FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Mobilní aplikace Důchody
Student:	Bc. Adam Sekereš
Vedoucí:	Ing. Josef Gattermayer
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

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 iOS.
11) Otestujte serverovou část a mobilní aplikaci.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 6. listopadu 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Mobilní aplikace Důchody

Bc. Adam Sekereš

Katedra softvérového ižinierstva
Vedúci práce: Ing. Josef Gattermayer

7. mája 2018

Poďakovanie

Chcel by som sa poďakovať Ing. Josefovi Gattermayerovi za ochotu a čas, ktory mi venoval pri odbornom vedení diplomovej práce.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 7. mája 2018 .....

Odkaz na túto prácu

Sekereš, Adam. *Mobilní aplikace Důchody*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Diplomová práca sa zaoberá tvorbou mobilnej aplikácie pre zariadenia s operačným systémom iOS a serverovej časti aplikácie. Práca sa postupne zaoberá analýzou, návrhom, vývojom a testovaním mobilnej aplikácie a serverovej časti. Cieľom výslednej aplikácie je uľahčiť užívateľom získavanie informácií o dôchodkovom systéme v Českej republike. Aplikácia čerpá dáta zo zdrojov otvorených dát o dôchodkovom systéme. Serverová časť je realizovaná v programovacom jazyku Node.js. Mobilná aplikácia je realizovaná v programovacom jazyku Swift.

Klíčová slova mobilná aplikácia, serverová aplikácia, dôchodkový systém, Node.js, Swift, iOS

Abstract

This diploma thesis deals with the creation of a mobile application for devices with iOS operating system and the server part of the application. The work is gradually analyzing, designing, developing and testing the mobile application and the server part. The goal of the resulting application is to make it easier for users to share information about the pension system in the Czech Republic. The application draws data from open data sources on the pension

system. The server part is implemented in the programming language Node.js. The mobile application is implemented in the Swift programming language.

Keywords mobile application, server application, pension system, Node.js, Swift, iOS

Obsah

Zoznam obrázkov

Úvod

S postupom času, ako ľudia začnú dospievať a starnúť, začínajú sa zamýšľať nad svojou budúcnosťou. Začnú sa viac zaujímať o svoj život na dôchodku. Zisťujú čo dôchodok prináša, aký veľký dôchodok budú mať, kedy pôjdu do dôchodku a podobné informácie. Niektorí tieto záležitosti riešia skôr, iní zas neskôr.

V dnešnej dobe existuje niekoľko zdrojov otvorených dát a informácií, ktoré sa vzťahujú k dôchodkovému systému. Taktiež existuje množstvo kalkulačiek s rôznou funkcionalitou. Problém spočíva v tom, že proces získavania informácií je zdĺhavý a náročnejší v porovnaní s očakávaniami. Človek pri získavaní informácií musí navštíviť viacero webových stránok alebo stiahnuť viacero aplikácií, ktoré sa zameriavajú na jednu konkrétnu vec. Na základe týchto poznatkov sa autor práce rozhodol vytvoriť aplikáciu, ktorá umožní jednoduchšie získavanie informácií o dôchodkovom systéme v Českej republike. Vzhľadom ku zväčšujúcemu sa počtu mobilných zariadení je vhodné aplikáciu zamerať práve na mobilnú platformu.

Práca sa postupne zaoberá analýzou, návrhom, implementáciou a testovaním mobilnej aplikácie spolu so serverovou častou systému.

Kapitola 1

Cieľ práce

Cieľom práce je vyvinúť mobilnú aplikáciu pre operačný systém iOS, ktorá uľahčí užívateľom zisťovanie informácií o dôchodkovom systéme v Českej republike. Súčasťou práce je implementácia vlastnej serverovej časti, ktorá umožňuje komunikáciu a poskytovanie dát mobilnej aplikácii.
Detailný cieľ práce sa skladá z nasledujúcich bodov:

- analýza existujúcich mobilných aplikácii, ktoré sa zaoberajú dôchodkovým systémom
- analýza zdrojov otvorených dát dôchodkového systému v Českej republike
- výber relevantných dátových zdrojov a návrh vhodnej funkcionality mobilnej aplikácie
- spracovanie návrhu užívateľského rozhrania vo forme drôteného modelu
- prevedenie užívateľského prieskumu so zameraním na drôtený model a funkcionalitu aplikácie
- vytvorenie požiadaviek na serverovú časť, ktorá bude vyťažovať dáta z otvorených zdrojov dát a poskytovať ich cez štandardizované aplikačné rozhranie mobilnej aplikácie
- prevedenie analýzy technických riešení serverovej časti
- implementácia serverovej časti a mobilnej aplikácie pre operačný systém iOS
- testovanie serverovej časti a mobilnej aplikácie

Kapitola 2

Analýza a návrh

Nasledujúca kapitola je zameraná na analýzu a návrh výsledného systému. Analýza a návrh sú rozdelené na dve fázy: mobilnú aplikáciu a serverovú časť. Fáza serverovej časti nadväzuje na výsledky fázy mobilnej aplikácie. Jednotlivé časti tejto kapitoly sú v práci na seba naväzované podľa toho, kedy boli realizované. Z tohto dôvodu je v kapitole uvedená analýza serverovej časti až po návrhu mobilnej aplikácie.
Analýza sa zameriava na analýzu existujúcich riešení, dátoveho zdroja, požiadaviek, technických riešení, databázových technológií a operačného systému iOS. Cieľom analýzy je analyzovať a zhodnotiť všetky predchádzajúce body. Návrh je zameraný na architektúru systému, funkcionalitu a užívateľské rozhranie mobilnej aplikácie a návrh funkcionality serverovej časti. Na základe analýzy sú definované technológie a komponenty, nad ktorými bude výsledný systém realizovaný.
Pojem *systém* je v kontexte tohto projektu chápaný ako celkový výsledný produkt, ktorý sa skladá z mobilnej aplikácie a serverovej časti.

2.1 Mobilná aplikácia

Táto časť opisuje fázu analýzy a návrhu mobilnej aplikácie. V nasledujúcej časti sú postupne analyzované exitujúce riešenia a dostupné dátové zdroje. Na základe analýzy a užívateľského dotazníku je prevedený návrh funkcionality a špecifikácia požiadaviek. Táto časť návrhu je doplnená o vhodné diagramy. Záver časti sa venuje opisu vytvoreného drôteného modelu mobilnej aplikácie.

2.1.1 Existujúce riešenia

V tejto časti práce sú popísané existujúce mobilné a webové aplikácie. Analyzované aplikácie možno rozdeliť do troch kategórií: aplikácie, ktoré sa priamo zaoberajú problematikou dôchodkov, čiastočne sa zaoberajú dôchodkovou prob-

lematikou a aplikácie, ktoré sa nezaoberajú dôchodkovou problematikou, ale sú určené prevažne ľuďom v dôchodkovom veku. Analýza je zameraná hlavne na funkcionalitu, užívateľské rozhranie (UI) a užívateľskú skúsenosť (UX). Zahŕňa mobilné aplikácie s operačným systémom iOS a Android. Analýza zahŕňa domáce a zahraničné aplikácie.

2.1.1.1 Finanční kalkulačky (iOS)[1]

Aplikácia ponúka po spustení užívateľovi štyri základné druhy kalkulačiek: výpočet čistej mzdy, nemocenskej dávky, kalkulačku sporenia a hypotéky. Pri vstupe do aplikácie nie je potrebná žiadna autorizácia ani autentikácia. V prípade čistej mzdy a nemocenskej dávky výpočet prebieha po stlačení príslušného tlačidla. Výpočet sporenia a hypotéky je interaktívny a zobrazuje sa hneď po zmene hodnôt na posuvníkoch. Výpočet hypotéky je obohatený o tabuľku a grafické zobrazenie zostatku hypotéky v prieberu nasledujúcich rokov. Výhoda aplikácie spočíva v plnom fungovaní aj bez internetového pripojenia a jednoduchom užívateľskom rozhraní. Užívateľská skúsenosť je negatívne ovplyvnená reklamami, ktoré sa vyskytujú v aplikácii na každej obrazovke. Aplikácia neponúka možnosť výpočtu dôchodku.

2.1.1.2 Finanční kalkulačky (Android)[2]

Vezria pre mobilné telefóny a tablety s operačným systémom android ponúka rozšírenú funkcionalitu oproti iOS verzii (2.1.1.1). Aplikácia ponúka navyše výpočet výšky dôchodku a výšku daňových odvodov osôb samostatne zárobkovo činných. Výpočet výšky dôchodku počíta s päťročnými obdobiami, do ktorých užívateľ zadá priemernú výšku hrubej mzdy, počet odpracovaných rokov a počet mesiacov predčasného alebo neskorého odchodu do dôchodku. Algoritmus výpočtu neberie do úvahy všetky vstupné premenné, ktoré sa odzrkadľujú na výške dôchodku. Výpočet nie je úplne exaktný, ale iba približný. Aplikácia ponúka navyše štatistické dáta vývoja čistej mzdy na území Českej republiky. Tie sú prezentované tabuľkou a grafickou formou. Okrem iného umožňuje poslať tvorcom osobný názor na aplikáciu a vypnúť reklamy za poplatok.
Výhodou aplikácie Finanční kalkulačky je rozsiahlejšia ponuka funkcionalít, ktorá zahŕňa výpočet výšky dôchodku a štatistické dáta. Výpočet dôchodku predstavuje iba približnú výšku, nakoľko algoritmus neberie do úvahy všetky vstupné premenné a počíta s priemernými hodnotami za určité obdobie. Štatistické ukazovatele obsahujú iba jednu dátovú sadu.

2.1.1.3 68 - The Pension Calculator [3]

Mobilná aplikácia, ktorá umožňuje výpočet výšky dôchodku pre obyvateľov Írska. Po spustení je zobrazená úvodná obrazovka, ktorá vyžaduje súhlas s pravidlami a podmienkami používania. Prístup do aplikácie je plne anonymný, bez nutnosti prihlásenia alebo registrácie. Užívateľovi je umožnené dodatočne

vyplniť kontaktné údaje priamo v aplikácii. Po udelení súhlasu je umožnený vstup do hlavnej časti aplikácie, ktorá pozostáva z výpočtu výšky dôchodku. Po vyplnení potrebných informácií je zobrazená odhadovaná výška dôchodku spolu s jednoduchým grafom. Aplikácia vyžaduje internetové pripojenie, bez ktorého hlavná funkcia nefunguje. Užívateľské rozhranie je jednoduché a ľahko pochopiteľné. Hlavná funkcionalita je dostupná ihneď po vstupe do aplikácie. Okrem výpočtu výšky dôchodku ponúka možnosť kontaktovať emailom alebo telefonicky autorskú spoločnosť. Aplikácia je špecifická pre Írsky dôchodkový systém a neumožňuje výpočet dôchodku podľa dôchodkového systému v Českej republike.
Aplikácia 68 - The Pension Calculator poskytuje jednoduché a intuitívne užívateľské rozhranie spolu s výhodou anonymného prihlásenia. Nevýhodou je úzka škála užívateľských funkcií a nepoužiteľnosť aplikácie bez internetového pripojenia.

2.1.1.4 Financie.online [4]

Mobilná aplikácia predstavuje digitálneho finančného asistenta na Slovensku. Do aplikácie je umožnený anonymný prístup s obmedzenou funkcionalitou. Registrácia je možná pomocou účtu na sociálnej sieti Facebook alebo cez autorizačnú službu Google. Financie.online ponúka obrovské množstvo dát a funkcionalít. Obsahuje množstvo kalkulačiek, finančnú poradňu s online expertom, články a slovníky z odboru ekonómie, možnosť porovnania hypoték, úverov, sporiacich účtov a iné produkty. Užívateľ má možnosť pridať si svoje finančné ciele, ich hodnotu a dátum dosiahnutia. Aplikácia následne uloží cieľ a zobrazí sumu peňazí, ktorú by mal užívateľ denne odložiť, aby cieľ dosiahol. Po pridaní vkladu k danému cieľu sa opraví zostávajúca suma a priemerná denná suma, ktorú by mal užívateľ odložiť.
Kalkulačka dôchodkového sporenia nezobrazuje reálnu výšku dôchodku, ale počíta výšku mesačného vkladu. Vklad je potrebné odložiť každý mesiac, aby užívateľ nasporil sumu, ktorá je dostatočná na vyplácanie súkromného dôchodku vo výške, ktorú si vybral. Kalkulačka interaktívne zobrazuje dáta spolu s grafom po zmene posuvníkov.
Výhodou Financie.online je široké spektrum ponúkaných dát a funkcionalít. Dôchodková kalkulačka ponúka odlišný model výpočtu oproti iným aplikáciám. Užívateľské rozhranie a grafické prevedenie aplikácie je na profesionálnej úrovni. Užívateľskú skúsenosť zhoršuje fakt, že z aplikácie sa nedá odhlásiť a zmazať, alebo zmeniť osobné údaje, ktoré boli poskytnuté pri registrácii. Mobilná aplikácia Financie.online bez internetového pripojenia funguje, ale ponúka veľmi obmedzenú funkcionalitu.

2.1.1.5 Výpočet.cz[5]

Webová aplikácia ktorá ponúka množstvo kalkulačiek rozdielnych druhov. Finančné kalkulačky, kalkulačky pre zdravý životný štýl ale aj kalkulačku dôchodkového veku a výšky dôchodku. Webová aplikácia obsahuje odkazy aj na mobilné aplikácie (2.1.1.1 a 2.1.1.1), ktoré sa nachádzajú v obchodoch App Store a Google Play. Kalkulačky ponúkané na webe pracujú nad rovnakými algoritmami ako v mobiných verziách a vyžadujú rovnaké vstupné dáta. Kalkulačka dôchodkového veku vyžaduje rok narodenia, pohlavie a počet detí. V hypotekárnej kalkulačne chýba oproti mobilnej verzii tabuľkové a grafové zobrazenie výšky zostatku hypotéky. Výhodou webovej aplikácie výpočet.cz je zoskupenie všetkých druhov kalkulačiek na jednej stránke. Kalkulačky pracujú nad rovnakými algoritmami ako v mobilných aplikáciách. Užívateľ webovej aplikácie by očakával rozšírenia v podobe pridania štatistických alebo grafových dát, alebo presnejšie a podrobnejšie výpočty oproti tým, ktoré sa nachádzajú v mobilných verziách. Tieto funkcionality však webovou aplikáciou nie sú poskytnuté.

2.1.2 Dátové zdroje

V nasledujúcej sekcii sú analyzované dátové zdroje ponúkajúce otvorené dáta spojené s dôchodkovým systémom v Českej republike. Analýza je vyhotovená len na dátové zdroje, ktoré poskytujú dáta v strojovo čitateľnej podobe.

2.1.2.1 Česká správa sociálneho zabezpečenia

Česká správa sociálneho zabezpečenia (ďalej len „ČSSZ“) na svojom portáli publikuje a vizualizuje dáta, ktoré vlastní a spravuje[6]. Otvorené dáta sú publikované ako dátove sady. Aktuálny katalóg dátových sád je dostupný na `https://data.cssz.cz/web/otevrena-data/katalog-otevrenych-dat`. Dátové sady sú voľne využívateľné. ČSSZ si však vyhradzuje právo z prevádzkových, odborných či iných dôvodov dočasne obmedziť prístup k zverejňovaným otvoreným dátam ČSSZ, a to aj bez predchádzajúceho upozornenia[7]. Dátové sady sú dostupné vo formátoch CSV a RDF/TriG.

2.1.2.2 Portál verejnej správy

Portál verejnej správy pod záštitou ministerstva vnútra Českej republiky poskytuje informácie v súlade so zákonom o slobodnom prístupe k informáciám[8]. Portál ponúka informácie a dáta z rôznych oblastí výkonu verejnej správy, medzi ktorými sú aj informácie priamo alebo vedľajšie spojené s dôchodkom. Informácie sú prezentované formou často kladených otázok na špecifický problém, situáciu alebo paragraf zákona, ktorý sa spája s dôchodkami rôznych druhov. Portál priamo odkazuje na aplikáciu Financie.online (2.1.1.4),

s ktorou zdieľa množstvo rovnakých informácií. Množstvo prezentovaných informácií presahuje rozsah zobrazený na webe ČSSZ. K prezentovaným informáciám však nie sú sprístupnené žiadne štatistické dáta ani dátové sady. Informácie, ktoré sú prezentované na webe nie sú pristupné cez API ani vo forme súborov. Portál verejnej správy je vhodný zdroj informácií, na ktorý je možné sa odkazovať v implementovanom riešení.

2.1.2.3 Český štatistický úrad

Český štatistický úrad (ďalej len „ČSÚ“) poskytuje dáta, analýzy, grafy, kartogramy a informácie z rôznych štatistických okruhov. Štatistiky spojené s dôchodkom sú poskytované v rozličných formátov nezávisle na sebe. Najčastejšie sú dáta prezentované textovou formou na webe alebo súbormi vo formátoch: XLS, GIF, DOC, PDF, JPG alebo CSV. Dáta vo formátoch XLS, GIF, DOC, PDF, JPG sú už väčšinou spracované do formy tabuliek alebo grafov. Z tohto dôvodu sa tieto informácie považujú za vhodný odkazový materiál, sú ale nevhodné pre ďalšie strojové spracovanie v mobilnej aplikácii alebo na serverovej časti. Pre jednoduché spracovanie dát na serveri sú pri výbere uprednostnené dátové sady, ktoré su poskytované vo formáte CSV.

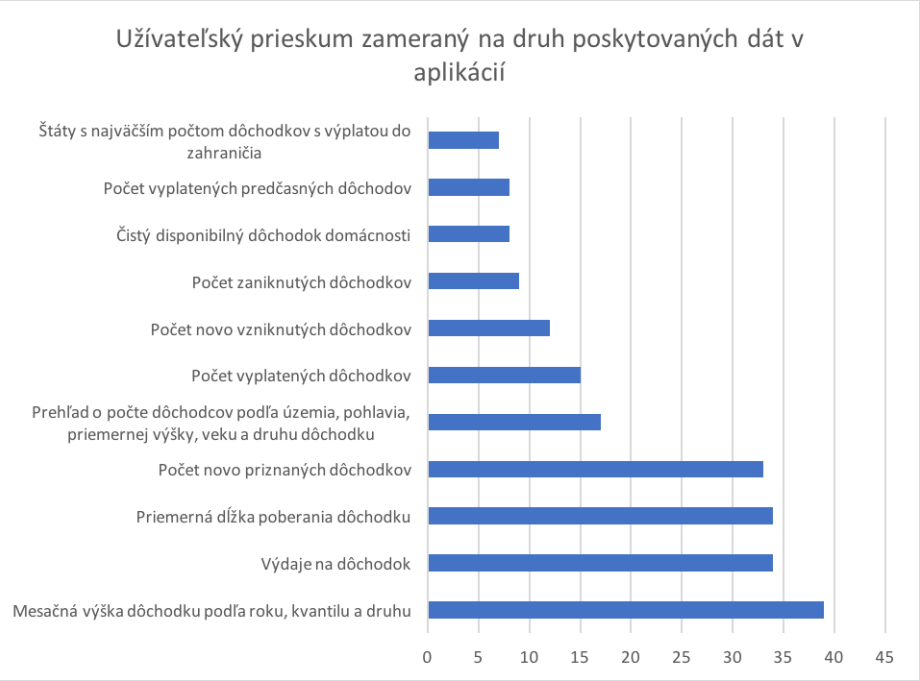
2.1.2.4 Výber relevantných zdrojov dát

Pred konečným výberom použitých dátových sád a informácií v aplikácii je zvolený širší okruh možností, ktorý je validovaný pomocou užívateľského prieskumu. Spomedzi dátových sád sú zvolené otvorené dáta ČSSZ, nakoľko ako jediné sú jednoducho strojovo čitateľné. Dáta poskytované ČSÚ nemajú jasne definovaný formát a preto nie sú jednoducho strojovo čitateľné. Z otvorených dát ČSÚ sú zvolené štatistické výstupy, ktoré môžu byť užívateľovi sprostretkované formou odkazov. Príklad zvolenej štatistiky z ČSÚ je Čistý disponibilný dôchodok domácností. Portál verejnej správy neposkytuje žiadne otvorené dáta ani API rozhranie na čerpanie informácií a preto je vhodný ako odkazový materiál. Užívateľský prieskum je zameraný na povedomie užívateľov o aplikáciách, ktoré približujú dôchodkové systémy, požiadavky na ich funkcionalitu a obsah dát ktoré zobrazujú. V dotazníku sú obsiahnuté nasledujúce dátové sady a štatistické výstupy:

- Mesačná výška dôchodku podľa roku, kvantilu a druhu
- Počet novo priznaných dôchodkov
- Priemerná dĺžka poberania dôchodku
- Výdaje na dôchodok
- Prehľad o počte dôchodcov podľa územia, pohlavia, priemernej výšky, veku a druhu dôchodkov

- Počet vyplatených dôchodkov
- Počet zaniknutých dôchodkov
- Počet novo vzniknutých dôchodkov
- Počet vyplatených predčasných dôchodkov
- Štáty s najväčším počtom dôchodkov s výplatou do zahraničia
- Čistý disponibilný dôchodok domácnosti

Okrem okruhu ponúkaných možností má užívateľ právo zadať iný typ infomácie o ktorú sa zaujíma a nie je v ponuke obsiahnutá. Do prieskumu bolo zapojených viac než 50 ľudí. Dotazník bol podávaný rôznym vekovým kategóriám, aby bolo zaručené, že aplikácia bude spĺňať požiadavky rôznych vekových kategórii. Zo spracovaných odpovedí sú vytvorené grafy, ktoré slúžia na jednoduchšiu ilustráciu výsledkov. Konkrétne odpovede na dotazník vo formáte PDF spolu so všetkými graficky spracovanými odpoveďami sú dostupné na pribalenom CD v zložke `/attachement/survey`.



Obr. 2.1: Graf užívateľských preferencií pri výbere dátových sád

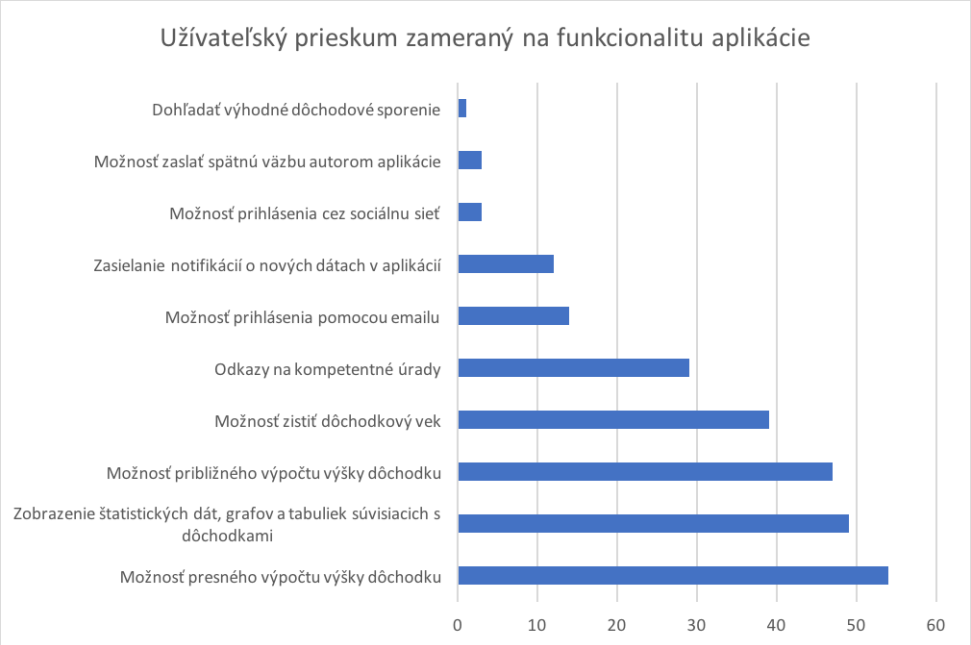
Na základe grafu užívateľských preferencii (2.1) je viditeľné, že najväčší záujem medzi opýtanými má dátová sada zobrazujúca výšku dôchodku poďla roku (39 z 58 opýtaných), najmenej zaujímavá z pohľadu opýtaných je dátová

sada „Štáty s najväčším počtom dôchodkov s výplatou do zahraničia“. Pre prvotný prototyp aplikácie sú zvolené dátové sady, ktoré v dotazníku dosiahli minimálne 25% úspešnosť:

1. Mesačná výška dôchodku podľa roku, kvantilu a druhu
2. Výdaje na dôchodok
3. Priemerná dĺžka poberania dôchodku
4. Počet novo priznaných dôchodkov
5. Prehľad o počte dôchodcov podľa územia, pohlavia, priemernej výšky, veku a druhu dôchodkov
6. Počet vyplatených dôchodkov

2.1.3 Analýza a špecifikácia požiadaviek

Pred samotnou špecifikáciou požiadaviek bol definovaný a navrhnutý širší okruh funkcionalít, ktoré by výsledná aplikácia mohla poskytovať. Okruh funkcionalít prešiel rovnako ako dátové sady v predchádzajúcej sekcii užívateľským prieskumom (2.1.2.4). Dotazník obsahoval iba okruh funkčných požiadaviek na mobilnú aplikáciu. Nefunkčné požiadavky vychádzajú zo zadania práce alebo sú definované autorom práce.



Obr. 2.2: Graf užívateľských preferencií zameraných na funkcionalitu mobilnej aplikácie

Na základe užívateľského prieskumu (2.2) a predošlej analýzy dát a existujúcich riešení je vytvorená špecifikácia požiadaviek.

2.1.3.1 Funkčné požiadavky

Funkčné požiadavky definujú, čo môže užívateľ očakávať od mobilnej aplikácie z hľadiska jej funkcionality a správania.

FP1: Registrácia a prihlásenie
Registrácia užívateľa je povinná. Užívateľ má možnosť sa registrovať zadaním povinných vstupných informácií alebo účtom Google. Užívateľ musí pri registrácii pomocou účtu Google schváliť poskytnutie vyžiadaných informácií. Prihlásenie do existujúceho účtu je umožnené pomocou emailovej adresy a hesla alebo pomocou registrovaného Google účtu.

FP2: Automatické prihlásenie
V prípade, ak prihásený užívateľ vypne aplikáciu, aplikácia presmeruje užívateľa do hlavnej ponuky po opätovnom zapnutí automaticky bez nutnosti prihlásenia.

FP3: Grafické zobrazenie dátových a štatistický sád
Mobilná aplikácia spracúva a graficky zobrazuje dátové a štatistické dáta poskytované zo serverovej časti.

FP4: Presný výpočet dôchodku
Mobilná aplikácia umožňuje užívateľovi vypočítať presnú výšku dôchodku podľa aktuálne platnej legislatívy.

FP5: Zjednodušený, približný výpočet dôchodku
Mobilná aplikácia umožňuje užívateľovi vypočítať približnú, zjednodušenú výšku dôchodku.

FP6: Výpočet dôchodkového veku
Mobilná aplikácia umožňuje užívateľovi vypočítať dôchodkový vek.

FP7: Zobrazenie odkazov na webové stránky úradov, ktoré sa zaoberajú dôchodkovým systémom
Mobilná aplikácia odkazuje užívateľa na webové stránky kompetentných úradov, ktoré sú spojené s dôchodkovým systémom.

FP8: Aktualizácia dátových sád
Mobilná aplikácia dokáže reagovať na zmenu dátových sád a umožní bez nutnosti obnovovania zobrazovať nové dátové sady.

FP9: Lokálne ukladanie dát
Mobilná aplikácia ukladá potrebné dáta do lokálnej pamäte zariadenia.

FP10: Aktualizácia a nastavenie účtu
Mobilná aplikácia umožňuje užívateľovi meniť nastavenia účtu, základné informácie o účte a prihlasovacie údaje týkajúce sa daného účtu.

2.1.3.2 Nefunkčné požiadavky

Nefunkčné požiadavky definujú obmedzujúce podmienky na nefunkčnú časť systému, medzi ktoré patrí napríklad výkon, kvalita, platforma alebo dostupnosť výslednej aplikácie.

NP1: Platforma a operačný systém
Mobilná aplikácia je dostupná pre zariadenia s operačným systémom iOS 10.0 a novšie verzie.

NP2: Podpora zariadení
Mobilná aplikácia je dostupná pre telefóny iPhone 5s a novšie verzie. Verzia pre tablety je dostupná pre zariadenia iPad Air a novšie.

NP3: Nutnosť internetového pripojenia
Mobilná aplikácia pre svoje fungovanie vyžaduje pripojenie na internet. Bez internetového pripojenia je funkcionalita mobilnej aplikácie nedostupná.

NP4: Rest API
Mobilná aplikácia dokáže komunikovať so serverovou časťou pomocou Rest API.

NP5: Súlad s návrhovými pokynmi operačného systému
Mobilná aplikácia je vytvorená v súlade s návrhovými pokynmi tvorby aplikácií cieľovej platformy a operačného systému.

NP6: Nedostupnosť dát
Mobilná aplikácia v prípade zmazania, alebo nedostupnosti dátových sád na webe ČSSD pracuje s posledne dostupnými dátami.

NP7: Závislosť na serverovej časti
Mobilná aplikácia je závislá na fungovaní serverovej časti. Pri nedostupnosti serverovej časti aplikácia ponúka obmedzenú funkcionalitu.

NP8: Zabezpečenie
Mobilná aplikácia komunikuje so serverovou časťou pomocou šifrovanej komu-

nikácie.

2.1.4 Prípady použitia

Slovami tvorcu konceptu prípadov použitia Ivara Jakobsona, prípady použitia sú všetky spôsoby používania systému za účelom dosiahnutia špecifického cieľa špecifickým užívateľom. Všetky prípady použitia predstavujú užitočný spôsob použitia systému a ilustrujú hodnotu, ktorú systém poskytuje[9].
Prípad použitia popisuje užívateľské zaobchádzanie so systémom a následnú odozvu aplikácie na podnet. Prípad použitia sa skladá so sekvencie akcií, ktoré začínajú cieľom užívateľa a končia ked je cieľ naplnený. Prípady použitia typicky reprezentujú detailnejšiu formu funkčných požiadaviek aplikácie. Niekoľko prípadov použitia je často mapovaných na rovnakú funkčnú požiadavku[10]. Každý prípad použitia je definovaný scenárom, ktorý má jeden alebo niekoľko spôsobov (ciest) ako je možné jeho cieľ naplniť. Scenáre jednotlivých prípadov použitia sa definujú textovou formou alebo diagramom prípadov použitia.
V nasledujúcej časti práce sú textovo opísané prípady použitia mobilnej aplikácie. Zaujímavejšie prípady použitia sú rozpísané do detailov s prislúchajúcimi detailnými scenármi.

PP1 Registrácia

1. Prípad použitia začína, keď sa užívateľ chce zaregistrovať v aplikácii.
2. Aplikácia zobrazí úvodnú obrazovku s možnosťami prihlásenia, registrácie pomocou emailu alebo pokračovaním pomocou služby Google.
   a) Užívateľ vyberie možnosť registrácie pomocou emailu.
      i. Aplikácia zobrazí registračný formulár s informáciami, ktoré je nutné vyplniť za účelom registrácie.
      ii. Užívateľ vyplní registračný formulár a pošle požiadavku na registráciu.
      iii. Mobilná aplikácia spracuje požiadavku na registráciu a pošle dáta serverovej časti. V prípade neúspešnej registrácie je zobrazené chybové hlásenie a užívateľovi je umožnená zmena informácií. Prípad použitia pokračuje v bode i. V prípade úspešnej registrácie aplikácia zobrazí potvrdzovaciu hlášku a vráti užívateľa na úvodnú obrazovku.
   b) Užívateľ vyberie možnosť pokračovania pomocou služby Google.
      i. Aplikácia otvorí rozhranie autorizačnej služby Google.
      ii. Užívateľ vyplní požadované informácie autorizačnej služby a odošle požiadavku na spracovanie. V prípade neúspešnej autorizácie je zobrazené chybové hlásenie a prípad použitia po-

kračuje v bode i. V prípade úspešnej autorizácie je užívateľ presmerovaný do hlavnej ponuky aplikácie.

3. Prípad použitia končí.

PP2 Prihlásenie

1. Prípad použitia začína, keď sa užívateľ chce prihlásiť do aplikácie.
2. Aplikácia zobrazí úvodnú obrazovku s možnosťami prihlásenia, registrácie pomocou emailu alebo pokračovaním pomocou služby Google.
   a) Užívateľ vyberie možnosť prihlásenia pomocou emailu.
      i. Aplikácia zobrazí prihlasovací formulár pre zadanie užívateľského hesla a emailu.
      ii. Užívateľ vyplní prihlasovací formulár a pošle požiadavku na prihlásenie.
      iii. V prípade neúspešného prihlásenia je zobrazené chybové hlásenie a užívateľovi je umožnená zmena informácií. Prípad použitia pokračuje v bode i. V prípade úspešnej registrácie je užívateľ presmerovaný do hlavnej ponuky aplikácie.
      iv. Prípad použitia končí.
   b) Užívateľ vyberie možnosť pokračovania pomocou služby Google.
      i. Užívateľ bol autorizovaný službou Google v minulosti.
         A. Aplikácia presmeruje užívateľa do hlavnej ponuky. Prípad použitia končí.
      ii. Užívateľ sa autorizuje službou Google prvýkrát.
         A. Aplikácia otvorí rozhranie autorizačnej služby Google.
         B. Užívateľ vyplní požadované informácie autorizačnej služby a odošle požiadavku na spracovanie. V prípade neúspešnej autorizácie je zobrazené chybové hlásenie a prípad použitia pokračuje v bode 2. V prípade úspešnej autorizácie je užívateľ presmerovaný do hlavnej ponuky aplikácie a prípad použitia končí.

PP3 Automatické prihlásenie

Automatické prihlásenie do aplikácie po opätovnom zapnutí aplikácie je umožnené užívateľom, ktorí sa neodhlásili z aplikácie pred posledným vypnutím aplikácie.

1. Prípad použitia začína po zapnutí aplikácie.
   a) Aplikácia zistí, že užívateľ bol pred posledným vypnutím aplikácie prihlásený.

i. Aplikácia na pozadí pošle požiadavku na obnovu autorizačných práv a následne prebehne pokus o prihlásenie. V prípade úspešnej autorizácie je užívateľ presmerovaný do hlavnej ponuky a prípad použitia končí. Pokiaľ sa nepodarí úspešne obnoviť autorizačné práva, prípad použitia končí s chybou.

b) Aplikácia zistí, že užívateľ nebol pred posledným vypnutím aplikácie prihlásený. Prípad použitia končí s chybou.

PP4 Výpočet približnej výšky dôchodku

1. Prípad použitia začína, keď chce užívateľ vypočítať presnú výšku dôchodku.
2. Aplikácia zobrazí formulár s nutnými informáciami na realizovanie výpočtu.
3. Užívateľ vyplní formulár a odošle požiadavku.
4. Aplikácia spracuje vyplnené dáta a odošle požiadavku na serverovú časť. V prípade úspechu je zobrazená výška dôchodku. V opačnom prípade je zobrazená chybová hláška a prípad použitia pokračuje v bode 2.
5. Prípad použitia končí.

PP5 Výpočet presnej výšky dôchodku

Prípad použitia sa zhoduje s prípadom použitia 4 (2.1.4). Požadovaný formulár sa líši v druhu a množstve dát, ktoré je potrebné od užívateľa získať.

PP6 Grafické zobrazenie štatistických dát

1. Prípad použitia začína, keď chce užívateľ zobraziť štatistické dáta.
2. Aplikácia zobrazí zoznam dostupných štatistických dát.
3. Užívateľ si zvolí jednu položku zo zoznamu dostupných dátových sád.
4. Aplikácia odošle požiadavku serverovej časti na získanie štatistických dát s automatickými hodnotami filtrovacích parametrov.
5. Aplikácia spracuje odpoveď zo serverovej časti a zobrazí graf štatistických dát s automatickými hodnotami filtrovacích parametrov, alebo prázdny graf pokiaľ neexistuje záznam spĺňajúci zadané parametre.
6. Prípad použitia končí.

PP7 Výpočet dôchodkového veku

1. Prípad použitia začína, keď chce užívateľ vypočítať dôchodkový vek.
2. Aplikácia zobrazí formulár s povinnými informáciami na realizovanie výpočtu.
3. Užívateľ vyplní formulár.
4. Aplikácia počas vypĺňania formuláru reaguje na zmeny hodnôt a zobrazuje vypočítanú hodnotu dôchodkového veku.
5. Prípad použitia končí.

PP8 Zmena filtrovacích parametrov grafického zobrazovania dát

Aplikácia umožňuje užívateľovi meniť hodnoty filtrovacích parametrov. Po potvrdení hodnôt, aplikácia prekreslí graf na základe nových dát.

PP9 Odhlásenie

Aplikácia poskytuje užívateľovi možnosť odhlásenia sa z mobilnej aplikácie. Užívateľ je po odhlásení presmerovaný na úvodnú obrazovku.

PP10: Aktualizácia dátových sád

Aplikácia v prípade zistenia dostupnosti nových dátových sád automaticky na pozadí aktualizuje lokálne úložisko.

PP11: Aktualizácia dátových sád

Aplikácia umožňuje ukladanie dát do lokálneho úložiska a cache pamäti.

PP12: Presmerovanie na užitočné zdroje informácií

Aplikácia umožňuje užívateľovi zobraziť zoznam odkazov na webové stránky orgánov Českej republiky, ktoré sa zaoberajú dôchodkovým systémom v Českej republike. Po kliknutí na odkaz aplikácia automaticky otvorí webový prehliadač a presmeruje užívateľa na konkrétnu stránku.

PP13: Zmena hesla

Zmena hesla je umožnená užívateľom, ktorí sa registrovali do aplikácie pomocou emailu. Užívatelia, ktorí sa registrovali inou formou, nemajú možnosť zmeny hesla a ani im v aplikácii táto možnosť nie je zobrazená. Nasledujúci scenár opisuje prípad použitia pre užívateľov, ktorí sa do aplikácie registrovali pomocou emailu.

1. Prípad použitia začína, keď chce užívateľ zmeniť heslo.

2. Aplikácia zobrazí formulár na zmenu hesla.

3. Užívateľ vyplní formulár a odošle požiadavku.

4. Aplikácia vykoná validáciu starého hesla a kontrolu požiadavky na nové heslo. Pokiaľ heslá spĺňajú všetky kritériá je požiadavka odoslaná na serverovú časť. V opačnom prípade je zobrazená validačná chybová hláška a prípad použitia pokračuje bodom 2.

5. Aplikácia na základe odpovede zo serveru v prípade úspechu zobrazí potvrdenie o vykonaní zmeny. V opačnom prípade je zobrazená chybová hláška.

6. Prípad použitia končí.

PP14: Zmena emailu

Zmena emailu je umožnená užívateľom, ktorí sa do aplikácie registrovali pomocou emailu. Užívateľom, ktorí sa registrovali inou formou táto možnosť nie je umožnená ani zobrazená v aplikácii.

1. Prípad použitia začína, keď chce užívateľ zmeniť email.

2. Aplikácia zobrazí formulár na zmenu emailu.

3. Užívateľ zadá nový email a potvrdí zmenu zadaním hesla.

4. Aplikácia vykoná validáciu hesla a odošle požiadavku na serverovú časť v prípade, že validácia prebehne úspešne. V prípade neúspešnej validácie hesla aplikácia zobrazí chybovú hlášku a prípad použitia pokračuje v bode 2.

5. Aplikácia odošle požiadavku na zmenu emailu. Na základe odpovede zo serveru v prípade úspechu zobrazí potvrdenie o vykonaní zmeny. V opačnom prípade je zobrazená chybová hláška.

6. Prípad použitia končí.

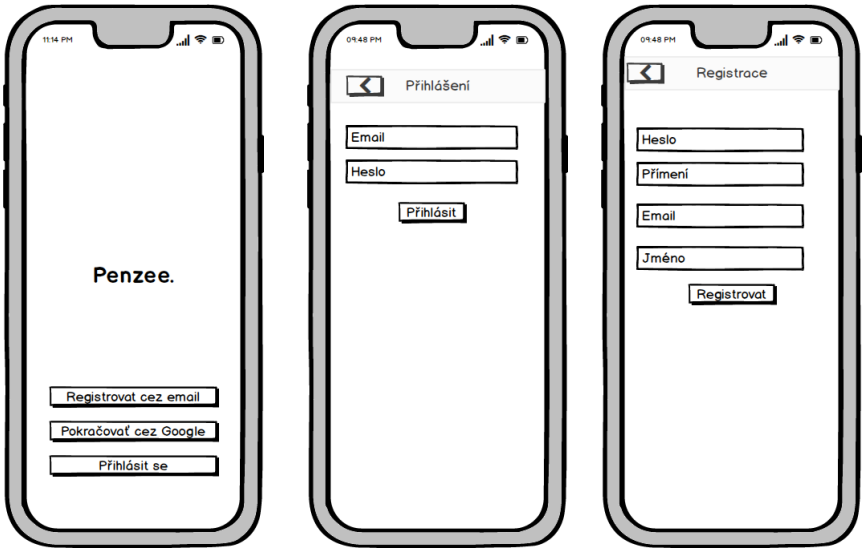
PP15: Zmena užívateľoveho mena

Prípad použitia sa zhoduje s prípadom použitia 14 (2.1.4). Namiesto emailu je užívateľom zadávané nové meno.

2.1.5 Návrh užívateľského rozhrania

V nasledujúcej časti je popísaný návrh užívateľského rozhrania mobilnej aplikácie. Užívateľské rozhranie mobilnej aplikácie musí byť jednoduché a ľahko pochopiteľné užívateľom. Pred samotnou implemetáciou je navrhnutý a vytvorený drôtený model (angl. „wireframe“) mobilnej aplikácie. Tento model slúži na pochopenie aplikácie užívateľmi a doladenie prípadných chýb ešte pred samotnou implementáciou, čím je ušetrený programátorský čas. Drôtený model aplikácie je dvojdimenzionálna ilustrácia užívateľského rozhrania, ktorá sa zameriava na rozloženie elementov na obrazovke, prioritizáciu obsahu stránky, dostupnú funkcionalitu a správanie. Drôtený model zachytáva prechody medzi jednotlivými obrazovkami aplikácie. Drôtený model neobsahuje žiadne štylistické prvky, farby alebo grafiku[11].
Drôtený model aplikácie je vytvorený v programe Balsamiq[1]. V nasledujúcej časti sú popísané jednotlivé obrazovky. Celý drôtený model spolu s jednotlivými prechodmi medzi obrazovkami a akciami je dostupný vo formáte PDF na pribalenom CD v zložke `/attachement/wireframes/v1`. Po vytvorení modelu je prevedené užívateľské testovanie a validácia modelu.

2.1.5.1 Úvodná obrazovka, prihlásenie a registrácia



Obr. 2.3: Drôtený model úvodnej obrazovky, prihlásenia a registrácie

Prvá z trojice obrazoviek je úvodná obrazovka mobilnej aplikácie. Táto obrazovka slúži ako rozcestník. Sú na nej tri tlačidlá určené na prihlásenie pomocou emailu, účtu google alebo na registráciu pomocou emailu.

[1] https://balsamiq.com/

Druhá z obrazoviek je obrazovka určená na prihlásenie pomocou emailu. Obsahuje textové polia pre email spolu s heslom a tlačidlom „Přihlásit“. V prípade úspešného prihlásenia je užívateľ presmerovaný do hlavnej ponuky. V prípade chyby je zobrazený formulár s chybovou hláškou.
Tretia obrazovka zachytáva registráciu pomocou emailu. Obsahuje textové polia pre meno, priezvisko, email, heslo a tlačidlo určené na registráciu. V prípade úspešnej registrácie sa zobrazí úvodná obrazovka. V opačnom prípade je zobrazené chybové hlásenie.

2.1.5.2 Obrazovky spojené s dôchodkom

Obr. 2.4: Drôtený model približnej a presnej výšky dôchodku a obrazovky doplňujúcich informácií

Prvá z trojice obrazoviek je obrazovka, na ktorej sa počíta približná výška dôchodku. Táto obrazovka je zobrazená po úspešnom prihlásení do aplikácie. Pod navigačnou lištou je segment na prepínanie medzi približným a presným dôchodkom. V navigačnej lište je tlačidlo odkazujúce na obrazovku s doplňujúcimi informáciami. Spodná lišta obrazovky slúži na prepínanie medzi jednotlivými záložkami hlavnej ponuky aplikácie. Hlavné telo obrazovky je tvorené textovými poliami, ktoré slúžia na zadávanie potrebných informácií pre výpočet dôchodku.
Druhá z obrazoviek je obrazovka určená na výpočet presnej výšky dôchodku. Od predošlej obrazovky sa líši iba druhom a množstvom dát, ktoré je potrebné zadať do výpočtu dôchodku.
Tretia obrazovka zobrazuje doplnkové informácie. Umožňuje vypočítať dôchod-

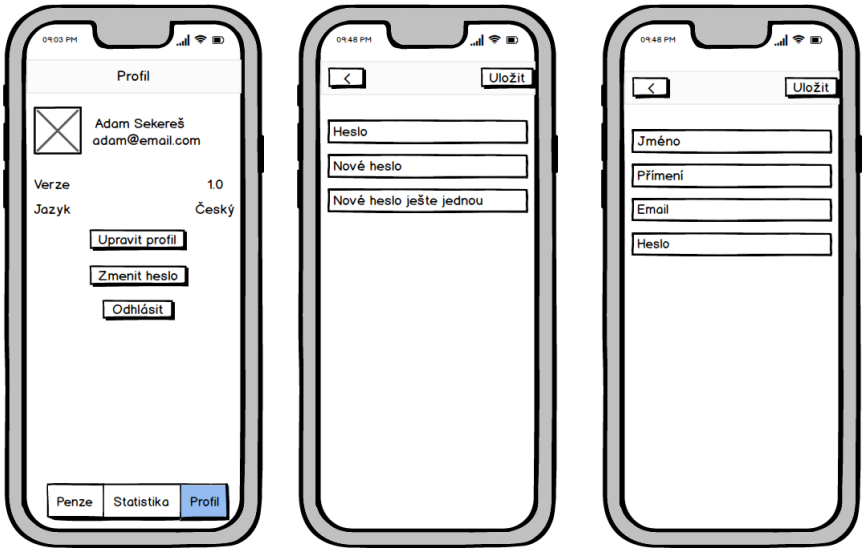
kový vek užívateľa a zobrazuje odkazy na inštitúcie spojené s dôchodkovým systémom.

2.1.5.3 Štatistické dáta

Obr. 2.5: Drôtený model zoznamu a grafického zobrazenia dátových sád

Prvá z dvojice obrazoviek je obrazovka, ktorá zobrazuje množinu dostupných dátových sád. Zobrazené názvy dátových sád sú usporiadané v tabuľke.
Druhá z obrazoviek vykresľuje grafické zobrazenie konkrétnej dátovej sady. Obsahuje množinu parametrov, podľa ktorých je dátová sada filtrovaná. Obrazovka je dynamicky vykresľovaná na základe počtu a druhu filtrovacích parametrov. Pod filtrovacími parametrami je graf, ktorý zobrazuje filtrované dáta z konkrétnej dátovej sady.

2.1.5.4 Profil



Obr. 2.6: Drôtený model profilu, úpravy hesla a úpravy profilových informácií

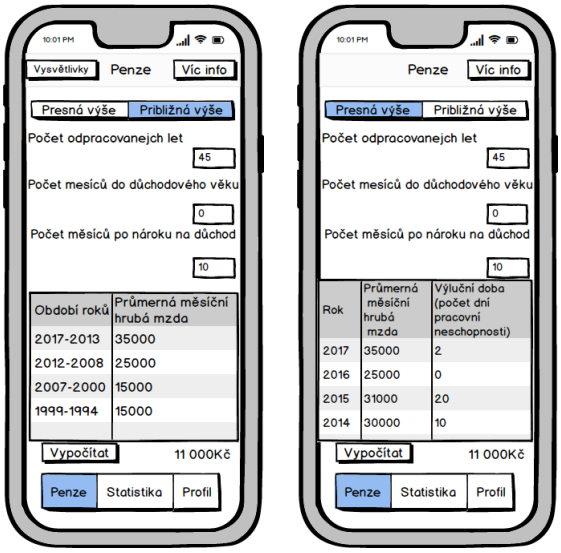
Prvá z trojice obrazoviek zobrazuje profil užívateľa. Táto obrazovka okrem profilových informácií obsahuje tlačidlá pre odhlásenie, zmenu hesla a profilových informácií. Tlačidlá pre zmenu hesla a profilových informácií sú zobrazené iba v prípade, že sa užívateľ do aplikácie prihlásil pomocou emailu a hesla. V prípade, že užívateľ využil možnosť prihlásenia pomocou služby Google, tieto dve tlačidlá nie sú zobrazené.
Druhá z obrazoviek je obrazovka určená na zmenu hesla. Je zobrazená iba v prípade, že sa užívateľ prihlásil do aplikácie pomocou emailu a hesla.
Tretia obrazovka je obrazovka určená na zmenu profilových informácií. Táto obrazovka je podobne ako druhá obrazovka zobrazovaná iba v prípade, že sa užívateľ prihlásil do aplikácie pomocou emailu a hesla.

2.1.6 Validácia drôteného modelu a funkcionality

Validácia drôteného modelu a funkcionality napomáha odhaliť možné nedostatky mobilnej aplikácie ešte pred jej samotným programovaním. Validácia sa skladá z iterácií, v ktorých je užívateľom predstavená funkcionalita aplikácie a jej odpovedajúci drôtený model. Od testovacích užívateľov je získaná spätná väzba, na základe ktorej môže viesť k prepracovaniu návrhu drôteného modelu alebo funkcionality. Do užívateľskej validácie drôteného modelu a funkcionality je zapojených päť ľudí, z ktorých každý patrí do inej vekovej kategórie špecifikovanej v dotazníku (2.1.2.4).

2.1.6.1 1. Iterácia

V prvej iterácii je užívateľom predstavená zvolená funkcionalita na základe dotazníku. Spolu s fukcionalitou je následne zobrazený drôtený model, ktorý je zobrazený v návrhu (2.1.5).
Validácia funkcionality mobilnej aplikácie prebehla bez problémov. Testeri sa zhodovali s výberom funkcionality mobilnej aplikácie, ktorá je zvolená na základe dotazníku.
Validácia drôteného modelu odhalila problémy užívateľov s pochopením formulárov na výpočet presnej a približnej výšky dôchodku. Testeri si v niektorých prípadoch neboli istí, aké informácie sú od nich požadované. Okrem iného poukázali na chýbajúce texty v záhlaví obrazoviek na zmenu hesla a profilových údajov. Žiadne iné problémy testeri pri testovaní nezaznamenali.



Obr. 2.7: Drôtený model približnej a presnej výšky dôchodku po zapracovaní pripomienok

Na základe spätnej väzby je obrazovka pre presný a približný výpočet výšky dôchodku upravená tak, aby obsahovala výstižnejšiu špecifikáciu jednotlivých položiek (2.1.6.1). Pridané sú aj chýbajúce textové popisy do záhlavia obrazoviek na zmenu hesla a užívateľských informácií. Verzia drôteného modelu po zapracovaní pripomienok z prvej iterácie je dostupná na priloženom CD v zložke `/attachement/wireframes/v2`.

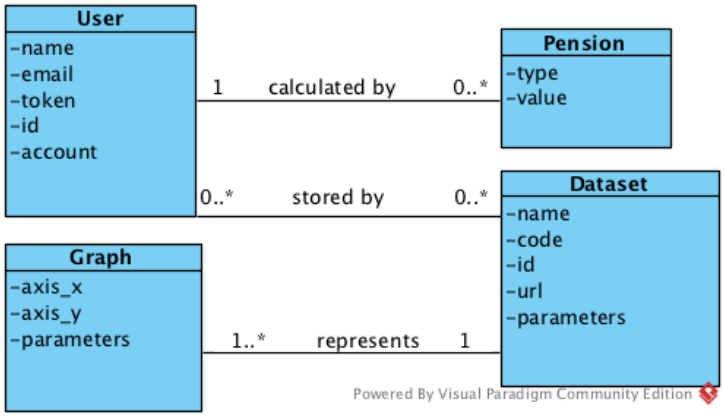
2.1.6.2 2. Iterácia

V druhej iterácii je testerom predstavený drôtený model, ktorý obsahuje zapracované pripomienky z prvej iterácie a funkcionalita aplikácie. Testeri ne-

poukázali na žiadny nedostatok drôteného modelu.

2.1.7 Doménový model

Konceptuálne doménové modelovanie, taktiež nazývané konceptuálne modelovanie alebo modelovanie domény je v knihe *The Object Primer: Agile Model-Driven Development with UML 2.0*[12] definované ako proces vytvárania entít reprezentujúcich predmety, koncepty a ich vzájomné vzťahy, ktoré súvisia s modelovaným problémom. Z iného pohľadu je konceptuálne modelovanie chápané ako spôsob zňazornenia detailného chápania problému. Doménový model pomáha pri návrhu systému, ktorý je udržovateľný, testovateľný a vhodný pre inkrementálny vývoj. Vedené čiastočne z objektovo orientovaných návrhových prístupov, modelovanie domény predpokladá riešenie ako súbor doménových objektov, ktoré spolupracujú na splnení scenárov na úrovni systému[13]. V nasledujúcej časti je zachytený doménový model mobilnej aplikácie a opísané jednotlivé entity, ktoré v modeli vystupujú.



Obr. 2.8: Doménový model mobilnej aplikácie

2.1.7.1 User

User je entita reprezentujúca užívateľa. Obsahuje meno, email, token, id a účet. Užívateľ môže vypočítať neobmedzený počet dôchodkov a disponuje ľubovoľným množstvom dátových sád.

2.1.7.2 Pension

Pension je entita reprezentujúca dôchodok. Obsahuje typ a výšku daného dôchodku. Každý dôchodok je vypočítaný práve jedným užívateľom.

2.1.7.3 Dataset

Dataset je entita reprezentujúca dátovú sadu. Dátová sada je špecifická svojím menom, url, parametrami sady a kódom dátovej sady. Patrí ľubovoľnému počtu užívateľov a je vizuálne prezentovateľná minimálne jedným grafom.

2.1.7.4 Graph

Entita reprezentujúca grafové zobrazenie. Obsahuje názvy jednotlivých osí grafu, parametre a ich prislúchajúce hodnoty. Každý graf patrí práve jednej dátovej sade, ktorú reprezentuje.

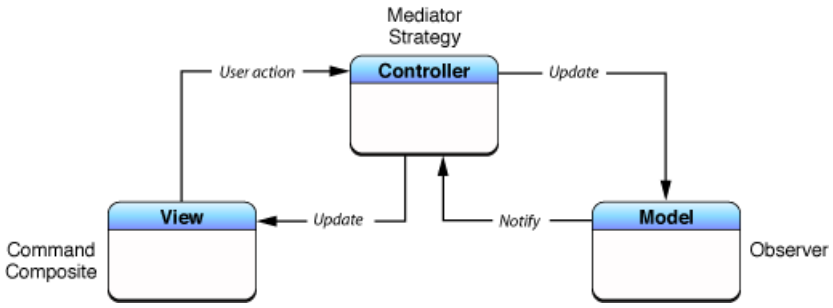
2.1.8 Návrhové vzory mobilnej aplikácie

V nasledujúcej časti sú analyzované návrhové vzory podľa ktorých môže byť implementovaná mobilná aplikácia. Analýza rozoberá základné komponenty, vzťahy medzi komponentami, kladné a záporné stránky. Vhodné použitie návrhového vzoru musí okrem iného zabezpečiť jednoduchú použiteľnosť, udržovateľnosť, škálovateľnosť, testovateľnosť a znovupoužitie jednotlivých komponentov z ktorých sa skladá. V závere časti je zvolený najvhodnejší návrhový vzor, ktorý bude realizovaný v implementačnej časti mobilnej aplikácie.

2.1.8.1 Model-View-Controller

Model-View-Controller (ďalej len „MVC“) je návrhový vzor, ktorý bol prvýkrát publikovaný v osemdesiatych rokov minulého storočia spoločnosťou Xerox[2] ako súčasť projektu Smalltalk. Spoločnosť Apple definuje návrhový vzor nasledovne. Je postavený z troch hlavných druhov objektov: model, controller a view objekt. Model objekt zapúzdruje dáta a základnú logiku manipulovania s nimi. Model objekt upozorňuje view o zmene dát a vykonáva zmeny na dátach na základe podnetu od controller objektu.
View objekt reprezentuje to čo vidí užívateľ v aplikácii. Posiela controller objektu informácie o užívateľskej interakcii a zobrazuje zmeny na základe podnetu od controller objektu.
Controller objekt funguje ako sprostredkovateľ medzi modelovými a view objektami. Posiela správy o aktualizáciách view a model objektom a definuje aplikačnú logiku[14].
Nevýhodou MVC je nízka znovupoužiteľnosť kódu a testovateľnosť masívnych controllerov, ktoré navyše obsahujú aj celú navigačnú logiku, referencie na view a model objetky.

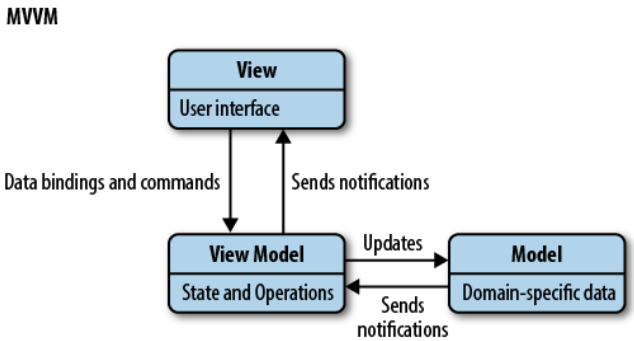
[2] `http://www.xerox.com`



Obr. 2.9: Cocoa verzia MVC návrhového vzoru[15]

2.1.8.2 Model-View-ViewModel

Model-View-ViewModel (ďalej len „MVVM“) bol predstavený prvýkrát v roku 2005 Johnom Grossmanom. Hlavným cieľom MVVM je oddelenie view vrstvy od aplikačnej logiky za účelom testovateľnosti uživateľského rozhrania. Raffaele Garofalo vo svojej knihe[16] definuje MVVM z troch hlavných komponentov: model, view, viewmodel. Model komponent plní rovnakú funkciu ako v MVC. View komponent je zodpovedný za vykresľovanie grafických komponentov a dát na obrazovku aplikácie. Jedná sa o obdobu controller komponentu z MVC bez aplikačnej logiky. Aplikačná logika je obsiahnutá vo viewmodel objekte. View objekt drží referenciu na viewmodel objekt, ktorý danú aplikačnú logiku poskytuje. View objekt dozerá na hodnoty, ktoré sú mu poskytované viewmodel objektom a v prípade zmeny vykoná adekvátnu akciu.
Výhodami MVVM sú znovupoužiteľnosť kódu, znížený počet väzieb medzi užívateľským rozhraním a ostatnými časťami kódu a jednoduchá testovateľnosť aplikačnej logiky[16]. Nevýhoda MVVC pramení z nutnosti vyššej náročnosti na množstvo písaného kódu, ktorý pri menších aplikáciách často prevyšuje výhody MVVM.



Obr. 2.10: Návrhový vzor MVVM[17]

2.1.8.3 Model-View-ViewModel-Coordinator

Model-View-ViewModel-Coordinator (ďalej len „MVVMC“) je mladý návrhový vzor, ktorý vychádza z MVVM a je doplnený o coordinator komponent. Coordinátor komponent má za úlohu navigáciu medzi obrazovkami v aplikácii. Inicializuje potrebné objekty pred zmenou obrazovky a realizuje samotný prechod a tým sa stáva view objekt v MVVMC oproti MVVM jednoduchším z pohľadu funkcionality. Výhodou MVVMC je oddelená navigačná logika aplikácie v samostatnom komponente, ktorá poskytuje schopnosť jednoducho meniť tok aplikácie. [18]

2.1.8.4 Voľba návrhového vzoru

Pre implementačnú časť mobilnej aplikácie je zvolený návrhový vzor MVVM. Oproti klasickému MVC nebudú vytvárané masívne controller objekty. Pomocou MVVM a viewmodel objektov sa zníži previazanosť užívateľského rozhrania s logikou aplikácie a tým sa zvýši testovateľnosť a znovupoužiteľnosť kompenent aplikácie. MVVMC nie je zvolený z dôvodu slabšej dokumentácie a rozšírenosti celého návrhového vzoru naprieč komunitou. Voľbou MVVM však nie je úplne zrušená možnosť prechodu na MVVMC v budúcnosti. V prípade usúdenia potreby MVVMC je pridanie coordinator objektu realizovateľné bez výrazného narušenia MVVM.

2.1.9 Databázový model

Na základe predošlej analýzy a návrhu použitých návrhových vzorov spolu s definovaním doménového modelu je možné konkrétne definovať dátové štruktúry, ktoré sa budú v mobilnej aplikácii ukladať. Mobilná aplikácia bude ukladať informácie spojené s užívateľom a dáta o jednotlivých dostupných datasetoch na serveri. V nasledujúcej časti sú definované databázové štruktúry ktoré budú uložené v mobilnom zariadení. V nasledujúcej časti sú opísané aj štruktúry, ktoré budú ukladané do lokálnej databázy, užívateľských predvolených nastavení alebo kľúčenky iOS.

2.1.9.1 Dataset

Dataset je objekt reprezentujúci dátovú sadu. V mobilnej aplikácii budú ukladané základné informácie a štruktúra každého dostupného datasetu. Ukladá sa názov, unikátny kód, url, unikátne id, parametre dátovej sady a disjunktná množina hodnôt parametrov dátovej sady. Disjunktná množina hodnôt parametru predstavuje možnosti filtrovacích hodnôt daného parametru. Dataset má práve jeden graf a každý graf patrí práve jednému datasetu.

2.1.9.2 Graph

Graph je objekt reprezentujúci grafické zobrazenie dátovej sady. Obsahuje id, názvy svojich osí, prislúchajúcich parametov, orientáciu osí a množinu filtrovacích parametrov.

2.1.9.3 SortingParameter

SortingParameter je objekt reprezentujúci filtrovací parameter. Filtrovací parameter má názov parametru, ktorý je definovaný kódom spolu s ekvivalentom v spisovnej verzii a id. Každý filtrovací parameter patrí jednému grafu.

2.1.9.4 DistinctValue

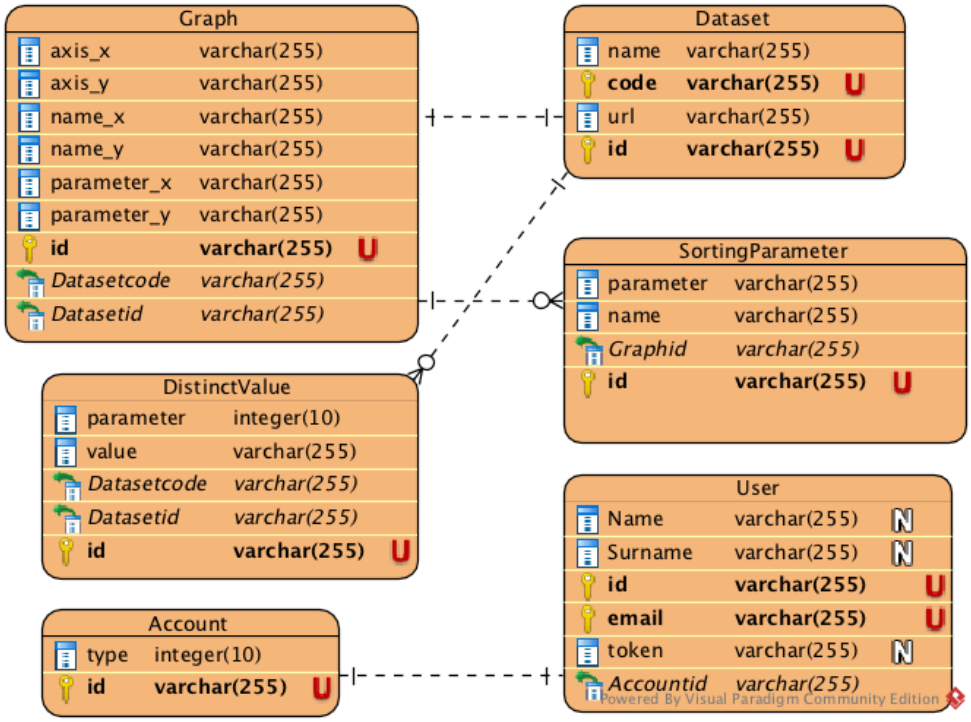
DistinctValue je objekt reprezentujúci disjunktnú množinu hodnôt parametru grafu. Je zložený z názvu parametra a hodnoty. Tento objekt patrí práve jednému grafu a každý graf môže mať jeden alebo viac disjunktných hodnôt.

2.1.9.5 User

User je objekt reprezentujúci užívateľské dáta. Tento objekt nie je ukladaný priamo do lokálnej databázy ale do šifrovanej kľúčenky iOS. Skladá sa z mena, id, tokenu. Každý užívateľ patrí práve jednému účtu.

2.1.9.6 Account

Account je objekt reprezentujúci užívateľský účet. Je zložený z id a typu účtu. Účet má priradený práve jedného užívateľa. Tento objekt je ukladaný do užívateľských predvolených nastavení.



Obr. 2.11: Databázový diagram mobilnej aplikácie

2.2 Architektúra systému

Architektúra je v knihe *Web Engineering, The Discipline of Systematic Development of Web Applications* definovaná ako štruktúra softwarového systému, ktorá rozdeľuje celý systém na podsystémy a definuje vzťahy medzi komponentami systému. Na základe architektúry je systém jednoduchšie pochopiteľný. Tvorba architektúry záleží na rôznych faktoroch, ako napríklad funkčné požiadavky, kvalita (výkon, škálovateľnosť, znovupoužiteľnosť), technické aspekty (operačný systém, middleware vrstva) a iné[19]. V nasledujúcej časti sú definované základné vrstvy architektúry a opísané možné riešenia spolu s ich kladnými a zápornými vlastnosťami. V závere časti je zvolená vhodná architertúra pre implementačnú časť práce.

2.2.1 Vrstvy architektúry

V nasledujúcej časti je definovaná prezentačná, aplikačná a dátová vrstva architektúry.

2.2.1.1 Prezentačná vrstva

Prezentačná vrstva (taktiež známa ako „klienstká vrstva“) je najvyššia vrstva architektúry. Predstavuje to, čo užívateľ vidí keď používa daný systém. Prezentačná vrstva zbiera užívateľské vstupy a posiela ich nižším vrstvám.

2.2.1.2 Aplikačná vrstva

Aplikačná vrstva (taktiež známa ako „business vrstva“) predstavuje logiku systému. Aplikačná vrstva typicky zabezpečuje validáciu dát, výpočty, autorizáciu, úpravu dát v dátovej vrstve a iné. Celkovo tým poskytuje funkcionalitu systému. Vykonáva komunikačné rozhranie medzi prezenčnou a dátovou vrstvou, čím zabezpečuje obojstrannú komunikáciu.

2.2.1.3 Dátová vrstva

Dátová vrstva má za úlohu ukladanie a poskytovanie dát ostatným vrstvám. Dátová vrstva poskytuje metódy pre pripojenie k databáze, získavanie, aktualizáciu, vkladanie, mazanie dát a iné.

2.2.2 Druhy architektúr

V nasledujúcej časti sú predstavené a analyzované rôzne druhy architektúr. V závere časti je zvolená vhodná architektúra, ktorá sa bude využívať pri tvorbe výsledného systému.

2.2.2.1 Jednovrstvová architektúra

Jednovrstvová architektúra sa skladá z prezentačnej, aplikačnej a dátovej vrstvy, ktoré sú všetky v jednej aplikácii. Logika a všetky dáta sú uložené lokálne v aplikácii.

2.2.2.2 Dvojvrstvová architektúra

Dvojvrstvová architektúra je zložená z oddelenej prezentačnej a dátovej vrstvy s ktorou priamo kumunikuje cez komunikačné rozhranie. Výhodami tejto architektúry je rýchlosť komunikácie, jednoduchá modifikácia a údržba. Nevýhodou je výkonnosť systému, ktorý sa znižuje s narastajúcim počtom užívateľom.

2.2.2.3 Trojvrstvová architektúra

Architektúra je zložená z prezentačnej, aplikačnej a dátovej vrstvy. Oproti jednovrstvovej architektúre sú všetky vrstvy od seba oddelené. Medzi výhody architektúry patrí vyššia výkonnosť, škálovateľnosť, znovupoužiteľnosť, zvýšená bezpečnosť a iné. Nevýhody, ktoré vyplývajú z trojvrstvovej architektúrý sú spojené so zvýšenou komplexnosťou a úsilím pri realizácii systému.

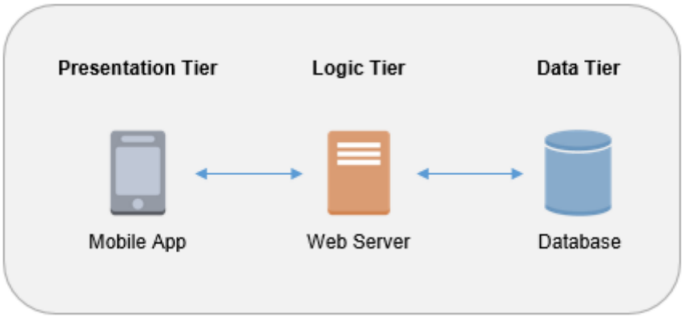
2.2.2.4 N-vrstvová architektúra

N-vrstvová architektúra (taktiež známa ako „distribuovaná aplikácia“) predstavuje ďalšiu vrstvu systému. Je podobná ako trojvrstvová architektúra, ale na rozdiel od nej, je zložená z väčšieho počtu aplikačných serverov za účelom distribúcie aplikačnej logiky. To je považované za hlavnú výhodu oproti trojvstvovej architektúre. S väčším počtom aplikačných serverov vzrastá finančná náročnosť a komplexnosť systému.

2.2.3 Voľba architektúry systému

Vzhľadom na špecifikáciu požiadaviek a typ vyvíjaného systému bolo zvolená trojvstvtová architektúra.
Na základe analýzy požiadaviek a dátových zdrojov je zhodnotené, že výsledný systém nie je realizovateľný pomocou jednovrstvovej architektúry. Dátové zdroje neposkytujú žiadne API, ktoré by umožnovali efektívne získavať do mobilnej aplikácie potrebné dáta priamo zo zdrojov. Jednorázové stiahnutie dát by zapríčinilo zbytočne veľkú veľkosť mobilnej aplikácie a spracovávanie veľkého množstva dát, čo je výpočtovo a časovo neprípustné. Aktualizácia dátových sád by v prípade jednovrstvovej architektúry bez existujúceho API vyžadovala opätovné nahrávanie aplikácie na App Store alebo kompletné stiahnutie všetkých dátových sád. V prípade nefunkčnosti alebo zmene URL zo strany otvorených dát by aplikácia prišla o dáta a tým o značnú časť svojej funkcionality. Nevýhodou jednovrstvovej architektúry je nutnosť prítomnosti celej logiky systému na mobilnom zariadení.
Dvojvrstvová architektúra oproti jednovrstvovej odstraňuje nutnosť nahrávania aplikácie na App Store v prípade aktualizácie otvorených dát. Na druhú stranu vyžaduje manuálnu aktualizáciu dát v databáze administrátorom aplikácie. Ten by sa v prípade dvojvrstvovej architektúry musel starať o dáta v databáze a pravidelne ich manuálne aktualizovať. Výhodou oproti jednovrstvovej architektúre je aj to, že v prípade zmeny alebo zániku URL dátovej sady sa nič neprejaví na funkčnosti mobilnej aplikácii. Dvojvrstvová architektúra, podobne ako aj jednovrstvová, vyžaduje prítomnosť celej aplikačnej logiky v mobilnom zariadení.
Trojvstvová architektúra umožňuje udržiavať aplikačnú logiku na aplikačnom serveri. To má za následok odľahčenie mobilnej aplikácie od zložitých strojových výpočtov a tým jej zrýchlenie. Takiež umožňuje tvorbu middlewarových služieb ako napríklad autentikácia a autorizácia užívateľov, správa užívateľských účtov a výpočty ktoré sú jedny z hlavných funkčných požiadaviek systému. Oproti dvojvrstvovej architektúre ponúka možnosť lepšie zabezpečiť dáta. V prípade, keď nie je dostupné žiadne API od poskytovateľov dát, trojvrstvová architektúra umožňuje poloautomatickú aktualizáciu dát. Administrátor môže špecifikovať na aplikačnom serveri zoznam URL, z ktorých budú v preriodickom čase získavané a aktualizované dáta, čím sa aktualizácia

oproti iným architektúram výrazne zjednoduší.
N-vrstvová architektúra nie je zvolená z dôvodu finančnej náročnosti a komplexnosti riešenia. V budúcnosti pri prípadnom veľkom záujme o aplikáciu a s narastajúcou funkcionalitou pripadá do úvahy rozdeliť aplikačnú logiku na viac serverov.



Obr. 2.12: Trojvrstvová architektúra systému

Na obrázku 2.12 je znázornená obecná ilustrácia trojvstvovej architektúry systému. Diagram komponent a sekvenčný diagram komunikácie medzi komponentami výslednej architektúry je predstavený v implementačnej časti.

2.3 Serverová časť

Súčasťou výsledného systému je aj serverová časť, ktorá je jednou z cieľov tejto práce. Pod pojmom serverová časť je chápaná aplikačná a dátová vrstva výsledného systému. Pred samotnou implementáciou serverovej časti je zhotovená analýza a špecifikácia požiadaviek. Následne autor na základe špecifikácie požiadaviek analyzuje možné technické riešenia. Záver tejto časti je venovaný výberu vhodných technológií a architektúry celého systému.

2.3.1 Analýza požiadaviek

Za účelom splnenia všetkých funkčných požiadaviek mobilnej aplikácie a výsledného systému je potrebný návrh a realizácia serverovej časti. Serverová časť prijíma požiadavky od mobilnej aplikácie, zastrešuje prevažnú časť logiky celého systému a sprostredkuje odpovede a dáta naspäť klientskej časti systému. Konkrétne požiadavky na serverovú časť systému sú:

Komunikačné rozhranie
Serverová časť poskytuje aplikačné a komunikačné rozhranie mobilnej aplikácii.

Vytvorenie užívateľského účtu

Serverová časť systému zabezpečuje vytvorenie užívateľského účtu, a zabezpečuje unikátnosť účtov na základe emailovej adresy.

Aktualizácia užívateľského účtu
Serverová časť systému umožňuje autorizovanému užívateľovi aktualizovať emailovú adresu, heslo a užívateľove meno.

Prihlásenie do klientskej aplikácie
Serverová časť systému validuje platnosť prihlasovacích údajov užívateľa.

Zabezpečená komunikácia
Serverová časť poskytuje zabezpečenú komunikáciu.

Odlásenie z klientskej aplikácie
Serverová časť systému monitoruje a umožňuje odhlásenie užívateľa.

Výpočet približnej a presnej výšky dôchodku
Serverová časť poskytuje aplikačnú logiku výpočtu presnej a približnej výšky dôchodku. Klientskej aplikácii je umožnený prístup k tejto fukncionalite cez konumikačné rozhranie.

Autorizácia užívateľov a tvorba tokenov
Serverová časť zabezpečuje autorizovaný prístup k API na základe tokenov. Serverová aplikácia zabezpečuje validáciu a tvorbu rôznych autorizačných práv vo forme tokenov.

Komunikácia s databázou
Serverová časť umožňuje komunikáciu s databázou a vloženie, získavanie, aktualizáciu a mazanie dát.

Automatická aktualizácia dát
Aktializácia dátových záznamov je umožnená automaticky v pravidelnom periodickom čase alebo implicitne, na podnet administrátora.

Vyťažovanie dát
Serverová časť umožnuje vyťažovať dáta zo zdrojov otvorených dát

Validácia dát
Serverová časť umožňuje validáciu dátových štruktúr jednotlivých prichádzajúcich požiadaviek. Požiadavky s neplatnou štruktúrou alebo obsahom dát sú odmietnuté.

Administrátorský prístup
Serverová časť poskytuje administrátorský prístup na základe administrátor-

ských oprávnení k niektorým výlučným funkciám systému. Jedná sa prevažne o funkcie spojené s aktualizáciami a modifikáciou dátových sád.

Šifrovanie hesiel
Serverová časť zabezpečuje ukladanie šifrovaných hesiel užívateľov.

2.3.2 Komunikačné protokoly

V nasledujúcej časti sú analyzované možné komunikačné protokoly, pomocou ktorých môže mobilná aplikácia a serverová časť spolu komunikovať. V závere časti je zvolený komunikačný protokol, ktorý sa bude v implementácii používať.

2.3.2.1 SOAP

Simple Object Access Protocol (ďalej len „SOAP“) je protokol webových služieb, ktorý definuje tri hlavné entity: poskytovateľ služby (angl. „service provider“), register služby (angl. „service registry“) a žiadateľ služby (angl. „service requester“). Poskytovateľ služby je sieťovo adresovateľná entita, ktorá prijíma a vykonáva požiadavky od konzumentov. Konzument je aplikácia, služba, alebo nejaký druh softwarového modulu, ktorý vyžaduje služby od poskytovateľa. Register služby je sieťový slovník, ktorý obsahuje dostupné služby. Konzument služby nájde popis danej služby v slovníku, ktorý je publikovaný poskytovateľom služby. Následne pomocou popisu konzument začne interakciu so službou. Komunikácia medzi jednotlivými komponentami je založená na XML formáte a SOAP protokole. SOAP správa sa skladá z obálky, hlavičky a tela správy. Hlavička obsahuje informácie o požiadavke a odpovedi. Obálka pomáha identifikovať XML dokument ako SOAP správu, ktoré sú posielané pomocou transportných protokolov SMTP, FTP alebo HTTP. Nevýhodou SOAP oproti HTTP protokolu je veľkosť správy, ktorej spracovanie vyžaduje vyššie nároky na výpočetné zdroje.

2.3.2.2 REST

Pojem REST (angl. „Representational state transfer“), bol prvýkrát definovaný Royom Fieldingom. RESTová architektúra je architektúra klient-server v ktorej klient posiela požiadavky a server vracia odpovede. Tieto požiadavky sú postavené na prenose stavu zdroja. Zdroj je niečo, čo je identifikovateľné URI. Reprezentácia zdroja je typicky dokument, ktorý zachytáva aktuálny stav zdroja. Návrhové princípy RESTu dodržiavajú adresovateľnosť, bezstavovosť a univerzálne rozhranie. REST podporuje serverové odpovede v rôznych formátoch ako napríklad XML, JSON, HTML a iné. Adresovateľnosť RESTových modelov spočíva v možnosti operovať so zdrojmi, ktoré sú označené URI. Univerzálne rozhranie slúži k prístupu zdrojov pomocou pevne stanovených HTTP metód. Každá transakcia je nezávislá na predchádzajúcej trans-

akcii. Server neudržuje informácie o reláciách a z toho dôvodu sú serverové odpovede nezávislé. Tieto princípy robia RESTové aplikácie jednoduchými a ľahkými. Aplikácie, ktoré dodržiavajú RESTovú architektúru sa nazývajú RESTful služby. RESTful služba používa GET, PUT, POST a DELETE http metódy na získavanie, vytvorenie, aktualizáciu a mazanie zdrojov [20].

2.3.2.3 Voľba komunikačného protokolu

Na základe analýzy je zvolený REST ako protokol, ktorý bude využívaný na komunikáciu medzi serverovou časťou a mobilnou aplikáciou. REST je menej typovo orientovaný ako protokol SOAP. REST optori SOAP nevyžaduje vo formáte správy obálku a hlavičku. Hlavnou výhodou REST protokolu oproti HTTP je jeho rýchlosť a výkon. Štúdia [20], ktorá porovnáva výkon REST a SOAP protokolov tvrdí, že RESTful správa má 9 až 10-krát menšiu veľkosť a spracovanie a prenos trvá 5 až 6 krát kratší čas ako s použitím SOAP protokolu.

2.3.3 Formát komunikačných správ

V nasledujúcej časti su analyzované možné formáty správ, pomocou ktorých môže mobilná aplikácia komunikovať so serverovou časťou. Na záver je zvolený výsledný formát, ktorý sa bude v implementácii používať.

2.3.3.1 XML

XML (angl. „Extensible Markup Language“) opisuje triedy dátových objektov nazývaných XML dokumenty a čiastočne opisuje správanie počítačových programov, ktoré tieto triedy spracúvajú. XML je podmnožinou SGML, z ktorého čiastočne vychádza. XML dokument slúži na uschovanie jednotiek nazývaných entity, ktoré obsahujú rozobraté alebo nerozobraté dáta. Rozobraté dáta sa skladajú zo znakov. Niektoré tvoria znakové údaje a niektoré z nich tvoria značku. Značka kóduje popis rozloženia a logickú štruktúru dokumentu. XML procesor je softwarový modul, ktorý sa používa na čítanie a tvorbu XML dokumentov. XML procesor poskytuje prístup k obsahu a štruktúre dokumentu. Procesor vykonáva svoju úlohu v mene iného modulu, ktorý sa nazýva aplikácia. Aplikácia špecifikuje ako má procesor čítať alebo vytvárať XML dokument. Medzi základné návrhové ciele XML okrem iných patria[21]:

- použiteľnosť naprieč internetom
- podpora širokej škály aplikácií
- jednoduchosť písania aplikácií, ktoré spracúvajú XML dokumenty
- kompatibilita so SGML
- jednoduchosť tvorby a čitateľnosť dokumentov

2.3.3.2 JSON

JSON je navrhnutý s cieľom slúžiť ako jazyk určený na výmenu dát, ktorý je ľudsky čitateľný a jednoduchý na strojové spracovanie, rozloženie a použitie. JSON objekt sa skladá z dvoch štuktúr:

- kolekcia zložená z názvu a hodnoty
- list hodnôt

Každá hodnota má jeden z nasledujúcich dátových typov: reťazec, číslo, JSON objekt, boolean, pole alebo null hodnotu[22]. Formát dát je opísaný pomocou JSON schémy. Schéma sa využíva pri validácii, testovaní štruktúry dát alebo ako ľudsky a strojovo čitateľná dokumentácia[23]. JSON je priamo podporovaný v rámci programovacieho jazyku JavaScript a tým poskytuje značné výkonnostné výhody oproti XML a YAML, ktorý vyžaduje dodatočné knižnice na získanie dát z objektového modulu dokumentu. Výkonnostné výhody sú podložené a analyzované v štúdiách *Comparison of JSON and XML Data Interchange Formats: a Case Study*[24] a *Comparison between JSON and YAML for data serialization*[25].

2.3.3.3 YAML

YAML je formát slúžiaci na serializáciu dát vo forme, ktorá je pre človeka ľahko čitateľná. YAML je inšpirovaný konceptmi z programovacích jazykov ako C, Perl a Python. YAML položka sa skladá z postupnosti dvojíc kľúč/hodnota. Hodnoty v YAML sú dátového typu reťazec, číslo, boolean, dátum, mapa, sekvencia alebo null. V dokumente *YAML Ain't Markup Language YAML*[26] vidia hlavné rozdiely medzi YAML a JSON formátoch v ich prioritách. Zatiaľ čo návrh JSON dokumentov je založený na jednoduchosti a univerzálnosti, z čoho plynie možnosť triviálneho generovania a rozkladania dát za cenu zníženej čitateľnosti. Hlavným cieľom YAML je ľudská čitateľnosť dokumentu a podpora serializácie ľubovoľných dátových štruktúr. YAML poskytuje extrémne čitateľné súbory, ktoré sú zložitejšie na generovanie a analýzu. V štúdii, ktorá porovnáva výkon serializácie formátov YAML a JSON je dokázaná značne vyššia výkonnosť pri používaní JSON formátu[25] oproti YAML.

```
{
  "menu": {
        "id": "file",
        "value": "File",
        "popup": {
        "menuitem": [
          {"value": "New", "onclick": "CreateNewDoc()"},
          {"value": "Open", "onclick": "OpenDoc()"},
          {"value": "Close", "onclick": "CloseDoc()"}
         ]
        }
  }
}
```

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

```
---
menu:
  id: file
  value: File
  popup:
    menuitem:
    - value: New
      onclick: CreateNewDoc()
    - value: Open
      onclick: OpenDoc()
    - value: Close
      onclick: CloseDoc()
```

Obr. 2.13: Porovnanie zápisu dokumentu vo formáte JSON, XML a YAML

2.3.3.4 Výber formátu komunikačných správ

Formát JSON je zvolený pre komunikáciu medzi webovou a serverovou časťou. Oproti YAML a XML umožňuje rýchlejšiu tvorbu a spracovanie dát. Navyše JSON formát poskytuje dostatočne ľudsky čitateľnú formu zápisu.

2.3.4 Databázové technológie

V nasledujúcej časti sú analyzované databázové technológie. Na záver je zvolená konkrétna databáza, do ktorej budú ukladané dáta spojené s výsledným systémom.

Rank	DBMS	Score	Changes
1.	Oracle	1289.79	+ 0.18
2.	MySQL	1226.40	-2.46
3.	Microsoft SQL Server	1095.51	-9.28
4.	PostgreSQL	395.47	-3.88
5.	MongoDB	341.41	+ 0.89
6.	DB2	188.95	+ 2.28
7.	Microsoft Access	132.22	+ 0.27
8.	Elasticsearch	131.36	+ 2.81
9.	Redis	130.11	-1.12
10.	Cassandra	119.09	-4.40
Copyright © April 2018 DB-Engines.com			

Obr. 2.14: 10 najpoužívanejších databáz podľa indexu DB engines[27]

2.3.4.1 Relačné databázy

Relačná databáza je založená na relačnom dátovom modeli. Relačná databáza poskytuje SQL rozhranie a ACID transakcie. Schéma tabuľky, známa tiež ako relačné schéma je definovaná názvom tabuľky a fixným počtom atribútov s prislušnými dátovými typmi. Záznam odpovedá jednému riadku v tabuľke, ktorý obsahuje hodnoty každého atribútu. Relácia sa skladá z množiny jednotných záznamov. Nad množinami relácií sú definované množinové operácie ako napríklad selekcia, projekcia a spojenie. Taktiež sú podporované základné operácie ako vytvorenie, modifikácia, znazanie, operácie pre kontrolu transakcií a užívateľský manažment[28]. Najrozšírenejšia relačná databáza sa nazýva Oracle (2.14).

2.3.4.2 Kľúč-hodnota databázy

Databáza typu hodnota-kľúč je databáza NoSQL druhu, ktorá ukladá iba páry hodnota-kľúč. Hodnoty z databázy je možné dostať iba pomocou daného kľúča. Databázy tohto druhu umožňujú okrem ukladania dát aj replikáciu, verzovanie, zamykanie, transakcie, radenie a iné funkcionality. Klientské rozhranie umožňuje vkladanie, mazanie a indexovanie[29]. Tieto systémy nie sú zvyčajne vhodné pre komplexné aplikácie, ale môžu nastať aj prípady kedy

je extrémna jednoduchosť vhodná. Databázy typu kľúč-hodnota sú používané vo vstavaných systémoch[30]. Najrozšírenejšia databáza typu kľúč-hodnota sa nazýva Redis (2.14).

2.3.4.3 Dokumentovo orientované databázy

Dokumentovo orientované databázy majú uložené dáta v dokumentoch. Na rozdiel od databáz typu kľúč-hodnota podporujú druhotné indexy a viac druhov dokumentov v databáze. Dokumentovo orientované databázy oproti relačným databázam nepodporujú transakčné vlastnosti ACID[29]. Dáta v dokumentoch nepodliehajahú žiadnej pevne definovanej schéme. Databáza sa skladá z množiny kolekcií, kde každá kolekcia je zložená z dokumentov. Dokumenty nemusia mať jednotnú štruktúru. Z toho dôvodu môžu mať rozdielne stĺpce. Dátové typy jednotlivých stĺpcov môžu byť taktiež odlišné. Oproti relačným databázam a databázam typu kľúč-hodnota môže jeden atribút nadobúdať viac hodnôt, ktoré sú definované v poli. Každý dokument môže mať v sebe vstavanú ďalšiu štruktúru. Voľná štuktúra dokumentov zapríčiňuje vyšší výkon a lepšiu celkovú škálovateľnosť databázy. Voľná štruktúra dát zároveň negatívne ovplyvňuje udržovateľnosť databázy, nakoľko užívateľ musí sám zaistiť kontrolu správnosti dát, cudzích kľúčov a podobne. Dokumentovo orientované databázy nepodporú pri dotazovaní operácie typu spojenie, čo má za následok menšiu flexibilitu pri dotazovaní sa na dáta. Dokumentovo orientované databázy často používajú interné zápisy, ktoré môžu byť spracovávané priamo aplikáciami ako napríklad JSON[31]. Najrozšírenejšia dokumentovo orientovaná databáza sa nazýva MongoDB (2.14).

2.3.4.4 Grafové databázy

Dáta v grafovej databáze sú reprezentované v grafových štruktúrach ako uzly a hrany. Hrany grafu reprezentujú vzťahy medzi jednotlivými uzlami, ktoré obsahujú konkrétne dáta. Grafové databázy umožňujú jednoduché spracovanie dát a jednoduché výpočty vlastností grafu, ako napríklad počet krokov k prejdeniu z jedného špecifického uzlu do druhého. Grafové databázy oproti ostatným sú vhodné na rýchle zistenie vzťahu medzi uzlami (dátami). Nevýhoda grafových databáz spočíva v tom, že neposkytujú indexy na všetky uzly a tým pádom nie je možný priamy prístup k hodnotám všetkých atribútov[32]. Taktiež nie sú vhodné na dotazovanie veľkého množstva analytických dát[29]. Najrozšírenejšia grafová databáza sa nazýva Neo4j (2.14).

2.3.4.5 Priestorové databázy

Priestorové databázy (angl. „wide column stores“) ukladajú dáta v záznamoch so schopnosťou udržiavať veľmi veľké množstvo dynamických stĺpcov. Názvy stĺpcov a kľúče jednotlivých záznamov nie sú pevne dané a každý záznam môže

mať až miliardy stĺpcov. Priestorové databázy môžu byť považované ako dvojdimenzionálne databázy typu kľúč-hodnota. Priestorové databázy majú podobnú charakteristiku voľných schém ako dokumentovo orientované databázy ale ich implementácia je veľmi rozdielna[33]. Na rozdiel od dokumentovo orientovaných databáz nepodporujú vstavané objekty ale iba skalárne typy[29]. Najrozšírenejšia databáza typu kľúč-hodnota sa nazýva Cassandra (2.14).

2.3.4.6 Výber databázy

Pre ukladanie dát sú zvolené dekumentovo orientované databázy. Na základe rôznorodosti dátových sád a ich rozdielnych parametroch je pevné schéma definované relačným modelom v relačnej databáze považované za nevhodné. Databázy typu kľúč-hodnota nie sú vhodné pre komplexnejšie aplikácie čo by mohlo viesť k limitácii aplikácie. Grafové databázy nie sú považované za vhodné pri dotazovaní vo veľkom množstve analytických dát. Dokumentovo orientované databázy sú uprednostnené pred priestorovými z dôvodu celkovej škálovateľnosti, možnosti uložiť vstavané objekty a podpory interných zápisov ako napríklad vo formáte JSON. Spomedzi dokumentovo orientovaných databáz je zvolená MongoDB z dôvodu najväčšej rozšírenosti.

2.3.5 Technické riešenia serverovej časti

V nasledujúcej časti sú analyzované použiteľné technológie a technické riešenia pri implementácii serverovej časti. Analyzované sú programovacie jazyky Java, Python, Ruby, Javascript a PHP. Na záver je zvolený najvhodnejší z nich.

2.3.5.1 Java

Java je podľa TIOBE indexu najrozšírenejší programovací jazyk na svete [34]. Je objektovo orientovaný, distribuovaný, robustný a silne typovaný programovací jazyk. Java je navrhnutá s cieľom mať čo najmenej závislostí medzi komponentami. Programovací jazyk bol prvýkrát predstavený v roku 1995 a odvtedy bol vytvorený robustný ekosystém obsahujúci množstvo dokumentácie, frameworkov, knižníc rôznych druhov. Java je prenositeľný programovací jazyk, ktorý nie je závislý na platforme. Využíva sa napríklad aj v mobilných telefónoch alebo rôznych zabudovaných systémoch. Java rozlišuje chyby na tie, ktoré vznikli v čase kompilácie a tie, ktoré vznikli v čase behu programu. Jedná sa o vyšší programovací jazyk, ktorý je kompilovaný do množiny bytekódových inštrukcií. Inštrukcie sú následne interpretované pomocou virtuálneho stroja Java (angl. „Java virtual machine“)[35]. Použitie programovacieho jazyka Java je vhodný pri tvorbe rozsiahlych informačných systémov s dôrazom na robustnosť a bezpečnosť riešenia. V porovnaní s ostatnými riešeniami vyžaduje väčšie množstvo práce, čo môže zavážiť pri tvorbe menších systémov.

2.3.5.2 Python

Python je interpretovaný, dynamicky typovaný vyšší programovací jazyk na všeobecné použitie. Python má vlastný manažment pamäte. Prvýkrát bol predstavený v roku 1991 a dnes už existuje množstvo knižníc a frameworkov. Medzi ďalšie výhody patria jeho jednoduchá škálovateľnosť a podpora mnohých systémov a platforiem. Python skripty dokážu spúšťať C/C++ skripty, Java alebo .NET komponenty, čo z neho robí vhodný nástroj na tvorbu rozšírení a úpravu softwaru na mieru. Mark Lutz vo svojej knihe[36] tvrdí, že množstvo Python kódu sa rovná jednej tretine kódu s rovnakou funkcionalitou napísaného v C++ alebo Jave. Python poskytuje množstvo konštrukcií, ktoré spolu s dodržiavaním viacerých programovacích paradigmov umožňujú vytvárať udržovateľný, čistý kód malých ale aj veľkých rozmerov. Nevýhodou Pythonu je jeho rýchlosť, ktorá je pomalšia oproti kompilovaným jazykom.

2.3.5.3 Ruby

Ruby je interpretovaný skriptovací programovací jazyk. Ruby je plne objektovo orientovaný jazyk čo znamená, že všetko v Ruby je objekt. Ruby má vlastný systém na správu knižníc a frameworkov nazývaný RubyGems. Najznámejším frameworkom pre tvorbu serverových aplikácii je Ruby on Rails. Rails používa MVC architektúru pre organizáciu aplikácie. Poskytuje základné štruktúry pre databázy, webové služby a webové stránky. Výhodou Rails je, že vychádza z množstva softwarových vzorov a paradigm ako napríklad DRY, CoC alebo vzor aktívneho záznamu (angl. „active record paterm“), ktorý definuje prácu s dátovými zdrojami[37]. Nevýhodou jazyka je jeho rýchlosť oproti kompilovaným jazykom, závislosť na interpretovi a slabšia dokumentácia oproti ostatným jazykom.

2.3.5.4 Javascript

JavaScript je multiplatformový, objektovo orientovaný programovací jazyk. Jeho syntax patrí do rodiny jazykov C/C++/Java, ale od týchto jazykov je odlišný. Javascript je slabo typovo orientovaný, čo znamená, že premenné nemusia mať definovaný typ. Momentálne je JavaScript často používaný ako interpretovaný jazyk pre webové stránky, ktorý je vkladaný priamo do HTML kódu stránky[38]. JavaScript je používaný aj pre tvorbu serverových aplikácií a to najmä pomocou frameworku Node.js, ktorý bol vytvorený v roku 2009. Pre spracovanie požiadavky Node.js používa neblokujúci asynchrónny model bežiaci na jednom vlákne. To zabezpečuje minimalizáciu réžie procesoru a tým maximalizáciu výkonu. Napríklad v prípade prístupu do databázy namiesto blokovania vlákna do ukončenia úlohy je úloha spustená a po skončení sa zavolá spätné volanie, ktoré indikuje ukončenie úlohy. Node.js sa stará o paralelné prevedenie daných úkonov, ktoré bežia na pozadí, čím odpadá záťaž programátorom. Node.js sa skladá z Javascript enginu a niekoľkých štandardných

knižníc. Node.js podporuje mnoho druhov databáz a umožňuje jednoduchú synchronizáciu. Je navrhnutý hlavne na aplikácie typu vstup/výstup, čo v tomto prípade zapríčiňuje značne lepšie výkonnostné výsledky oproti iným programovacím jazykom a knižniciam[39]. Vzhľadom na to, že sa jedná o JavaScriptový framework je jednoducho synchronizovateľný s databázami dokumentového typu, ktoré používaju JSON formát pre zápis dokumentov.

2.3.5.5 PHP

PHP je najvyužívanejší programovací jazyk na strane serveru určený pre tvorbu webových stránok[3]. PHP je v knihe *Programing PHP*[40] definovaný ako jednoduchý objektovo orientovaný, dynamicky typovaný programovací jazyk navrhnutý na tvorbu HTML obsahu. PHP bol prvýkrát predstavený v roku 1995 a momentálne poskytuje množstvo frameworkov a knižníc. Primárne použitie programovacieho jazyka PHP sa rozdeľuje na tri hlavné kategórie: tvorba serverových skriptov, tvorba skriptov v príkazovom riadku a tvorba grafického užívateľského rozhrania. Tvorba serverových skriptov bola spočiatku navrhnutá na vytváranie dynamického obsahu webových stránok. Webové skripty sú určené na spracovávanie, tvorbu a posielanie webových dát ako aj tvorbu XML dokumentov, animácií, PDF súborov a podobne. PHP dokáže spúšťať skripty z príkazového riadku podobne ako Perl, Ark alebo Unix shell. Za pomoci knižnice PHP-GTK[4] je možné vytvoriť multiplatformné GUI aplikácie v PHP. PHP beží na všetkých hlavných operačných systémoch a všetkých popredných webových serveroch. Výhodou PHP je, že podporuje okrem iného veľké množstvo databázových technológií, väčšinou sa však jedná o databázy SQL typu. Medzi PHP sa zaraďuje napríklad slabší výkon, nižšia modularita a nekonzistencia pomenovania funkcií (napríklad strpos, strchr, str_replace, str_pad).

2.3.5.6 Výber technického riešenia

Pre tvorbu serverovej časti práce je zvolený Javacsriptový framework Node.js. Ten poskytuje podľa testovaní[5] oproti ostatným interpretovaným jazykom značné výkonnostné výhody. Navyše je navrhnutý konkrétne pre aplikácie typu vstup/výstup. Programovací jazyk Java síce poskytuje lepší výkon v testovaniach, ale tvorba serverových aplikácií vyžaduje značne väčšie množstvo kódu a úsilia na tvorbu. Java je vhodná na veľké a robustné informačné systémy. Výhodou Node.js je jednoduchá synschronizácia a používanie dokumentovej databázy MongoDB, ktorá je zvolená pre ukladanie dát. Ďalšou výhodou je natívna podpora JSON formátu bez nutnosti integrácie doplkových knižníc.

[3] `https://w3techs.com/technologies/overview/programming_language/all`
[4] `http://gtk.php.net`
[5] `https://benchmarksgame-team.pages.debian.net/benchmarksgame/`

Kapitola 3

Implementácia

Nasledujúca kapitola je zameraná na popis implementačného procesu výsledného systému ako aj samotné nasadenie do produkčného prostredia. Zachytáva implementačný proces z pohľadu mobilnej aplikácie, serverovej časti a systému ako celku. V kapitole sú zhrnuté použité technické nástroje, knižnice, softwarové služby, platformy ale aj problémy a postupy, ktoré počas implementácie autor riešil a používal.

3.1 Inicializácia projektu

Pred samotnou implementáciou je realizovaná inicializácia projektov mobilnej aplikácie a serverovej časti. V nasledujúcej časti sú opísané použité softwarové nástroje, vývojové prostredia, poskytovatelia služieb a platformy, ktoré sú využívané pri implementácii a nasadení mobilnej aplikácie a serverovej časti. Táto časť popisuje niektoré elementy všeobecne a nezachádza do hĺbky. Konkrétne využitie pri implementácii a ukážky použitej funkcionality a kódu sú detailne rozoberané v nasledujúcich častiach.

3.1.1 Xcode

Mobilná aplikácia je vyvíjaná vo vývojovom prostredí Xcode 9.3, ktorý umožňuje vyvíjať mobilné aplikácie aj pre najnovší operačný systém iOS 11.
Xcode je profesionálne vývojové prostredie od spoločnosti Apple, ktoré ponúka nástroje na vývoj aplikácií pre operačné systémy iOS, macOS, watchOS a tvOS. Xcode je voľne dostupný na App Store výhradne pre zariadenia s operačným systémom macOS. Xcode podporuje programovacie jazyky ako napríklad Java, Python, PHP, Objective-C, C, C++ alebo Swift. Hlavnými súčasťami vývojového prostredia sú: vstavaný kompilátor s terminálom, editor zdrojového kódu, asistent editovania kódu, nástroje na verzovanie systému, kompletná dokumentácia, simulátor, integrovaný builder užívateľského rozhrania, grafický

debugger, testovacie frameworky, nástroje na nepretržitú integráciu a mnoho iných užitočných nástrojov[41].

3.1.2 Webstorm

Serverová časť je vyvíjaná vo vývojovom prostredí WebStorm 3.3. WebStorm je vývojové prostredie od spoločnosti JetBrains, ktorý podporuje bezplatnú študentskú licenciu. WebStorm je zameraný na programovacie jazyky Javascript, TypeScript, jazyky zamerané na štylistiku ako napríklad CSS a ich frameworky. Zo serverových frameworkov je podporovaný Node.js a Meteor. Hlavnými súčasťami vývojového prostredia sú debugger, vstavaný terminál, nástroje na spúšťanie, testovanie, záruku kvality, verzovanie systému, balíčkovací manažér a iné[42].

3.1.3 MongoDB

Pri vývoji systému sú používané dve inštancie databázy MongoDB. Inštancia, ktorá beži na lokálnom stroji je využívaná v testovacom prostredí na testovanie a ladenie systému. Druhá inštancia beží na databázovom serveri v produkčnom prostredí. V oboch prostrediach je nasadená verzia 3.4.10.

3.1.4 Git

Git je v knihe [43] definovaný ako systém na správu verzií. Tento verzovací systém je použitý pri implementácii systému. Verzovací systém je nevyhnutný pri práci v tímoch ale užitočný aj pre lokálne použitie. Git je systém určený na revíziu a správu verzii systému. Hlavným cieľom Gitu a verzovacích systémov obecne je zaistiť vývoj a udržiavanie obsahu úložiska, prístup k historickým verziám každého záznamu a zachytávať všetky zmeny vykonané v čase. Git pracuje na princípe uzol-uzol, čo znamená že každý uzol má lokálne uložené kompletné úložisko. V prípade lokálneho použitia sa môže zdať používanie verzovacieho systému zbytočné, ale v prípade rozšírenia vývojárskeho tímu sa môže nový vývojár hneď zapojiť do projektu. Ďalším dôvodom použitia Gitu je možnosť vrátiť sa k historickým, funkčným verziám systému a tým v prípade nefunkčnosti vyvíjanej aplikácie ušetriť čas pri opravovaní. Pomocou hostingovej služby Bitbucket sú vytvorené a spravované dve rozdielne Git úložiská, do ktorých je počas implementácie nahrávaný kód. Jedno úložisko je určené pre mobilnú aplikáciu a druhé pre serverovú časť systému. V každom projekte je vytvorený súbor `.gitignore`. Gitignore definuje súbory, ktoré sú ignorované a nebudú monitorované systémom Git. Zvyčajne sa jedná o súbory operačného systému, frameworky, balíčkovacie systémy alebo oprávnenia.

3.1.5 Apiary

Pri implementácii je použítá webová služba Apiary[6]. Apiary umožňuje rýchlo navrhnúť, zdokumentovať a testovať API serverovej časti systému. Pomocou Apiary môže každý programátor alebo používateľ systému prehľadne zistiť čo dostupné API ponúka a dokáže. Apiary navyše dokáže jednotlivé sieťové požadavky zavolať a tým slúži aj ako vhodný nástroj na testovanie. Pomocou Apiary je zdokumentované vernejné API serverovej časti. Vytvorená dokumentácia API pomocou neplatenej verzie Apiary je verejná a z toho dôvodu neobsahuje privátne API, ktoré je dostupné len administrátorom serverovej časti. Prevažne sa jedná o API spojené s konfiguráciou a administrátorskou modifikáciou dát. Odkaz na Apiary je na priloženom CD v súbore `/readme.txt`.

3.1.6 Google cloud platform

Google Cloud Platform[7] (ďalej len „GCP“) je sada verejných cloudových služieb ponúkaných spoložnosťou Google. Služby GCP možno rozdeliť do piatich hlavných karegórii: výpočtová technika a hosting, úložisko, sieťové služby, veľké dáta, strojové učenie. Výsledný systém využíva služby výpočtovej techniky a hostingu spolu s cloudovým úložiskom.
Kategória výpočtovej techniky a hostingu ponúka štyri modely cloudových služieb.

- Cloudové funkcie Google - funkcia ako služba (FaaS) je model, ktorý ponúka bezserverové spustiteľné prostredie na vytváranie a spájanie cloudových služieb. Programátor vytvorí jednoduché jednoúčelové funkcie, ktoré sú pripojené k udalostiam, ktoré vznikajú v cloudovej infraštruktúre. Cloudové funkcie sú spustené v okamihu keď nastane udalosť na architektúre. Udalosťou je typicky HTTP požiadavka na určitú URL, ktorá je definovaná pri tvorbe funkcie. Použitím cloudových funkcií odpadá nutnosť tvorby serveru. V prípade HTTP požiadavky programátor len špecifikuje konkrétnu URL a druh požiadavky, na ktorú sa má vykonať daná funkcia.

- Aplikakačná platforma - platforma ako služba (PaaS) je model, ktorý ponúka platformu. Model poskytuje užívateľom cloudové prostredie, v ktorom môžu vyvíjať, spravovať a dodávať aplikácie. Užívateľ má možnosť využívať automaticky spravovanú, škálovateľnú a zabezpečenú platformu. Platforma umožňuje jednoduché nasadenie aplikácie, monitorovanie, spúštanie, flexibilné prostredie, prístup k dátovým úložiskám a podobne.

- Výpočtové kontainery - kontainer ako služba (CaaS) je model ktorý ponúka počítačový kontainer. Vďaka počítačovému kontajneru sa uží-

[6] `https://apiary.io`
[7] `https://cloud.google.com`

vateľ môže zamerať na kód aplikácie namiesto nasadenia a integrácie do hostiteľského prostredia.

- Virtuálne stroje - infraštruktúra ako služba (IaaS) je model, ktorý poskytuje užívateľom prístup k infraštruktúre. Infraštruktúra zahŕňa napríklad výpočtové prostriedky, systém pre ukladanie dát, sieťové prostredie a podobne. Programátor si v tomto prípade sám vyberá a stará sa o operačný systém, vývojové nástroje, jazyky, frameworky, služby a ostatné softwarové technológie, ktoré preferuje.

Implementácia serverovej časti využíva aplikačnú platformu od Google cloud s nakonfigurovaným prostredím pre Node.js. Aplikácia je nasadzovaná do produkčného prostredia pomocou jedného príkazu z terminálu. GCP zobrazuje ovládací panel na ktorom sa monitoruje počet požiadaviek na API, vyťaženosť serveru, aktuálnu konfiguráciu, chybové hlásenia a iné užítočné informácie o bežiacej službe.
GCP služby cloudového úložiska ponúkajú perzistenciu dát v štyroch základných kategóriách: SQL databázy, NoSQL dátové uložiská, perzistentný disk priamo na výpočtovej technike alebo relačné databázy. Serverová časť využíva pre perzistenciu dát NoSQL dátové úložisko a perzistentný disk priamo na serveri. NoSQL dátové úložisko je využívané na ukladanie textových konfiguračných súborov a dočasné ukladanie CVS súborov dátových sád vo formáte CVS. Na perzistentnom disku serverov je nasadená inštancia dokumentovej mongoDB databázy, ktorá slúži na perzistenciu dátových sád a informáciách o uživateľovi vo formáte JSON. MongoDB databáza je nasadená do produkčného prostredia pomocou služby mLab[8], ktorá je na stránkach CGP odporúčaná ako jedna zo spôsobov nasadenia.

3.1.7 Firebase

Firebase je vývojová platforma mobilných a webových aplikácií, ktorá patrí pod spoločnosť Google. Firebase je možné definovať výrazom „backend ako služba“ (BaaS). Umožňuje programátorom mobilných aplikácií rýchlejšie a efektívnejšie vytvárať aplikácie bez nutnosti programovať vlastné riešenie daných služieb. Firebase je podobne ako GCP produkt poskytovaný spoločnosťou Google a tým je zaručená vysoká kompatibilnosť týchto produktov. Niektoré zo služieb Firebase sa navzájom prekrývajú so službami CGP a naopak. Služby Firebase sú však zamerané konktrétne na mobilné aplikácie. Firebase poskytuje oproti CGP autentikáciu užívateľov, monitorovanie pádov a výkonu mobilných aplikácii, služby sprostredkujúce posielanie správ a oznámení do mobilných zariadení, predikcie a iné fukncionality. Vysledný systém používa od Firebase službu autentikácie užívateľov a monitorovanie pádov mobilnej

[8] `https://mlab.com/`

aplikácie. Podrobné použitie a integrácia služieb je opísana v nasledujúcich častiach tejto kapitoly.

3.2 Mobilná aplikácia

V nasledujúcej časti sú opísane komponenty, technológie, postupy a problémy, ktoré sú implementované v mobilnej aplikácii.

3.2.1 RxSwift

Mobilná aplikácia je napísaná v programovacom jazyku Swift. Oproti klasickému funkcionálnemu programovaniu je v implementácii použitý reaktívny framework RxSwift[9]. Reaktívne systémy sú v knihe *Synchronous Programming of Reactive Systems*[44] definované ako systémy, ktoré neustále reagujú na prostredie rýchlosťou, ktorá záleži na danom prostredí. Na rozdiel od klasických systémov, ktoré majú vstupy dané na začiatku behu a vracajú výstup na konci behu, reaktívne systémy neprestajne reagujú s prostredím v perióde, ktorá je definovaná prostredím.
Reaktívne programovanie je programovanie založené na asynchrónnych tokoch dát a udalostí. Tok udalostí predstavuje napríklad vstupy klávesnice, stlačenie tlačidla, gestá, aktualizácia GPS a podobne. Pomocou reaktívneho programovania a RxSwift je možné jednoducho sledovať tieto toky dát a udalostí a adekvátne na ne reagovať. V klasickom objektovom programovaní je toto zabezpečené delegátmi, notifikáciami, vnorenými spätnými volaniami a podobne. Tieto konštrukcie často vedú k neprehľadnosti kódu, zvýšeniu závislostí medzi jednotlivými časťami aplikácie a celkovo sa aplikácia stáva ťažšie pochopiteľná. RxSwift a celkovo reaktívne programovanie je založené na návhovom vzore Pozorovateľ (angl. „Observer“). Návrhový vzor sa skladá z pozorovateľa a subjektu (angl „subject“). Každý subjekt má zoznam svojich pozorovateľov a v prípade akejkoľvek zmeny svojho stavu subjekt oboznámi všetkých pozorovateľov o tejto zmene. Oznámenie o zmene je zvyčajne realizované zavolaním nejakej metódy pozorovateľa. V RxSwift pozorovateľ pozoruje objekty typu `Observable`. Každý subjekt a pozorovateľ implementuje nejakú podmnožinu metód z trojice: `onNext`, `onError`, `onCompleted`, pomocu ktorých oznamuje subjekt pozorovateľom zmenu svojho stavu. Inicializácia pozorovania objektu v RxSwift je možná pomocou rôznym metód, z ktorých najzákladnejšia je metóda `subscribe`.

[9] `https://github.com/ReactiveX/RxSwift`

```
func rx_signinWith(email e: String, password pass: String)
        -> Observable<Bool> {
        return Observable.create { [unowned self] observer in
            Auth.auth().signIn(withEmail: e, password: pass,
            completion: { [unowned self] user, error in
                if let error = error {
                    observer.onError(error)
                } else {
                    observer.onNext(true)
                    self.account = .email
                    observer.onCompleted()
                }
            })
            return Disposables.create()
        }
}
```

Obr. 3.1: Ukážka funkcie vytvárajúcej subjekt typu Observable

```
rx_signinWith(email: email.value, password: password.value)
        .subscribe(onNext: { [unowned self] status in
            if status {
                self.successfulSignInAction()
            } else {
                self.failedSignInAction()
              }
            }, onError: { error in
                self.showError()
                }, onCompleted: {
                    self.signInCompletedAction()
        })
```

Obr. 3.2: Ukážka inicializácie pozorovania subjektu

Ukážka kódu 3.1 zachytáva funkciu na prihlásenie do aplikácie pomocou emailovej adresy a hesla. Funkcia vracia objekt typu observable, ktorý reprezentuje subjekt. Na základe priebehu autorizácie vytvorený objekt volá metódy `onNext`, `onError` a `onCompleted` na všetkých svojich pozorovateľoch, ktorý daný subjekt pozorujú. Ukážka kódu 3.2 zachytáva inicializáciu odoberania udalostí a dátových tokov pozorovateľom. Pozorovateľ implementuje metódy `onNext`, `onError` a `onCompleted`, ktoré môže subjekt na pozorovateľovi zavolať. V prípade, že subjekt zavolá na pozorovateľovi nejakú metódd

z danej trojice, ktorú pozorovateľ nemá implementovanú, nevyvolá sa žiadna akcia ani chyba.

3.2.2 Konfigurácia aplikácie

Celá aplikácia je implementovaná s dôrazom na obecnosť, dynamickosť, rozširovateľnosť a jednoduchú úpravu aplikácie. Každá dátová sada je rôzna. Obsahuje rôzne dáta, ktoré su filtrované odlišnými parametrami. Je potrebné vyriešiť problém ako vytvoriť aplikáciu, ktorá dokáže fungovať, reagovať na zmeny a zabezpečiť funkčnosť mobilnej aplikácie bez toho, aby bol nutný zásah do kódu mobilnej aplikácie. Dátové sady sa môžu časom meniť, pridávať a mazať. Môže nastať potreba zmeniť grafickú reprezentáciu dát, zmeniť filtrovacie parametre alebo mnoho iných vecí. S cieľom umožniť konfiguráciu aplikácie dynamicky podľa potrieb a tým vyriešiť tento problém, je vytvorený konfiguračný súbor Manifest.json.

```
{
 "datasets": [
 {
  "name": "Výdaje na důchody podle roku a druhu důchodu",
  "code": "ds_04",
  "parameters": [
    {"name": "druh_duchodu_kod"},{ "name":"druh_duchodu"},
    {"name": "referencni_obdobi"},
    {"name":"vydaje_na_duchody_opravene_o_zalohy_v_tis_kc"}
  ],
  "graph": {
    "axis": [
        {"axis": "x","parameter": "referencni_obdobi","name":
        ↪  "Rok"},
        {"axis": "y", "parameter":
        ↪  "vydaje_na_duchody_opravene_o_zalohy_v_tis_kc",
        "name":"Výdaje v tis. Kč"}
    ],
    "sort": [
        {"parameter": "druh_duchodu","name": "Druh důchodu"}
    ]
  },
  "url": "https://data.cssz.cz/dump/vydaje-na-duchody-v-cr.csv"
 }]
}
```

Obr. 3.3: Ukážka manifestu s jednou dátovou sadou

Mobilná aplikácia vždy po úspešnom prihlásení zistí, či je na serveri nová verzia mafinestu. V prípade, že sa na serveri nachádza novšia verzia ako je uložená v mobilnom zariadení, potom mobilná aplikácia požiada server o novú verziu. Tú server poskytne v tele odpovedi na požiadavku vo formáte JSON. Mobilná aplikácia rozparsuje dáta a uloží ich na perzistentné úložisko. Na základe dát z manifestu sú dynamicky zostavené jednotlivé obrazovky aplikácie a ich komponenty. Manifest je zostrojený tak, aby mohol byť v budúcnosti rozšíriteľný o ďalší druh informácií a nič neprekážalo aktuálnej implementácii. Momentálne definuje zoznam dostupných dátových sád, ich parametre a spôsob, ako sa majú graficky vykresliť. Pomocou manifestu je možné jednoducho meniť, pridávať alebo odoberať dátové sady. Taktiež je možné nastaviť filtrovacie parametre pomocou ktorých budú zobrazené dáta filtrované.

3.2.3 Prihlásenie, registrácia a odhlásenie

V nasledujúcej časti sú opísané prihlásenie, registrácia a odhlasovanie z aplikácie. Na všetky funkcionality je použítá služba Firebase Authentication. Mobilná aplikácia pri registrácii, prihlásení a odhlásení komunikuje priamo s Firebase backendom. Mobilná aplikácia v tomto prípade nekomunikuje s vytvoreným Node.js serverom. Pri týchto funkcionalitách sú využívané knižnice FirebaseAuth a GoogleSignIn.

3.2.3.1 Registrácia

Registrácia v aplikácii je možná pomocou emailu, hesla a mena alebo autorizačnej služby Google. Každá registrácia je založená na unikátnej emailovej adrese, ktorá nemôže byť rovnaká pre dva rôzne účty.
Registrácia pomocou emailu, hesla a mena je implementovaná pomocou triedy Auth, ktorá v prípade úspešnej registrácie vracia vytvorený užívateľský účet. Požiadavka na registráciu pomocou služby Google je realizovaná pomocou triedy GIDSignIn. Trieda GIDSignIn inicializuje užívateľské rozhranie potrebné pre autorizáciu službou Google. Služba podľa platnosti autorizácie vracia v delegát metóde užívateľský účet, ktorý je zaregistrovaný do Firebase alebo chybu.

3.2.3.2 Prihlásenie

Prihlásenie je rovnako ako registrácia možné pomocou emailu a hesla alebo pomocou autorizačnej služby Google. Prihlásenie pomocou emailu a hesla je implementované pomocou triedy Auth, ktorá v prípade úspešného prihlásenia vracia vytvorený užívateľský účet.
Prihlásenie do aplikácie pomocou autorizačnej služby Google má dve fázy. Prihlásenie je zabezpečené triedami GIDSingIn a GoogleAuthProvider. Trieda GIDSingIn slúži na autorizáciu v službe Google. Trieda GIDSignIn vráti užívateľský účet alebo chybu. Užívateľský účet je potrebné pre prihlásenie do

aplikácie overiť ešte u autorizačnej autority Google. Na to slúži trieda GoogleAuthProvider. GoogleAuthProvider na základe oprávnení, ktoré obsahuje užívateľský účet z GIDSignIn autorizuje užívateľa vo Firebase alebo vráti chybu.

3.2.3.3 Automatické prihlásenie

Automatické prihlásenie je realizované pomocou služby Firebase. Tá v prípade, že sa užívateľ pred vypnutím aplikácie neodhlásil Firebase eviduje užívateľa stále prihláseného v systéme. Firebase udržuje užívateľa prihláseného vo svojom systéme permanentne až do kým nie je explicitne zavolaná metóda `Auth.auth().signOut()`. Aplikácia pri opätovnom spustení zistí hodnotu premennej `Auth.auth().currentUser`, ktorá môže byť typu FIRUser alebo nil. V prípade, že je hodnota premennej currentUser nenulová, aplikácia požiada o obnovenie tokenu. Následne je na pozadí kontrolovaná dostupnosť konfiguračného súboru `Manifest.json` (3.2.2). Tým je ošetrený prípad, že užívateľ úplne zmazal aplikáciu a opätovne ju nainštaloval. Užívateľ je do aplikácie vpustený v prípade existencie platného tekonu a konfiguračného súboru.

3.2.3.4 Odhlásenie

Odhlásenie je implementované triedou Auth z knižnice Firebase. V prípade odhlásenia nezáleží na spôsobe prihlásenia a je realizované metódou `logout`. V prípade úspešného odhlásenia aplikácia zmaže všetky špecifické dáta, ktoré sa týkajú daného uživateľa ako napríklad typ prihlásenia alebo token. Aplikácia po odhlásení presmeruje užívateľa na úvodnú stránku.

3.2.4 Vykresľovanie grafov

Pre vykresľovanie grafov je použitá knižnica Charts[10]. Knižnica ponúka množstvo druhov grafov ako napríklad koláčové, čiarové, bodové, stĺpcové a iné. V mobilnej aplikácii sú implementované stĺpcové grafy. Vytvárané grafy su rendrované z grafického kontextu, ktorý je dynamicky dopočítavaný. To umožňuje približovať a oddiaľovať graf alebo posúvať graf do strán. Hodnoty grafu sa nastavujú pomocou dvoch polí. Okrem nastavenia grafu umožňujú zobrazovať legendy, farby, maximálny a minimálny počet zobrazených stĺpcov a podobne. Nevýhoda frameworku Charts spočíva v tom že nie je možné vykresliť legendu zvyslej osy y. Počas implementácie bol odhalený tento nedostatok a nepodarilo sa nájsť lepšiu alternatívnu náhradu za tento framework. Ostatné frameworky buď taktiež nepodporujú túto funkcionalitu alebo nespĺňajú iné závažné požiadavky.

[10] `https://github.com/danielgindi/Charts`

3.2.5 Komunikácia so serverom a autorizácia

Komunikácia so serverom je realizovaná cez HTTP protokol s využitím knižnice Alamofire[11]. Telo a hlavičky jednotlivých správ sú kodované vo formáte JSON. Mobilná aplikácia komunikuje so serverovou časťou cez poskytnuté API. Všetky požiadavky, ktoré sú odoslané na server pomocou HTTP požiadavky vyžadujú prítomnosť autorizačného parametru v hlavičke požiadavky. Autorizačným parametrom je užívateľský IDToken, ktorý prihlásený užívateľ získa z Firebase. V prípade neplatnej autorizácie server vracia na požiadavky HTTP stavový kód 401. Zakaždým, keď mobilná aplikácia príjme od serveru stavový kódu 401 je automaticky na pozadí vytvorená požiadavka na Firebase o nový IDToken pomocou metódy getIDToken.

```
func getIDToken() -> Observable<String?> {
return Observable.create({ [unowned self] observer in
    guard let user = Auth.auth().currentUser
    else { observer.onNext(nil); return Disposables.create() }
    user.getIDToken { [unowned self] (token, error) in
        guard let token = token
        else { observer.onNext(nil); return}
        self.setUserToken(token: token)
        observer.onNext(token)
        observer.onCompleted()
    }
    return Disposables.create()
})
}
```

Obr. 3.4: Ukážka funkcie slúžiacej na získanie nového IDTokenu

Funkcia na získanie nového tokenu vytvorí subjekt typu Observable, ktorý daný pozorovateľ začne pozorovať. V prípade úspešného získania nového tokenu je predošlá neúspešná požiadavka opätovne odoslaná.

3.2.6 Parsovanie a perzistencia dát

Perzistenia dát v mobilnej aplikácii je implementovaná pomocou frameworkov Realm[12], KeychainSwift[13] a užívateľských nastavení NSUserDefaults. Databáza Realm je alternatíva pre SQLite a Code Data. Vďaka jej návrhu je rýchlejšia ako objektovo-relačné mapovanie a SQLLite[45]. Navyše ponúka jednoduché rozhranie a možnosť automatickej synchronizácie. Do databázy

[11] https://github.com/Alamofire/Alamofire
[12] https://realm.io
[13] https://github.com/evgenyneu/keychain-swift

sú ukladané objekty dátových sád, grafy, filtrovacie parametre a hodnoty parametrov. Konkrétne atribúty jednotlivých objektov sú znázornené v databázovom modeli (2.11). Užívateľský účet sa ukladá do užívateľských nastavení. Z objektu užívateľa sa v zariadení ukladá iba token, ktorý je ukladaný do šifrovaných kľúčeniek pomocou frameworku KeychainSwift.
Parsovanie dát je realizované pomocou framerowku SwiftyJSON[14]. Ten ponúka jednoduchý wrapper sa parsovanie dát čím odpadá nutnosť používať zložité konštrukcie. Parsovanie dát je realizované pri spracovávaní serverových odpovedí.

3.2.7 Výpočty spojené s dôchodkom

Mobilná aplikácia ponúka možnosť výpočtu presnej výšky dôchodku, približnej výšky dôchodku a dôchodkového veku. Presná a približná výška dôchodku sú počítané pomocou serverovej časti systému. Mobilná aplikácia zbiera dáta z formulárov, ktoré následne upravuje do požadovanej formy, ktorá je definovaná na Apiary. Pre získanie výšky dôchodku sú odoslané požiadavky typu POST na dané URL. Na strane mobilnej aplikácie prebieha čiastočná validácia vstupov. Počet mesiacov do dôchodkového veku a počet mesiacov do nároku na dôchodok nesmú byť zároveň dve nenulové čísla. Iné hodnoty vstupov nie sú kontrolované v mobilnej aplikácii. V prípade úspešnej odpovede aplikácia zobrazí hodnotu dôchodku. V opačnom prípade je zobrazené chybové hlásenie zo serveru na základe HTTP stavového kódu alebo správy z tela serverovej odpovede.
Kalkulačka dôchodkového veku je zobrazená na obrazovke doplňujúcich informácií. Dôchodkový vek aplikácia počíta sama bez serverovej časti. Nakoľko nie sú dáta o dôchodkovom veku jednoducho strojovo čitateľné, je manuálne vytvorený súbor s vhodnou štruktúrou, ktorý odpovedá informáciám z webu ČSSZ[15]. Tento súbor je pri prvom prihlásení jednorázovo stiahnutý a uložený do mobilného zariadenia. Pri opätovných prihláseniach sa následne kontroluje len dostupnosť novších verzií. Mobilná aplikácia pracuje s týmto súborom lokálne, čo umožňuje dynamicky dopočítavať dôchodkový vek bez nutnosti zakaždým posielať požiadavku na server. Mobilná aplikácia reaktívne mení hodnotu dôchodkového veku automaticky po zmene jednej z hodnôt zobrazených na obrazovke (dátum narodenia, pohlavie, počet detí).

3.2.8 Aktualizácia profilu a zmena hesla

V nasledujúcej časti je opísaná aktualizácia profilových informácií a zmena hesla. Tieto funkcionality sú implementované pomocou Firebase, ktorý poskytuje jednoduché API. Táto funkcionalita je v aplikácii dostupná iba v prípade že je užívateľ prihlásený pomocou emailu a hesla. V prípade, že užívateľ je

[14] `https://github.com/SwiftyJSON/SwiftyJSON`
[15] `http://www.cssz.cz/cz/duchodove-pojisteni/davky/starobni-duchody.htm`

prihlásený pomocou autorizačnej služby Google tak táto funkcionalita na profilovej obrazovke nie je dostupná.

3.2.8.1 Zmena emailu a mena

Uživateľ má možnosť na jednej obrazovke zmeniť svoj prihlasovací email a meno. Pre zaistenie bezpečnosti je od užívateľa pri každej zmene vyžiadané heslo s cieľom zabrániť neautorizovanej zmene údajov. Mobilná aplikácia najprv skontroluje či došlo ku zmene údajov, následne zvaliduje zadané údaje a potom odošle požiadavku na aktualizáciu do Firebase. Pre zmenu užívateľského emailu slúži funkcia `updateEmail(to:)`, ktorá v spätnom volaní vracia chybu. Proces zmeny osobných údajov sa skladá z dvoch fáz. V prvej fáze je vytvorená požiadavok na zmenu pomocou metódy createProfileChangeRequest. Následne sa požiadavka naplní dátami, ktoré chce užívateľ zmeniť. Zmena informácií sa následne potvrdí pomocou metódy commitChanges. V prípade, že všetko prebehne bez chýb, je užívateľ presmerovaný naspäť na obrazovku profilu, kde sú už zobrazené aktualizované dáta. V prípade, že sa vyskytne chyba mobilná aplikácia zobrazí chybovú hlášku.

3.2.8.2 Zmena hesla

Pri zmene hesla aplikácia vyžaduje tri heslá. Nové, staré a potvrdenie nového hesla. Mobilná aplikáciá zvaliduje zadané heslá a v prípade že validácia prebehne bez chýb, je odoslaná požiadavka na zmenu na Firebase. Zmena hesla je realizovaná pomocou funkcie `updatePassword(to:)`, ktorá vracia v spätnom volaní chybový objekt alebo hodnotu nil.

3.2.9 Manažér objekty

Implementácia mobilnej aplikácie používa manažér objekty, ktoré sú implementované s cieľom znížiť počet závislostí medzi jednotlivými komponentami a zamedziť opakované vkladanie a linkovanie niektorých knižníc. Manažér objekt zapúzdruje a poskytuje určitý druh špecifickej funkcionality ako napríklad sieťové požiadavky, prístup a práca s databázou, parsovanie dát a podobne. Manažér objekt reprezentuje návrhový vzor unikát (angl. „singleton“). V mobilnej aplikácii sú implementované nasledujúce manažér objekty: AccountManager, FileManager, NetworkManager, DataManager a JSONParser. Account manager slúži na správu užívateľského účtu. FileManager sa stará u ukladanie dát do užívateľských nastavení. NetworkManager sa stará o sieťovú komunikáciu so serverom. DataManager zabezpečuje perzistenciu, modifikáciu a získavanie dát z databázy. JSONParser slúži na parsovanie správ, ktoré sú vo formáte JSON a vytváranie triednych inštancií.

3.3 Serverová časť

V nasledujúcej časti sú opísané komponenty, technológie, postupy a problémy, ktoré sú implementované v serverovej časti systému.

3.3.1 Express.js

Express.js[16]je webový framework, ktorý je založený na Node.js http module a sieťových komponentoch, ktoré sú taktiež nazývané middleware. Pomocou express.js je v implementácii vytvorený webový server. Middleware je software, ktorý poskytuje služby softwarovým aplikáciám nad rámec služieb, ktoré sú poskytované operačným systémom. V implementácii je vytvorený middleware určený napríklad na autorizáciu užívateľa alebo administrátora, validáciu dát a podobne. Express.js má obvykle jeden hlavný súbor, ktorý vykonáva nasledujúcu réžiu:

- vkladanie závislostí knižníc tretích strán, modulov, modelov alebo pomocných súborov
- konfiguráciu express.js aplikácie
- definovanie middlewarových funkcií, cookies, statických ciest zložiek a súborov
- definovanie ciest (angl. „routes“)
- pripojenie k databáze
- spustenie aplikácie

Po spustení express.js aplikácia počúva na požiadavky. Tie sú spracúvané na základe postupnosti definovaných middleware funkcií. Na základe týchto postupností je možné kontrolovať tok aplikácie[46]. Express.js obsahuje triedu Router. Router je smerovací middleware, ktorý slúži na smerovanie požiadaviek. V implementácii sú využívané metódy ako napríklad get, post, delete, update, use a iné.

[16] `https://expressjs.com`

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;
app.use(express.static('public'));
app.use(morgan('dev'));
app.use(bodyParser.json());
const pensionRouter = require('./api/v1/controllers/pensions.js');
app.use('/v1/pension',pensionRouter);
app.listen(PORT, () => {
    console.log(`Server is listening on port: ${PORT}`);
});
```

Obr. 3.5: Ukážka kódu z app.js zobrazujúca inicializáciu, konfiguráciu, definovanie cesty a spustenie aplikácie

V hlavnom súbore app.js sú definované smerovače, ktoré aplikácia používa. Na základe url sú požiadavky presmerované na zodpovedný smerovač, ktorý je následne zodpovedný za spracovanie požiadavky. Na pensionRouter z ukážky 3.5 sú presmerované všetky požiadavky začínajúce na `/v1/pesion`. Logika každého smerovaču je definová v novom controller objekte. Tým je zaistená modulárnosť a vyššia prehľadnosť aplikácie. Express aplikácia je pred jej spustením nakonfigurovaná potrebnými knižnicami alebo hodnotami premenných.

```
mongoose.connect(mongoUri);
var db = mongoose.connection;
db.on('error', function () {
throw new Error('Unable to connect to database at '+mongoUri);
});
db.once('open', function(){
console.log(`Connected to the database at uri: ${mongoUri}`);
});
```

Obr. 3.6: Ukážka z app.js, ktorá zobrazuje pripojenie aplikácie k databáze

3.3.2 Perzistencia dát

Pre jednoduchšiu prácu s dátami je v implementácii použitá framework Mongoose.js[17]. Mongoose poskytuje jednoduché rozhranie pre modelovanie dát uložených v MongoDB ako aj CRUD a iné operácie s dátami. Umožňuje okrem pripojenia k databáze vytvárať schému jednotlivých objektov a tým poskytuje validačné funkcionality. Tým je zamedzený pokus o vloženie objektu do

[17] `http://mongoosejs.com`

databázy, ktorý neodpovedá schéme. Pri dodržiavaní správnych názvov modelov mongoose poskytuje automatické mapovanie týchto modelov na kolencie dokumentov z databázy. Napríklad model s názvom Pension je v databáze mapovaný na kolekciu pensions.

```
var mongoose = require('mongoose');
var pensionSchema = mongoose.Schema({
    code: {
        type: String,
        required: true
    }
}, {strict: false});
PensionModel = mongoose.model('Pension', pensionSchema);
```

Obr. 3.7: Definícia schémy objektu dôchodku

Nakoľko záznamy z dátových sád nemajú pevne stanovené žiadne povinné parametre, schéma dôchodku vyžaduje len existenciu kódu daného záznamu. Každý záznam z dátovej sady, ktorý je pridaný do databázy musí obsahovať kód dátovej sady z ktorej pochádza. Okrem kódu má pri vytvorení každý záznam povinne parameter `_id`. Identifikátor je v prípade, že ho nešpeficikuje užívateľ vytvorený pomocou mongoose. Konštrukcia {`strict:false`} definuje možnosť pridávania ľubovoľných ďalších parametrov do objektu, ktoré nie sú definované v schéme.

3.3.3 Vytažovanie a aktualizácia dát

Server musí umožňovať pravidelne vyťažovať, aktualizovať, pridávať, alebo meniť dátové sady. Aktualizácia a vyťažovanie dát sú implementované dvoma spôsobmi: pomocou explicitného volania HTTP požiadavky na server, ktoré vyžaduje administrátorské práva alebo automaticky s pravidelným opakovaním pomocou cronu.
Cron je softwarový démon, ktorý v operačnom systéme automaticky spúšťa v určitom čase nejaký príkaz alebo script. Pre vytvorenie cron príkazu je v implementácii použitá knižnica cron[18]. Cron príkaz je v implementácii spúšťaný raz za mesiac vždy prvý deň v mesiaci. Spustená funkcia aktualizuje všetky dátové sady podľa aktuálneho manifestu. Dáta sú získavané z URL adries. V prípade, že sa URL adresa v priebehu času stane neplatná, dátová sada nie je z databázy zmazaná. Je na uvážení administrátora serverovej časti, aby dátovú sadu zmazal z manifestu. Záznamy z dátových sád, ktoré sa v aktuálnom manifeste nenachádzajú sú zmazané.
Aktualizácia explicitným volaním HTTP požiadavky spúšťa rovnakú aktu-

[18] `https://github.com/kelektiv/node-cron`

alizačnú funkciu ako cron. Rozdiel je v tom, že z bezpečnostných dôvodov je vyžadované administrátorské oprávnenie. Užívateľ, ktorý chce vzdialene vyvolať aktualizáciu, musí do hlavičky požiadavky pridať administrátorský kľúč.

3.3.4 Autorizácia

Na serveri je implementovaná užívateľská a administrátorská autorizácia. Funkcie `tokenAuthorization` a `adminAuthorization`, ktoré implementujú autorizačnú logiku sú v aplikácii využívané ako middleware. Ten zabezpečuje, aby boli zdroje zo serveru dostupné iba autorizovaným užívateľom. Každá prijatá požiadavka je na serveri vždy pred spracovaním autorizovaná. Autorizácia je vždy prevedaná ako prvá middleware funkcia v poradí. V prípade že autorizácia zlyhá, server nepokračuje v spracovaní požiadavky a vracia stavový kód 401.
Užívateľská autorizácia je implementovaná pomocou knižnice Firebase-admin. Firebase-admin je administrátorská knižnica pre Node.js, ktorá umožňuje komunikovať s Firebase zo strany serveru. Pred použitím je nutné službu nakonfigovať a do projektu vložiť súbor `keyfile.json`, ktorý obsahuje privátny kľúč a iné informácie umožňujúce komunikáciu s Firebase. Verifikácia tokenu, ktorý klientská aplikácia posiela v hlavičke poziadavky prebieha pomocou funkcie `verifyIdToken(token:)`. Funkcia po skončení zavolá v asynchrónne spätné volanie, ktoré obsahuje identifikovaného užívateľa alebo chybu. V prípade úspešnej autorizácie je funkcia ukončená volaním `next()`, čím je spustený nasledujúci middleware v poradí.
Administrátorská autorizácia je implementovaná bez použitia knižníc tretích strán. Middleware kontroluje či sa token z hlavičky rovná administrátoskému kľúču, ktorý je definovaný v súbore `admin.json`. Následný postup vyhodnotenia je rovnaký ako v prípade užívateľskej autorizácie.

3.3.5 Validácia dát

Validácia dát je implementovaná aj na strane serveru s cieľom zaistiť správnu funkcionalitu. Validácia dát podlieha dokumentácii z Apiary. Validácia je zameraná na celkovú štruktúru tela požiadavku ale aj hodnoty, ktoré jednotlivé kľúče nadobúdajú. Validácia dát je implementovaná pomocou knižnice Joi. Pred samotnou validáciou je vytvorený Joi objekt, v ktorom sú definované jednotlivé kľúče a ich obor hodnôt. Následne validácia prebieha pomocou funkcie `validate`, ktorá má za parametre joi objekt a dáta, ktoré majú prejsť validáciou. V prípade neúspešnej validácie server vracia stavový kód 400 a validačnú správu vyprodukovanú knižnicou Joi.

```
const validateExactPensionInput = function(req, res, next) {
 const exactPensionSchema = Joi.object().keys({
 "years": Joi.number().min(0).max(100).required(),
 "early-retirement": Joi.number().min(0).max(200).required(),
 "late-retirement": Joi.number().min(0).max(200).required(),
 "salaries": Joi.array().min(30).items(
    Joi.object().keys({
        "year": Joi.number().required(),
        "salary": Joi.number().min(0).required(),
        "non-worked-days":
        Joi.number().min(0).max(366).required()
    })
 ).required()
 });

    const result = Joi.validate(req.body,
            exactPensionSchema, function(err, value){
        if(err){
            res.status(400).send(err);
        } else {
            next();
        }
    });
};
```

Obr. 3.8: Ukážka validácie tela požiadavky na výpočet presnej výšky dôchodku

3.3.6 Štatistické dáta a výpočty

V nasledujúcej časti je opísaný algoritmus na výpočet presnej a približnej výšky dôchodku a spôsob poskytovania štatistických dát mobilnej aplikácii.

3.3.6.1 Výpočet dôchodku

Presný výpočet dôchodku je vypočítaný na základe nasledujúcich informácií:

- počet odpracovaných rokov
- počet mesiacov do dôchodového veku v prípade predčasného dôchodku
- počet mesiacov po nároku na dôchodok v prípade neskoršieho odchodu do dôchodku
- priemerný mesačný hrubý zárobok každý rok od roku 1986 až 2017

- výlučná doba[19] každý rok počas rokov 1986 až 2017

Presný výpočet dôchodku odpovedá matematickému postupu, ktorý je predstavený na webovej stránke `https://www.vypocet.cz/popis-vypoctu-duchodu`. Tento postup vychádza z informácií, ktoré sú dostupné na webe ČSSZ[20].
Výpočet približnej výšky výpočtu slúži ako zjednodušený model. Tento model oproti presnému výpočtu počíta s nulovou výlučnou dobou pre každý rok a priemerný mesačný hrubý zárobok je udávaný v päťročných intervaloch. Model počíta s rovnakou priemernou výškou hrubej mzdy v daných časových intervaloch. Konkrétny algoritmus výpočtu je rovnaký ako v prípade výpočtu presnej výšky dôchodku.
Konkrétna implementácia výpočtov výšky dôchodkov je v súbore `calculations.js`.

3.3.6.2 Štatistické dáta

Server poskytuje štatistické dáta mobilnej aplikácii na základe filtrovacích parametrov z požiadavky a kódu datasetu. Po prijatí požiadavky server z prijatých informácií vygeneruje požiadavku na mongoDB databázu. Záznamy z databázy sú spracované a odoslané mobilnej aplikácii vo forme množiny slovníkov, z ktorých každý slovník má dva údaje kľúč-hodnota. Tie predstavujú hodnoty osí x a y.

3.4 Nasadenie systému

V nasledujúcej časti je opísaný postup nasadenia systému z pohľadu mobilnej aplikácie a serverovej časti. V závere časti sú zachytené všetky komponenty z pohľadu systému ako celku a ich vzájomné prepojenie.

3.4.1 Serverová časť

Nasadenie Node.js aplikácie do produkčného prostredia je realizované pomocou terminálu. Pred nasadením projektu do produkčného prostredia je potrebné si nainštalovať Google Cloud Shell a uistiť sa, že hlavný súbor v ktorom sa spúšťa serverová aplikácia sa nazýva `app.js`. To z toho dôvodu že Google po nasadení do produkčného prostredia spúšta v produkčnom prostredí súbor `app.js`. Nasadenie zdrojového kódu je realizované príkazom `gcloud app deplay --project id_projektu`. Nasadenie aplikácie trvá niekoľko minút, po ktorých je aplikácia dostupná na URL

[19] počet dní dočasnej pracovnej neschopnosti, doba poberania invalidného dôchodku tretieho stupňa, doba po ktorú je uchádzač o zamestnanie v evidencii pracovného úradu alebo doba štúdia po dobu 6 rokov po dosiahnutí 18 rokov

[20] `http://www.cssz.cz/cz/duchodove-pojisteni/duchodova-kalkulacka/kalkulacka-pro-orientacni-vypocet-starobniho-duchodu.htm`

`https://id_projektu.appspot.com`. Konkrétne URL na ktorej je dostupná serverová časť systému je špecifikované v súbore `readme.txt`.
Nasadenie databázového serveru prebieha pomocou webovej konzole. Užívateľ zvolí verziu databázy mongoDB, lokalizáciu serveru a nadefinuje požadované počiatočné kolekcie. Webová služba sa následne postará o celú konfiguráciu a nasadenie databázy. Po úspešnom nasadení sú v konzole zobrazené prístupové údaje k databáze.

3.4.2 Mobilná aplikácia

Nasadenie mobilnej aplikácie na App Store je umožnené iba majiteľom vývojárskeho účtu Apple. Každá mobilná aplikácia, ktorá je na App Store musí byť podpísaná certifikátom pre distribúciu aplikácií, ktorý vydala spoločnosť Apple. Tento certifikát je umožnený vygenerovať majiteľom vývojárskeho účtu. Proces nasadenia začína vytvorením archívu aplikácie. Archív obsahuje vytvorenú aplikáciu a `.DSYM` súbory, ktoré sú potrebné pre dekódovanie hlásení o páde aplikácie. Následne pomocou webovej stránky iTunesConnect je vytvorená nová aplikácia. Aplikácia vyžaduje špecifikovať unikátny identifikátor balíka (angl. „bundle id“), názov a popis aplikácie, kľúčové slová, cenu aplikácie, kontaktné údaje autora aplikácie a vytvoriť ikonu aplikácie a minimálne tri obrázky presne stanovených rozmerov pre každú podporovanú platformu. Tieto obrázky sú následne zobrazené v App Store. Po vyplnení a nahratí všetkých potrebných údajov je možné aplikáciu poslať spoločnosti Apple na preskúmanie a schválenie. Medzi možné dôvody zamientutia aplikácie patrí: pády a chyby aplikácie, neštandardné užívateľské rozhranie, nefunkčné odkazy, nekompletné informácie, reklamy, opakované podanie podobnej aplikácie, zavádzanie užívateľov a mnoho daľších dôvodov[21]. Implementovaná mobilná aplikácia prešla kontrolou Apple a je dostupná na `https://itunes.apple.com/us/app/penzee/id1375326141?l=sk&ls=1&mt=8`.
Výsledná mobilná aplikácia, ktorá je nasadená na App Store sa nazýva `Penzee`.

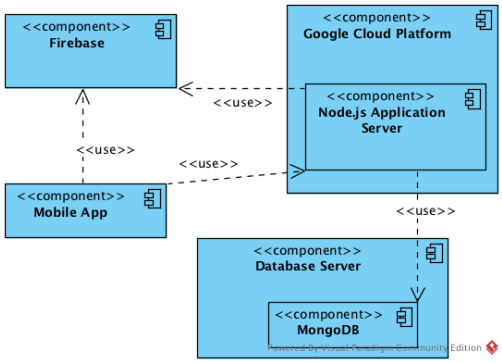
3.4.3 Výsledný systém

V nasledujúcej časti sú zobrané diagramy, ktoré znázorňujú komponenty celého systému a ich vzájomné prepojenie.

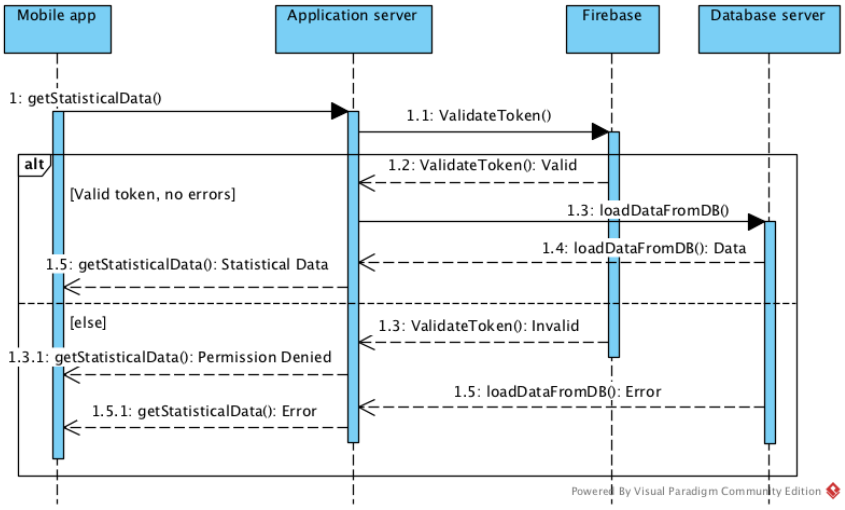
Diagram komponent slúži na znázornenie a zjednodušenie pochopenia fungovania celého systému. Diagram zachytáva komponenty výsledného systému a ich vzájomné vzťahy. Sekvenčný diagram znázorňuje získavanie štatistických dát zo serverovej časti. Tento diagram ilustruje všeobecný postup spolupráce komponent v rámci systému pri spacovávaní ľubovoľnej požiadavky. Mobilná aplikácia odošle požiadavku na server. Server pomocou Firebase zvaliduje token. V prípade validného tokenu je požiadavka spracovaná, prevedená a je odoslaná odpoveď mobilnej aplikácii. Pri prevedení požiadavky môže serverová

[21] `https://developer.apple.com/app-store/review/rejections/`

aplikácia podľa potreby komunikovať a získavať dáta z databáze. V prípade, že požiadavka neobsahuje validný token, server nepokračuje v ďalšom spracovaní požiadavky. Tým je zabezpečené využívanie servervových služieb výhradne autorizovanými užívateľmi.



Obr. 3.9: Diagram komponent výsledného systému



Obr. 3.10: Sekvenčný diagram znázorňujúci získavanie štatistických dát

Kapitola 4

Testovanie

V nasledujúcej kapitole je opísaný testovací proces systému, ktorý sa skladá z viacerých druhov testov. Testovanie aplikácie je realizované počas celého vývoja systému a aj po jeho skončení.

4.1 Programátorské testy

Programátorské testy sú prevedené programátorom ihneď po vytvorení kódu. Tento test prebieha na úrovni zdrojového kódu samotným programátorom. Pokiaľ testovanie nie je dôkladné, chybu odhalí až tester a následne ju nahlási programátorovi. Tým narastá réžia a s ňou spojené náklady[47].
Pri implementácii mobilnej aplikácií a serverovej časti bol kladený veľký dôraz na programátorské testy. Cieľom testov bolo zaistiť čo najmenej chýb v implementácii a tým čo najviac uľahčiť prácu testerom. Programátorské testy boli realizované pri implementácii mobilnej aplikácie ako aj serverovej časti.

4.2 Unit testy

Unit test je kus kódu, ktorý volá ďalší kus kódu a kontroluje správnosť predpokladaného správania. Ak sa predpoklad ukáže ako nesprávny, unit test vypíše chybové hlásenie. Unit testy testujú jednotlivé triedy a ich implementované metódy[47].
Testovanie má za účel overiť funkčnosť implementovaných tried na rôzne vstupy. V testovaní sú použité aj nesprávne vstupy, ktoré by mohli zapríčiniť nesprávne fungovanie alebo pád aplikácie. Unit testy mobilnej aplikácie sú napísané pomocou frameworku XCTest[22]. XCtest testovací framework od spoločnosti Apple, ktorý je automaticky integrovaný do vývojového prostredia Xcode. Testy sú zamerané hlavne na funcionalitu jednotlivých viewmodel a manažér objektov. Unit testy servrovej časti sú implementované pomocou

[22] `https://developer.apple.com/documentation/xctest`

testovacích frameworkov chai[23] a mocha[24]. Testy serverovej časti sú zamerané na správnosť ponúkanej funkcionality, analytických výpočtov, validáciu dát a autorizáciu užívateľov. Testy sú spolu zo zdrojovými kódmi na pribalenom CD v zložkách `/src/server/tests` a `/src/mobile/MyPensionTests`.

4.3 Testovanie API

Testovanie aplikačného rozhrania serverovej časti je nevyhnutné počas celej doby implementácie. Serverová časť bola testovaná na lokálnom stroji ale aj po nasadení na server do produkčného prostredia. Pre testovanie API sú vytvorené automatické testy v Node.js s využitím knižnice chai-http[25], ktorá umožňuje simulovať HTTP požiadavky na server. Testované sú hlavne návratové kódy, správna štuktúra vstupných a výstupných dát, existencia povinných hlavičiek, správny formát posielaných správ a podobne.
Testovanie API prebiahalo okrem automatických testov aj pomocou softwarových nástrojov Apiary a Postman. Postman je vývojové prostredie pre tvorbu API. Postman umožňuje vytvárať automatizované testy, dokumentáciu, monitorovať alebo debugovať aplikáciu. Pomocou nástrojov Apiary a Postman boli manuálne zasielané HTTP požiadavky na server s cieľom odhaliť a odladiť chyby systému. Kontrolované boli prevažne návratové HTTP stavové kódy a telá serverových odpovedí. Zdrojové kódy testov sú na pribalenom CD v zložke `/src/server/tests`.

4.4 Užívateľské testy

Užívateľské testovanie testuje hotovú aplikáciu ako celok a overuje ju z pohľadu užívateľa. Tento druh testovaní sa zameriava prevažne na funkčné požiadavky systému a užívateľskú prívetivosť aplikácie. Testovanie môže prebiehať vo viacerých kolách, v ktorých sú nájdené chyby a nedostatky opravené a následne opäť testované[47].
Pred samotným testovaním aplikácie sú vytvorené testovacie scenáre na základe prípadov použitia. Cieľom testovania bolo zistiť, či sú užívatelia schopní intuitívneho ovládania aplikácie a vedia využiť všetky funkcionality. Na záver testovania boli zozbierané užívateľské poznatky a pripomienky k aplikácii. Aplikácia bola testovaná užívateľmi rôznych vekových kategórii, s cieľom zaistiť čo najväčšiu výpovednú hodnotu testovania. Testovanie odhalilo, že niektorí užívatelia nepochopili textovým popisom pri vypĺňaní formulárov. Iní považovali funkciu presného výpočtu dôchodku za zdĺhavú, nakoľko vyžaduje príliš veľké množstvo informácií. Textové popisky aplikácie boli na základe

[23] `https://github.com/chaijs/chai`
[24] `https://mochajs.org`
[25] `https://github.com/chaijs/chai-http`

týchto poznatkov modifikované. Výpočet presnej výšky dôchodku ostal nezmenený, nakoľko sa jedná o jednu z požiadaviek systému. V budúcnosti je však na zvážení túto funkcionalitu upraviť alebo úplne odstrániť, ak sa ukáže ako málo využívaná.

4.5 Reportovanie pádov aplikácie

Ďalším užitočným nástrojom pri testovaní aplikácie je monitorovanie a reportovanie pádu aplikácie. Pre monitorovanie pádov aplikácii bola nasadená služba Firebase Crashlytics, ktorá je integrovaná do Firebase. Záznamy o pádoch aplikácie obsahujú napríklad čas a počet pádov, verziu systému a aplikácie, ale aj výpis zásobníku volaní[26]. Programátor môže teda identifikovať v ktorom mieste aplikácia spadla. Firebase Crashlytics beží aj naďalej v produkčnom prostredí čím, je zabezpečené neustále monitorovanie funkčnosti aplikácie.

[26] `https://firebase.google.com/docs/crashlytics/`

Záver

Cieľom diplomovej práce bola analýza, návrh, tvorba drôteného modelu, implemetácia a testovanie mobilnej aplikácie pre operačný systém iOS, ktorá uľahčí užívateľom zisťovanie informácií o dôchodkovom systéme v Českej republike. Práca si tiež kládla za cieľ analyzovať, navrhnúť, implementovať a otestovať vlastnú serverovú časť, ktorá bude vyťažovať dáta zo zdrojov otvorených dát a poskytovať ich cez štandardizované aplikačné rozhranie mobilnej aplikácii. Autor považuje cieľ práce za splnený.
Na začiatku práce bola vytvorená analýza existujúcich riešení a dátových zdrojov. Následne bola navrhnutá funkcionalita mobilnej aplikácie, ktorá bola validovaná formou užívateľského dotazníku. Na základe výstupov z dotazníku bol vytvorený drôtený model aplikácie, ktorý bol spolu s funkcionalitou aplikácie opätovne iteratívne validovaný skupinou testerov. Diplomová práca pokračovala analýzou požiadaviek a technických riešení serverovej časti, výberom technických riešení spolu s návrhom celkovej architektúry systému. Na základe návrhu a analýzy prebehla implementácia mobilnej aplikácie a serverovej časti systému. Po implementácii, ale aj po jej skončení, boli jednotlivé aplikácie otestované a odstránené nedostatky. Výsledkom implementačnej časti diplomovej práce je serverová časť a mobilná aplikácia pre zariadenia s operačným systémom iOS. Serverová časť je nasadená do produkčného prostredia na aplikačný server. Mobilná aplikácia je publikovaná v obchode App Store pod názvom `Penzee`.

Budúca práca

Hlavným krokom do budúcnosti je zbieranie spätnej väzby od užívateľov spolu s monitorovaním aplikácie. Následne rozšíriť aplikáciu o funkcionality a informácie z ďalších, zahraničných dôchodkových systémov mimo Českej republiku. S tým prichádza potreba aplikáciu preložiť do ďalších jazykov, čo by mohlo viesť k väčšej užívateľskej základni. Autor vidí priestor na zlepšenie mobilnej aplikácie v jej grafickom návrhu. Nakoľko autor nemá profesionálne grafické cítenie ani zručnosti, musí byť zmena konzultovaná s profesionálnym

grafikom. Na základe konzultácie s grafikom bude v budúcnosti upravený grafický vzhľad mobilnej aplikácie.

Literatúra

[1] Dine4Fit, a.: Finanční kalkulačky [online]. [cit. 2018-04-16]. Dostupné z: https://goo.gl/MM8Shw

[2] Dine4Fit, a.: Finanční kalkulačky - výpočet čisté mzdy 2018 [online]. [cit. 2018-04-16]. Dostupné z: https://play.google.com/store/apps/details?id=cz.psc.android.financnikalkulacky&hl=cs

[3] Púca: 64 - The Pension Calculator [online]. [cit. 2018-04-16]. Dostupné z: https://itunes.apple.com/us/app/64-the-pension-calculator/id808093095?mt=8

[4] s.r.o., F. S.: Financie.online [online]. [cit. 2018-04-16]. Dostupné z: https://itunes.apple.com/sk/app/financie-online/id1244667204?l=sk&mt=8

[5] autor, N.: Výpočet.cz [online]. [cit. 2018-04-16]. Dostupné z: https://www.vypocet.cz

[6] Česká správa sociálního zabezpečení: Česká správa sociálního zabezpečení [online]. [cit. 2018-04-16]. Dostupné z: https://data.cssz.cz/web/otevrena-data/

[7] Česká správa sociálního zabezpečení: Otevřená data - Podmínky užití [online]. [cit. 2018-04-16]. Dostupné z: https://data.cssz.cz/web/otevrena-data/podminky-uziti

[8] vnitra, M.: Portál veřejné správy [online]. [cit. 2018-04-16]. Dostupné z: https://portal.gov.cz

[9] Jacobson, I.: Use-Case 2.0, The Guide to Succeeding with Use Cases. [cit. 2018-04-16]. Dostupné z: https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf

[10] Kanisová, H.; Muller, M.: *UML srozumitelně*. Computer press, 2006.

[11] SAP: What is a Wireframe? [cit. 2018-04-16]. Dostupné z: `https://experience.sap.com/basics/post-143/`

[12] Ambler, S. W.: *The object primer: Agile model-driven development with UML 2.0*. Cambridge University Press, 2004.

[13] Leffinngwell, D.: Use-Case 2.0, Domain modeling. [cit. 2018-04-16]. Dostupné z: `https://www.scaledagileframework.com/domain-modeling/`

[14] Inc., A.: Model-View-Controller. [cit. 2018-04-16]. Dostupné z: `https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html`

[15] Apple: Cocoa version of MVC as a compound design pattern. [cit. 2018-04-16]. Dostupné z: `https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Art/cocoa_mvc.gif`

[16] Garofalo, R.: *Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern*. Microsoft Press, 2011.

[17] Online, S. B.: MVVM pattern. [cit. 2018-04-16]. Dostupné z: `https://www.safaribooksonline.com/library/view/learning-javascript-design/9781449334840/httpatomoreillycomsourceoreillyimages1547825.png`

[18] Khanlou, S.: Coordinators Redux. [cit. 2018-04-16]. Dostupné z: `http://khanlou.com/2015/10/coordinators-redux/`

[19] Gerti Kappel, B. P. (editor): *Web Engineering, The Discipline of Systematic Development of Web Applications*. John Wiley and Sons, 2006.

[20] Mumbaikar, S.; Padiya, P.; aj.: Web services based on soap and rest principles. *International Journal of Scientific and Research Publications*, ročník 3, č. 5, 2013: s. 1–4.

[21] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: Extensible markup language (XML). *World Wide Web Journal*, ročník 2, č. 4, 1997: s. 27–66.

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

[23] Pezoa, F.; Reutter, J. L.; Suarez, F.; aj.: Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2016, s. 263–273.

[24] Nurseitov, N.; Paulson, M.; Reynolds, R.; aj.: Comparison of JSON and XML data interchange formats: a case study. *Caine*, ročník 9, 2009: s. 157–162.

[25] Eriksson, M.; Hallberg, V.: Comparison between JSON and YAML for data serialization. *The School of Computer Science and Engineering Royal Institute of Technology*, 2011.

[26] Ben-Kiki, O.; Evans, C.; Ingerson, B.: Yaml ain't markup language (yaml™) version 1.1. *yaml. org, Tech. Rep*, 2005: str. 23.

[27] solid IT gmbh: DB-Engines Ranking. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/ranking`

[28] solid IT gmbh: Relational DBMS. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/article/Relational+DBMS?ref=RDBMS`

[29] Cattell, R.: Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, ročník 39, č. 4, 2011: s. 12–27.

[30] solid IT gmbh: Key-value Stores. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/article/Key-value+Stores`

[31] solid IT gmbh: Document Stores. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/article/Document+Stores`

[32] solid IT gmbh: Graph DBMS. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/article/Graph+DBMS`

[33] solid IT gmbh: Wide Column Stores. [cit. 2018-04-16]. Dostupné z: `https://db-engines.com/en/article/Wide+Column+Stores`

[34] software BV, T.: TIOBE Index. [cit. 2018-04-16]. Dostupné z: `https://www.tiobe.com/tiobe-index/`

[35] Gosling, J.: *The Java language specification*. Addison-Wesley Professional, 2000.

[36] Lutz, M.: *Learning Python: Powerful Object-Oriented Programming*. Ö'Reilly Media, Inc.", 2013.

[37] Bächle, M.; Kirchberg, P.: Ruby on rails. *IEEE software*, ročník 24, č. 6, 2007.

[38] Flanagan, D.: *JavaScript: the definitive guide*. Ö'Reilly Media, Inc.", 2006.

[39] Tilkov, S.; Vinoski, S.: Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, ročník 14, č. 6, 2010: s. 80–83.

[40] Lerdorf, R.; Tatroe, K.; MacIntyre, P.: *Programming Php*. Ö'Reilly Media, Inc.", 2006.

[41] Inc., A.: Xcode IDE. [cit. 2018-04-16]. Dostupné z: `https://developer.apple.com/xcode/features/`

[42] s.r.o, J.: WebStorm. [cit. 2018-04-16]. Dostupné z: `https://www.jetbrains.com/webstorm/features/`

[43] Loeliger, J.; McCullough, M.: *Version Control with Git: Powerful tools and techniques for collaborative software development*. Ö'Reilly Media, Inc.", 2012.

[44] Halbwachs, N.: *Synchronous programming of reactive systems*, ročník 215. Springer Science & Business Media, 2013.

[45] Realm: Realm Database. [cit. 2018-04-16]. Dostupné z: `https://realm.io/docs`

[46] Mardan, A.: *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[47] Patton, R.: *Testování softwaru*. Computer press, 2002.

Dodatok **A**

Zoznam použitých skratiek

GUI Graphical User Interface

XML Extensible Markup Language

RDF/TriG Resource Description Framework

CSV Comma Separated Values

JSON JavaScript Object Notation

XLS Excel Spreadsheet

GIF Graphics Interchange Format

UI User Interface

UX User Experience

DOC Document

JPEG Document

SGML Standard Generalized Markup Language

YAML Ain't Markup Language

ACID Atomicity, Consistency, Isolation, Durability

PHP Hypertext Preprocessor

DRY Don't Repeat Yourself

CoC Convention over Configuration

API Application Programming Interface

CRUD Create, Read, Update, Delete

DSYM Debug Symbols

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `mobile` ..... zdrojové kódy mobilnej aplikácie
  - `server` ..... zdrojové kódy serverovej aplikácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `thesis.pdf` ..... text práce vo formáte PDF
- `attachement` ..... priložené elektronické dokumenty
  - `wireframes` ..... drôtené modely aplikácie
  - `survey` ..... výstupné materiály z užívateľského dotazníku