonomicky aktivního obyvatelstva počítají i nezaměstnaní).

Výsledek této funkce je dán jako počet důchodců na jednoho pracujícího.

3.1.2.2 Srovnání nákladů na důchody v ČR podle roku

Funkce *Srovnání nákladů na důchody v ČR podle roku* vychází z datové sady *Výdaje na důchody*. Získání požadovaných dat spočívá pouze ve filtraci datové sady (filtrují se pouze starobní důchody pro kalendářní roky).

3.1.2.3 Procentuální nárůst počtu důchodců za období

Funkce *Procentuální nárůst počtu důchodců za období* vychází z datové sady *Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle druhu důchodu*. Zpracování dat pro tuto funkci spočívá v získání údajů o počtu starobních důchodců pro obě pohlaví pro území České republiky za určité období. V úvahu připadá období čtvrtletí (záznamy v datové sadě jsou poskytována za čtvrtletí), nebo období rok (funkce by pracovala pouze se záznamy posledního čtvrtletí roku). Rozhodnutí o velikosti období bude případně řešeno v sekci 4.1 (v případě, že funkce bude uživateli vybrána pro implementaci).

3.1.2.4 Průměrná délka pobíraní důchodu

Funkce průměrná délka pobírání důchodu vychází z datové sady *Průměrná délka pobírání starobního důchodu*. Datová sada poskytuje data ve vhodném formátu, zpracování dat spočívá v selekci potřebných atributů z každého záznamu datové sady. Funkce popisuje průměrnou délku pobírání důchodu dle pohlaví a dle roku ukončení důchodu.

3.1.2.5 Výše důchodu dle území

Funkce *Výše důchodu dle území* vychází z datové sady *Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle*

druhu důchodu. Funkce popisuje průměrnou výši důchodu dle kraje, okresu a nejaktuálnějšího roku v datové sadě (v úvahu připadá i možnost dle různých roků z datové sady). Rozhodnutí, zda funkce bude pracovat pouze s nejaktuálnějším rokem, nebo se všemi roky z datové sady bude případně řešeno v sekci 4.1 (v případě, že funkce bude uživateli vybrána pro implementaci).

Zpracování dat z datové sady pro tuto funkci je prováděno následujícím způsobem:

1. Filtrace záznamů podle druhu důchodu a pohlaví (funkce pracuje pouze se záznamy popisující starobní důchod pro obě pohlaví)
2. Získání průměrné výše důchodu pro jednotlivé kraje za jednotlivé roky (záznamy v datové sadě jsou uváděny za čtvrtletí, průměrná výše důchodu pro konkrétní rok bude vypočtena jako průměrná hodnota za všechna čtvrtletí konkrétního roku)
3. Získání průměrné výše důchodu pro jednotlivé okresy za jednotlivé roky stejným způsobem jako jako v bodě 2
4. Přiřazení okresů k příslušným krajů podle roku (nebo pouze pro nejaktuálnější rok z datové sady)

3.1.2.6 Počet důchodců dle území

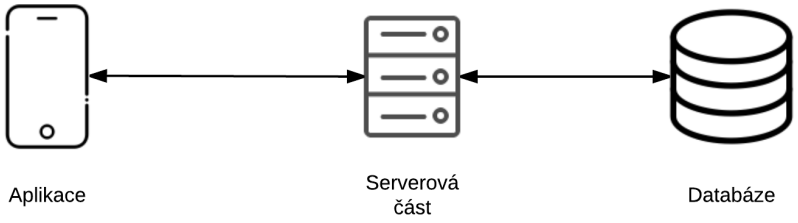
Funkce *Počet důchodců dle území* vychází z datové sady *Přehled o počtu důchodců podle území, pohlaví, průměrné výše důchodu, průměrného věku a podle druhu důchodu*. Funkce popisuje počet důchodců dle kraje, okresu a nejaktuálnějšího roku v datové sadě (v úvahu připadá i možnost dle různých roků z datové sady). Rozhodnutí, zda funkce bude pracovat pouze s nejaktuálnějším rokem, nebo s všemi roky z datové sady bude případně řešeno v sekci 4.1 (v případě, že funkce bude uživateli vybrána pro implementaci). Zpracování dat pro tuto funkci se provádí stejným způsobem jako u funkce *Výše důchodu dle území* (3.1.2.5) s rozdílem vyhledávaného atributu (vyhledává se atribut počet důchodců namísto atributu průměrná výše důchodu).

3.2 Architektura

Celé řešení se bude skládat z následujících částí:

- mobilní aplikace jako klientská část řešení (požadavek plyne ze zadání práce)
- serverová část (požadavek plyne ze zadání práce)
- databáze (databáze bude použita pro persistenci datových struktur pro vybrané funkce a uživatelských dat)

Schéma architektury celého řešení je popsáno obrázkem 3.2.



Obrázek 3.2: Architektura systému

3.3 Mobilní aplikace

Tato sekce popisuje základní návrhové rozhodnutí a specifikaci požadavků pro mobilní aplikaci.

3.3.1 Cílová platforma

Cílová platforma mobilní aplikace je již specifikována v zadání práce. Je jí operační systém Android.

Podle [16] je operační systém Android nejrozšířenější mobilní platformou a ve třetím čtvrtletí roku 2016 bylo 86.8% všech prodaných chytrých mobilních telefonů určeno právě pro tuto platformu.

Poslední verzí operačního systému Android je verze 7.1 Nougat (API 25). Mobilní aplikace by měla podporovat i starší verze operačního systému Android, neboť podle tabulky 3.3 většina mobilních zařízení používá starší verze tohoto operačního systému.

Vzhledem k procentuálnímu zastoupení používaných verzí operačního systému Android v tabulce 3.3 byla zvolena minimální podporovaná verze 5.0 Lollipop (API 21). Volba této minimální verze zapříčiní, že mobilní aplikace bude dostupná pro zhruba 60% mobilních zařízení s operačním systémem Android a při implementaci bude možné použít relativně nové funkcionality tohoto operačního systému přidané v pozdějších verzích API. Další výhodou této volby je, že od této verze přichází Android s novým vzhledem grafických komponent (tzv. Material design). V implementaci mobilní aplikace bude možné tyto komponenty použít.

3.3.2 Požadavky

Specifikace požadavků na implementaci mobilní aplikace vychází z předpokladu, že mobilní aplikace bude poskytovat jak uživatelské, tak statistické

Tabulka 3.3: Procentuální zastoupení používaných verzí operačního systému Android [1]

Verze	Název	API	Zastoupení
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.7%
4.3		18	1.6%
4.4	KitKat	19	21.9%
5.0	Lollipop	21	9.8%
5.1		22	23.1%
6.0	Marshmallow	23	30.7%
7.0	Nougat	24	0.9%
7.1		25	0.3%

funkce (výběr, návrh a implementace poskytovaných funkcí je popsán v sekci 4.1).

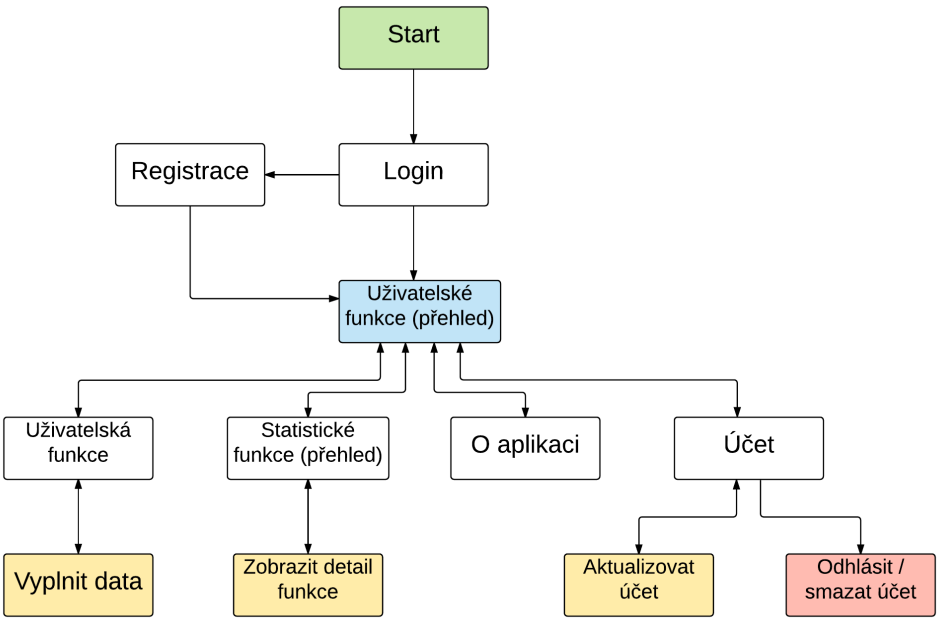
Pro implementaci mobilní aplikace byly specifikovány následující požadavky:

- mobilní aplikace bude vhodným způsobem prezentovat vybrané uživatelské funkce
- mobilní aplikace bude vhodným způsobem prezentovat vybrané statistické funkce
- přístup do aplikace bude podmíněn vytvořením uživatelského účtu za účelem zjednodušení používání uživatelských funkcí (možnost uložení již vypočteného výsledku funkce při dalším použití aplikace či automatický výpočet funkce na základě údajů uvedených při registraci)
- mobilní aplikace nebude provádět zpracování dat a výpočty pro vybrané funkce (cílem je přesun výpočtů a zpracování dat na serverovou část za účelem minimalizace výpočetní náročnosti mobilní aplikace a minimalizace přenášených dat mezi serverovou částí a mobilní aplikací)
- mobilní aplikace bude získávat data pro vybrané funkce od serverové části přes standardizované komunikační rozhraní

3.3.3 Uživatelské rozhraní

Návrh uživatelského rozhraní vychází z předpokladu, že mobilní aplikace bude poskytovat jak uživatelské, tak statistické funkce (výběr, návrh a implementace poskytovaných funkcí je popsán v sekci 4.1).

Pro mobilní aplikaci bylo navrženo rozdělení do následujících částí:



Obrázek 3.3: Task graf popisující základní strukturu uživatelského rozhraní mobilní aplikace

.

- část s přehledem uživatelských funkcí (domovská obrazovka aplikace)
- část s přehledem statistických funkcí
- část pro detailní zobrazení vybrané statistické funkce
- část pro administraci uživatelského účtu
- část se základními informacemi o aplikaci

Ve vhodných případech je možné uvažovat o rozšíření mobilní aplikace o další části (např. pro více prostorově náročné funkce).

Vzhledem k požadavkům z podsekce 3.3.2 bude přístup do aplikace podmíněn vytvořením uživatelského účtu. Uživatelské rozhraní tedy musí řešit mechanismus přihlášení do aplikace a registrace uživatelského účtu.

Základní návrh struktury uživatelského rozhraní je popsán task grafem na obrázku 3.3.

Detailní návrhy vzhledu jednotlivých obrazovek ve formě drátěného modelu jsou řešeny v sekci 4.1, neboť v době návrhu nebyly známy funkce, které aplikace bude nabízet.

3.4 Serverová část

Tato sekce popisuje základní návrhové rozhodnutí a specifikaci požadavků pro serverovou část.

3.4.1 Požadavky

Specifikace požadavků na implementaci serverové části vychází z předpokladu, že mobilní aplikace bude poskytovat jak uživatelské, tak statistické funkce (výběr, návrh a implementace poskytovaných funkcí je popsán v sekci 4.1).

- serverová část bude poskytovat standartizované komunikační rozhraní pro komunikaci s mobilní aplikací
- serverová část bude poskytovat algoritmy pro výpočet vybraných uživatelských funkcí
- serverová část bude poskytovat data pro vybrané funkce
- serverová část bude umožňovat správu uživatelských účtů
- serverová část bude zabezpečovat části komunikačního rozhraní s uživatelskými funkcemi a údaji o uživatelském účtu
- serverová část bude předzpracovávat a ukládat data z příslušných datových sad pro vybrané funkce
- serverová část bude periodicky aktualizovat data pro vybrané funkce
- serverová část bude snadno konfigurovatelná s ohledem na nasazení v různých prostředích

3.4.2 Technologie

Pro implementaci serverové části byla vybrána technologie Node.js. Technologie Node.js byla vybrána z následujících důvodů:

- Vzhledem k požadavkům uvedeným v podsekci 3.4.1 nebyl identifikován žádný potencionální problém. Jsou dostupné knihovny řešící problémy plynoucí z těchto požadavků.
- Navrhované komunikační rozhraní mezi serverovou částí a mobilní aplikací (popsáno v sekci 3.5) bude pracovat s daty ve formátu JSON. Pro reprezentaci objektů v technologii Node.js se používá formát JSON, tudíž nebude nutné převádět data mezi různými formáty.
- Node.js umožňuje snadný převod mezi formáty CSV a JSON (datové sady jsou k dispozici ve formátu CSV).

- Nasazení aplikace v technologii Node.js do produkčního prostředí je jednoduché a existuje dostatečné množství hostingových služeb, které tuto technologii podporují.
- Node.js je moderní technologie nabývající na popularitě, o čemž svědčí obrázek 3.4 popisující vývoj počtu dostupných knihoven pro jednotlivé analyzované technologie za dané časové období.
- Jazyk Java nebyl vybrán z důvodu náročnějšího nasazení aplikace do produkčního prostředí (zejména nutnost použití a administrace aplikačního serveru). Dále je předpoklad využití pouze malé podmnožiny funkcionality, kterou tento jazyk nabízí.
- Jazyk PHP nebyl vybrán kvůli nízké modularitě, což by mohlo představovat problém při budoucím rozšiřováním serverové části o nové funkce. Druhým důvodem je nezkušenost autora práce s vývojem pod touto platformou.
- Framework Ruby on Rails z důvodu nutnosti dodržení architektury frameworku při implementaci serverové části (záměr vyvarovat se případnému obcházení architektury frameworku).

3.5 Komunikační rozhraní

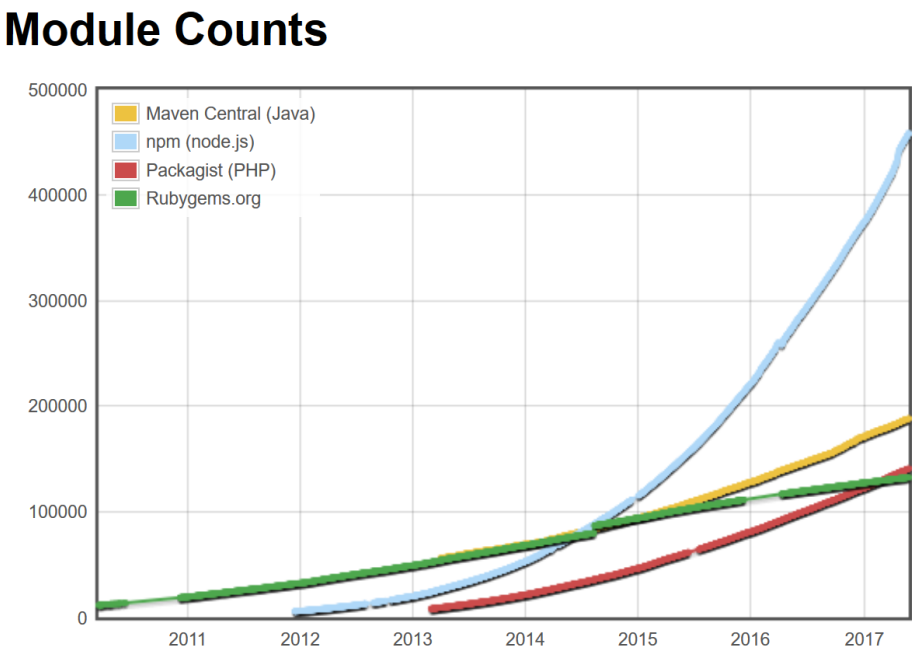
Jako komunikační rozhraní mezi mobilní aplikací a serverovou částí bylo vybráno aplikační rozhraní REST (dále jen REST API). Jako formát přenášených dat byl vybrán formát JSON.

3.5.1 REST API

REST API je komunikační rozhraní mezi komponentami aplikace, které splňuje pravidla REST architektury (podrobný popis REST architektury je dostupný z [17]). Pro přenos dat se používá protokol HTTP.

Základním pojmem pro REST je tzv. zdroj. Zdroj reprezentuje přenášená data spolu s operacemi, které jsou nad těmito daty dostupné. Jednotlivé zdroje jsou jednoznačně identifikovány svou URI (v případě protokolu HTTP se jedná o příslušnou URL), které se používají pro přístup k příslušným zdrojům. Pro manipulaci se zdroji definuje 4 základní CRUD (create, read, update, delete) operace, které jsou při použití HTTP protokolu implementovány příslušnými HTTP metodami:

- metoda GET pro získání zdroje
- metoda POST pro vytvoření zdroje



Obrázek 3.4: Vývoj počtu dostupných knihoven pro jednotlivé analyzované technologie za dané časové období [2]

- metoda PUT pro aktualizaci zdroje
- metoda DELETE pro smazání zdroje

Pro popis výsledku operací se používají HTTP stavové kódy (klient se z odpovědi na daný požadavek dozví, zda operace byla provedena úspěšně či nastal nějaký problém).

3.5.1.1 Princip HATEOAS

Princip HATEOAS (Hypermedia as the engine of application state) říká, že reprezentace zdroje obsahuje odkazy na související zdroje [17]. To v důsledku znamená, že klient při komunikaci nepotřebuje znát celou strukturu REST API (klient je schopen získat každý zdroj pomocí kořenového URL a těchto odkazů).

3.5.2 JSON

JSON (JavaScript Object Notation) je textově orientovaný formát pro serializaci strukturovaných dat. Je odvozen od objektových literálů jazyka Ja-

vascript. JSON může reprezentovat čtyři primitivní datové typy (string, number, boolean a null) a dva strukturované typy (object a array) [18].

Příklad zápisu objektu ve formátu JSON je popsán níže.

```
{
    "stringAttribute": "Sample text",
    "numberAttribute": 7,
    "booleanAttribute": true,
    "arrayAttribute": [ "first", "second", "third" ]
}
```

3.5.3 Zabezpečení

Jelikož některé z navrhovaných funkcí vyžadují osobní údaje uživatele, je nutné příslušné části komunikačního rozhraní zabezpečit (v případě, že mobilní aplikace bude tyto funkce poskytovat). Pro zabezpečení byly vybrány následující technologie:

- protokol HTTPS
- JSON Web Token

3.5.3.1 JSON Web Token

JSON Web Token (dále jen JWT token) je řetězec vzniklý zašifrováním (případně podepsáním) určitých informací (ve formě JSON objektu) sloužící pro zabezpečení přenosu dat mezi účastníky komunikace [19].

Pro implementaci celého řešení bylo navrženo použití JWT tokenu pro autentizaci uživatele při vstupu do aplikace a autorizaci požadavků uživatelských funkcí. Schéma autentizace pomocí JWT tokenu je znázorněno na obrázku 3.5

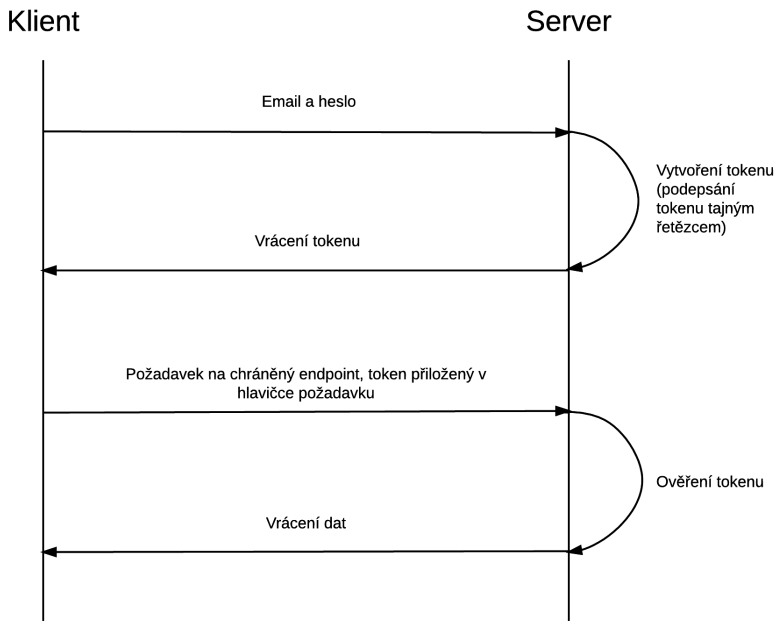
3.5.3.2 Protokol HTTPS

Protokol HTTPS vychází z protokolu HTTP, avšak data jsou přenášena v šifrované podobě. Zabezpečení přenosu spočívá v použití protokolu TLS (případně SSL) protokolu na transportní vrstvě [20].

3.6 Databáze

Vzhledem k povaze dat a vybrané technologii pro implementaci serverové části je vhodné použít NoSQL dokumentovou databázi podporující uložení dat ve formátu JSON.

Výběr dokumentové databáze vychází z předpokladu, že databáze bude primárně sloužit pouze k ukládání předzpracovaných dat v podobě, která bude dostupná přes komunikační rozhraní. Veškeré zpracování dat bude řešeno na



Obrázek 3.5: Autentizace pomocí JWT tokenu

aplikační úrovni. Předpokládané schéma databáze je pro každou každou funkci jedna databázová struktura (dokumentová kolekce). Dokumentové kolekce budou obsahovat veškeré informace pro konkrétní funkci, tudíž databázový dotaz na získání dat bude mít rychlou odezvu (upřednostnění rychlosti odezvy na úkor normalizace dat). Dalším důvodem výběru dokumentové databáze je vybraná technologie pro implementaci serverové části a navržené komunikační rozhraní. Node.js technologie s JSON objekty interně pracuje a komunikační rozhraní rovněž používá objekty tohoto datového formátu. Oproti použití klasické SQL databáze odpadá režie spojená s mapováním JSON objektů na SQL tabulky a naopak.

Dokumentových databází podporující uložení dat ve formátu JSON existuje poměrně velké množství. Jako příklad lze uvést:

- MongoDB
- CouchDB
- OrientDB
- HyperDex

Pro implementaci serverové části byla zvolena databáze MongoDB, neboť se jedná a poměrně rozšířenou technologii a integrace technologie Node.js

s toutu databází je široce podporována. Dalším důvodem výběru této databáze je existence hostingových služeb pro tuto databázi.

Kapitola 4

Realizace

V této kapitole je popsán proces vývoje mobilní aplikace a serverové části, vybrané části implementace a technologie použité při implementaci.

4.1 Uživatelský průzkum a vývoj

Vývoj funkcí mobilní aplikace byl řízen prostřednictvím jednotlivých iterací. Každá iterace zahrnuje následující kroky:

1. uživatelský průzkum (5ti respondentům byl předložen seznam s možnými funkcemi mobilní aplikace uvedenými v sekci 3.1 za účelem výběru preferované funkce, již vyvinuté funkce byly v pozdějších iteracích ze seznamu postupně odebírány)

2. zpracování návrhu uživatelského rozhraní funkce ve formě drátěného modelu (wireframe)

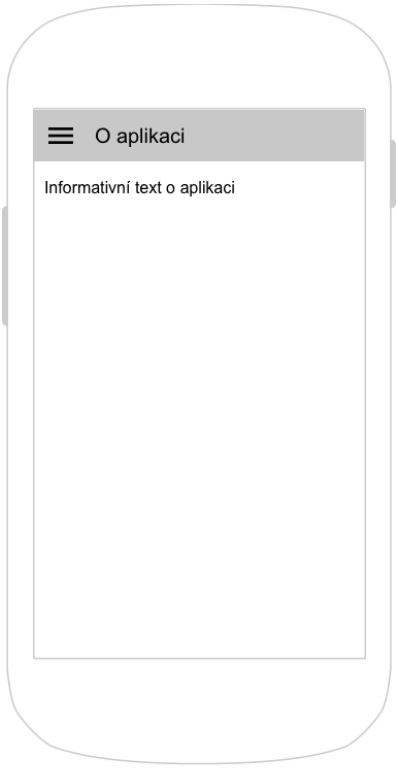
3. předložení drátěného modelu funkce respondentům za účelem získání zpětné vazby a doladění návrhu funkce

4. zpracování připomínek z kroku 3, případná úprava drátěného modelu a konzultace s respodenty (v případě velkých změn modelu)

5. implementace funkce

4.1.1 Vývoj základní struktury mobilní aplikace

Před samotným vývojem funkcí byla s respondenty konzultována základní struktura mobilní aplikace (uvedená v 3.3.3) spolu s konceptem uživatelských a statistických funkcí. Dále byl vytvořen návrh uživatelského rozhraní ve formě drátěného modelu vyjma funkcí samotných.



Obrázek 4.1: Drátěný model pro obrazovku se základními informacemi o aplikaci

.

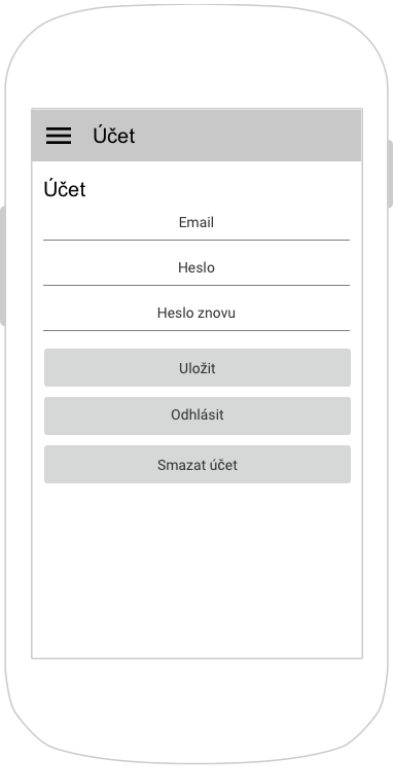
4.1.1.1 Drátěný model

- drátěný model pro obrazovku se základními informacemi o aplikaci (obrázek 4.1)
- drátěný model obrazovky pro administraci uživatelského účtu (obrázek 4.2)
- drátěný model obrazovky pro přihlášení (obrázek 4.3)
- drátěný model obrazovky pro registraci (obrázek 4.4)

Výše zmíněné drátěné modely mohou být upraveny v pozdějších fázích vývoje (po zpracování výstupů ze zpětných vazeb z jednotlivých iterací).

4.1.1.2 Výstupy ze zpětné vazby

K návrhu nebyly uvedeny žádné připomínky.



Obrázek 4.2: Drátěný model obrazovky pro administraci uživatelského účtu

.

4.1.2 1. iterace

4.1.2.1 Uživatelský průzkum

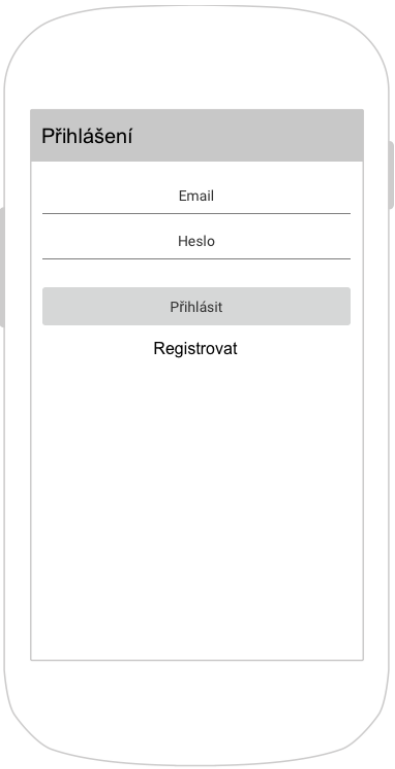
Výsledky uživatelského průzkumu 1. iterace jsou uvedeny v tabulce 4.1. Jako nejvíce preferovanou funkcí v 1. iteraci byla vybrána funkce *Kalkulačka důchodového věku*.

4.1.2.2 Drátěný model

Drátěný model pro funkci *Kalkulačka důchodového věku* je zobrazen na obrázku 4.5.

4.1.2.3 Výstupy ze zpětné vazby

- vstupy funkce (datum narození, pohlaví a počet dětí) budou uváděny již při registraci (možnost aktualizace bude v části pro administraci uživatelského účtu)

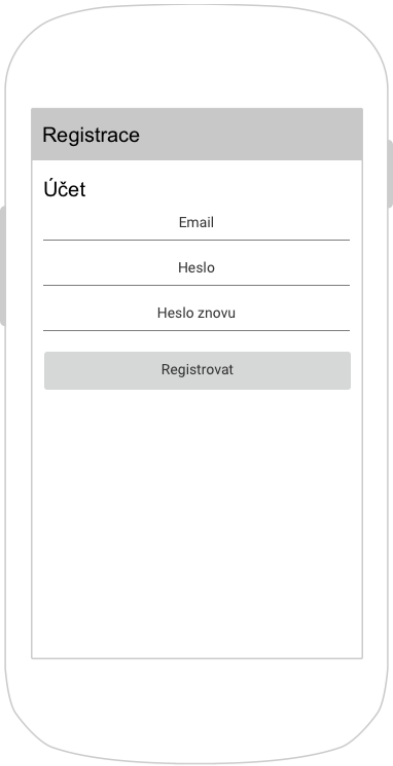


Obrázek 4.3: Drátěný model obrazovky pro přihlášení

.

Tabulka 4.1: Uživatelský průzkum – 1. iterace

Název funkce	Počet hlasů
Kalkulačka předpokládané výše důchodu	2
Kalkulačka důchodového věku	3
Porovnání počtu pracujících a důchodců	0
Srovnání nákladů na důchody v ČR podle roku	0
Procentuální nárůst počtu důchodců za období	0
Průměrná délka pobíraní důchodu	0
Výše důchodu dle území	0
Počet důchodců dle území	0



Obrázek 4.4: Drátěný model obrazovky pro registraci

.

- funkce bude obsahovat grafické zobrazení procentního naplnění důchodového věku (např. pomocí koláčového grafu)

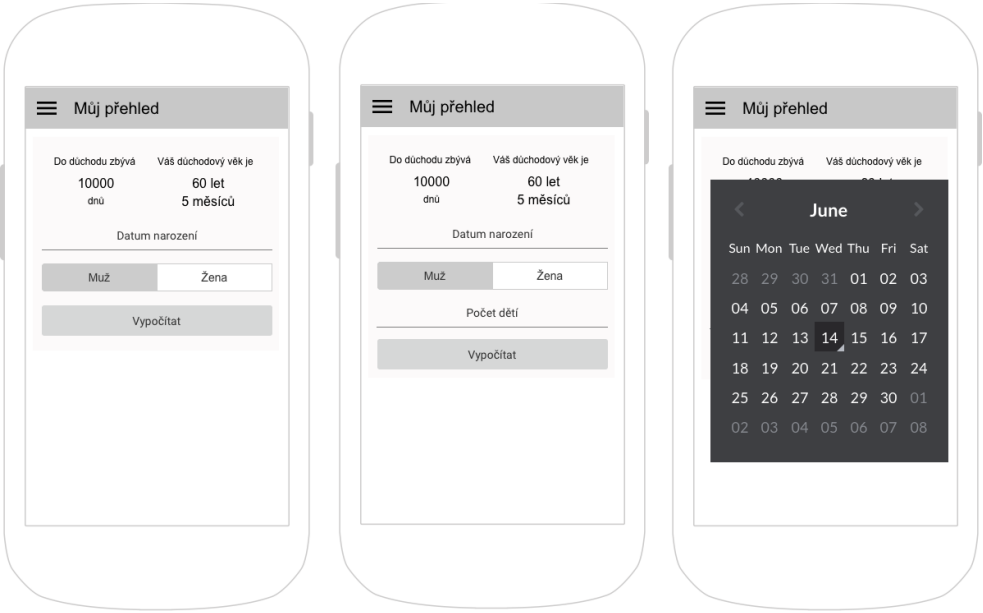
4.1.2.4 Úpravy drátěného modelu

Upravený drátěný model pro funkci *Kalkulačka důchodového věku* je zobrazen na obrázku 4.6. Dále byl upraven drátěný model obrazovky pro registraci (obrázek 4.7) a drátěný model obrazovky pro administraci uživatelského účtu (obrázek 4.8).

4.1.3 2. iterace

4.1.3.1 Uživatelský průzkum

Výsledky uživatelského průzkumu 2. iterace jsou uvedeny v tabulce 4.2. Jako nejvíce preferovanou funkcí v 2. iteraci byla vybrána funkce *Kalkulačka předpokládané výše důchodu.*



Obrázek 4.5: Drátěný model pro funkci *Kalkulačka důchodového věku*

.

Tabulka 4.2: Uživatelský průzkum – 2. iterace

Název funkce	Počet hlasů
Kalkulačka předpokládané výše důchodu	4
Porovnání počtu pracujících a důchodců	0
Srovnání nákladů na důchody v ČR podle roku	1
Procentuální nárůst počtu důchodců za období	0
Průměrná délka pobíraní důchodu	0
Výše důchodu dle území	0
Počet důchodců dle území	0

Obrázek 4.6: Drátěný model pro funkci *Kalkulačka důchodového věku* (po zpracování připomínek ze zpětné vazby)

.

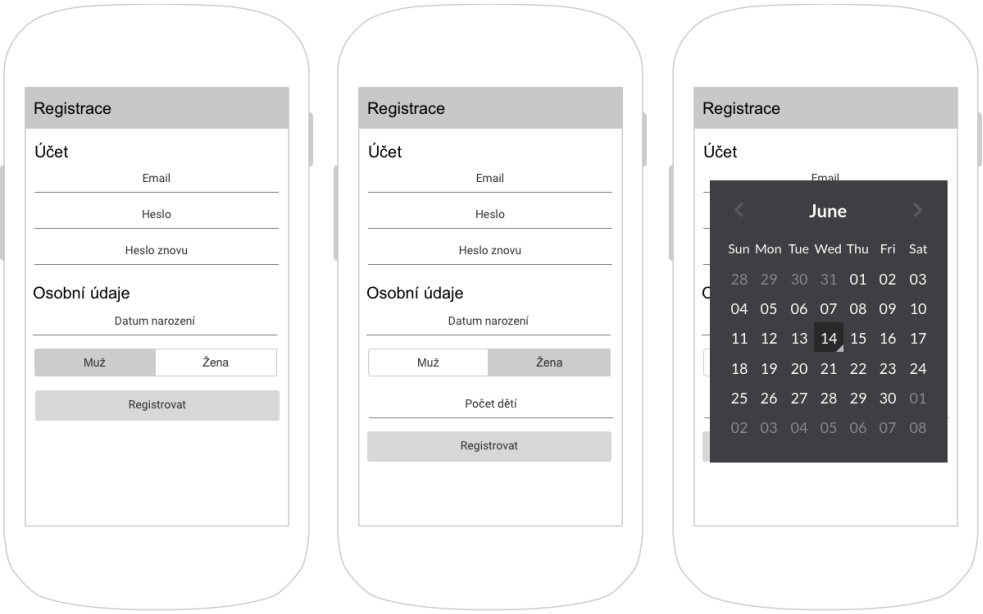
4.1.3.2 Drátěný model

Drátěný model pro funkci *Kalkulačka předpokládané výše důchodu* je zobrazen na obrázku 4.9.

4.1.3.3 Výstupy ze zpětné vazby

- část s přehledem uživatelských funkcí bude zobrazovat poslední vypočtenou předpokládanou výši důchodu spolu s porovnáním s průměrnou výší důchodu

- část s přehledem uživatelských funkcí bude zobrazovat dobu minimální účasti na sociálním pojištění (rozšíření je možné, neboť funkce *Kalkulačka důchodového věku* je již implementována)



Obrázek 4.7: Drátěný model obrazovky pro registraci (po zpracování připomínek ze zpětné vazby z 1. iterace)

.

Tabulka 4.3: Uživatelský průzkum – 3. iterace

Název funkce	Počet hlasů
Porovnání počtu pracujících a důchodců	1
Srovnání nákladů na důchody v ČR podle roku	3
Procentuální nárůst počtu důchodců za období	0
Průměrná délka pobíraní důchodu	1
Výše důchodu dle území	0
Počet důchodců dle území	0

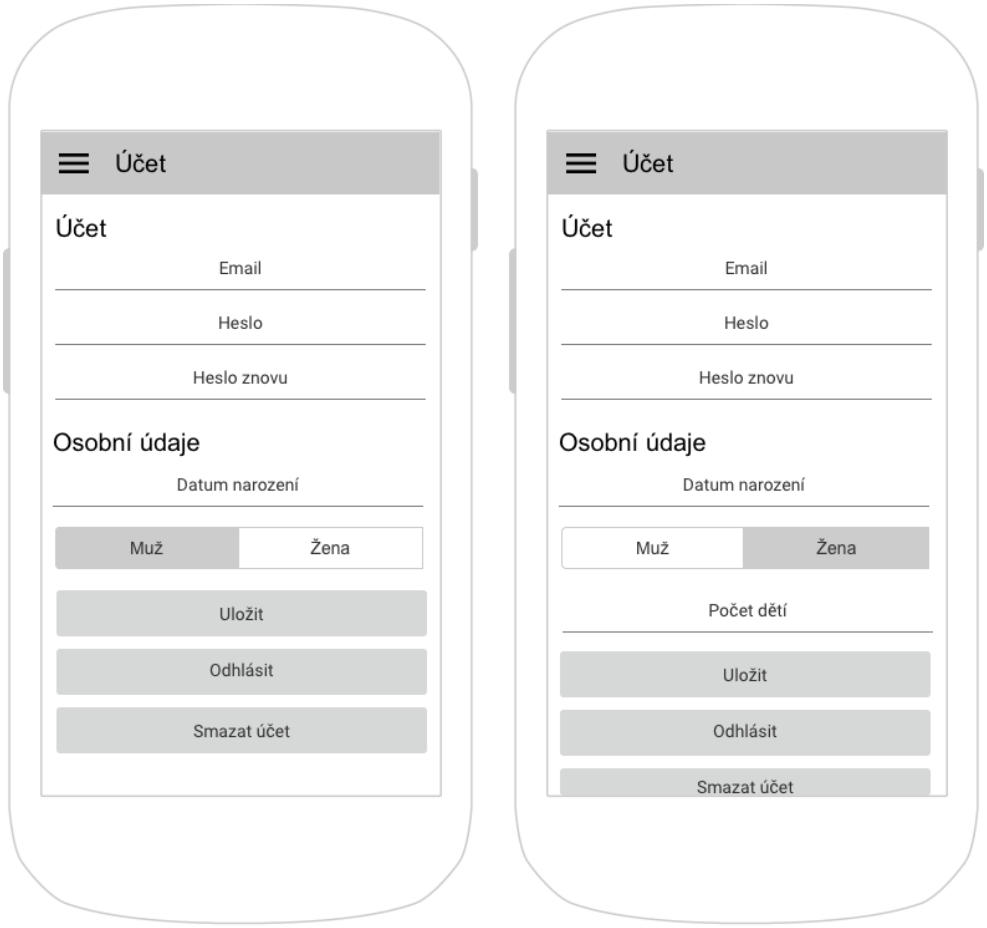
4.1.3.4 Úpravy drátěného modelu

Drátěný model pro rozšíření funkce *Kalkulačka předpokládané výše důchodu* plynoucí ze zpětné vazby funkce je zobrazen na obrázku 4.10.

4.1.4 3. iterace

4.1.4.1 Uživatelský průzkum

Výsledky uživatelského průzkumu 3. iterace jsou uvedeny v tabulce 4.3. Jako nejvíce preferovanou funkcí ve 3. iteraci byla vybrána funkce *Srovnání nákladů na důchody v ČR podle roku.*



Obrázek 4.8: Drátěný model obrazovky pro administraci uživatelského účtu (po zpracování připomínek ze zpětné vazby z 1. iterace)

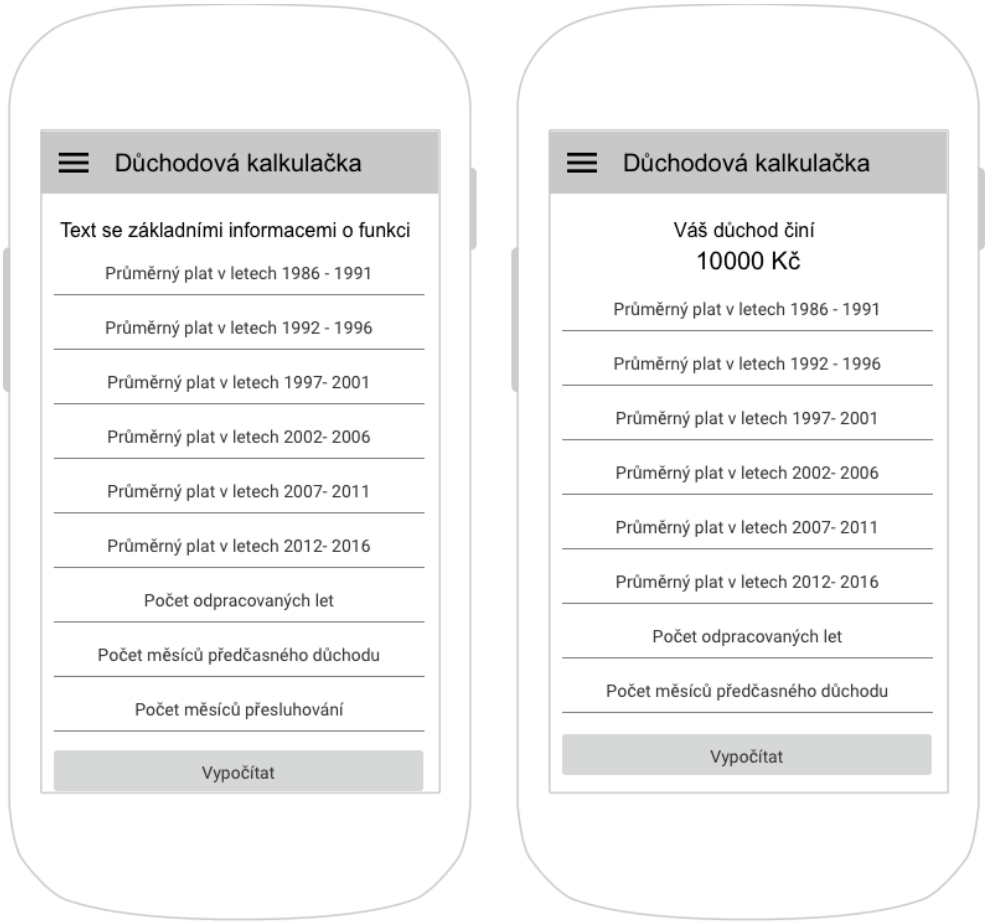
.

4.1.4.2 Drátěný model

Drátěný model pro funkci *Srovnání nákladů na důchody v ČR podle roku* je zobrazen na obrázku 4.11.

4.1.4.3 Výstupy ze zpětné vazby

- funkce bude poskytovat informace o meziročním procentním nárůstu nákladů na důchody (je možné zobrazit v titulku vybrané hodnoty v grafu)



Obrázek 4.9: Drátěný model pro funkci *Kalkulačka předpokládané výše důchodu*

.

4.1.5 4. iterace

4.1.5.1 Uživatelský průzkum

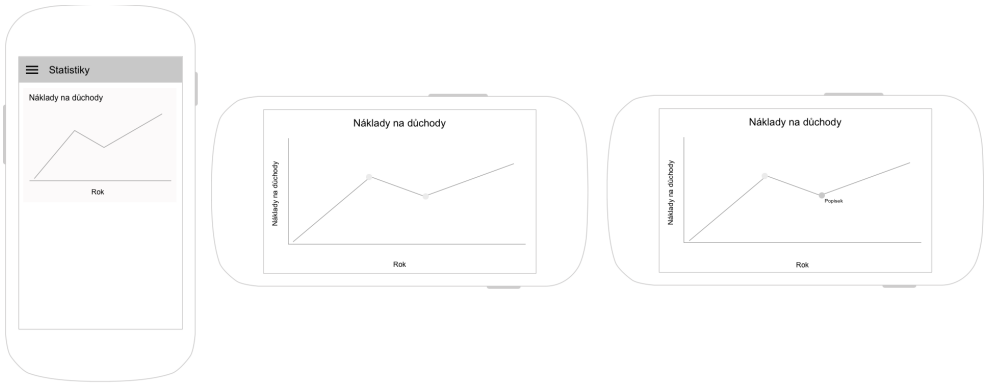
Výsledky uživatelského průzkumu 4. iterace jsou uvedeny v tabulce 4.4. Jako nejvíce preferovanou funkcí ve 4. iteraci byla vybrána funkce *Porovnání počtu pracujících a důchodců*.

4.1.5.2 Drátěný model

Drátěný model pro funkci *Porovnání počtu pracujících a důchodců* je zobrazen na obrázku 4.12.

Obrázek 4.10: Drátěný model pro rozšíření funkce *Kalkulačka předpokládané výše důchodu*

.

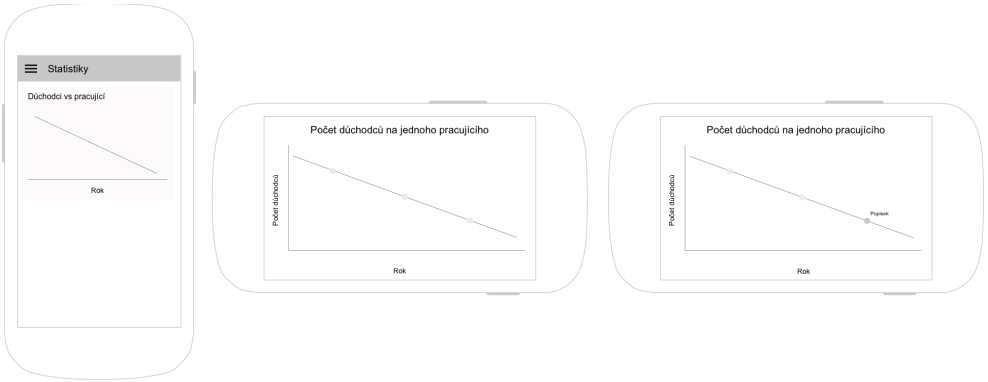


Obrázek 4.11: Drátěný model pro funkci *Srovnání nákladů na důchody v ČR podle roku*

.

Tabulka 4.4: Uživatelský průzkum – 4. iterace

Název funkce	Počet hlasů
Porovnání počtu pracujících a důchodců	2
Procentuální nárůst počtu důchodců za období	0
Průměrná délka pobíraní důchodu	1
Výše důchodu dle území	1
Počet důchodců dle území	1



Obrázek 4.12: Drátěný model pro funkci *Porovnání počtu pracujících a důchodců*

4.1.5.3 Výstupy ze zpětné vazby

- funkce bude zobrazovat i počet pracujících na jednoho důchodce

4.1.5.4 Úpravy drátěného modelu

Upravený drátěný model pro funkci *Porovnání počtu pracujících a důchodců* je zobrazen na obrázku 4.13.

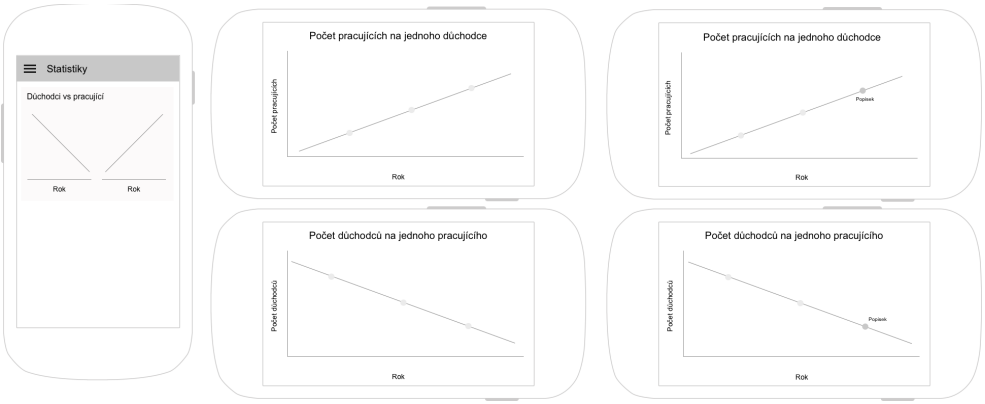
4.1.6 5. iterace

4.1.6.1 Uživatelský průzkum

Výsledky uživatelského průzkumu 5. iterace jsou uvedeny v tabulce 4.5. Jako nejvíce preferovanou funkcí v 5. iteraci byla vybrána funkce *Průměrná délka pobíraní důchodu*.

4.1.6.2 Drátěný model

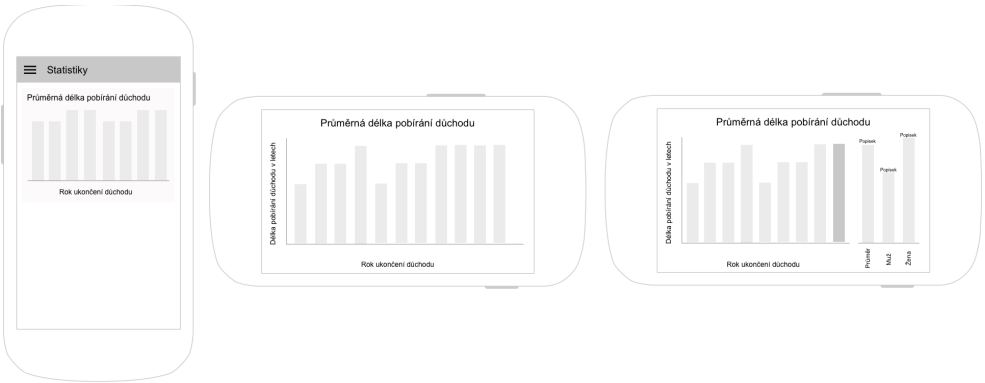
Drátěný model pro funkci *Průměrná délka pobíraní důchodu* je zobrazen na obrázku 4.14.



Obrázek 4.13: Drátěný model pro funkci *Porovnání počtu pracujících a důchodců* (po zpracování připomínek ze zpětné vazby)

Tabulka 4.5: Uživatelský průzkum – 5. iterace

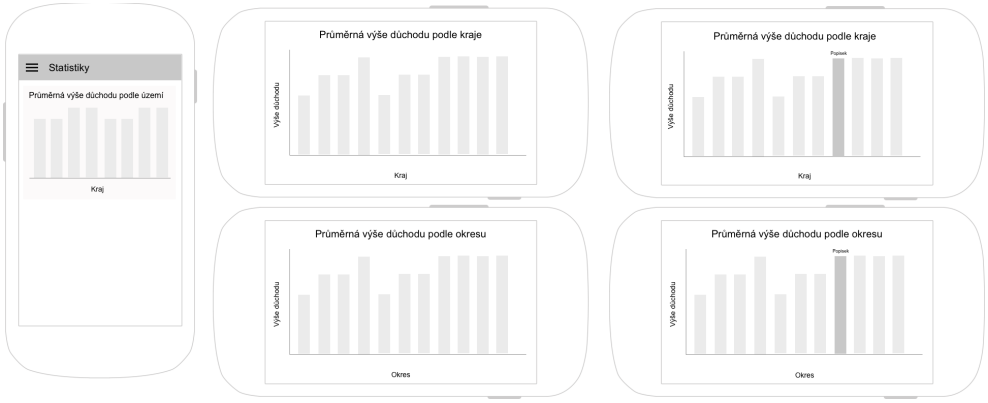
Název funkce	Počet hlasů
Procentuální nárůst počtu důchodců za období	0
Průměrná délka pobíraní důchodu	3
Výše důchodu dle území	1
Počet důchodců dle území	1



Obrázek 4.14: Drátěný model pro funkci *Průměrná délka pobíraní důchodu*

Tabulka 4.6: Uživatelský průzkum – 6. iterace

Název funkce	Počet hlasů
Procentuální nárůst počtu důchodců za období	0
Výše důchodu dle území	4
Počet důchodců dle území	1



Obrázek 4.15: Drátěný model pro funkci *Výše důchodu dle území*

4.1.6.3 Výstupy ze zpětné vazby

K návrhu nebyly uvedeny žádné připomínky.

4.1.7 6. iterace

4.1.7.1 Uživatelský průzkum

Výsledky uživatelského průzkumu 6. iterace jsou uvedeny v tabulce 4.6. Jako nejvíce preferovanou funkcí v 6. iteraci byla vybrána funkce *Výše důchodu dle území*.

4.1.7.2 Drátěný model

Drátěný model pro funkci *Výše důchodu dle území* je zobrazen na obrázku 4.15.

4.1.7.3 Výstupy ze zpětné vazby

- funkce bude zobrazovat výši důchodu dle území pouze pro aktuální rok

4.2 Implementace

Tato sekce popisuje vybrané části implementace mobilní aplikace a serverové části.

4.2.1 Ukládání uživatelských hesel

Problematika ukládání uživatelských hesel se týká snad každého systému, který s uživatelskými účty pracuje. Hesla jsou velmi citlivá data, neboť odcizení hesel může mít na uživatele zvlášť negativní dopad. Negativní dopad odcizení hesel ještě zvyšuje nezodpovědné chování uživatelů, kteří navzdory doporučením o bezpečném chování na internetu opakovaně používají stejná hesla pro různé systémy. Odcizení hesla v jednom systému může v důsledku znamenat narušení uživatelských účtů i v jiných systémech. Z tohoto důvodu je nutné věnovat problematice ukládání hesel zvláštní péči.

Rozhodně špatným přístupem je ukládat hesla v jejich čisté podobě (tzv. v *plain textu*). V případě narušení databáze má útočník k dispozici hesla pro všechny uživatele.

Vhodnějším přístupem je ukládat hesla v jejich hashované podobě. Na uživatelského heslo je aplikována hashovací funkce a teprve její výstup je uložen do databáze. Díky vlastnostem hashovacích funkcí z uloženého hashe hesla není možné zpětně dopočítat hodnotu původního hesla. Takto uložená hesla jsou již lépe zabezpečena, ale stále není ošetřen stav, kdy různí uživatelé používají stejné heslo. V případě znalosti hesla a hashe pro určitého uživatele je útočník schopen získat i hesla uživatelů, kteří používají stejné heslo (stejný vstup do hashovací funkce generuje stejný výstup). Z tohoto důvodu se používá technika solení hesla. Technika solení hesla spočívá ve vygenerování náhodného řetězce a přidání tohoto řetězce k heslu před aplikací hashovací funkce. Do hashovací funkce vstupuje heslo rozšířené o sůl. Tento princip zajistí, že každá stejná hesla mají různý hash. V databázi se pak ukládá sůl a hash hesla. Ověření hesla se provádí tak, že k příchozímu heslu je připojena jeho sůl (pro daný uživatelský účet), takto vzniklý řetězec je zahashován a výstup z hashovací funkce je porovnán proti hashi uloženému v databázi.

Pro ukládání hesel byl v serverové části implementován mechanismus hashování a solení hesla.

4.2.2 Asynchronní exekuce požadavků

Jelikož veškerá data zobrazovaná v mobilní aplikaci pochází od serverové části, je nutné se zabývat problémem horší odezvy. V případě, že uživatel bude mít pomalejší připojení k internetu, získání dat od serverové části bude trvat delší dobu. Problémem mnohých aplikací je, že v tomto stavu uživatelské rozhraní zamrzne a čeká, až data od serverové části obdrží. Toto řešení je z uživatelského hlediska nevhodné, neboť uživatel neví, zda aplikace pracuje či nikoliv.

Z tohoto důvodu je vhodné vykonávat požadavky na serverovou část asynchronně. V uživatelském rozhraní je poté vhodné zobrazit nějaký indikátor, že aplikace pracuje (např. progress bar). Části uživatelského rozhraní, které nezávisí na vykonání požadavku, pak nejsou blokovány dlouhotrvajícími požadavky a reagují na uživatelský vstup.

V implementaci mobilní aplikace byl použit asynchronní model exekuce požadavků na serverovou část.

4.2.3 Automatické přihlašování do aplikace

Vzhledem k nutnosti použití uživatelského účtu pro přístup do mobilní aplikace, je nutné se při každém použití mobilní aplikace přihlásit. Standardem bývá, že uživatel se přihlásí či registruje pouze jednou a při dalším použití mobilní aplikace je tato akce vykonána automaticky. Pro implementaci této funkcionality poskytuje operační systém Android třídu `android.content.SharedPreferences`. Tato třída umožňuje uložení dat v mobilním zařízení ve formě klíč-hodnota. Fyzicky jsou data uložena v souboru. Práva pro přístup k tomuto souboru jsou definovány při získávání objektu třídy `android.content.SharedPreferences`. Pro uložení uživatelského jména a hesla je téměř nutností nastavit práva pro přístup k tomuto souboru pouze dané aplikaci. V mobilní aplikaci jsou přihlašovací údaje při první přihlášení či registraci uloženy a při opakovaném použití použity pro přihlášení.

4.3 Použité technologie a knihovny

Tato sekce popisuje technologie a knihovny použité při implementaci.

4.3.1 Serverová část

Tato podsekce popisuje vybrané technologie a knihovny použité při implementaci serverové části.

4.3.1.1 body-parser

Body-parser slouží k předzpracování příchozích požadavků na serverovou část a zjednodušuje tvorbu odchozích požadavků. Příchozí požadavek je vývojáři předložen již rozparsovaný (např. tělo požadavku je dostupné přes atribut *body*).

V implementaci tato knihovna primárně slouží k ulehčení obsluhy příchozích požadavků.

4.3.1.2 csvtojson

Knihovna csvtojson slouží ke převodu dat ve formátu CSV do JSON objektů.

Protože většina dat z použitých zdrojů je ve formátu CSV, tato knihovna je použita pro převod stažených dat do JSON objektů, se kterými technologie Node.js interně pracuje.

4.3.1.3 express

Express je framework pro tvorbu webových aplikací v technologii Node.js.

V serverové části je použit jako hlavní technologie pro tvorbu komunikačního rozhraní (REST API).

4.3.1.4 morgan

Knihovna morgan slouží k logování požadavků na REST API.

Při implementaci byla knihovna použita zejména pro sledování požadavků na REST API serverové části a ověření správnosti jejich vykonání.

4.3.1.5 express-hateoas-links

Tato knihovna je rozšířením frameworku express, který umožňuje přikládání odkazů popisující REST API k odchozím požadavkům.

V implementaci tato knihovna přikládá odkazy na dostupné zdroje do odchozích požadavků (např. dotaz na kořenový zdroj REST API vratí v atributu *links* seznam odkazů na jednotlivé části REST API).

4.3.1.6 cheerio a cheerio-tableparser

Knihovny cheerio a cheerio-tableparser implementují jádro knihovny jQuery a mimo jiné umožňuje jednoduše parsovat HTML stránky (např. vyhledávání elementů pomocí názvů nebo selektorů).

V serverové části jsou tyto knihovny použity pro získávání dat z HTML stránek.

4.3.1.7 jsonwebtoken

Knihovna jsonwebtoken poskytuje funkce pro manipulaci s JWT tokenem, který slouží k zabezpečení REST API aplikace. Mezi tyto funkce patří vytvoření (a podepsání) tokenu, jeho verifikaci a funkce pro získání dat z tokenu. Pro podepsání tokenu je možno použít buď certifikát, nebo tajný řetězec (tzv. secret) definovaný na straně serverové části.

V implementaci je tato knihovna použita pro zabezpečení chráněných zdrojů REST API a pro autentizaci uživatele při vstupu do aplikace. Pro podepisování tokenu byl použit tajný řetězec. Nastavení vlastností zabezpečovacího tokenu je možné v konfiguračním souboru `config.js`. Zde je možné konfigurovat délku platnosti tokenu (po jakou dobu je token validní) a tajný řetězec pro podepisování tokenu.

4.3.1.8 mongoose

Mongoose je nástroj pro integraci databáze MongoDB a aplikace vyvíjené v technologii Node.js. Jedná se o obdobu ORM frameworku pro technologii Node.js a databázi MongoDB. Framework umožňuje tvorbu a validaci databázových schémat, datovou persistenci a tvoru databázových dotazů založených na jazyce Javascript.

Pomocí tohoto nástroje je v implementaci serverové části řešena veškerá persistence dat a dotazování nad uloženými daty. Nastavení připojení k databázi je specifikováno v konfiguračním souboru `config.js`.

4.3.1.9 node-schedule

Knihovna node-schedule umožňuje periodické spouštění úloh. Interval spouštění úloh je možné nastavit pomocí cron intervalu, intervalu na základě datumu a implementovaných pravidel.

V implementaci je tato knihovna použita pro pravidelné spouštění skriptů pro aktualizaci dat z použitých zdrojů. Interval spouštění těchto aktualizací je možné nastavit v konfiguračním souboru `config.js`. Dále je zde také možnost aktualizace vypnout.

4.3.1.10 underscore

Knihovna underscore poskytuje velké množství funkcí pro operace nad kolekcemi a zjednodušuje jejich zpracování.

V implementaci je tato knihovna použita zejména pro zpracovávání stažených dat z použitých zdrojů. Vzhledem k rozsáhlosti některých použitých datových sad, je jejich zpracování náročné. S použitím této knihovny je možné zpracovávat tyto data s minimem použitého kódu.

4.3.2 Aplikace

V této podsekci jsou popsány technologie a knihovny použité při implementaci mobilní aplikace.

4.3.2.1 HelloCharts

HelloCharts je grafická knihovna pro operační systém Android pro tvorbu interaktivních grafů. Knihovna nabízí poměrně velké množství typů grafů spolu s velkou škálou možností přizpůsobitelnosti grafů požadavkům mobilní aplikace.

Grafy byly využity zejména pro vizualizaci uživatelských a statistických funkcí.

4.3.2.2 Retrofit

Retrofit je HTTP klient pro aplikace vyvíjené na platformě Android nebo Java. Tento framework výrazně zjednodušuje implementaci komunikace přes REST API. Mimo jiné umožňuje snadnou konverzi mezi JSON a Java objekty, synchronní a asynchronní exekuci požadavků, obsluhu odpovědí na odeslané požadavky a mnoho dalších problémů spojených s komunikací přes REST API.

Následující seznam popisuje základní komponenty frameworku a jejich použití v implementaci.

1. **Model**
   Model slouží k definici přenášených dat prostřednictvím obyčejných javovských tříd. Při odesílání dat jsou instance těchto tříd konvertovány do JSON objektů a dále odesílány. Při příchozí komunikaci jsou naopak JSON objekty převáděny do objektů těchto tříd. Tato konverze je prováděna automaticky a vývojář je od ní plně odstíněn.

   Třídy popisující datový model aplikace se nachází v balíku `gregemar.fit.cvut.dpapp.rest.model`.

2. **Komunikační rozhraní**
   Druhou částí frameworku je definice komuničního rozhraní prostřednictvím javovských rozhraní (Java interface) s metodami popisující jednotlivé požadavky. Tyto metody jsou řízeny pomocí anotací, které slouží ke specifikaci HTTP metody, těla požadavku a případně dalších atributů požadavku. Příklad takové metody je popsán níže.

   ```
   @POST("/api/auth/register")
   Call<AuthResult> register(@Body Settings settings);
   ```

   Jak je vidět na příkladu, každá taková metoda navrací objekt datového typu `Call`. Požadavek popsaný tímto objektem umožňuje synchronní nebo asynchronní exekuci.

   Javovské rozhraní popisující komunikační rozhraní jsou umístěny v balíku `gregemar.fit.cvut.dpapp.rest.service`.

3. **Poskytovatel služby (service provider)**
   Tato třída umožňuje vytvářet konkrétní instance rozhraní s komunikačními metodami. Rovněž se v této třídě nastavují veškeré parametry komunikace. V mobilní aplikaci je tato funkcionalita implementována ve třídě `RestProvider`.

Po implementaci potřebných tříd a rozhraní z výše popsaného seznamu je aplikace schopná komunikovat přes REST rozhraní pomocí HTTP protokolu. Komunikace pomocí protokolu HTTPS je možná pouze v případě použití veřejně ověřitelného certifikátu na straně serveru. V případě použití certifikátu

podepsaného sám sebou (self-signed certifikátu) je nutné přiložit tento certifikát k aplikaci a vynutit jeho důvěryhodnost ve třídě nastavující parametry komunikace (třída výše popsaná jako poskytovatel služby). Použití sám sebou podepsaného certifikátu je implementaci mobilní aplikace konfigurovatelné.

Další funkcí, kterou tento framework poskytuje, je zabezpečení komunikace. Pro zabezpečení komunikace se serverovou částí byl použit JWT token. Pro obsluhu JWT tokenu byly implementovány následující třídy:

- `AuthInterceptor`
- `TokenAuthenticator`

Třída `AuthInterceptor` implementuje rozhraní `okhttp3.Interceptor` a její funkce je automatické přibalování JWT tokenu ke každému odchozímu požadavku.

Třída `TokenAuthenticator` implementuje rozhraní `okhttp3.Authenticator`. Má na starost obsluhu požadavků, které skončí stavovým kódem 401 Unauthorized. Pokud aplikace odešle požadavek na chráněný zdroj REST API serverové části a vrátí se odpověď se stavovým kódem 401 Unauthorized, `TokenAuthenticator` provede následující posloupnost kroků:

1. odchytí odpověď se stavovým kódem 401 Unauthorized
2. provede přihlášení do aplikace (synchronní požadavek)
3. aktualizuje přístupový JWT token
4. opakuje odmítnutý požadavek
5. vrátí výsledek opakovaného požadavku

Implementace tříd `AuthInterceptor` a `TokenAuthenticator` je popsána níže.

```
public class AuthInterceptor implements Interceptor {

    @Override
    public Response intercept(Chain chain)
    throws IOException {
        Request result = chain.request();
        String token = AuthManager.getAccessToken();
        if (!TextUtils.isEmpty(token)) {
            result = result.newBuilder()
                .header("x-access-token", token)
                .build();
        }
        return chain.proceed(result);
    }

}

public class TokenAuthenticator implements
    Authenticator {

    @Override
    public Request authenticate(Route route,
        Response response) throws IOException {
        Log.d(getClass().getCanonicalName(),
            "Refreshing access token");
        AuthService authService = RestProvider
            .createService(AuthService.class);
        Call<AuthResult> loginCall = authService
            .login(AuthManager.getAutoLogin());
        String refreshedToken = loginCall.execute()
            .body().getToken();
        AuthManager.setAccessToken(refreshedToken);
        return response.request().newBuilder()
            .header("x-access-token", refreshedToken)
            .build();
    }

}
```

Ve výsledku Retrofit představuje komplexní nástroj pro komunikaci přes REST rozhraní umožňující snadné volání požadavků a v jisté míře i automatizaci obsluhy JWT tokenu.

Kapitola 5

Testování

Kapitola popisuje testování mobilní aplikace a serverové části.

5.1 Testování serverové části

Při testování serverové části byl kladen důraz zejména na správnou funkčnost poskytovaného REST API, neboť bez správně fungujícího REST API nemůže mobilní aplikace fungovat.

Pro testování serverové části byly použity tyto nástroje:

- Postman
- Apiary

5.1.1 Postman

Postman je nástroj pro testování REST API. Umožňuje tvorbu požadavků na REST API, jejich exekuci a testování odpovědí. Pro pohodlnější provádění testů nástroj poskytuje možnost vytvoření prostředí s uživatelem definovanými proměnnými. Tvorba požadavků se provádí pomocí grafického rozhraní nástroje, testy odpovědí se popisují pomocí jazyka Javascript.

5.1.1.1 Testy

Pro testování REST API serverové části bylo vytvořeno prostředí s proměnnými, které popisují adresu REST API, cesty k jednotlivým zdrojům REST API a uživatelské jméno a heslo pro testovacího uživatele. Dále je toto prostředí použito pro ukládání tokenu pro přístup ke chráněným zdrojům REST API.

Testy je pokryto celé REST API (pro každý zdroj REST API je vytvořen test) dle následujícího scénáře (v případě nástroje Postman se pro testovací scénáře používá pojem *kolekce*):

1. vytvoření uživatelského účtu
2. přihlášení do aplikace
3. získání dat pro uživatelské funkce
4. provedení funkce *Kalkulačka předpokládané výše důchodu*, ověření aktualizace poslední vypočtené předpokládané výše důchodu k danému uživatelskému účtu
5. získání dat o uživatelském účtu, aktualizace uživatelského účtu a ověření, zda se aktualizace uživatelského účtu provedla správně
6. získání dat pro statistické funkce
7. smazání uživatelského účtu

Odpověď na každý testovací požadavek je testována na HTTP stavový kód. U odpovědí vracející data jsou mimo testu na stavový kód HTTP odpovědi provedeny následující testy:

- test přítomnosti HTTP hlavičky `Content-Type`
- test datového formátu přijatých dat (HTTP hlavička `Content-Type` musí obsahovat hodnotu `application/json`)
- test struktury dat odpovědi testovacího požadavku (validace přijatých dat proti v testu definovanému schématu)
- test hodnot dat odpovědi testovacího požadavku (test je prováděn pouze u požadavků, u kterých dopředu známe hodnotu dat - např. u požadavku na získání informací o uživatelském účtu očekáváme hodnoty dat stejné jako hodnoty použité v požadavku pro registraci)

Soubory popisující testy a soubor s výsledky testů jsou přiloženy ve složce `test/server/postman` k přiloženém CD. Jedná se o soubor `DPServer_test_env.postman_environment` popisující prostředí pro proměnné použité v testech, soubor `DPServer_test_collection.postman_collection` popisující scénář testů (testovací kolekci) a soubor `DPServer_test_collection.postman_test_run` popisující výsledky testů.

Pro provedení testů REST API je nutné importovat soubory popisující prostředí a testy do nástroje Postman.

5.1.2 Apiary

Apiary je služba pro dokumentaci a testování REST API. Pro popis REST API se používá jazyk Blueprint. Na základě tohoto popisu služba vygeneruje interaktivní dokumentaci. V dokumentaci je mimo jiné možné provádět požadavky na REST API a tím i provádět testování.

Soubor s popisem REST API v jazyce Blueprint je přiložen k přiloženému CD. Jedná se o soubor `api_doc.blueprint` ve složce `test/server/apiary`. V současné době služba neposkytuje možnost exportu dokumentace do formátu PDF nebo HTML, dokumentace je tedy dostupná pouze v samotné službě.

5.2 Testování mobilní aplikace

Platforma Android poskytuje 2 základní typy testů. Prvním typem jsou testy prováděné na mobilním zařízení. Druhým typem jsou testy, které pro exekuci mobilní zařízení nepotřebují. Druhý typ testů je oproti prvnímu limitován v absenci aplikačního kontextu (během testů aplikace neběží), tudíž pomocí nich nelze testovat komponenty aplikace, které s aplikačním kontextem pracují.

Vzhledem k implementaci aplikace byly použity testy prováděné na mobilním zařízení. Pro testování mobilní aplikace byly použity následující technologie:

- JUnit
- Espresso
- Awaitility

5.2.1 JUnit

JUnit je framework pro tvorbu jednotkových testů (unit testů).

Při testování aplikace jsou jednotkové testy použity pro ověření správné funkčnosti komunikace se serverovou částí (schopnost vykonat požadavky na REST API serverové části a získat data ve správném formátu), pro ověření výpočtů nad přijatými daty a k ověření spustitelnosti aplikace (kontrola přítomnosti konfiguračních proměnných).

5.2.2 Espresso

Espresso je framework pro testování uživatelského rozhraní mobilní aplikace pro operační systém Android.

V rámci testování mobilní aplikace slouží testy vytvořené v tomto frameworku k ověření správnosti chování viditelných částí aplikace (jednotlivých

obrazovek) a k simulaci uživatelského chování. V testech se primárně ověřuje přítomnost (případně viditelnost) grafických komponent a reakce aplikace na dostupné uživatelské akce. Účelem těchto testů je minimalizace vzniku neočekávaného pádu aplikace.

5.2.2.1 Testy

Před provedením každého testu vyžadující uživatelský účet (testy grafických komponent, které vykonávají požadavky na zabezpečené zdroje REST API serverové části) je vytvořen testovací účet. Testovací účet je provedení testů následně smazán.

V mobilní aplikaci byly implementovány následující testy uživatelského rozhraní:

- `LoginActivityTest`
  V tomto testu je testována aktivita `LoginActivity`. Jsou prováděny testy přítomnosti grafických komponent a testy simulující úspěšný a neúspěšný požadavek na přihlášení do aplikace.

- `RegisterActivityTest`
  V tomto testu je testována aktivita `RegisterActivity`. Jsou prováděny testy přítomnosti grafických komponent a testy simulující úspěšný a neúspěšný požadavek na registraci do aplikace.

- `MainActivityTest`
  V tomto testu je testována aktivita `MainActivity`. Nejprve je otestována správná funkčnost menu. Následně jsou procházeny jednotlivé části mobilní aplikace (simulace průchodu aplikací) a je testována správnost zobrazeného obsahu.

- `PensionCalculatorTest`
  V tomto testu je detailněji testována funkce *Kalkulačka předpokládané výše důchodu*. Jsou prováděny testy přítomnosti grafických komponent a testy simulující úspěšný a neúspěšný požadavek na vykonání této funkce.

Vyjma výše popsaných testů byly vytvořeny testy pro aktivitu `DetailStatisticActivity`, která slouží pro detailní zobrazení vybrané funkce. Tato aktivita je popsána více testovacími třídami, neboť grafický obsah závisí na parametrech, které jsou aktivitě předávány při vytváření.

5.2.3 Awaitility

Framework Awaitility poskytuje podporu pro testování asynchronních částí aplikace (v případě mobilní aplikace jsou všechny požadavky na REST API serverové části volány asynchronně). Framework Awaitility byl použit spolu s frameworkem Espresso k testování asynchronních uživatelských akcí (např.

simulace úspěšného přihlášení). Příklad použití tohoto frameworku je popsán níže.

```
@RunWith(AndroidJUnit4.class)
public class ExampleTestClass {
    ⋮
    @Test
    public void exampleTest() {
        ⋮
        await().until(new Callable<Boolean>() {
            @Override
            public Boolean call() throws Exception {
                return someAsyncCondition;
            }
        }, equalTo(true));
    }
}
```

Exekuce testů byla prováděna pomocí virtuálního mobilního zařízení a vývojového nástroje Android Studio, do kterého jsou výše popsané testovací frameworky integrovány.

Soubor s výsledky testů mobilní aplikace byl přiložen k přiloženému CD. Jedná se o soubor `test_result.html` ve složce `test/app`.

Jelikož mobilních zařízení s operačním systémem Android existuje obrovské množství, není možné mobilní aplikaci otestovat zvlášť pro každé zařízení. Z tohoto důvodu je posledním krokem testování zpřístupnění mobilní aplikace uživatelům (nasazení mobilní aplikace je popsáno zvlášť v sekci 6.2) a analýza chování aplikace na zařízeních uživatelů, kteří si aplikaci nainstalovali. Analýza chování aplikace je řešena pomocí služby Fabric.

5.2.4 Fabric

Služba Fabric slouží pro analýzu chování mobilní aplikace a chování uživatelů v mobilní aplikaci. Konkrétněji služba Fabric nabízí následující funkce:

- analýza aktivity uživatelů (počet uživatelů, kteří mobilní aplikace použili za určité období a průměrná doba používání mobilní aplikace)
- analýza růstu počtu uživatelů za určité období
- analýza aktivity uživatelů dle území (státu)
- analýza výskytu chyb v aplikaci (pro tuto funkci slouží modul Crashlytics)

Veškeré výstupy výše popsaných funkcí jsou dostupné přes webové rozhraní služby.

Pomocí modulu Crashlytics je možné sbírat informace o neodhalených chybách aplikace včetně detailního popisu (k dispozici je celý záznam o vyhozené výjimce). Modul Crashlytics také nabízí filtrování výskytu chyb dle konkrétního mobilního zařízení. V případech opakovaného výskytu určité chyby je plánována její oprava v další verzi mobilní aplikace.

Integrace služby Fabric do mobilní aplikace je poměrně jednoduchou záležitostí, neboť služba poskytuje plugin pro vývojový nástroj Android Studio, ve kterém byla aplikace vyvíjena. Integrace služby Fabric byla provedena pomocí průvodce v tomto pluginu (veškeré změny ve zdrojovém kódu jsou obstarány samotným pluginem). Podmínkou použití služby je vytvoření uživatelského účtu v této službě.

Kapitola 6

Nasazení

Tato kapitola popisuje nasazení mobilní aplikace a serverové části do předprodukčního (testovacího) prostředí.

6.1 Nasazení serverové části

Serverovou část lze provozovat na serveru, na kterém jsou nainstalované následující technologie:

- Node.js ve verzi 6.9.5 nebo vyšší
- databáze MongoDB

Alternativou může být použití cloud hostingových služeb, které podporují výše zmíněné technologie. Výhoda použití těchto služeb spočívá v již předpřipravené infrastruktuře a ve snadném použití. Za úskalí použití těchto služeb lze považovat limitované množství funkcí, které tyto služby poskytují v neplacených verzích, což může představovat problém vzhledem k nutnosti integrace aplikace s databází.

Pro nasazení byla vybrána služba Heroku.

6.1.1 Heroku

Heroku je cloudová hostingová služba. Služba podporuje nejen aplikace vytvořené v technologii Node.js (dalšími podporovanými jazyky jsou např. Java, Ruby, PHP).

Heroku pro hosting aplikací používá tzv. dyno. Dyno je linuxový kontejner umožňující zpracování maximálně jednoho příkazu současně. Tyto kontejnery lze rozdělit do následujících kategorií:

- **Web dynos**
  Tyto kontejnery slouží ke zpracování HTTP požadavků. Ostatní kontejnery tuto funkcionalitu neumožňují.

- **Worker dynos**
  Kontejnery v této kategorii jsou určeny pro úlohy jiné než obsluha HTTP požadavků. Jedná se např. o výpočetně náročné úlohy běžící na pozadí, periodicky spouštěné úlohy atd.

- **One-off dynos**
  Tato kategorie kontejnerů slouží zejména pro administraci (např. migrace databáze).

Pro nasazení byla použita neplacená verze této služby. Neplacená verze je poskytována v následující konfiguraci:

- 1 dyno kontejner (Web/Worker)
- 512 MB RAM
- automatické uspání aplikace po 30ti minutách neaktivity

Automatické uspávání aplikace může zapříčinit delší odezvu při prvním požadavku na REST API serverové části. V několika případech při testování mobilní aplikace nastalo i neúspěšné vykonání požadavku a požadavek se musel zopakovat.

6.1.1.1 Integrace databáze

Služba Heroku také poskytuje možnost hostingu MongoDB databáze, avšak její použití je placené. Z toho důvodu byla pro hosting databáze zvolena služba mLab, která nabízí v neplacené vezi 0,5 GB databázového prostoru. Integrace mezi službami mLab a Heroku je zajištěna pomocí addonu mLab MongoDB, který je poskytován službou Heroku. Administraci databáze lze provádět přes webové rozhraní služby mLab. Připojení k databázi z aplikace (serverové části) je následně možné nastavením údajů o hostované databázi v konfiguraci databázového připojení v konfiguračním souboru `config.js`.

6.1.1.2 Postup nasazení

Před samotným nasazením aplikace na tuto službu, je nutné vytvořit uživatelský účet a nainstalovat klienta, který umožňuje obsluhu služby prostřednictvím příkazové řádky. Nutnou podmínkou je také použití verzovacího systému Git. Nasazení se porovádí v adresáři aplikace.

Následující seznam popisuje potřebné kroky k nasazení aplikace:

1. přihlášení se ke službě pomocí příkazu `heroku login`, uživatel zadá email a heslo použité uživatelského účtu
2. příkazem `heroku create` vytvoření nové Heroku aplikace

3. nahrání zdrojových kódů do Heroku repozitáře použitím příkazu `git push heroku master` (pro ilustraci je nahrávána `master` větev repozitáře projektu do `master` větve repozitáře ve službě Heroku, je možné nahrát i jiné větve)

4. spuštění aplikace za pomoci příkazu `heroku ps:scale web=1` (po provedení tohoto příkazu aplikace poběží v jednom dyno kontejneru)

Po provedení výše popsaných kroků by aplikace měla být dostupná. Pro ověření je možno aplikaci otevřít v prohlížeči příkazem `heroku open`. Užitečným příkazem je také příkaz `heroku logs`, který zobrazí logy aplikace a dyno kontejneru.

6.2 Nasazení mobilní aplikace

Provoz mobilní aplikace je možný na mobilním telefonu s operačním systémem Android v minimální verzi 5.0 Lollipop (API 21). Mobilní aplikace vyžaduje oprávnění pro přístup k internetu.

Pro distribuci aplikací pro operační systém Android se používají APK balíčky. Z tohoto balíčku je pak možné v mobilním zařízení aplikaci nainstalovat. Takový postup instalace aplikací je uživatelsky nepohodlný a rovněž sám operační systém má k instalaci výhrady (je nutné povolit instalaci aplikací z neověřených zdrojů). Instalace aplikací z neověřených zdrojů rovněž znamená bezpečnostní riziko v zanesení škodlivého kódu do mobilního zařízení. Z tohoto důvodu byla mobilní aplikace nasazena na distribuční službu Google Play.

Mobilní aplikace byla nasazena na distribuční službu Google Play ve fázi předběžného přístupu (early access). Odkaz na aplikaci je součástí přiloženého CD (v souboru `readme.txt`). Aplikace je veřejně dostupná (v sekci pro předběžný přístup), avšak počet uživatelů aplikace je omezen na maximálně 1000.

Závěr

Osobní přínos

Osobní přínos práce spočívá zejména v osvojení si nových technologií jak pro vývoj serverových aplikací, tak pro vývoj mobilních aplikací pro operační systém Android. Dále jsem měl možnost se detailně seznámit se službami a nástroji, které výrazně zjednodušují činnosti prováděné v jednotlivých fázích softwarového projektu. Konkrétně se jedná o služby Heroku pro hosting serverových aplikací, Fabric pro sledování chování aplikace na mobilních zařízeních, Google Play sloužící k distribuci mobilních aplikací, Apiary pro testování a dokumentaci aplikačního rozhraní a o nástroj Postman pro testování aplikačního rozhraní. Přínosem práce je rovněž osvojení si principů agilní vývoje.

Zhodnocení cílů

V práci byly splněny všechny cíle definované v kapitole 1.

Byla provedena analýza zdrojů (otevřených i dalších možných) dat důchodového systému v ČR a analýza existujících mobilních aplikací věnujících se důchodovému systému, která byla rozšířena i o webové aplikace. Na základě těchto analýz byly vybrány vhodné zdroje a navrženy vhodné funkce pro mobilní aplikaci. Na základě analýzy technických řešení pro serverovou část, analýzy databázových technologií a specifikace požadavků na serverovou část byly vybrány vhodné technologie pro implementaci serverové části. Vývoj mobilní aplikace a serverové části byl řízen pomocí iterací, kde v každé iteraci byly provedeny tyto kroky: uživatelský průzkum pro výběr nejvíce preferované funkce, vytvoření drátěného modelu funkce, konzultace drátěného modelu s respondenty, doladění návrhu funkce a zpracování připomínek (případná úprava drátěného modelu), implementace funkce. Mobilní aplikace a serverová část byly otestovány.

Výsledkem této práce je funkční mobilní aplikace. V současné době je mobilní aplikace dostupná ve fázi předběžného přístupu z distribuční služby Go-

ogle Play a serverová část je nasazena na hostingové službě Heroku.

Výhled do budoucna

V nejbližší době je plánována analýza a oprava případných chyb plynoucích z použití mobilní aplikace na různých mobilních zařízeních. Následně je možné mobilní aplikaci zveřejnit v produkčním režimu na distribuční službě Google Play. V případě kladných ohlasů je možné mobilní aplikaci rozšířit o další vhodné funkce.

Literatura

[1] Google Inc.: Android Developers [online]. 2017. Dostupné z: https://developer.android.com

[2] DeBill, E.: Module Counts [online]. 2017, [cit. 2017-05-05]. Dostupné z: http://www.modulecounts.com

[3] Kalkulačka pro orientační výpočet starobního důchodu [online]. 2017, [cit. 2017-03-05]. Dostupné z: http://www.cssz.cz/cz/duchodove-pojisteni/duchodova-kalkulacka/kalkulacka-pro-orientacni-vypocet-starobniho-duchodu.htm

[4] Starobní důchody [online]. 2017, [cit. 2017-03-05]. Dostupné z: http://www.cssz.cz/cz/duchodove-pojisteni/davky/starobni-duchody.htmm

[5] Chlapek, D.; Kučera, J.; Nečaský, M.: Metodika publikace otevřenỳch dat veřejné správy ČR. 2012. Dostupné z: http://www.korupce.cz/assets/partnerstvi-pro-otevrene-vladnuti/otevrena-data/Metodika_Publ_OpenData_verze_1_0.pdf

[6] Wonderlich, J.: Ten principles for opening up government information. *Sunlight Foundation*, ročník 11, 2010.

[7] Otevřená data [online]. 2017, [cit. 2017-03-05]. Dostupné z: https://data.cssz.cz

[8] Výsledky sčítání lidu, domů a bytů 2011 (SLDB 2011) [online]. [cit. 2017-03-05]. Dostupné z: https://www.czso.cz/csu/czso/otevrena_data_pro_vysledky_scitani_lidu_domu_a_bytu_2011_sldb_2011

[9] Finanční kalkulačka [online]. [cit. 2017-03-05]. Dostupné z: https://play.google.com/store/apps/details?id=cz.okhelp.finance

[10] Irish Life Corporate Business: My Pension. [cit. 2017-05-07]. Dostupné z: https://play.google.com/store/apps/details?id=com.irishlife.mypension.activities

[11] Watson, T.: Track My Pension. [cit. 2017-03-07]. Dostupné z: https://play.google.com/store/apps/details?id=com.watsonwyatt.pensions

[12] Výpočet.cz [online]. 2017, [cit. 2017-03-05]. Dostupné z: https://www.vypocet.cz

[13] The Computer Language Benchmarks Game [online]. [cit. 2017-05-15]. Dostupné z: http://benchmarksgame.alioth.debian.org/

[14] Maier, D.: *Theory of relational databases*. Computer Science Pr, 1983.

[15] Pritchett, D.: Base: An acid alternative. *Queue*, ročník 6, č. 3, 2008: s. 48–55.

[16] Smartphone OS Market Share, 2016 Q3 [online]. 2016, [cit. 2017-04-24]. Dostupné z: http://www.idc.com/promo/smartphone-market-share/os

[17] Masse, M.: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011.

[18] Crockford, D.: The application/json media type for javascript object notation (json). 2006.

[19] Jones, M.; Bradley, J.; Sakimura, N.: Json web token (jwt). Technická zpráva, 2015.

[20] Rescorla, E.: Http over tls. 2000.

Příloha **A**

Seznam použitých zkratek

API Application programming interface

APK Android package kit

CSV Comma-separated values

ČSSZ Česká správa sociálního zabezpečení

HATEOAS Hypermedia as the engine of application state

HTML Hypertext markup Language

HTTP Hypertext transfer protocol

HTTPS Hypertext transfer protocol secure

ICT Information and communication technologies

JSON JavaScript object notation

JVM Java virtual machine

JWT JSON web token

NoSQL Not only SQL

ORM Object-relational mapping

OSVČ Osoba samostatně výdělečně činná

PHP Hypertext preprocessor

REST Representational state transfer

RDF Resource description framework

SQL Structured query language

SSL Secure sockets layer

TLS Transport layer security

URI Uniform resource identifier

URL Uniform resource locator

XLS MS Excel file extension

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `apk` ..... adresář s instalačním souborem mobilní aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `app` ..... zdrojové kódy implementace mobilní aplikace
    - `server` ..... zdrojové kódy implementace serverové části
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `test`
  - `app` ..... adresář se soubory popisující testy mobilní aplikace
  - `server` ..... adresář se soubory popisující testy serverové části
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `wireframes` .... adresář s návrhy uživatelského rozhraní mobilní aplikace