FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Integrace aplikací na společný dashboard
Student:	Simona Kurňavová
Vedoucí:	Ing. Vojtěch Jirkovský
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 vytvořit platformu (dashboard) pro integraci webových aplikací a služeb (widgetů) do jedné obrazovky a vytvořit pro ni několik vzorových widgetů.

1. Proveďte rešerši existujících služeb s obdobným cílem.
2. Navrhněte vlastní řešení s přihlédnutím ke kladům a záporům zjištěným v rešerši a s důrazem na snadnou rozšiřitelnost.
3. Implementujte dashboard a alespoň 3 widgety, které do něj bude možné začlenit. Pro implementaci backendu použijte framework Django, pro frontend framework Angular.
4. Všechny implementované části řádně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 19. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Integrácia aplikácií na spoločný dashboard

Simona Kurňavová

Katedra Softvérového inženýrství
Vedúci práce: Ing. Vojtěch Jirkovský

11. mája 2018

Poďakovanie

Rada by som poďakovala všetkým ktorí mi ochotne pomáhali s touto bakalárskou prácou. Predovšetkým poďakovanie patrí vedúcemu práce Ing. Vojtěchovi Jirkovskému, za odborné rady a dohľad pri písaní práce. Tiež by som chcela poďakovať svojej rodine a priateľom za podporu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracovala samostatne, a že som 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. V súlade s ustanovením § 46 odst. 6 tohto zákona, týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, vrátane všetkých počítačových programov, ktoré sú jej súčasťou, alebo prílohou, taktiež dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý nezhoršuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného, či spracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu, a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom, a v zrovnateľnom rozsahu ako je sprístupnený zdrojový kód Diela.

V Prahe 11. mája 2018 .....

Odkaz na túto prácu

Kurňavová, Simona. *Integrácia aplikácií na spoločný dashboard.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Táto práca sa zaoberá analýzou súčasných riešení problematiky správy väčšieho množstva aplikácií z užívateľského pohľadu, návrhom vhodnejšieho a vylepšeného riešenia a jeho následnou implementáciou.

Cieľom práce je správne navrhnúť systém, ktorý spája niekoľko aplikácií do jedného dashboardu, jeho implementácia a otestovanie funkcionalít. Zameriava sa aj na prívetivé užívateľské rozhranie a intuitívnu prácu so systémom.

Pri vývoji sa využívajú framework technológie Django a Angular 5 pre dve oddelené aplikácie predstavujúce klienta a server systému, prepojené pomocou REST API.

Kľúčové slová integrácia aplikácií, dashboard, widget

Abstract

This thesis focuses on analysis of the present technologies dedicated to the management of many applications from the user point of view. It provides design and architecture of better and more convenient solution and its implementation.

The aim of this thesis is to design system, which connects bigger amount of applications into single dashboard, implement and test the functions of the system. It is also focused to create amiable user interface and intuitive interaction with the system.

Technologies used during development are frameworks Django and Angular 5 for the two separate applications representing client and server side of the system, connected via REST API.

Keywords integration of aplications, dashboard, widget

Obsah

Zoznam obrázkov

Zoznam kódov

Úvod

V súčasnosti existuje veľké množstvo aplikácií, ktoré ľudia využívajú k rôznym činnostiam. Najmä v pracovnej sfére toto množstvo aplikácií narastá so zvyšujúcou sa automatizáciou jednotlivých činností – písanie a archivácia dokumentov, odosielanie a prijímanie správ, organizácia tímov a podobne. Správa týchto aplikácií je náročná ako pre užívateľa, tak pre výkon zariadení.

Táto práca chce poskytnúť vylepšenie súčasných riešení tejto správy a zjednotiť aplikácie do dashboardu, ktorý poskytne pracovnú plochu na ukladanie a interakciu s aplikáciami. Zámerom je znížiť množstvo otvorených okien a kariet prehliadača, zjednodušiť manipuláciu s aplikáciami a zjednotiť ich správu. Toto riešenie bude realizované formou webovej aplikácie, čo zabezpečí multiplatformovosť.

Práca obsahuje analýzu súčasných technológií zameraných na túto problematiku, a na ich základe navrhuje systém spájajúci vhodné funkcie existujúcich riešení. Snaží sa naviac o elimináciu ich nedostatkov v čo najväčšej miere. Obsahuje implementáciu daného systému a tiež sa zameriava na otestovanie jednotlivých funkcionalít.

Výsledná aplikácia je určená pre rôzne druhy užívateľov. Od bežného užívateľa po skúsených vývojárov, ktorým bude naviac poskytovaná možnosť implementácie vlastnej aplikácie a jej integrácia do systému.

Kapitola 1

Ciele práce

Cieľom práce je tvorba systému na integráciu aplikácií umožňujúci správu a interakciu s aplikáciami, so zameraním na prívetivé užívateľské rozhranie, multiplatformovosť a jednoduché rozšírenie. Rovnako je zámerom vytvorenie troch ukážkových aplikácií a ich integrácia do systému.

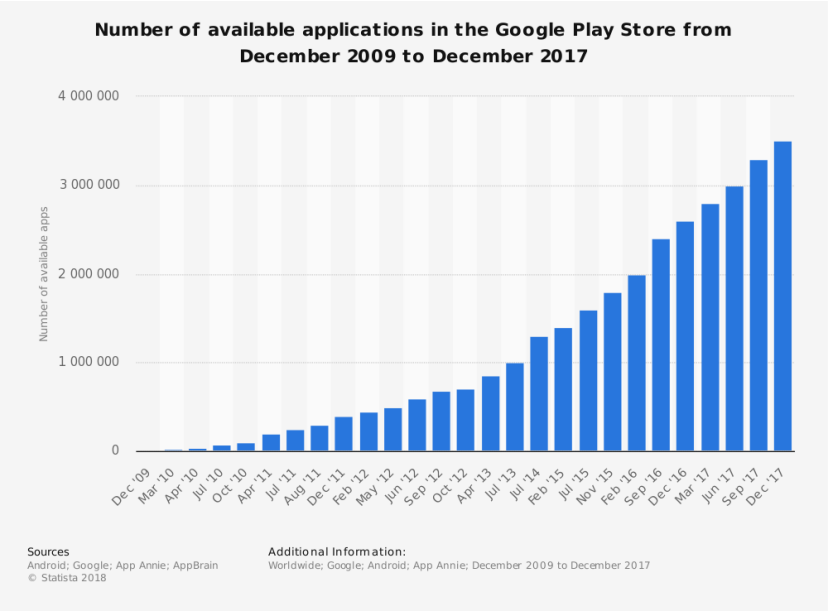
Kapitola 2

Analýza

Táto kapitola sa bude zaoberať prieskumom súčasných aplikácií a technológií, nejakým spôsobom integrujúcich alebo spájajúcich aplikácie, za účelom zvýšenie pohodlia používania a zjednodušenie prístupu.

Cieľom analýzy je porozumieť súčasným technológiám, zhodnotiť ich výhody a nevýhody a na základe získaných informácií špecifikovať požiadavky pre vhodnejšie riešenie.

2.1 Nárast množstva aplikácií na trhu



Obr. 2.1: Množstvo aplikácií na Google Play v rokoch 2009-2017 [27]

Súčasný trh je plný rozličných aplikácií. Od sociálnych sietí, až po aplikácie uľahčujúce pracovné postupy. Vzhľadom na aktuálny stav rozrastajúceho sa trhu je bezpečné predpokladať, že množstvo aplikácií sa bude zvyšovať. Na tento nárast poukazuje aj štatistika množstva aplikácii na Google Play zobrazená na obrázku 2.1. V roku 2009 bolo množstvo aplikácii v tomto obchode okolo 16 tisíc podľa portálu Statista [27]. V súčasnosti je to vyše 3,7 miliónov podľa štatistík AppBrain [1].

Podobne je na tom aj množstvo užívateľov aplikácií. Podľa portálu Emarketer množstvo užívateľov sociálnych sietí stále narastá. V roku 2017 to bolo 2,46 miliárd užívateľov, čo v porovnaní s rokom 2016 znamená nárast o 8.2 % [5].

2.2 Súčasné technológie integrujúce aplikácie

Predovšetkým v pracovnej sfére, či už v oblasti vývoja softvéru, marketingu, ekonomiky, alebo mnohých ďalších odvetví, sa využíva niekoľko rozličných aplikácií súčasne. Toto množstvo aplikácií vyúsťuje do zástupov otvorených okien a kariet prehliadača, a následnú neprehľadnosť a zníženie výkonu zariadení.

Problematika veľkého množstva aplikácií a účtov je dobre známa, a je niekoľko technológií ktoré sa pokúšajú užívateľom túto situáciu zjednodušiť a poskytujú správu väčšieho množstva účtov a aplikácií. Predstavím len niekoľko, podľa môjho názoru najvýznamnejších, technológií. Existuje však mnoho ďalších, ktoré takisto predstavujú podporu pre danú problematiku.

2.2.1 Vzájomné prepojenie aplikácií

Jedným so spôsobov ako zjednodušiť používanie niekoľkých aplikácií súčasne je vzájomne ich prepojiť. To je možné dosiahnuť pomocou nasledujúcich technológií, ktoré bližšie popíšem.

2.2.1.1 Zjednotenie účtov

Mnoho aplikácií implementuje možnosť registrácie a následného prihlásenia sa pod účtom inej aplikácie (napríklad Google alebo Facebook). Najväčšou výhodou toho je, že aplikácia preberá už raz užívateľom zadané prihlasovacie údaje z inej aplikácie, čím sa urýchľuje registrácia a budúci prístup k aplikácii. Znižuje sa množstvo údajov ktoré si užívateľ musí pamätať. Nerieši to však všetky problémy. Stále sú to samostatné aplikácie, len sa vzájomne odkazujú na rovnaké prihlasovacie údaje.

2.2.1.2 Prepojenie sociálnych sietí

Ďalšiu technológiu, alebo skôr funkcionalitu samotných aplikácií, ktorú by som rada zmienila je možnosť prepojenia účtov rôznych sociálnych sietí medzi se-

bou. Túto funkciu majú sociálne siete ako napríklad Facebook, ktorý umožňuje pridanie účtov aplikácií ako sú Instagram alebo Twitter do profilu užívateľa. Táto funkcia poskytuje napríklad možnosť zdieľať príspevky pridané na jednej zo sociálnych sietí na druhú. Užívateľ môže tak jednoducho pridať jeden príspevok na niekoľko sietí bez nutnosti kopírovania.

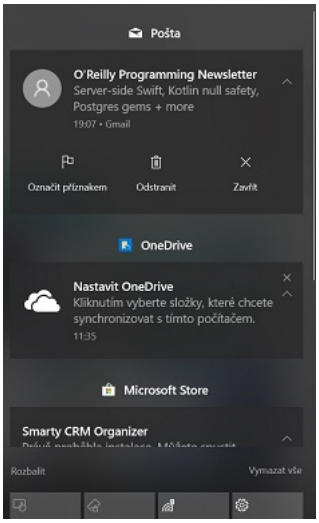
Znovu však narážame na problém s obmedzenou funkcionalitou prepojenej aplikácie. Rovnako sa táto funkcia nevzťahuje na veľké množstvo aplikácií, a často je možné ju pridať len samotnými vývojármi aplikácie s rozsiahlejším zásahom do aplikácie.

2.2.2 Integrácia aplikácií v rámci operačných systémov

Ďalšou skupinou technológií sú funkcie a možnosti operačných systémov, ktoré umožňujú prepojenie s ďalšími aplikáciami.

2.2.2.1 Notifikácie

Takmer každý novodobý operačný systém (ďalej len OS) umožňuje odoberanie notifikácií od rôznych aplikácií. Najznámejšími príkladmi sú funkcie MS Windows, MacOS, ale aj mnohých Linuxových distribúcií. Táto funkcia sa aktivuje pridaním účtu k zvolenej aplikácii, od ktorej chce užívateľ dostávať notifikácie. Notifikácie predstavujú oznámenia o správach, novinkách a akciách v rámci aplikácie. OS si pamätá jednotlivé účty, ktoré sú väčšinou viazané k nainštalovaným aplikáciám v systéme. Na základe týchto poskytnutých informácií je OS schopný zobrazovať aktuálne notifikácie užívateľovi vhodnou formou. Tou je najčastejšie zoznam v bočnom paneli, vyskakovacie okná, prípadne iné spôsoby zobrazení.



Obr. 2.2: Windows notifikácie

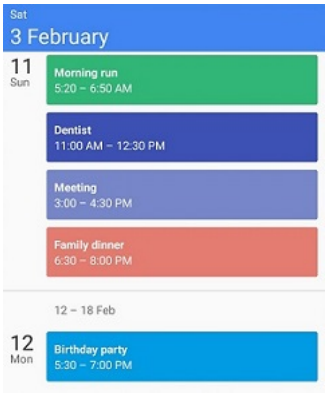
Najväčšou výhodou tohto riešenia je, že všetky „diania“ v aplikáciách užívateľ vidí pohromade a je jednoduché sa v nich orientovať. Má možnosť bližšieho preskúmania notifikácie, prípadne roztvorenia samotnej aplikácie. Je včas informovaný o stave v aplikáciách, a nie je nutné im venovať nepretržitú pozornosť, chodiť ich pravidelne kontrolovať, alebo ich udržiavať otvorené na pozadí.

Na obrázku 2.2 je zobrazená ukážka Windows notifikácií v bočnom paneli.

Toto riešenie má však aj pár nevýhod. Keďže sú to funkcionality samotných OS nie sú prístupné na iných zariadeniach a platformách. Čo znamená, že k notifikáciám nastaveným na MS Windows užívateľ nebude mať prístup v rámci MacOS, alebo napríklad na mobilnom telefóne s Android OS. Takisto množstvo aplikácií ktoré je možné takto prepojiť s určitým OS je obmedzené. Väčšina umožňuje prístup len k najpoužívanejším aplikáciám ako sú napríklad Facebook, Gmail, Twitter a podobne. V prípade využívania menšej a menej známej aplikácie užívateľ nemá možnosť túto aplikáciu pripojiť.

2.2.2.2 Android dashboard

Ako som spomínala v predchádzajúcej sekcii, väčšina operačných systémov poskytuje notifikácie. Android okrem notifikácií ponúka aj dashboard s widgetami. Widgety sú miniaplikácie, ktoré sú istým spôsobom sprostredkovateľom funkcionalít a informácií poskytovaných určitou aplikáciou, na ktorú sú naviazané. Android dashboard obsahuje maticu, na ktorú si môže užívateľ ukladať jednotlivé widgety podľa potreby. Má možnosť rozhodovať o ich rozmiestnení, a v mnohých prípadoch aj veľkosti (pokiaľ to daná aplikácia povoľuje).



Obr. 2.3: Google Calendar Android widget

Vhodným príkladom widgetu je napríklad Google Calendar widget (obrázok 2.3), ktorý preberá dáta a funkcie od aplikácie Google Calendar a umožňuje zobrazenie prehľadu týždňa alebo mesiaca na ploche Android zariadenia.

Hlavnou nevýhodou tejto technológie je, že nie každá aplikácia má svoj widget, prípadne má len obmedzené funkcionality. Na druhej strane, jej najvý-

raznejšou výhodou podľa môjho názoru je vynikajúca prehľadnosť, jednoduchá organizácia a intuitívnosť pre užívateľa.

2.3 Intuitívnosť a jednoduchosť súčasných technológií

Dôležitou otázkou pri porovnaní technológií zmienených v predchádzajúcej sekcii je, nakoľko je používanie týchto technológií zrozumiteľné a jednoduché pre bežného užívateľa. Predstavila som niekoľko typov UI: webové rozhranie, notifikácie a dashboard. Každé z nich má svoje výhody a nevýhody a častokrát záleží najmä na účele a preferencii užívateľa. Webové rozhranie a dashboard sú schopné poskytnúť viac informácií ako napríklad notifikácie. Tie sú však preferované v situáciách keď užívateľ nemôže mať neustále otvorený prehliadač na rovnakej karte, ale potrebuje byť informovaný o zmenách v aplikáciách. Nemôžem povedať že existuje práve jedno správne užívateľské rozhranie ktoré bude vyhovovať všetkým. Práve preto si myslím, že najužitočnejšie prevedenie UI je také, ktoré užívateľovi umožňuje čo najviac možností úprav a prispôsobení prostredia. Čo v tomto prípade najviac spĺňa Android dashboard.

2.4 Zhrnutie

Z tejto analýzy môžem usúdiť, že existuje niekoľko riešení danej problematiky. Vo všetkých spomenutých prípadoch sa však objavuje problém s multiplatformovosťou. Väčšina týchto technológií je viazaná na operačné systémy, takže užívateľ nedokáže zdieľať nastavenia notifikácií a widgetov medzi rôznymi systémami a zariadeniami. Mnohokrát chýba rozšíriteľnosť funkcionalít a pridanie ďalších aplikácií, pre ktoré ešte nie je podpora v oblasti danej technológie. Mnoho ľudí využíva niekoľko technológií súčasne, a to hlavne v prípade využívania viacerých zariadení s rôznymi OS. Týmto nedostatkom by som sa v rámci môjho systému rada vyhla.

Postrehla som však niekoľko vlastností, ktoré by podľa môjho názoru bolo vhodné prebrať a využiť. Android dashboard a možnosť správy aplikácií na ploche je podľa môjho názoru prívetivé riešenie zo strany užívateľského rozhrania a umožňuje jednoduchú orientáciu pre užívateľa. Rovnako tak riešenie systému formou webovej aplikácie je jedno z najistejších zaručení multiplatformovosti a jednoduchého prístupu bez nutnosti inštalácie.

2.5 Špecifikácia požiadaviek

Na základe predchádzajúcej analýzy už existujúcich technológií som zostavila požiadavky na systém, ktoré preberajú vhodné funkcionality v praxi použí-

vaných technológií a rozširujú ich o ďalšie, vhodnejšie a efektívnejšie riešenia problematiky.

Požiadavky boli zostavené v súlade s SRS (Software Requirment Specification) štandardom [10], ktorý hovorí o vhodných vlastnostiach požiadaviek.

2.5.1 Terminológia

Na začiatok špecifikujem pojmy ktoré budem v rámci systému využívať:

Dashboard Plocha na zobrazovanie a organizáciu widgetov.

Aplikácia Implementácia aplikácie

Widget Aplikácia nachádzajúca sa na dashboarde užívateľa s určitou veľkosťou, pozíciou, prípadne priradeným účtom (ak nejaký vyžaduje)

2.5.2 Obecné mimofunkčné požiadavky

- P1 Multiplatformovosť – podpora v rámci rôznych systémov a zariadení
- P2 Autentifikácia – nutnosť registrácie a prihlásenia sa
- P3 Rozšíriteľnosť systému – jednoduché pridanie ďalších funkcionalít

2.5.3 Obecné funkčné požiadavky

- P4 Hlavný komponent – interaktívny dashboard s widgetami
- P5 Nastavenia profilu – možnosti dodatočnej úpravy profilu
- P6 Správa účtov aplikácií – možnosť odobrania účtu (o pridanie sa budú starať samotné aplikácie)

2.5.4 Funkčné požiadavky na dashboard

- P7 Forma matice, do ktorej buniek bude možné vkladať widgety a prispôsobovať ich umiestnenie
- P8 Rozšíriteľnosť – jednoduché pridanie widgetu do systému
- P9 Dva módy:
  - Normálny: interakcia s widgetami a ich funkcionalitami, možnosť zväčšenia widgetu do modálneho okna, uzamknuté všetky možnosti úprav
  - Editovací: možnosť pridania, odobrania, premiestnenia alebo zmeny veľkosti widgetu na ploche dashboardu, uzamknutá možnosť interakcie s funkcionalitami widgetov

2.5.5 Požiadavky na widgety

- P10 Unifikované rozhranie jednotlivých widgetov
- P11 Možnosť zobrazenia zväčšenej verzie widgetu v modálnom okne

2.5.6 Požiadavky na GUI

- P12 Intuitívnosť – ľahká orientácia v systéme aj pre neskúseného užívateľa
- P13 Jednoduchosť – v rámci farebnej schémy a rozloženia elementov, minimálne množstvo grafických prvkov
- P14 Prehľadnosť dashboardu – dedikovať čo najväčší možný priestor samotnému dashboardu, vytvorenie čo najprívetivejšieho vzhľadu widgetov a editovacieho módu

2.5.7 Technologické požiadavky

- P15 Forma webovej aplikácie
- P16 Využitie frameworkov Django a Angular 5 – klient a server dve samostatné aplikácie komunikujúce cez REST API
- P17 Dokumentácia kódu
- P18 Tri funkčné ukážkové widgety integrované do systému

2.6 Technológie a návrhové vzory

V tejto sekcii bližšie rozoberiem technológie a vzory, ktoré budú pri vývoji využité. Ide o implementáciu webovej aplikácie, takže za týmto účelom budú použité webové frameworky. Klient a server budú tvoriť dve samostatné aplikácie, prepojené pomocou REST API.

2.6.1 Framework

Softvérový framework, alebo aplikačný rámec, je štruktúra navrhnutá k zjednodušeniu vývoja softvéru. Môže obsahovať podporné programy, definované konvencie, podporu pre návrhové vzory, rôzne pomocné nástroje a ďalšie funkcionality zefektívňujúce vývojový proces. Predstavuje istú kostru softvéru, zvyčajne zameranú na typ alebo časť aplikácie – webová aplikácia, testovanie softvéru, UI.

Vhodnými príkladmi sú napríklad Nette, Laravel, NodeJS alebo UI frameworky ako je Semantics UI.

2.6.2 Django

Django je open-source webový backend framework napísaný v jazyku Python. Obsahuje objektovo-relačné mapovanie, ktoré je sprostredkovateľom medzi triedami v Pythone a relačnou databázou. Umožňuje teda vytvárať tabuľky databázy formou modelov v Pythone. Poskytuje tiež vlastný server na vývoj a testovanie [4].

2.6.3 DRY

DRY (don't repeat yourself) je princíp programovania, ktorý hovorí o neopakovaní informácie. Definuje prístup k štruktúre dát tak, že sa v kóde nič nenachádza viac ako jedenkrát [28]. Django je jeden z frameworkov ktorý sa snaží práve o tento prístup pomocou preddefinovaných tried a pomocných funkcií.

2.6.4 Python

„*If the implementation is hard to explain, it's a bad idea.*“ (The Zen of Python [24])

Python je interpretovaný vysokoúrovňový jazyk s podporou objektového programovania, ale aj ďalších paradigmát. Má dynamické typovanie a automatickú správu pamäte. Používa minimalistický kód, čo umožňuje rýchle programovanie a obsahuje mnoho funkcií a knižníc na podporu efektívneho vývoja [25].

Využíva sa v mnohých oblastiach ako napríklad testovanie softvéru alebo webový vývoj. U mnohých Linuxových distribúcií je súčasťou základnej inštalácie.

2.6.5 MVC a MVT

MVC (Model View Controller) je často používaná softvérová architektúra, ktorá rozdeľuje dátovú časť, logiku a užívateľské rozhranie do troch častí, ktoré majú na seba čo najmenší vplyv [2].

- Model: Popisuje dáta a ich tvar.
- View: Užívateľské rozhranie, ktoré má priamu interakciu s užívateľom. Umožňuje mu pridanie alebo zmenu dát.
- Controller: Predstavuje logiku, spracovanie požiadaviek užívateľa, a stará sa o vykresľovanie View.

Existuje niekoľko poňatí tohto modelu. Často sa používa napríklad MVP – Model View Presenter. Django implementuje vlastnú architektúru podľa tohto modelu. Môžeme ju nazvať MVT – Model View Template. Táto architektúra neobsahuje Controller, pretože o to sa stará samotné Django [18].

- Model: Popisuje dáta a ich tvar.
- View: Obsahuje logiku aplikácie a predstavuje istého sprostredkovateľa medzi modelom a templatom.
- Template: Prezentačná vrstva, ktorá určuje výzor a štruktúru stránky.

2.6.6 Model

Vďaka objektovo-relačnému mapovaniu Django umožňuje jednoduché vytváranie databázy pomocou Modelov a bez databázového jazyka ako je SQL. Modely sa implementujú ako triedy jazyka Python, pričom dedia z triedy Model poskytovanou Djangom (Django documentation: Models [4]).

```
from django.db import models

class Note(models.Model):
    name = models.CharField(max_length=20)
    content = models.CharField(max_length=200)
```

Kód 2.1: Príklad Django Model

Príklad modelu 2.1 obsahuje definíciu entity Note (poznámka), ktorá má dva atribúty „name“ a „content“ so zadaným maximálnym množstvom znakov.

Django tiež poskytuje bohaté API na ukladanie objektov do databázy a následné dotazovanie.

2.6.7 View

Ako už bolo spomenuté, View obsahuje logiku aplikácie a rozhoduje čo sa s dátami urobí, a čo sa zobrazí užívateľovi. Nasledujúci príklad 2.2 definuje View, ktorý zobrazí zoznam poznámok. Využíva model Note a renderuje template (Django documentation: Writing views [4]).

```
from django.shortcuts import render

def note_list(request):
    notes = Note.objects.all()
    return render(request, 'note_list.html', {'notes': notes})
```

Kód 2.2: Príklad Django View

2.6.8 Template

Templates, alebo šablóny, sú dokumenty, ktoré obsahujú HTML tagy definujúce štruktúru stránky a syntax pridanú Djangom na vytváranie dynamického obsahu. Na to sa využíva aj šablónovací jazyk Jinja (Django documentation: Templates [4]).

2.6.9 REST

REST, z anglického „Representational State Transfer“, je architektúra rozhrania navrhnutá pre distribuované prostredie. Používa sa pre jednoduchý prístup k dátam. Prístup a manipulácia s dátami sú definované štyrmi operáciami CRUD: create (vytvoriť), read (prečítať), update (editovať) a delete (zmazať). Tieto operácie sú implementované pomocou odpovedajúcich metód HTTP protokolu.

Pri vývoji bude využitá REST architektúra na komunikáciu medzi dvomi frameworkami (Django a Angular). Je to jeden z najjednoduchších a najpoužívanejších spôsobov ako dosiahnuť vzájomnú komunikáciu. Toto API bude vytvorené pomocou rozšírenia Djanga – Django rest framework, ktorý poskytuje funkcionality na podporu REST služieb.

2.6.10 Django Rest Framework

Django Rest Framework (DRF), je technológia, ktorá obsahuje nástroje potrebné na implementáciu webového API Django aplikácie. Obsahuje balíčky OAuth1 a OAuth2 užitočné pre implementáciu autentifikácie užívateľov.

REST API sa realizuje pomocou serializérov, ktoré spracujú dáta do požadovaného formátu, a odosielajú sa pomocou špeciálnych View tried [3].

2.6.11 Tokenová autentifikácia

Alebo token based authentication, je mechanizmus pre autentifikáciu API požiadaviek. Využívajú sa na to tokeny, ktoré klient získa pri úspešnej autentifikácii.

Priebeh je nasledovný: Klient sa autentifikuje pomocou svojich užívateľských údajov (zvyčajne meno a heslo). Tieto údaje sa odošlú na server, ktorý vráti prístupový token. Tento token je unikátny a identifikuje klienta. Všetky ďalšie požiadavky na server sa odosielajú s tokenom v HTTP hlavičke, ktorý slúži ako náhrada prihlasovacích údajov. Token má z bezpečnostných dôvodov obmedzenú dobu platnosti, takže po určitom čase je nutné ho obnoviť. To sa realizuje najčastejšie pomocou takzvaného refresh tokenu, ktorý klient obdrží spolu s prístupovým tokenom.

V Djangu sa táto autentifikácia dá jednoducho realizovať pomocou modulu OAuth2 [26], ktorý budem v tejto práci využívať. Rovnako tak sa tento typ autentifikácie bude využívať pri komunikácii s API rozličných aplikácií.

2.6.12 Angular

Angular je frontend framework založený na jazyku TypeScript. Vyvíja ho spoločnosť Google, a používa sa pre vývoj webových a mobilných aplikácií. Je to úplné prepísanie frameworku AngularJS. V súčasnosti je najvyššia verzia An-

gular 5, predchádzali ju verzie 2 a 4 (číslo 3 bolo preskočené a za verziu 1 sa pokladá framework AngularJS).

Aplikácie sa vyvíjajú pomocou šablón (templates), so špecifickou syntaxou pridanou Angularom. Jednou z najdôležitejších častí syntaxe sú dekorátory. Sú to funkcie, ktoré modifikujú TypeScriptové objekty. Pridávajú metadáta k triedam s informáciou o ich funkciách a spracovaní.

Jazyk TypeScript od Microsoftu je doporučený jazyk vývoja v Angulare. Je to nadstavba JavaScriptu, a pred spustením v prehliadači sa do neho musí skompilovať. Umožňuje objektovo-orientované programovanie a na rozdiel od JavaScriptu poskytuje statické typovanie.

Angular obsahuje mnoho pomocných modulov, ako je napríklad HttpClientModule na HTTP komunikáciu so serverom [7].

2.6.13 Angular CLI

Jednou z výhod Angularu je nástroj Angular CLI, ktorý poskytuje správu vývoja aplikácie z príkazového riadku. Dokáže založiť projekt, generovať časti kódu, spúšťať automatické testy a za behu kompilovať projekt do JavaScriptu vďaka detekcii zmien v kóde [8].

2.6.14 Moduly

Základnou stavebnou jednotkou Angularu sú moduly. Tento systém modulov sa nazýva NgModules. Predstavujú logické celky aplikácie a skladajú sa z komponentov, služieb, direktív a ďalších prvkov Angularu. Na menšie aplikácie často postačí jeden modul. Na rozsiahlejšie projekty, pokiaľ to má zmysel, ich môžeme využiť niekoľko. Každý projekt však musí obsahovať základný modul, ktorý sa volá AppModule (ukážka 2.3). Moduly sa označujú dekorátorom @NgModule. Tento dekorátor priraďuje informáciu o importovaných externých moduloch a prvkoch aplikácie (Angular docs: Fundamentals [7]).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Kód 2.3: Príklad Angular modulu

2.6.15 Komponenty

Komponenty obsahujú samotnú logiku aplikácie. Definujú sa vo vnútri tried a využíva sa dekorátor @Component.

```
@Component({
  selector:    'example-selector',
  templateUrl: './example.component.html',
  providers:  [ ExampleService ]
})
export class ExampleComponent {}
```

Kód 2.4: Príklad Angular komponentu

Logika komponentu sa implementuje dovnútra triedy. Atribúty dekorátora určujú názov selektora – tagu, ktorým môžem komponent zobraziť, cestu k šablóne patriacej komponentu a „providers“, čiže služby ktoré komponent využíva (Angular docs: Fundamentals [7]). Príklad komponentu je znázornený v ukážke 2.4

2.6.16 Data Binding

Ďalšou dôležitou súčasťou frameworku je data binding, ktorý predstavuje mechanizmus medzi komponentom a templatom. Pomocou neho medzi sebou tieto časti komunikujú. Podľa dokumentácie (Angular docs: Fundamentals, Components & Templates, Template Syntax [7]), Angular rozlišuje 4 druhy data bindingu: Interpolation (2.5), Property binding (2.6), Event binding (2.7) a Two-way binding (2.8).

```
<p> {{ variable }} </p>
```

Kód 2.5: Interpolation: Jednoduchý výpis premennej v šablóne

```
<child [property]="currentState"></child>
```

Kód 2.6: Property binding: Získanie hodnoty z rodičovského komponentu

```
<button (click)="eventCall()">Event button</button>
```

Kód 2.7: Event binding: Používa sa na volanie metódy zo šablóny

```
<input [(ngModel)]="variable">
```

Kód 2.8: Two-way binding: Kombinuje Event binding a Property binding

2.6.17 Services

Services, alebo Služby, predstavujú čokoľvek čo aplikácia potrebuje. Je to veľmi široký pojem. Využívajú sa na rôzne funkcie, ako je napríklad pripojenie k serveru (autentifikácia, odosielanie a prijímanie dát), konfigurácia aplikácie, výpočtové funkcie a podobne. Služby odľahčujú komponenty od množstva kódu

a logiky. Pomocou Dependency injection sú prístupné všetkým komponentom modulu (Angular docs: Tutorial, Services [7]).

2.6.18 Dependency injection

Dependency injection je návrhový vzor ktorý definuje akým spôsobom vytvárať objekty závislé na iných objektoch. Ak máme objekty ktoré majú ďalšie závislosti, pri vytvorení im tieto závislosti musíme poskytnúť. Dependency injection to zabezpečuje externou entitou, ktorá tieto závislosti dodá.

Výhodou je, že vďaka Dependency injection môžeme testovať, pridávať a upravovať jednotlivé objekty nezávisle na sebe.

Angular má vlastný framework, ktorého účelom je práve Dependency injection. Definícia závislosti sa vloží do konštruktora objektu a na základe toho Angular dokáže zabezpečiť závislosti (Angular docs: Fundamentals, Dependency Injection [7]).

2.6.19 Direktívy

Direktívy sú triedy, ktoré sú schopné dynamicky meniť DOM na základe inštrukcií ktoré sú im predané. Používajú sa na pridanie, odobranie alebo výmenu elementov stránky. Sú tiež užitočné pri výpise zoznamov.

```
<div *ngIf="showHelloWorld()">
    Hello world
</div>
```

Kód 2.9: Príklad Angular direktívy

V príklade 2.9 je použitá direktíva NgIf, ktorá sa správa veľmi podobne ako obyčajný „if“ príkaz v bežných programovacích jazykoch. Pokiaľ je návratová hodnota metódy showHelloWorld() true, vypíše text Hello world. V prípade že metóda vráti false, nevypíše v tomto prípade nič.

Existuje mnoho ďalších direktív poskytovaných Angularom. Ako napríklad NgFor, NgSwitch alebo NgStyle. Angular taktiež umožňuje vytvorenie vlastných direktív pomocou dekorátora @Directive (Angular docs: Fundamentals, Components & Templates, Attribute Directives [7]).

2.6.20 HTML

HTML (Hyper Text Markup Language) je značkovací jazyk určený pre tvorbu webových stránok. Popisuje štruktúru a z časti aj vzhľad stránky. Používa HTML tagy, ako napríklad <h1></h1>, ktorý popisuje nadpis. Text dokumentu stránky sa uzatvára do týchto tagov. Tagy je možné do seba vnárať a vytvárať tak hierarchickú štruktúru dokumentu [20].

2.6.21 CSS

CSS, alebo kaskádové štýly, je jazyk pre popis vzhľadu elementov jazykov ako HTML, XHTML alebo XML. Hlavným účelom tohto jazyka je oddelenie štruktúry webovej stránky od jej vzhľadu.

Syntakticky sa CSS skladá zo selektorov a deklarácií. Selektory určujú element ktorého vzhľad chceme špecifikovať a deklarácia sa skladá z vlastnosti a jej hodnoty [19].

2.6.22 Dokumentácia

Z hľadiska znovu použiteľnosti a dlhodobého vývoja je dokumentácia kódu vysoko užitočná, niekedy dokonca nevyhnutná. Existuje množstvo technológií ktoré tvorbu dokumentácie uľahčujú. Pri tomto systéme bude využitý Compodoc pre klientsku časť implementácie, a Django admin dokumentácia pre serverovú.

Kapitola 3

Návrh

Táto kapitola sa bude zaoberať návrhom systému podľa požiadaviek špecifikovaných v predchádzajúcej kapitole. Položím základ databáze, komponentom na strane klienta a užívateľskému rozhraniu.

3.1 Spôsoby použitia

Spôsoby použitia, alebo Use Cases, sú metódou v softvérovom inžinierstve ako opísať správanie systému v rozličných situáciách a interakciu medzi systémom a užívateľom. Často sú reprezentované v podobe Use Case diagramov a pomáhajú bližšie pochopiť a špecifikovať funkcie softvéru. Ich základ tvoria požiadavky na systém [11].

Diagramy v tejto sekcii boli vytvorené pomocou softvéru StarUML [17], ktorý slúži na tvorbu UML diagramov.

3.1.1 Nastavenia

V sekcii nastavenia (ktorá sa bude zobrazovať v obsahu modálneho okna) sa budú nachádzať nasledujúce položky:

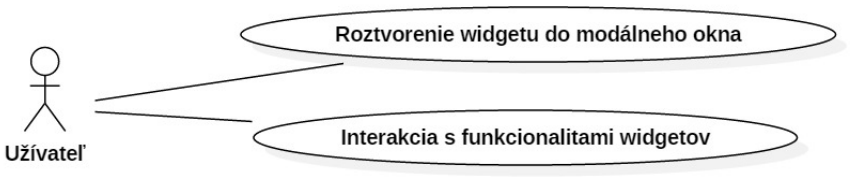
- Profil – meno a e-mail užívateľa s možnosťou úpravy týchto údajov
- Účty – Zoznam účtov patriacich užívateľovi s možnosťou ich zmazania
- Widgety – Zoznam widgetov nachádzajúcich sa na dashboarde užívateľa a informácie k nim (meno, popis, informácia o existencii priradeného účtu)

3.1.2 Dashboard

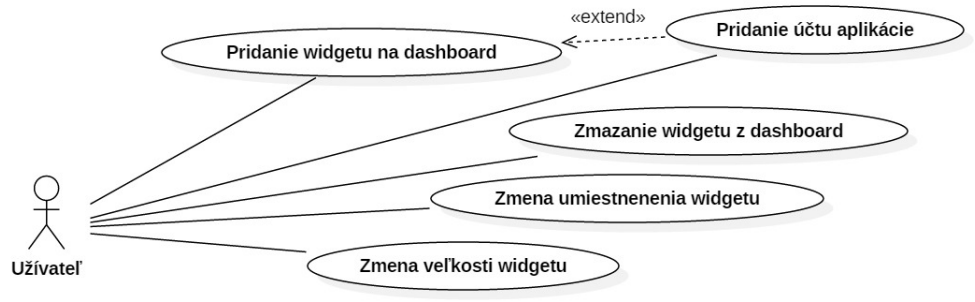
Samotný dashboard, ktorý bude tvoriť hlavnú funkcionalitu systému bude mať možnosť dvoch módov: normálneho a editovacieho. Diagramy oboch módov sú znázornené na obrázkoch 3.1 a 3.2.

V normálnom móde budú zakázané všetky možnosti úprav, povolená bude len interakcia s funkcionalitami (widgetami).

Editovací mód bude povoľovať možnosti úprav rozmiestnenia, zmenu veľkostí widgetov na dashboarde, pridanie a odobranie widgetu z plochy.



Obr. 3.1: Use Case diagram normálneho módu



Obr. 3.2: Use Case diagram editovacieho módu

3.2 Scenár autentifikácie

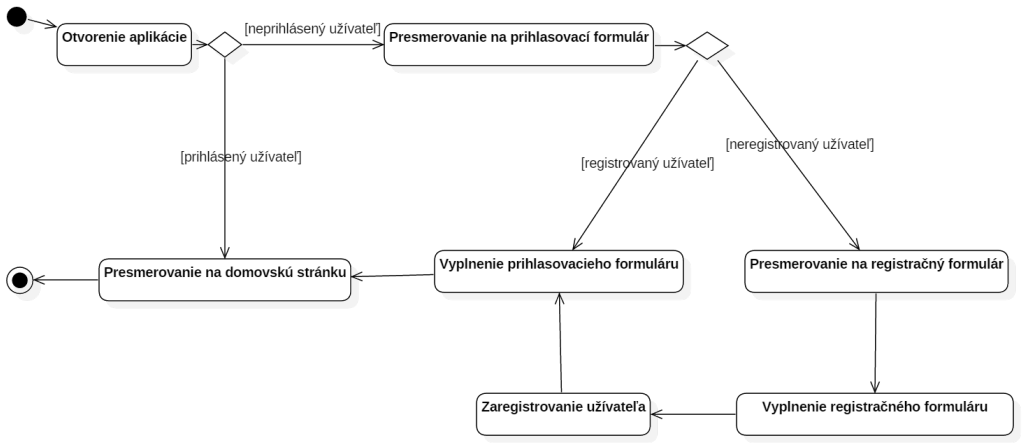
Ako bolo spomenuté v požiadavkách, autentifikácia užívateľa bude povinná, čo znamená, že pre prístup k systému bude užívateľ povinný prihlásiť sa, alebo v prípade nového užívateľa registrovať a následne sa prihlásiť. Po úspešnom prihlásení budú sprístupnené všetky funkcionality systému, ktoré budú viazané na účet užívateľa. Tieto funkcionality sa budú nachádzať na domovskej stránke.

Nasledujúci diagram 3.3 je vytvorený pomocou UML diagramu aktivít, ktorý je určený práve na modelovanie scenárov. V tomto diagrame z dôvodu prehľadnosti nie je zachytená validácia formulárov a riešenie chýb. Predstavuje teda ideálne fungovanie autentifikácie systému. Diagram bol zostavený na základe článku portálu UML diagrams[6].

Užívateľské a serverové chyby prihlásenia a registrácie budú ošetrené nasledujúcimi spôsobmi:

- Ošetrenie formulárov pomocou UI frameworku – užívateľ dostane oznámenie o chybe ešte pred odoslaním dát na server (nedostatočná dĺžky hesla, užívateľského mena, nesprávny tvar e-mailu)

- Notifikácie – zobrazia sa až po odpovedi zo servera (nesprávne heslo, neexistujúce užívateľské meno, už zaregistrovaný užívateľ)



Obr. 3.3: Diagram autentifikácie

3.3 Wireframe

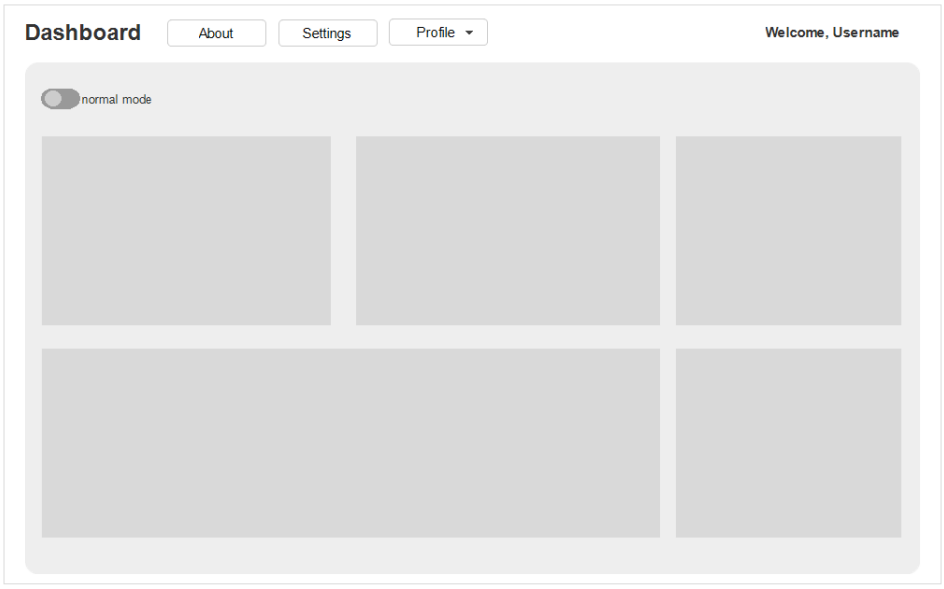
Wireframe je základný návrh hlavných častí systému. Neurčuje dizajn, ale rozloženie komponentov. Je dobrou pomôckou k samotnému dizajnu. V nasledujúcich sekciách predstavím návrhy hlavných stránok a ovládacích prvkov systému. Wireframy nie sú záväzné v rámci farebnej schémy a vzhľadu prvkov.

Nasledujúce návrhy boli vytvorené pomocou nástroja Justinmind [14].

3.3.1 Normálny mód dashboardu

Hlavnou stránkou systému bude mód interakcie medzi widgetami a užívateľom. Zvolené rozloženie častí je nasledovné: Aplikácia bude obsahovať horný panel s niekoľkými ovládacími prvkami a pod panelom sa bude nachádzať dashboard s widgetami.

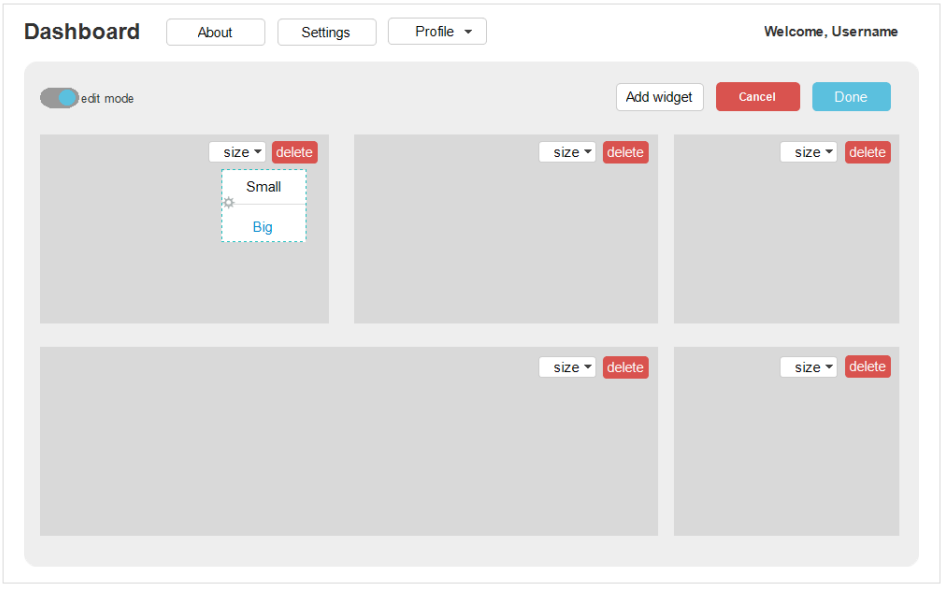
V normálnom móde bude dashboard obsahovať len prepínač do editovacieho módu. Ďalšie ovládacie prvky budú zabezpečené len jednotlivými widgetami 3.4.



Obr. 3.4: Wireframe normálneho módu

3.3.2 Editovací mód dashboardu

Pri editovacom móde bude umožnená správa widgetov pomocou tlačidiel „Size“ a „Delete“ (kde „Delete“ bude reprezentované ikonkou) pri každom widgete.



Obr. 3.5: Wireframe editovacieho módu dashboardu

Rovnako pribudne možnosť pridania nového widgetu na dashboard a ukon-

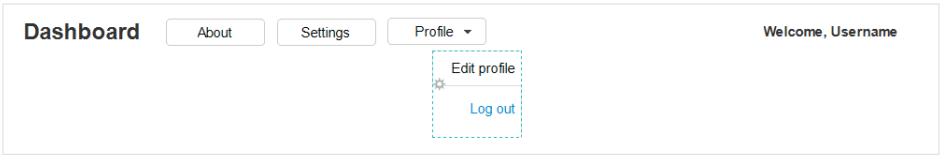
čenie editovacieho módu dvomi spôsobmi – tlačidlom „Cancel“, ktoré zahodí všetky doterajšie úpravy a tlačidlom „Done“, ktoré uloží zmeny a vráti dashboard do normálneho módu 3.5.

3.3.3 Menu

V predchádzajúcich wireframoch už bol znázornený aj horný panel, ktorý obsahuje niekoľko ovládacích prvkov systému.

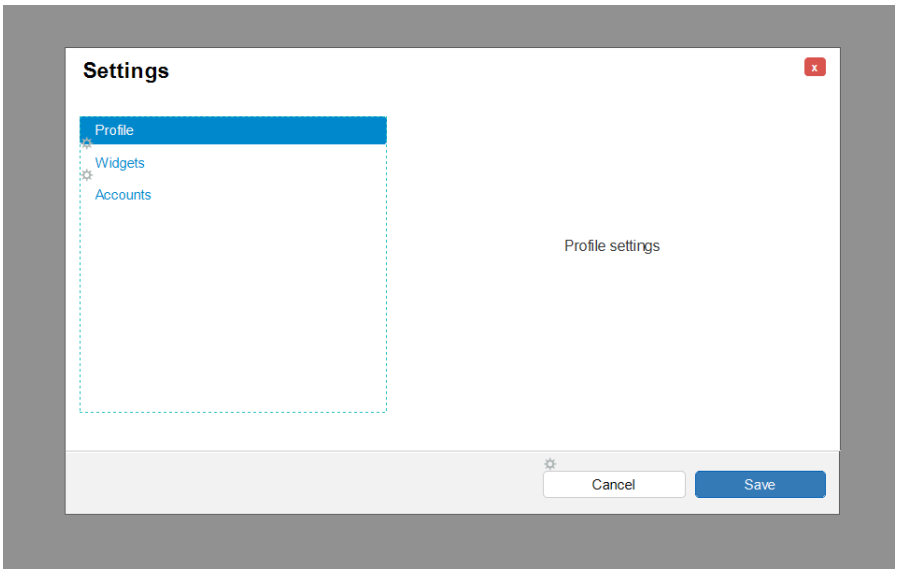
Tlačidlo „Settings“ bude odkazovať na modálne okno s rôznymi nastaveniami (ktoré bude bližšie ukázané v nasledujúcej sekcii), tlačidlo „About“ odkáže na ďalšie modálne okno s informáciami o aplikácii a tlačidlo „Profile“ bude otvárať dropdown s dvomi možnosťami: „Edit profile“ (nastavenia užívateľského profilu) a „Log out“ (okamžité odhlásenie zo systému).

Na pravej strane sa bude nachádzať text „Welcome Username“ s užívateľským menom aktuálne prihláseného užívateľa namiesto „Username“ 3.6.



Obr. 3.6: Wireframe roztvoreného menu

3.3.4 Modálne okná



Obr. 3.7: Wireframe modálneho okna nastavení

Modálne okno je vyskakovacie okno, ktoré je potomkom hlavného okna webovej aplikácie. Je zobrazené po určitej akcii a vytvára zvláštny režim stránky, kde nie je povolená interakcia s inými časťami aplikácie ako je samotné modálne okno.

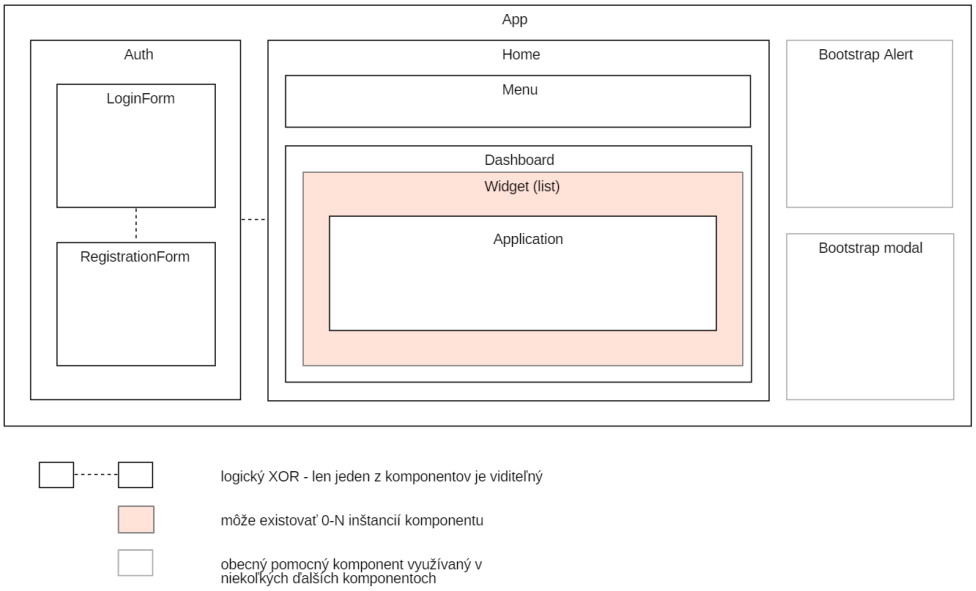
Tieto okná sa budú využívať na zobrazenie rôznych informatívnych správ, možnosti nastavení (profilu, správy účtov), a pre zväčšenie widgetov.

Okno s nastaveniami bude obsahovať bočné menu, vďaka ktorému sa užívateľ môže preklikávať medzi položkami nastavení, ktorých obsah sa bude zobrazovať na pravej strane okna 3.7.

3.4 Návrh komponentov

Po špecifikácii rozloženia a dizajnu prvkov sa zameriam na logiku a funkcie jednotlivých častí stránky. Rozloženie na jednotlivé komponenty je dôležitý úkon pred samotnou implementáciou, pretože nesprávne prevedenie môže viesť k rozsiahlemu a zbytočne zložitému kódu.

Framework Angular pracuje s komponentami, ktoré spoločne tvoria výslednú aplikáciu. Každý komponent má vlastnú logiku a preto je správne rozdelenie týchto komponentov kľúčové. Angular umožňuje aby každý komponent mohol v sebe obsahovať ďalšie komponenty. Túto funkciu budem v práci často využívať.



Obr. 3.8: Diagram Angular komponentov

Obrázok 3.8 popisuje návrh a hierarchiu komponentov. Hlavný komponent

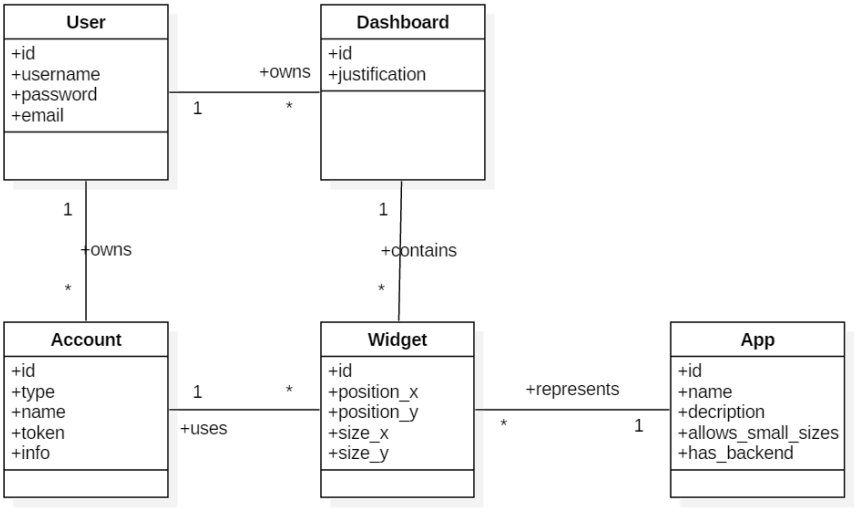
je daný Angularom, a to je komponent s názvom „app-component“. Ten bude zastrešovať všetky ostatné komponenty. Systém bude mať dve hlavné časti – autentifikačnú a domovskú stránku. V rámci autentifikačnej časti bude obsahovať buď prihlasovací, alebo registračný formulár. Domovská stránka bude obsahovať samotný dashboard s widgetami a ďalšími ovládacími prvkami.

Dôležitými komponentami sú Alerty a Modálne okná, ktoré sa budú využívať na niekoľkých miestach, a teda v niekoľkých komponentoch. Budú slúžiť na zobrazovanie informatívnych správ, hlásenie chýb, správu nastavení a podobne. Oba tieto komponenty nebudú definované v tejto práci, ale v UI frameworku Bootstrap, ktorý ich poskytuje. Budú však značne využívané v rámci celého projektu, preto sú spomenuté aj v tomto návrhu.

Systém bude tiež obsahovať niekoľko komponentov pre obsah modálnych okien, ktoré však pre jednoduchosť nie sú zobrazené na diagrame. Bude sa jednať o okná nastavení, sekciu s informáciami o tomto systéme a zobrazenie zväčšenej podoby aplikácií.

Ďalšie komponenty budú tvoriť jednotlivé aplikácie, ktoré sa budú do systému pridávať. Ich komponenty a celková šablóna aplikácie bude definovaná v implementačnej časti tejto práce.

3.5 Databáza



Obr. 3.9: UML diagram databázy

Návrh tabuliek databázy je jeden z najdôležitejších návrhov pred samotnou implementáciou. Je to základný stavebný prvok celého systému a správny návrh uľahčuje následný vývoj. Na modelovanie diagramu som opäť využila StarUML softvér a class diagram 3.9.

Tabuľka User (Užívateľ) je v tomto diagrame len orientačná, keďže Django už má implementáciu tohto modelu. Systém bude teda už len pracovať s inštanciami tohto objektu. V skutočnosti táto tabuľka obsahuje oveľa viac atribútov, ale v tomto diagrame sú pre jednoduchosť zobrazené len tie, ktoré sa budú v práci využívať.

Tabuľka Dashboard bude slúžiť najmä na identifikáciu skupiny widgetov, ktoré sa na dashboarde daného užívateľa nachádzajú. V systéme bude povolených aj niekoľko dashboardov na jedného užívateľa, ale nie naopak. Nepovolí zdieľané dashboardy a to najmä kvôli bezpečnosti, keďže by sa museli zdieľať aj užívateľské účty, čo nedáva zmysel. Atribút „justification“ hovorí o zarovnaní widgetov na dashboarde. Môže nadobúdať hodnoty left, right alebo center.

Tabuľka Account prestavuje účet užívateľa. Môže mať len jedného vlastníka, keďže obsahuje prístupy k jednotlivým externým účtom.

Tabuľka App predstavuje reálnu aplikáciu – názov a informácie o aplikácii. Má tiež atribút „allows_small_sizes“ (true alebo false), ktorý informuje o povolení alebo zakázaní menších veľkostí widgetov.

Tabuľka Widget bude spájať užívateľský účet k aplikácii a samotnú aplikáciu. Podmienkou k existencii inštancie je, že sa nachádza na dashboarde, ku ktorému je tiež viazaná a má definované umiestnenie a veľkosť.

3.6 REST API

Návrh rozhrania, cez ktoré bude komunikovať server a klient je veľmi jednoducho odvoditeľné z návrhu databázy. Budú využité serializéry pre jednotlivé entity databázy (na prevedenie dát do správneho formátu) pre klienta a potom špeciálne View triedy pre odoslanie dát.

Na klientskej strane budú entity reprezentované Interface objektami. O ich prijatie sa postarajú Services implementujúce CRUD operácie.

Kapitola 4

Implementácia serverovej časti

Táto kapitola sa zameriava na realizáciu serverovej časti aplikácie. Celý vývoj aplikácie prebieha v Pycharm IDE od spoločnosti JetBrains [12]. Serverová aplikácia je založená na Django frameworku [4] s použitím Django Rest Frameworku [3] pre vytvorenie webového rozhrania.

4.1 Štruktúra projektu

Na začiatok predstavím štruktúru Django projektu. Väčšina súborov bola automaticky vygenerovaná pri založení projektu v Pycharm IDE.

```
dash_app ..... adresár django aplikácie
    __init__.py ..... označuje python balíček
    admin.py ..... nastavenia administrácie
    apps.py ..... konfigurácia aplikácií
    models.py ..... implementácia modelov
    serializers.py ..... implementácia serializérov
    test ..... adresár obsahujúci súbory s testami
    tests.py ..... súbor pre spustenie testov
    urls.py ..... routing konfigurácia aplikácie
    views.py ..... implementácia Views
dashboard ..... adresár django projektu
    __init__.py ..... označuje python balíček
    settings.py ..... globálne nastavenia projektu
    urls.py ..... routing konfigurácia projektu
    wsgi.py ..... deployment konfigurácia
manage.py ..... nástroj na správu projektu z príkazového riadku
db.sqlite ..... Vývojová databáza
```

Aplikácia obsahuje dva hlavné adresáre: dash_app a dashboard. Adresár dashboard predstavuje projekt a jeho globálne nastavenia. dash_app je ad-

resár aplikácie. V tomto význame však slovo aplikácia nemusí predstavovať samostatne stojacu webovú aplikáciu. Je to členenie Django funkcionalít do logických celkov. Keďže serverová časť systému nie je veľmi rozsiahla, stačí mi jedna aplikácia v tomto projekte.

Súbory ___init___.py sa nachádzajú v oboch adresároch a predstavujú definíciu Python balíčkov. Sú to prázdne súbory, ktorých funkciou je informovať o tom, že daný adresár je balíčkom.

Súbor manage.py je spustiteľný súbor, ktorý slúži na správu aplikácie z príkazového riadku. Umožňuje vygenerovať štruktúry projektu, častí kódov, spustenie Django shellu a spustenie vývojového servera.

4.2 Model

Modely v Djangu predstavujú entity databázy. Preto som sa pri implementácii riadila diagramom databázy.

Definícia entít je veľmi jednoduchá. Sú reprezentované triedami, ktoré dedia z triedy Model poskytovanou Djangom. Stĺpce databázy sú definované pomocou atribútov. Stĺpec s ID – unikátnym identifikátorom, je pridaný automaticky ku každému modelu.

```
from django.db import models

class Account(models.Model):
    owner = models.ForeignKey('auth.User', related_name='accounts
        ', on_delete=models.CASCADE)
    type = models.CharField(max_length=40)
    name = models.CharField(max_length=40)
    token = models.CharField(max_length=300)
    info = models.CharField(max_length=300, blank=True, null=True
        )
```

Kód 4.1: Implementácia modelu Account

Dátové typy jednotlivých atribútov sa opäť definujú pomocou tried Djanga. CharField predstavuje reťazec znakov a vyžaduje nastavenie „max_length“ parametra, čiže maximálnej dĺžky reťazca. Ďalším typom je Integer, predstavujúci číslo a ForeignKey, ktorý nastavuje cudzí kľúč. V implementácii Account entity 4.1 je vidieť cudzí kľúč „owner“, ktorý spája účet s jeho vlastníkom. Implementácie Dashboard, Widget a App sú realizované veľmi podobne, takže ich tu nebudem uvádzať.

Tabuľka User je súčasťou Djanga, takže jej definícia nie je nutná a práca s ňou je rovnaká ako s ostatnými modelmi.

4.3 Administrácia

Django administrácia je veľmi užitočná funkcia tohto frameworku. Umožňuje pridanie, zmazanie alebo zmenu dát uložených v databáze. Je výhodná najmä

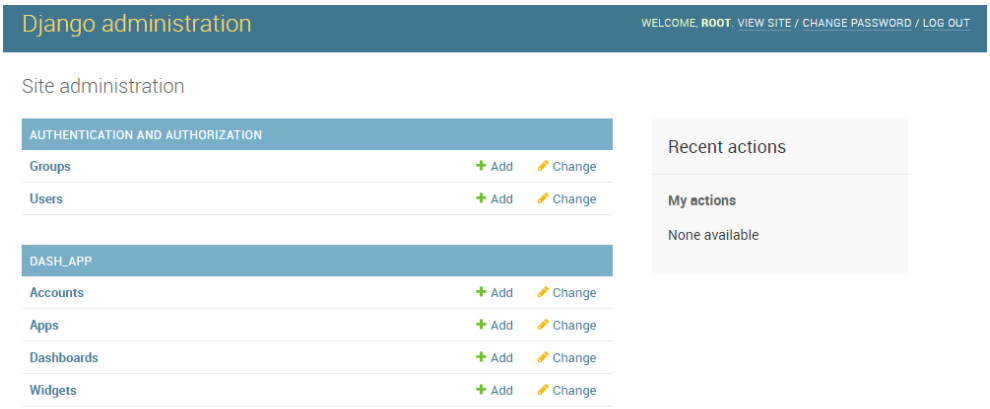
pri testovaní a vývoji.

Pre prístup k tejto funkcii je nutné založiť účet Superuser. Je to užívateľ, ktorý má práva k správe projektu. Vytvorenie tohto užívateľa sa realizuje pomocou príkazu 4.2.

```
python3 manage.py createsuperuser
```

Kód 4.2: Vytvorenie účtu Superuser v termináli

Po zadaní tohto príkazu sa Django spýta na užívateľské meno, heslo a email. Po zadaní údajov je účet Superusera vytvorený. Ďalej už len stačí registrovať vytvorené modely v admin.py súbore, a administrácia je pripravená na použitie. Predvolená hodnota je nastavená na URL „/admin“. Na obrázku 4.1 je ukážka administrácie po prihlásení sa so zoznamom databázových tabuliek.



Obr. 4.1: Ukážka Django administrácie

4.4 REST API

Pre tvorbu REST API som použila modul Django Rest Framework. Jeho pridanie do projektu a použitie pozostáva z niekoľkých jednoduchých krokov. Inštalácia prebieha v príkazovom riadku a modul sa pridá do INSTALLED_APPS v súbore settings.py.

4.4.1 Serializéry

Dôležitým bodom k vytvoreniu REST API je definícia serializérov, ktoré budú mať na starosti transformáciu dát do správneho tvaru pre HTTP prenos. Framework obsahuje triedy, ktoré výrazne uľahčujú implementáciu. V aplikácii som použila triedu ModelSerializer, z ktorej dedia všetky mnou definované serializéry.

```
from rest_framework import serializers
from dash_app.models import Account

class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ('id', 'owner', 'type', 'name', 'token', 'info')
```

Kód 4.3: Implementácia Account serializéra

Na príklade 4.3 vidieť jednoduchý a plne funkčný serializér pre Account model. Trieda ModelSerializer požaduje len názov modelu a atribúty, ktoré chceme transformovať. O zvyšok sa postará sám framework. Serializér musí existovať pre všetky modely, vrátane modelu User. Pre tento model sa bude serializér trochu líšiť.

```
from rest_framework import serializers
from django.contrib.auth.models import User

class UserSerializer(serializers.ModelSerializer):
    password = serializers.CharField(style={'input_type': '
        password'})
    class Meta:
        model = User
        fields = ('id', 'username', 'password', 'email')

    def create(self, validated_data):
        user = super(UserSerializer, self).create(validated_data)
        if 'password' in validated_data:
            user.set_password(validated_data['password'])
            user.save()
        return user
```

Kód 4.4: Implementácia User serializéra

Ako vidieť na ukážke implementácie 4.4, pribudla definícia password, kde je potrebné definovať špeciálny typ vstupu pre heslo. Metadáta triedy sú definované rovnako ako pri AccountSerializer, ale pribudla metóda create(), ktorá sa stará o vytvorenie novej inštancie modelu User. Je definovaná za účelom pridania hesla k objektu, čomu nestačí štandardný Djangom generovaný create(), ktorý sa pri ostatných serializéroch definovať nemusel.

Tento serializér je používaný pre účely registrácie. Keď však užívateľ potrebuje napríklad zistiť vlastné užívateľské meno, nemôže server odoslať kompletné užívateľské údaje, a to najmä z bezpečnostných dôvodov (napríklad pri hesle). Preto som vytvorila ďalší jednoduchý User serializér (CurrentUserSerializer), ktorý obsahuje len užívateľské meno a email. Využíva sa pri požiadavkách klienta o vlastné údaje a na úpravu týchto dvoch atribútov.

Fungovanie serializérov som si overila v Django shell a pomocou Django Unit testov, ktoré bližšie popíšem v kapitole o testovaní.

4.5 OAuth2

Pre autentifikáciu a bezpečnosť využívam modul OAuth2, ktorý zabezpečuje tokenovú autentifikáciu.

OAuth 2.0 je štandard, ktorý hovorí o postupe autentifikácie pre webové, mobilné a desktopové aplikácie. Je to protokol vyvíjaný IETF OAuth Working Group, ktorý nahradil svojho predchodcu OAuth [22].

4.5.1 Oprávnenia

V rámci modulov OAuth2 a Django Rest Framework sú špecifikované oprávnenia k API ako IsAuthenticated, čo znamená, že prístup je povolený len prihláseným užívateľom.

4.6 Views

Views, alebo pohľady v tejto aplikácií slúžia na odosielanie odpovedí zo servera ku klientskej aplikácii. Využíva sa pritom protokol HTTP. Pri implementácii som využila Django Rest Framework triedy ModelViewSet a APIView.

4.6.1 ModelViewSet

ModelViewSet trieda zabezpečuje implementáciu základných operácií s modelmi cez REST API. Čo znamená, že na základe priradeného modelu je schopná vytvoriť základné CRUD operácie pre daný model. Tieto operácie sú v podaní Djanga metódy, ktoré vracajú Response objekt. Tento objekt obsahuje stavový kód (HTTP status code, hovoriaci o výsledku operácie) a dáta.

V mnohých prípadoch som však tieto metódy preťažovala v rámci vlastných tried dediacich z ModelViewSet, a to najmä z dôvodov potrebných úprav výsledkov (zmena poradia, filtrovanie, pridanie údajov). Napríklad v prípade Account a Dashboard, kde je potrebné aby atribút „owner“ mal hodnotu aktuálneho užívateľa, som preťažila funkciu perform_create(), aby sa tento vlastník pridal automaticky pri vybavovaní požiadavky.

4.6.2 UserView

Pri Views užívateľa som musela využiť dva View, podobne ako pri serializéroch. Jeden View na manipuláciu s už existujúcim užívateľom UserView (dediaci z ModelViewSet) a druhý View pre registráciu užívateľov CreateUserView (dediaci z APIView). Dôvodom je, že pri registrácii užívateľ nemá existujúci účet, a teda nie je schopný sa autentifikovať, čo by mu zakázalo prístup k API (ktoré využíva OAuth2 autentifikáciu, takže neposkytne neoprávnený prístup). Preto som vytvorila View mimo OAuth2 a povolila tak prístup bez autentifikácie.

4.6.3 CORS

Cross-Origin Resource Sharing (CORS) je mechanizmus, ktorý používa HTTP hlavičky, aby boli klientovi poskytnuté potrebné oprávnenia k prístupu k dátam, ktoré požaduje od servera na inej doméne ako je on sám.

V tejto aplikácii som použila modul django-cors-headers pre pridanie týchto hlavičiek [32].

4.6.4 Routing

Aby užívateľ mohol pristupovať k dátam na serveri, je potrebné definovať URL na ktoré sa budú posielať požiadavky viazané na View. Nasledujúci kód 4.5 je zoznamom URL vzorov, ktoré API používa.

```
router = DefaultRouter()
router.register(r'apps', views.AppViewSet)
router.register(r'accounts', views.AccountViewSet)
router.register(r'widgets', views.WidgetViewSet)
router.register(r'dashboards', views.DashboardViewSet)
router.register(r'users', views.UserViewSet)

urlpatterns = [
    url(r'^', include(router.urls)),
    url(r'^o/', include('oauth2_provider.urls', namespace='
        oauth2_provider')),
    url(r'^admin/doc/', include('django.contrib.admindocs.urls'))
        ,
    url(r'^admin/', admin.site.urls),
    url(r'^api/', include('rest_framework.urls')),
    url(r'^users/register', views.CreateUserView.as_view()),
]
```

Kód 4.5: Implementácia smerovania

Na začiatku kódu je definícia Routera, ktorý generuje základné URL pre jednotlivé operácie všetkých modelov. Tieto URL sú štandardné a dajú sa nájsť v dokumentácii frameworku. Ďalej v zozname vidíme URL pre prístup do OAuth2 nastavení, Django admin dokumentácie, Django administrácie a Django Rest Frameworku. Špecifická je opäť samostatná definícia URL pre registráciu užívateľov používajúca CreateUserView.

Kapitola 5

Implementácia klientskej časti

V predchádzajúcej kapitole som rozobrala implementáciu serverovej časti. V tejto kapitole opíšem realizáciu klienta a jeho napojenie na vytvorené REST API. Pri vývoji klientskej aplikácie som využila WebStorm IDE od JetBrains [13] a framework Angular 5 [7].

5.1 Štruktúra projektu

Podobne ako pri implementácii servera, aj tu na začiatku predstavím najdôležitejšie súbory projektu a ich štruktúru.

- `app` ..... modul app
  - `applications` ..... Priečinok s komponentami aplikácií
  - `components` ..... Priečinok s komponentami logiky aplikácie
  - `services` ..... Priečinok so Services aplikácie
  - `app.module.ts` ..... Definícia modulu
  - `settings.ts` ..... Vlastné nastavenia
- `index.html` ..... Hlavná šablóna
- `styles.scss` ..... Kaskádové štýly
- `package.json` ..... Zoznam závislostí projektu
- `tsconfig.json` ..... Konfigurácia

Súbor settings.ts obsahuje moju vlastnú konfiguráciu aplikácie. To zahrňuje nastavenie adresy servera a ďalších konštánt využívaných v aplikácii.

V súbore app.module.ts sa nachádza definícia modulu. Do neho sú pridané deklarácie komponentov, importované moduly, registrované služby a ďalšie dôležité informácie o module.

5.2 Moduly a komponenty

Aplikácia obsahuje jeden modul a niekoľko komponentov, ktorých hierarchia a funkcie zodpovedajú návrhu komponentov na obrázku 3.8.

5.3 Routing

Angular umožňuje tvorbu takzvaných „Single-Page Application“ (jednostránkových aplikácií). To znamená, že v základnom nastavení neobsahuje smerovanie a všetko sa odohráva na jednej stránke (ani po preklikávaní medzi komponentami sa URL v prehliadači nebude meniť). V tomto systéme som však použila modul Router, ktorý zabezpečuje smerovanie medzi podstránkami systému. Nasledujúci kód 5.1 je jeho definícia:

```
RouterModule.forRoot([
    { path: '', component: AuthComponent },
    { path: 'home', component: HomeComponent },
    { path: 'login', component: LoginFormComponent },
    { path: 'register', component: RegistrationFormComponent },
]),
```

Kód 5.1: Implementácia smerovania

Path definuje reťazec ktorý bude pridaný na koniec URL hlavnej stránky. „Component“ je priradenie komponentu ktorý sa zobrazí keď užívateľ zadá celú túto URL do prehliadača. Router tiež umožňuje presmerovanie v kóde pomocou jednoduchého príkazu. To využívam najmä pri autentifikácii užívateľov.

Keďže väčšina logiky sa odohráva na jednej stránke – domovskej stránke, systém nepotrebuje rozsiahle smerovanie. Router definuje domovskú stránku s dashboardom – „home“, prihlasovací a registračný formulár – „login“ a „register“, a úvodnú stránku poskytujúcu základné informácie o aplikácii a odkaz na registračný a prihlasovací formulár.

5.4 Komunikácia s API

Pre komunikáciu s API serverovej časti aplikácie využívam Services triedy s Dependency injection HttpClientModule, čo je Angular modul umožňujúci komunikáciu cez HTTP.

Zoznam Services starajúcich sa o komunikáciu s API je nasledovný:

AuthService Sprostredkúva autentifikáciu pri prihlásení a pridáva HTTP hlavičky ostatným Services.

UserService Umožňuje vytvorenie užívateľa a editáciu profilu už existujúceho užívatela.

AccountService CRUD operácie nad entitou Account.

ApplicationService CRUD operácie nad entitou Application.

WidgetService CRUD operácie nad entitou Widget.

DashboardService CRUD operácie nad entitou Dashboard.

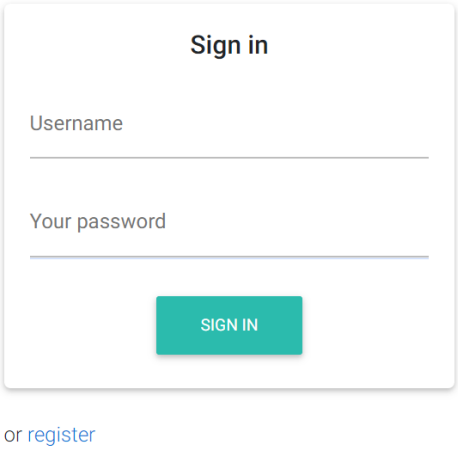
5.5 Autentifikácia

Ako bolo zmienené v predchádzajúcej sekcii, autentifikáciu a registráciu zabezpečujú AuthService a UserService. Pozrime sa však na užívateľskú stránku prihlásenia a registrácie.

5.5.1 Formuláre

Na prihlasovanie a registráciu slúžia jednoduché formuláre, ktoré sú definované v komponentoch Angularu: RegistrationForm a LoginForm. Tieto komponenty prijímajú užívateľský vstup z formulárov a predávajú ich AuthService alebo UserService k ďalšiemu spracovaniu – zaslanie požiadavky na server o prihlásenie alebo registráciu.

5.5.1.1 Užívateľské rozhranie



Obr. 5.1: Prihlasovací formulár

Formuláre sú komponenty UI frameworku MDB [16]. Na obrázku 5.1 sa nachádza prihlasovací formulár. Pod formulárom sa nachádza odkaz na registračný formulár, ktorý má rovnaký vzhľad, len pridané pole „Email“ (obrázok v prílohe C.6).

5.5.1.2 Ošetrenie užívateľských vstupov

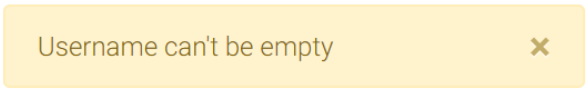
Tieto formuláre majú ošetrený užívateľský vstup, aby sa zjednodušil prístup pre užívateľa a zároveň sa zbytočne neodosielali neplatné dáta na server.

Základným ošetrením formulárov v takmer reálnom čase (po prekliknutí sa z Input poľa) je validácia frameworku MDB. Ten poskytuje základnú validáciu – minimálny alebo maximálny počet znakov, kontrola syntaxe emailu a podobne. Týmto spôsobom sa kontroluje e-mail a heslo.

Ďalší spôsob sú notifikácie pri pokuse užívateľa odoslať formulár. V notifikáciách sa zobrazujú chyby hlásené serverom (nesprávne údaje, užívateľ je už registrovaný), nedostupnosť servera alebo neúplnosť vyplnených údajov.

Ukážka 5.2 hlási chybu pri nesprávnom vyplnení poľa e-mailu (nesprávny tvar – chýba „@“ a „.“), zatiaľ čo ukážka 5.3 ukazuje Alert o nevyplnení prihlasovacieho mena.

Obr. 5.2: Chyba pri vyplňovaní registračného formulára



Obr. 5.3: Alert s upozornením pri prihlasovaní

5.6 Menu

Dôležitým ovládacím prvkov aplikácie je menu, ktoré obsahuje odkazy do nastavení, editácie profilu, informácií o aplikácii a umožňuje odhlásenie zo systému.

5.6.1 Užívateľské rozhranie

Pre menu, rovnako ako pri formulároch, bol využitý komponent z frameworku MDB – Navbar. Na obrázku 5.4 je výsledný vzhľad menu na základe návrhu.

Obr. 5.4: Menu aplikácie

5.7 Normálny mód dashboardu

Dashboard v normálnom móde umožňuje interakciu s widgetmi. Má jednoduchú logiku na zobrazovanie widgetov, ktoré sú reprezentované formou TypeScript poľa, hoci výsledné zobrazenie má formu matice. Na generovanie tejto matice je použitý grid MDB UI frameworku.

Jednotkou šírky pre widgety je stĺpec. Stĺpcov v riadku je dvanásť, preto najväčšia možná šírka widgetu je práve dvanásť. Výška widgetu je zadávaná v pixeloch. Výška riadku sa bude určovať podľa jeho najvyššieho widgetu. Ukážka tohto módu sa nachádza na obrázku v prílohe C.2.

5.8 Editovací mód dashboardu

Editovací mód používa rovnakú maticu ako normálny mód, ale pridáva možnosti úprav. Tie sú nasledovné:

- Posun widgetov: umožňuje pomocou metódy Drag and Drop presúvať widgety na ploche, len pomocou intuitívnych pohybov myši
- Zmena veľkosti widgetu: zmena šírky widgetu, čiže počtu stĺpcov ktorých priestor bude zaberať zároveň so zmenou výšky (v pixeloch)
- Odobranie widgetu: pomocou ikonky zmazania, ktorá odstráni widget z plochy
- Pridanie widgetu: formou vyskakovacieho okna, kde si užívateľ môže vybrať zo zoznamu aplikácii; widget sa pridá na posledné voľné miesto na dashboarde

Výsledný vzhľad dashboardu v editovacom móde možno vidieť na obrázku v prílohe C.3.

5.9 Aplikácie

V tejto sekcii poviem niečo o implementácii rozhrania a načítania komponentov predstavujúcich aplikácie (vo vnútri widgetov) na dashboarde.

5.9.1 Rozhranie

Rozhranie aplikácie, ako bolo spomenuté v požiadavkách, má byť unifikované, takže jednotné pre každú aplikáciu. Preto som vytvorila určitú šablónu ktorá uľahčuje vývoj. Je to zložka obsahujúca niekoľko povinných (prípadne nepovinných) súborov ktoré aplikácia potrebuje.

Obsah tejto zložky je nasledovný:

```
application ..... Zložka obsahujúca kód aplikácie
├── application.component.ts ..... Kód hlavného komponentu
├── application.component.html ..... Vzhľad hlavného komponentu
├── application-popup.component.ts ..... Kód popup komponentu
├── application-popup.component.html .... Vzhľad popup komponentu
└── application.service.ts ..... Multifunkčná service
```

Hlavný komponent aplikácie predstavuje komponent ktorý bude zobrazený na dashboarde užívateľa. Popup komponent je obsah popupu ktorý sa zobrazí po zväčšení widgetu (pri mnohých aplikáciách to môže byť len opätovné použitie hlavného komponentu, je však nutné aby bol definovaný aj popup komponent). Service v tejto zložke je nepovinná, avšak nutná z hľadiska prehľadnosti v prípadoch rozsiahlejšej logiky aplikácie, alebo komunikácie so serverom.

Do tejto zložky môžu byť pridané ďalšie komponenty a Services podľa potreby, v rozumnom množstve. Rovnako tak HTML súbory nie sú nevyhnutné pokiaľ sa vzhľad nachádza priamo pri TypeScript definícii komponentov.

Definície hlavného a popup komponentu sa musia držať základnej štruktúry, ktorá je nutná k správnemu fungovaniu widgetu. Definícia hlavného komponentu je nasledovná 5.2

```
@Component({
selector: 'name-application',
templateUrl: './name-application.component.html',
})
export class NameApplicationComponent extends
    ApplicationBaseComponent {
    constructor() {
        super()
    }
}

MAPPINGS['name-application'] = NameApplicationComponent;
```

Kód 5.2: Forma hlavného komponentu aplikácie

Trieda komponentu musí dediť z komponentu ApplicationBaseComponent, ktorý definuje Input atribúty. Tieto atribúty sú „state“ a „widget“. Sú nutné k odovzdaniu informácie o móde dashboardu (editovací alebo normálny) a widgetu, ktorý aplikáciu využíva (teda aj účtu, ktorý má daná aplikácia priradený).

Mapovanie reťazca názvu aplikácie a komponenty je nevyhnutné k dynamickému načítaniu aplikácie a je nutné aby názov pri mapovaní mal tvar názvu aplikácie v databáze, a za ním „-application“.

Definícia komponentu modálneho okna je veľmi podobná 5.3.

```
@Component({
selector: 'name-popup',
templateUrl: './name-popup.component.html',
```

```
})
export class NamePopupComponent extends PopupBaseComponent {
    constructor(public activeModal: NgbActiveModal) {
        super()
    }
}

MAPPINGS['name-popup'] = NamePopupComponent;
```

Kód 5.3: Forma komponentu modálneho okna aplikácie

Rozdiel od hlavného komponentu predstavuje koncovka mapovaného reťazca a konštruktor, ktorý musí obsahovať Dependency injection NgbActiveModal (aby mohol byť obsahom modálneho okna Bootstrapu).

Podrobný návod na to ako implementovať aplikácie pre dashboard bol vypracovaný a je prístupný vo forme PDF. Užívateľ si ho môže stiahnuť pomocou linku na úvodnej stránke aplikácie (na obrázku v prílohe C.1).

5.9.2 Načítanie komponentov

Angular umožňuje dynamické načítanie komponentov. Túto jeho funkciu som využila aj pri načítaní komponentov aplikácií. Pri implementácii som sa riadila článkom na portáli Medium [30], ktorý hovorí o načítaní komponentov z poľa namapovaných reťazcov na komponenty. Tieto reťazce ukladám do databázy ako názov aplikácie a pri načítaní dashboardu užívateľa prekladám tento reťazec na komponent, ktorý sa má zobraziť. Preto je dôležité aby každý nový komponent aplikácie bol uložený do tohto poľa, a mal správny a unikátny identifikačný reťazec. Kód komponentu, ktorý sa o toto načítanie stará, sa nachádza v prílohe D.1.

5.10 Lokalizácia

Lokalizácia je zabezpečená pomocou Angular Language Service (Angular docs: Internationalization [7]). Preklady sa ukladajú do XLF súborov a načítajú pri spustení. Aktuálne je dostupný len preklad do slovenčiny. Pre všetky ostatné lokácie je stránka v angličtine.

5.11 Dokumentácia

Aplikácia využíva automaticky generovanú dokumentácia Compodoc [23], ktorá obsahuje všetky potrebné informácie pre vývojárov: popis modulov, komponentov, Services, ich metód a atribútov. Zahrňuje tiež zoznam závislostí aplikácie a ďalšie užitočné informácie. Generuje sa nasledujúcim príkazom v príkazovom riadku 5.4.

```
npm run compodoc
```

Kód 5.4: Generovanie Compodoc dokumentácie

Kapitola 6

Implementácia aplikácií

V rámci tejto kapitoly rozoberiem implementáciu ukážkových aplikácií pre tento systém. Tieto aplikácie sa načítajú dynamicky a sú implementované formou Angular komponentov. Pre správne fungovanie je nutné mať záznam o aplikáciách v databáze.

6.1 Google Calendar aplikácia

Táto aplikácia sprostredkúva prístup a interakciu s už existujúcim Google kalendárom. Využíva pri tom API poskytnuté spoločnosťou Google [9]. Keďže toto API je určené pre jazyk JavaScript, využila som tiež modul types/gapi pre definíciu typov API v TypeScripte, aby som zaistila kompatibilitu.

Kalendár poskytuje základné funkcie Google Kalendáru – zobrazenie, pridanie, odobranie a zmenu udalostí.

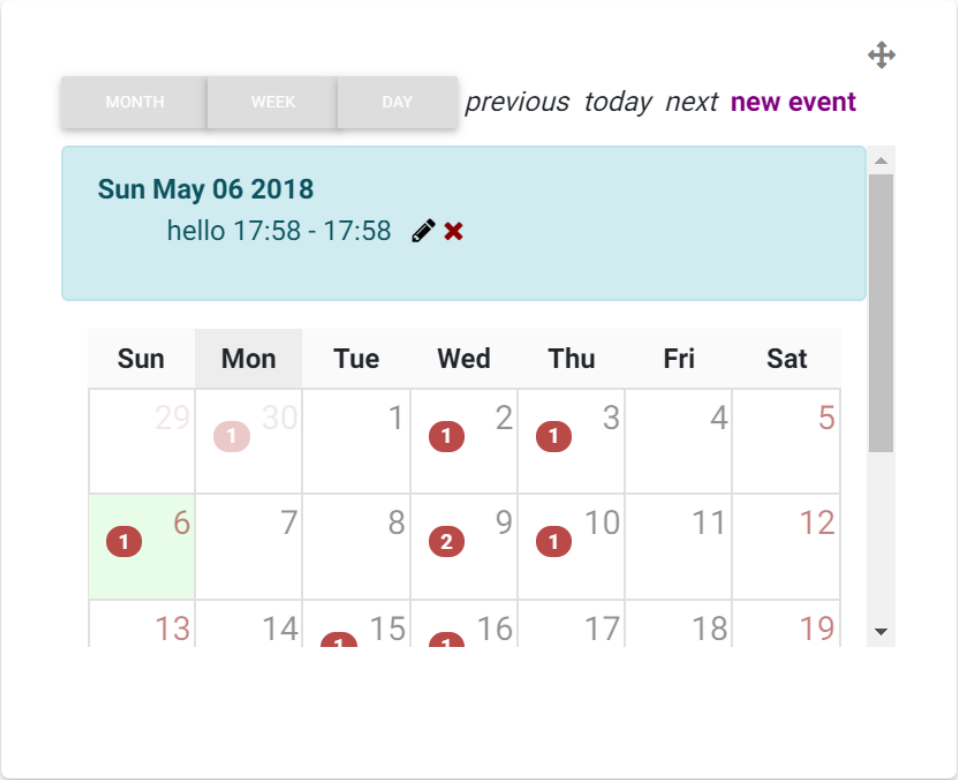
Pre grafické zobrazenie kalendára (všetky tri módy zobrazenia) bol použitý komponent angular 5.0+ calendar [15].

Kalendár obsahuje niekoľko módov zobrazenia, medzi ktorými sa môže užívateľ preklikávať za pomoci tlačidiel v záhlaví aplikácie:

- Mód mesiaca: Kalendár má minimalistické zobrazenie všetkých dní mesiaca. Po kliknutí na jeden z nich sa jeho dátum a všetky udalosti zobrazia v Alert komponente nad ním. Odtiaľ má užívateľ možnosť upravovať alebo mazať tieto udalosti.

- Mód týždňa: Dni v týždni sa zobrazia v riadku a udalosti sú zobrazené pod každým dňom.

- Mód dňa: Zobrazuje sa časová línia dňa, kde sa celodenné udalosti ukážu nad ňou a ostatné v daných časových úsekoch.



Obr. 6.1: Google Calendar widget v móde mesiaca

6.2 OneNote aplikácia

Microsoft Office OneNote je aplikácia od spoločnosti Microsoft na vytváranie poznámok. Poznámky sú uložené v poznámkových blokoch (notebooks), ktoré majú sekcie (sections) a v sekciách stránky (pages). Na tieto stránky sa píšu poznámky užívateľa.

Aplikácia komunikuje s Microsoft Graph API [29]. Umožňuje zobrazenie, vytváranie, úpravu a odstránenie poznámok. Na sprístupnenie funkcií vyžaduje prihlásenie sa pod účtom Microsoft.

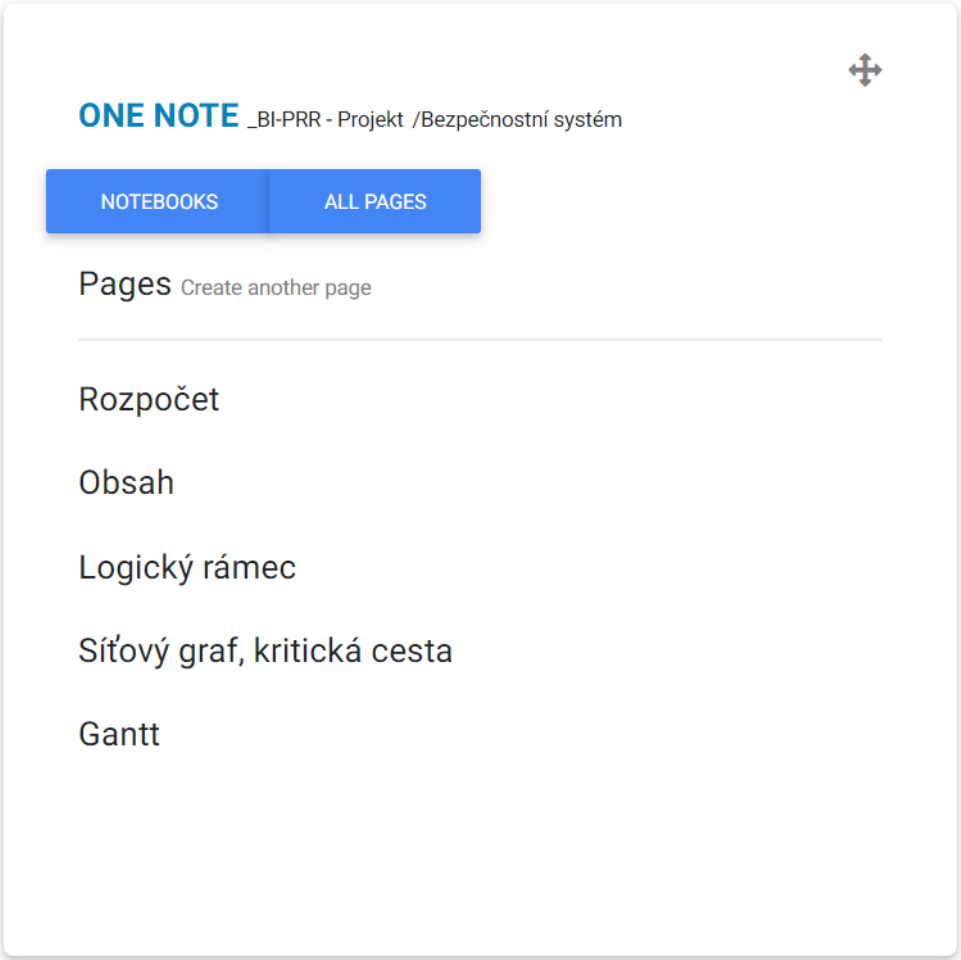
Prihlásenie k účtu sa realizuje pomocou implicitného flow, ktoré užívateľa presmeruje na prihlasovací formulár od Microsoftu a vyžiada schválenie práv pre aplikáciu na prístup a vytváranie zmien v aplikácii.

Umožňuje dva módy:

- Prechádzanie poznámkových blokov: Užívateľ sa môže preklikávať medzi blokmi, ich sekciami a v nich upravovať stránky s poznámkami. Pri prechádzaní sa zobrazuje cesta, kde sa užívateľ práve nachádza v záhlaví aplikácie.

- Zobrazenie všetkých stránok: Jednoduchý pohľad na všetky existujúce stránky, bez ohľadu na to v ktorých blokoch a sekciách sa nachádzajú.

Táto aplikácia je špecifická tým, že na rozdiel od Google Calendar aplikácie má napísaný vlastný Backend. Je to nevyhnutné z toho dôvodu, že Microsoft nepodporuje CORS hlavičky pri získavaní prístupového tokenu. Aplikácia teda nemôže získať token z klientskej aplikácie, preto sa odkazuje na server, ktorý zastupuje úlohu prostredníka pri komunikácii s API.



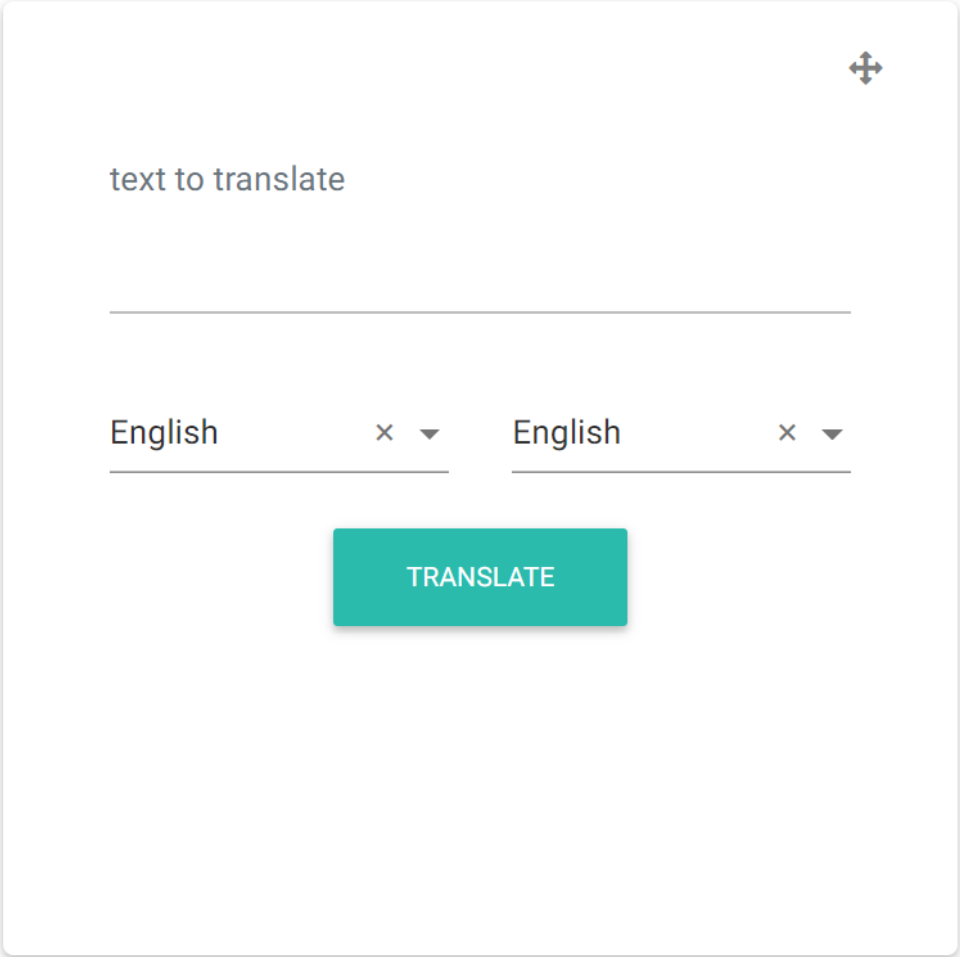
Obr. 6.2: OneNote widget v móde prechádzania notebookov

6.3 Translate aplikácia

Jednoduchá aplikácia pre prekladanie z a do cudzích jazykov. Používa Yandex Translate API [31], ktoré podporuje vyše 30 svetových jazykov. Aplikácia

nepotrebuje žiadny užívateľský účet pre svoje fungovanie.

UI aplikácie obsahuje formulár pre vloženie textu na preklad a voľbu jazykov. Výsledný preložený text sa zobrazí pod formulárom.



Obr. 6.3: Translate widget

6.4 Error aplikácia

Špeciálnou aplikáciou je Error aplikácia, ktorá slúži na zobrazenie chyby pri načítaní aplikácie. Napríklad po vymazaní widgetu, ktorý je však stále na dashboarde užívateľa, alebo akejkoľvek inej chyby pre ktorú sa nepodarilo zobraziť správnu aplikáciu. Error aplikácia má len jednu funkciu, a to je zobrazenie správy za účelom poskytnutia informácie užívateľovi, že nastala chyba.

Kapitola 7

Testovanie

Táto kapitola sa bude zaoberať testovaním funkcionalít implementovaných v predchádzajúcich častiach. Testovanie je realizované pomocou Unit testov a manuálneho testovania.

7.1 Unit testy

Unit testy sú testy zamerané na jeden menší logický celok aplikácie. Najčastejšie triedu alebo funkciu.

Django prináša vlastnú formu testovacích tried pomocou základnej triedy TestCase. Tieto testy sa spúšťajú v príkazovom riadku a nevykonávajú žiadne zmeny v reálnej databáze. Sú bezpečne spustené v izolovanom prostredí.

7.1.1 Testy modelov

Účelom týchto testov je overiť, že sú modely správne definované. Každý model má vlastnú testovaciu triedu. Kontrolujú povinné a nepovinné atribúty a testy majú danú nasledujúcu štruktúru metód:

setUp() Táto metóda sa spustí ako prvá. Vytvorí záznamy v databáze, s ktorými sa v jednotlivých testoch bude pracovať.

test_create() Testuje sa vytvorenie objektu modelu a jeho uloženie do databázy. V prípade nepovinných atribútov sa testuje uloženie bez nich, prípadne vyhodenie výnimky pri pokuse o uloženie nesprávnych dát.

test_get() Testuje sa získanie špecifického objektu z databázy, alebo prípadná výnimka, ak žiadaný objekt neexistuje.

test_filter() Testuje sa filtrovanie objektov na základe kritérií a kontroluje sa ich výsledný počet.

test_delete() Testuje sa zmazanie záznamu z databázy.

test_update() Testuje sa úprava záznamu v databáze.

7.1.2 Testy serializérov

Pomocou týchto testov zisťujem správnosť serializérov – validita, atribúty, prevod do JSON objektu a naspäť. Tieto testovacie triedy majú nasledujúce metódy:

setUp() Vytvorí záznamy v databáze, s ktorými sa v jednotlivých testoch bude pracovať.

test_create() Testuje vytvorenie serializéra.

test_edit() Testuje sa úprava už vytvoreného serializéra.

test_json() Testuje sa prevedenie serializéra na JSON objekt.

test_from_json() Testuje sa prevedenie z JSON objektu späť na Python objekt.

7.1.3 Spustenie automatických testov

Tieto testy boli spustené za každou väčšou zmenou v systéme (na serverovej strane), pre overenie správnosti pri vývoji. Obrázok 7.1 je screenshot výsledku testov po spustení v Bash termináli.

```
simona:dashboard$ python3 manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....................................
----------------------------------------------------------------------
Ran 36 tests in 0.125s

OK
Destroying test database for alias 'default'...
simona:dashboard$
```

Obr. 7.1: Výsledok Django Unit testov

7.2 Užívateľské testovanie

Pri tomto testovaní bola vybraná malá skupina ľudí, ktorej úlohou bolo otestovať zadané scenáre na tejto aplikácii. Tieto scenáre spolu s výsledkami testovania sú popísané v prílohe F.

7.3 Testovanie vo webových prehliadačoch

Aplikácia bola testovaná v niekoľkých prehliadačoch aby sa overila kompatibilita. Nasledujúci zoznam obsahuje tieto webové prehliadače a verzie v ktorých bola testovaná:

- Google Chrome verzia 65.0.3325.181
- Mozilla Firefox verzia 59.0.3
- Opera verzia 52.0.2871.99
- Microsoft Edge verzia 41.16299.371.0

7.4 Splnenie požiadaviek

Všetky požiadavky boli splnené a ich funkčnosť otestovaná. Multiplatformovosť aplikácie bola zabezpečná formou webovej aplikácie a jej rozšíriteľnosť použítím frameworkov (Angular - možnosť rozšírenia o ďalšie komponenty, Django - rozšírenie o aplikácie alebo modely). Widgety majú jednotné rozhranie a dynamické načítanie, čo umožňuje jednoduché pridanie ďalších widgetov do systému. Užívateľské rozhranie aplikácie má jednotný a minimalistický dizajn, a obsahuje len dôležité a funkčné prvky.

Záver

Cieľom tejto práce bolo vytvoriť interaktívny dashboard pre správu aplikácií, a implementovať niekoľkých ukážkových widgetov s dôrazom na multiplatformovosť a rozšíriteľnosť.

Systém pre správu aplikácií bol implementovaný formou webovej aplikácie, čo zabezpečuje jeho fungovanie na všetkých platformách, a tiež boli vyvinuté widgety pre demonštráciu funkcií systému. Tieto widgety majú unifikované rozhranie, vďaka ktorému na základe niekoľkých konvencií je možné vytvoriť nový widget a jednoducho ho integrovať do systému.

Všetky tieto funkcionality boli otestované formou automatických testov na serverovej strane aplikácie, a manuálnych užívateľských testov na klientskej strane.

Tento systém je len prvotnou verziou aplikácie v ktorej vývoji by som rada pokračovala do budúcna. To zahrňuje implementáciu ďalších funkcionalít systému, a vylepšenie a optimalizáciu už existujúcich. Taktiež by som rada do vývoja widgetov zapojila aj ďalších vývojárov, a rozšírila tak množstvo dostupných widgetov pre užívateľov.

Literatúra

[1] APPBRAIN. Number of Android applications [online]. 16. apríla 2018 [cit. 16. apríla 2018]. Dostupné z: http://www.appbrain.com/stats/number-of-android-apps

[2] Design Patterns. MVC Pattern [online]. In: tutorialspoint.com. 2018 [cit. 10. apríla 2018]. Dostupné z: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[3] Django Rest Framework v3.8.2 [online]. 2011-2017 [cit. 27. februára 2018]. Dostupné z: http://www.django-rest-framework.org/

[4] DJANGO SOFTWARE FOUNDATION. Django documentation 2.0 [online]. 2005-2018 [cit. 22. marca 2018]. Dostupné z: https://docs.djangoproject.com/en/2.0/

[5] EMARKETER. eMarketer Updates Worldwide Social Network User Figures [online]. 17. júl 2017 [cit. 16. apríla 2018]. Dostupné z: https://www.emarketer.com/Article/eMarketer-Updates-Worldwide-Social-Network-User-Figures/1016178

[6] FAKHROUTDINOV, Kirill. Activity Diagrams [online]. In: UML diagrams. 2009-2017 [cit. 18. februára 2018]. Dostupné z: https://www.uml-diagrams.org/activity-diagrams.html

[7] GOOGLE. Angular 5.2.9 [online]. 2010-2018 [cit. 19. marca 2018]. Dostupné z: https://angular.io

[8] GOOGLE. Angular CLI. A command line interface for Angular 1.7.4. [online]. 2010-2016 [cit. 19. marca 2018]. Dostupné z: https://cli.angular.io/

[9] GOOGLE. Google Calendar API v3 [online]. 8. marca 2018 [cit. 7. apríla 2018]. Dostupné z: https://developers.google.com/calendar/

[10] IEEE Guide for Software Requirements Specifications. In: IEEE Std 830-1984, vol., no., pp.1-26. 10. februára 1984 [cit. 10. apríla 2018]. doi: 10.1109/IEEESTD.1984.119205. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=278253&isnumber=6883

[11] Dr. JACOBSON, Ivar. Spence, Ian. Bittner, Kurt. USE-CASE 2.0: The Guide to Succeeding with Use Cases [online]. 2011 [cit. 20. februára 2018]. Dostupné z: https://www.ivarjacobson.com/publications/white-papers/use-case-ebook

[12] JETBRAINS. PyCharm 2017.3.2 [software]. 2000–2018 [cit. 4. apríla 2018]. Dostupné z: https://www.jetbrains.com/pycharm/?fromMenu

[13] JETBRAINS. WebStorm 2017.3.2 [software]. 2000–2018 [cit. 8. apríla 2018]. Dostupné z: https://www.jetbrains.com/webstorm/?fromMenu

[14] JUSTINMIND. Justinmind Prototyper 8.2.1 [software]. 2014-2018 [cit. 19. februára 2018]. Dostupné z: https://www.justinmind.com

[15] LEWIS, Matt. angular-calendar documentation 0.23.7 [online]. 2016 [cit. 28. marca 2018]. Dostupné z: https://mattlewis92.github.io/angular-calendar/docs/index.html

[16] MDBOOTSTRAP. Material Design for Bootstrap 4 (Angular) [online]. 2018 [cit. 10. apríla 2018]. Dostupné z: https://mdbootstrap.com/angular/

[17] MKLAB CO., LTD. StarUML 2 [software]. 2014-2018 [cit. 18. februára 2018]. Dostupné z: http://staruml.sourceforge.net/docs/user-guide(en)/toc.html

[18] The Model-View-Controller Design Pattern [online]. In: The django Book. 2017 [cit. 20. februára 2018]. Dostupné z: https://djangobook.com/model-view-controller-design-pattern/

[19] MOZILLA. CSS [online]. In: MDN web docs. 2018 [cit. 11. marca 2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[20] MOZILLA. HTML [online]. In: MDN web docs, 2018 [cit. 11. marca 2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[21] NG-BOOTSTRAP TEAM. Bootstrap 4 components, powered by Angular v1.1.2 [online]. [cit. 11. apríla 2018]. Dostupné z: https://ng-bootstrap.github.io/#/components/modal/examples

[22] OAuth 2.0 [online]. [cit. 2. mája 2018]. Dostupné z: https://oauth.net/2/

[23] OGLOBLINSKY, Vincent. Compodoc [online]. 2016-2017 [cit. 16. apríla 2018]. Dostupné z: https://compodoc.github.io/website/

[24] PETERS, Tim. PEP 20 – The Zen of Python [online]. 22. augusta 2014 [cit. 5. marca 2018]. Dostupné z: https://www.python.org/dev/peps/pep-0020/

[25] PYTHON SOFTWARE FOUNDATION. Python documentation 3.6.5 [online]. 2001-2018 [cit. 5. marca 2018]. Dostupné z: https://docs.python.org/3/

[26] RICHER, Justin. User Authentication with OAuth 2.0 [online]. [cit. 4. apríla 2018] Dostupné z: https://oauth.net/articles/authentication/

[27] STATISTA. Google Play Store: number of available apps 2009-2017 [online]. 2009-2017 [cit. 16. apríla 2018]. Dostupné z: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[28] SMITH, Steve. Don't Repeat Yourself [online]. In: O'Reilly commons. 2009 [cit. 10. apríla 2018]. Dostupné z: http://programmer.97things.oreilly.com/wiki/index.php/Don%27t_Repeat_Yourself

[29] MICROSOFT. Overview of Microsoft Graph. 2018 [cit. 30. apríla 2018]. Dostupné z https://developer.microsoft.com/en-us/graph/docs/concepts/overview

[30] VUIKA, Denys. Dynamic content in Angular. Multiple ways to create Angular components dynamically at runtime. In: Medium, 2016 [cit. 20. februára 2018]. Dostupné z: https://medium.com/@DenysVuika/dynamic-content-in-angular-2-3c85023d9c36

[31] YANDEX. Yandex Translate API. Developers. 2011-2018 [cit. 30. apríla 2018]. Dostupné z https://translate.yandex.com/developers

[32] YIU, Otto. django-cors-headers [online]. 2018 [cit. 10. apríla 2018]. Dostupné z: https://github.com/ottoyiu/django-cors-headers

Dodatok **A**

Zoznam použitých skratiek

OS Operation System (Operačný Systém)

UI User Interface (Užívateľské rozhranie)

GUI Graphical User Interface (Grafické užívateľské rozhranie)

API Application programming interface (Aplikačné programovacie rozhranie)

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

MVC Model View Controller

MVT Model View Template

CSS Cascading Style Sheets (Kaskádové štýly)

DOM Document Object Model

HTML HyperText Markup Language

IDE Integrated Development Environment

CRUD Create, Retrieve, Update, Delete

SRS Software Requirements Specification

UML Unified Modeling Language

MDB Material Design for Bootstrap (UI framework)

Dodatok **B**

Zoznam použitých programov

JetBrains Pycharm IDE pre vývoj v jazyku Python

JetBrains WebStorm IDE pre vývoj v jazyku TypeScript

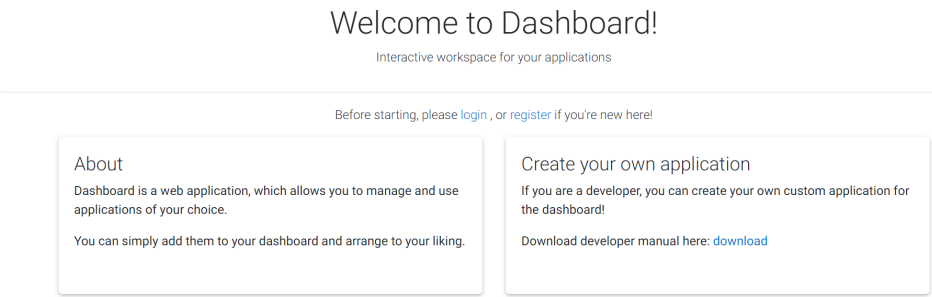
TEXworks Prostredie pre písanie práce v LaTeX

StarUML Software na tvorbu UML diagramov

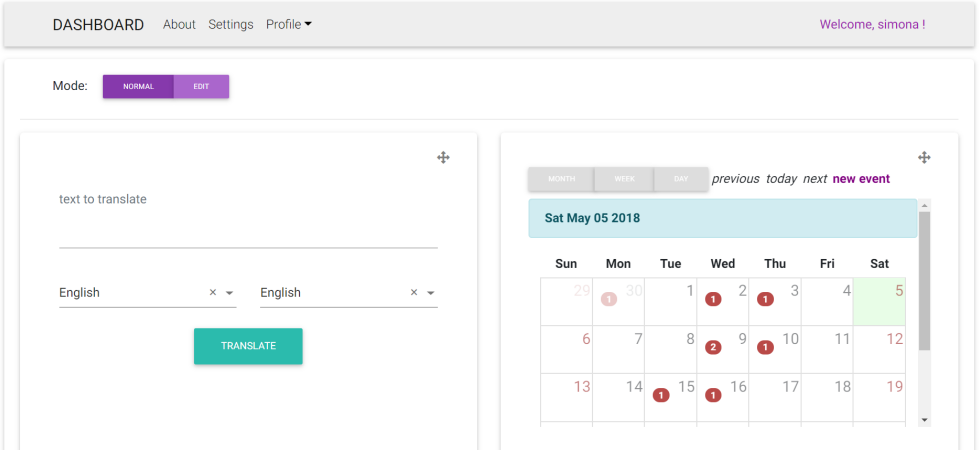
Justinmind Software na tvorbu Wireframes

Dodatok C

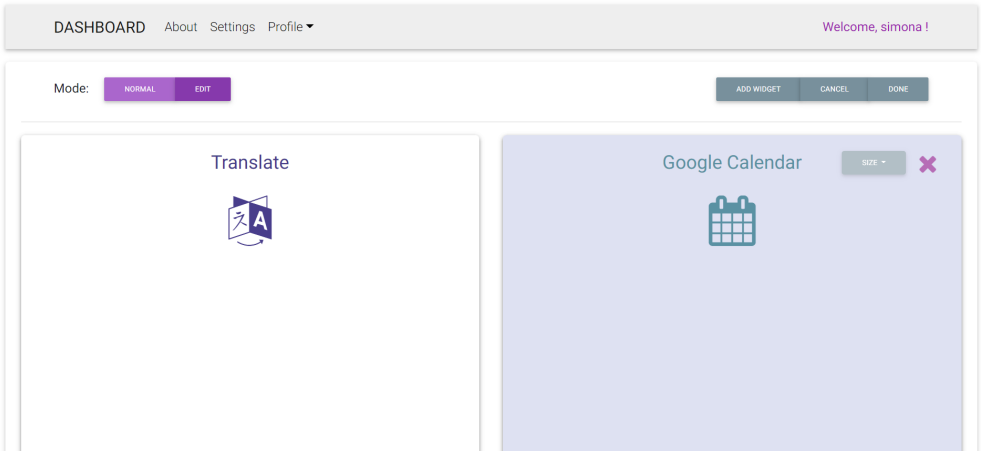
Ukážka výslednej aplikácie



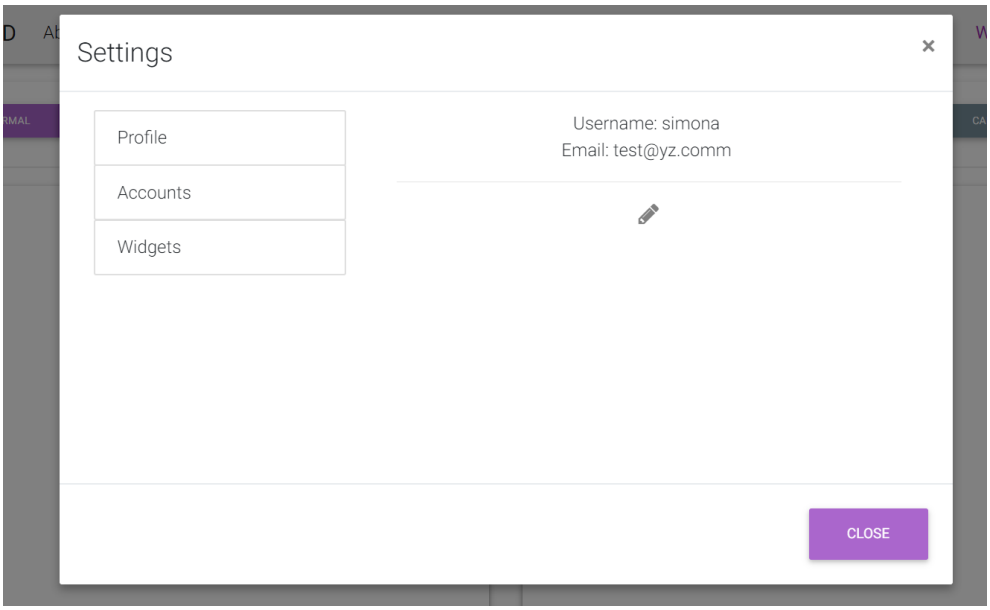
Obr. C.1: Úvodná stránka aplikácie



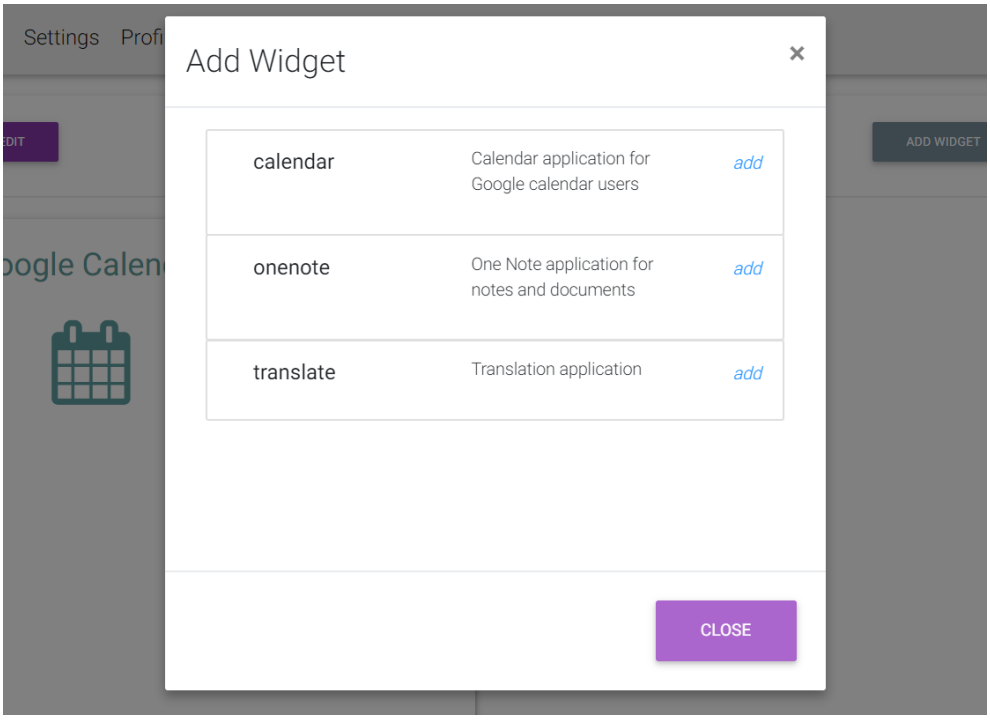
Obr. C.2: Domovská stránka v normálnom móde



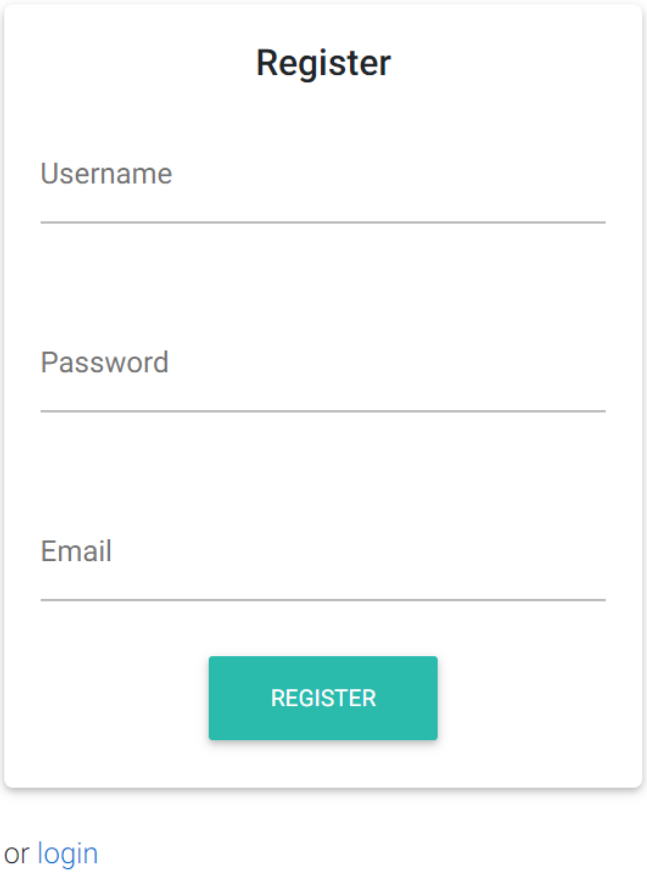
Obr. C.3: Domovská stránka v editovacom móde



Obr. C.4: Modálne okno s nastaveniami



Obr. C.5: Modálne okno pridávania widgetov



Obr. C.6: Stránka pre registráciu užívateľa

Dodatok D

Ukážka zdrojového kódu

```
/**
 * Mappings of the components and their names for dynamic loading
     of applications
 */
export let MAPPINGS = {};

/**
 * Loads application into widget dynamically and passes necessary
     values to it
 */
@Component({
  selector: 'application',
  templateUrl: './templates/application.component.html',
  providers: [ApplicationService, NgbModal]
})

export class ApplicationComponent implements OnInit, OnDestroy {
  /**
   * Widget object handed from dashboard
   */
  @Input() widget: WidgetInterface;
  /**
   * State of the dashboard
   */
  @Input() dashboardState: String;
  /**
   * Application in given widget
   */
  private application: ApplicationInterface;
  /**
   * Type of component to load
   */
  private type: String;
  /**
   * Reference to application component for dynamic loading
   */
  private componentRef: ComponentRef<{}>;
```

```
@ViewChild('container', { read: ViewContainerRef })
/**
 * Reference to container for the application component to load
 */
private container: ViewContainerRef;

/**
 * Returns component from the string name of application
 */
static getComponentType(typeName: string) {
  return MAPPINGS[typeName];
}

constructor(private componentFactoryResolver:
    ComponentFactoryResolver,
            private appService: ApplicationService,
            public popupService: NgbModal) {}

/**
 * Calls loading of application
 */
ngOnInit() {
  this.loadApplication();
}

/**
 * Loads Application component from app name and passes values
 */
loadComponent() {
  if (this.type) {
    let componentType = ApplicationComponent.getComponentType(
        this.type + '-application');
    if (!componentType) {
      componentType = ApplicationComponent.getComponentType('
          error-application');
    }
    const factory = this.componentFactoryResolver.
        resolveComponentFactory(componentType);
    this.componentRef = this.container.createComponent(factory)
        ;
    (<ApplicationBaseComponent>this.componentRef.instance).
        state = this.dashboardState;
    (<ApplicationBaseComponent>this.componentRef.instance).
        widget = this.widget;
  }
}

/**
 * Destroys component reference
 */
ngOnDestroy() {
  if (this.componentRef) {
    this.componentRef.destroy();
    this.componentRef = null;
```

```
    }
  }

  /**
   * Retrieves application data from server
   */
  loadApplication() {
    this.appService.retrieveAll().subscribe(
      data => {
        const applications = <ApplicationInterface[]>data['
            results'];
        for (let i = 0; i < applications.length; i++) {
          if (this.widget.app === applications[i].id) {
            this.application = applications[i];
            this.type = applications[i].name;
          }
        }
        this.loadComponent();
      },
      () => {
        this.application = null;
        this.type = 'error';
      }
    );
  }

  /**
   * Checks state of the component
   */
  isState(state: String) {
    return state === this.dashboardState;
  }

  /**
   * Opens application in popup if available, otherwise opens
       ErrorPopupComponent
   */
  openInPopUp() {
    let popupContent = ApplicationComponent.getComponentType(this
        .type + '-popup');
    if (!popupContent) {
      popupContent = ApplicationComponent.getComponentType('error
          -popup');
    }
    const popup = this.popupService.open(popupContent,
        {size: 'lg'});
    popup.componentInstance.widget = this.widget;
  }

}
```

Kód D.1: Angular komponent Application - Zabezpečujúci dynamické načítanie komponentov aplikácií

```
<div>
  <div class="container" *ngIf="isState('normal')">
    <div class="row">
      <div class="col" align="right">
        <a (click)="openInPopUp()"><i class="fa fa-arrows open-
            icon" aria-hidden="true"></i></a>
      </div>
    </div>
  </div>
  <div #container></div>
</div>
```

Kód D.2: Šablóna komponentu Application

Dodatok E

Inštalačná príručka

Aplikácia sa skladá zo serverovej a klientskej časti. Pred spustením klientskej aplikácie je nutné mať spustený server.

E.1 Server

Nasledujúce body popisujú postup spustenia Django aplikácie v termináli, počúvajúcej na porte 8000. Pri zmene tohto portu je potrebné upraviť aj port v klientskej aplikácii, a to v súbore settings.ts.

1. Inštalácia závislostí zo súboru requirements.txt: pip install -r requirements.txt

2. Vytvorenie Superuser účtu príkazom: python3 manage.py createsuperuser

3. Pripravenie databázy: python3 manage.py makemigrations dash_app python3 manage.py migrate dash_app

4. Prihlásenie sa do administrácie: `http://localhost:8000/admin` a následná registrácia klientskej aplikácie na: `http://localhost:8000/o/applications/`

5. Spustenie servera pomocou príkazu python3 manage.py runserver

E.2 Klient

Spustenie klientskej aplikácie prebieha nasledovne:

1. Inštalácia Angularu, Angular-CLI (`https://angular.io/guide/quickstart`) a závislostí zo súboru package.json

2. Pridanie získaného client ID a secret (z registrácie aplikácie na serveri) do konštánt CLIENT_ID a CLIENT_SECRET v súbore settings.ts
3. Spustenie vývojového servera a kompilácie TypeScriptu pomocou príkazu: ng serve
4. Po úspešnej kompilácii otvorenie v prehliadači na `http://localhost:4200`

E.3 Obsah databázy

Pre možnosť využívania vytvorených aplikácii je nutné pridať o nich záznam do databázy. Ich dáta sú nasledovné:

```
App.objects.create(name='calendar', allows_small_sizes=False,
    has_backend=False)
App.objects.create(name='onenote', allows_small_sizes=False,
    has_backend=True)
App.objects.create(name='translate', allows_small_sizes=False,
    has_backend=False)
```

Kód E.1: Objekty aplikácií

Tieto údaje je možné pridať jednoducho cez Django administráciu (po prihlásení pod účtom Superuser), alebo cez terminál v Django shell E.1.

Dodatok F

Testovacie scenáre

F.1 Autentifikácia

F.1.1 Registrácia

Podmienky: Neprihlásený užívateľ.

1. Otvorenie aplikácie na /register URL
2. Pokus o registráciu bez vyplnených údajov -> Chyba formou notifikácie
3. Čiastočne vyplnené údaje -> Chyba formou notifikácie
4. Nesprávny tvar e-mailu alebo krátke/primitívne heslo -> Chyba formou červeného formulára
5. Správne vyplnené údaje -> Úspešná registrácia

Výsledok: OK
Poznámka: Boli objavené malé UI nedostatky v prehliadači Microsoft Edge - Červené čiary po stranách formulárov pri nesprávnom vyplnení hesla alebo e-mailu. Tieto formuláre však pochádzajú z MDB frameworku, a tak nie je možnosť čiary odstrániť. Neobmedzujú však funkcionalitu aplikácie.

F.1.2 Prihlásenie

Podmienky: Neprihlásený, ale registrovaný užívateľ.

1. Otvorenie aplikácie na /login URL
2. Pokus o prihlásenie bez vyplnených údajov -> Chyba formou Alert notifikácie
3. Čiastočne vyplnené údaje -> Chyba formou Alert notifikácie

4. Chybné meno alebo heslo -> Chyba formou Alert notifikácie: Nesprávne prihlasovacie údaje

5. Správne vyplnené údaje -> Presmerovanie na /home URL

Výsledok: OK

F.1.3 Odhlásenie

Podmienky: Prihlásený užívateľ.

1. Prihlásenie

2. V menu: Profile – Log out -> Presmerovanie na /login URL

Výsledok: OK

F.2 Nastavenia

F.2.1 Nastavenia profilu

Podmienky: Registrovaný užívateľ.

1. Prihlásenie

2. V menu: Profile – Edit profile -> Otvorenie modálneho okna s nastavením profilu

3. Ikonka úpravy -> Otvorenie formuláru úpravy profilu

4. Zmena užívateľského mena -> Meno je zmenené a ukazuje sa zelená notifikácia

5. Zmena e-mailu -> E-mail je zmenený a ukazuje sa zelená notifikácia

6. Zmena e-mailu a mena -> E-mail aj meno sú zmenené a ukazuje sa zelená notifikácia

7. Uloženie bez zmeny -> Nič sa nezmenilo a ukazuje sa zelená notifikácia

Výsledok: OK

F.2.2 Nastavenie účtov

Podmienky: Priradené účty k widgetom.

1. Prihlásenie
2. V menu: Settings – Accounts -> Zobrazenie zoznamu účtov
3. Výber jedného z nich -> Otvorenie jeho detailu
4. Delete -> Zmazanie účtu a zobrazenie zelenej Alert správy
5. Delete všetkých účtov -> Prázdny zoznam účtov

Výsledok: OK

F.2.3 Nastavenia widgetov

Podmienky: Widgety na dashboarde.

1. Prihlásenie
2. V menu: Settings – Widgets -> Zobrazenie zoznamu aktívnych widgetov (ktoré sa nachádzajú na dashboarde užívateľa)
3. Rozkliknutie widgetu -> Zobrazenie detailu: názov aplikácie, opis aplikácie a informácia či má widget priradený účet (musí zodpovedať skutočnosti)

Výsledok: OK

F.3 Dashboard

F.3.1 'No widgets on your dashboard!' karta

Podmienky: Nový registrovaný užívateľ.

1. Prihlásenie -> Presmerovanie na /home a zobrazenie karty
2. Tlačidlo „Manage dashboard“ -> Prepnutie do editovacieho módu, zmiznutie karty
3. Prepnutie do normálneho módu -> Objavenie karty
4. Prepnutie do editovacieho módu, pridanie widgetu a uloženie zmien -> Zmiznutie karty, zobrazenie widgetu na dashboarde

Výsledok: OK
Poznámka: Bola objavená UI chyba vo Firefox prehliadači so zarovnaním tlačidla. Táto chyba bola opravená.

F.3.2 Pridávanie widgetov

Podmienky: Prihlásený užívateľ, prázdny dashboard.

1. Prepnutie do editovacieho módu
2. Tlačidlo „Add widget“ -> Zobrazenie modálneho okna so zoznamom aplikácii
3. Kliknutie na jednu z aplikácií -> Zmiznutie modálneho okna, objavenie nového widgetu na dashboarde na poslednej pozícii
4. Pridanie ďalších widgetov -> Objavenie na dashboarde v poradí pridania
5. Prepnutie do normálneho módu s pomocou tlačidla Done -> Widgety sú uložené a zobrazené v správnom poradí
6. Prepnutie do editovacieho módu a pridanie ďalších widgetov -> Objavenie widgetov na dashboarde
7. Uloženie -> Widgety sú na dashboarde

Výsledok: OK

F.3.3 Odobranie widgetov

Podmienky: Prihlásený užívateľ, widgety na dashboarde.

1. Prepnutie do editovacieho módu
2. Zmazanie jedného widgetu
3. Uloženie zmien -> Normálny mód bez zmazaného widgetu
4. Prepnutie do editovacieho módu
5. Zmazanie všetkých widgetov a uloženie -> Na dashboarde sa nachádza len ’No widgets on dashboard’ karta

Výsledok: OK

F.3.4 Presun widgetov

Podmienky: Prihlásený užívateľ, widgety na dashboarde.

1. Prepnutie do editovacieho módu
2. Presunutie/Vymieňanie widgetov
3. Uloženie zmien -> Normálny mód a správna pozícia každého widgetu
4. Opakovanie bodov 1-3 s rôznym počtom widgetov a rôznymi presunmi

Výsledok: OK

F.3.5 Zmena veľkosti widgetov

Podmienky: Prihlásený užívateľ, widgety na dashboarde.

1. Prepnutie do editovacieho módu
2. Zmena veľkosti jednotlivých widgetov
3. Uloženie zmien -> Normálny mód a widgety majú správne veľkosti

Výsledok: OK

F.3.6 Zrušenie editovacieho módu

Podmienky: Prihlásený užívateľ, widgety na dashboarde.

1. Prepnutie do editovacieho módu
2. Zmeny podľa scenárov 3.3 - 3.5
3. Zrušiť mód pomocou tlačidla Cancel -> Normálny mód a zmeny nie sú zaznamenané

Výsledok: OK

F.4 Aplikácie

Nasledujúce sekcie predstavujú oporné body pri testovaní aplikácií.

F.4.1 Translate

Podmienky: Translate widget na dashboarde.

- Preklady z jazykov - Kratšie aj dlhšie texty
- Zmena veľkosti widgetu - kontrola UI a zobrazení prekladov
- Test funkcií pri móde modálneho okna

Výsledok: OK

F.4.2 Google Calendar

Podmienky: Google Calendar widget na dashboarde.

- Pridanie účtu
- Zobrazenie udalostí - správny dátum, čas, text
- Pridanie udalosti

- Zmazanie udalosti
- Úprava udalosti
- Test funkcií pri móde modálneho okna

Výsledok: OK

F.4.3 OneNote

Podmienky: OneNote widget na dashboarde.

- Pridanie účtu
- Zobrazenie poznámok - poznámkové bloky, sekcie, stránky
- Pridanie poznámkového bloku, sekcie, stránky
- Odobranie stránky
- Úprava stránky
- Test funkcií pri móde modálneho okna

Výsledok: OK

Dodatok G

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `server` ..... implementácia serverovej časti
    - `client` ..... implementácia klientskej časti
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF