

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-79995

**CO-WORKING APLIKÁCIA PRE TEAMOVÉ
PROJEKTY
DIPLOMOVÁ PRÁCA**

2020

Bc. Karol Krist

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-79995

**CO-WORKING APLIKÁCIA PRE TEAMOVÉ
PROJEKTY**
DIPLOMOVÁ PRÁCA

Študijný program: Aplikovaná informatika
Číslo študijného odboru: 2511
Názov študijného odboru: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Ústav automobilovej mechatroniky
Vedúci záverečnej práce: Ing. Oto Haffner, PhD.
Konzultant: Ing. Erich Stark, PhD.

Bratislava 2020

Bc. Karol Krist

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Karol Krist
Diplomová práca:	Co-working aplikácia pre teamové projekty
Vedúci záverečnej práce:	Ing. Oto Haffner, PhD.
Konzultant:	Ing. Erich Stark, PhD.
Miesto a rok predloženia práce:	Bratislava 2020

Cielom tejto diplomovej práce je prieskum aktuálne existujúcich co-working aplikácií a analýza ich nedostatkov. Následne na základe tejto analýzy navrhnuť co-working aplikáciu, ktorá zahŕňa funkcionality existujúcich aplikácií ale zároveň odstraňuje nájdené nedostatky. Následne pomocou zvolených technológií túto aplikáciu implementovať a otestovať pomocou stanovených testov.

Kľúčové slová: Co-working, co-working aplikácie, teamové projekty, framework, FEI STU

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Karol Krist
Master's thesis:	Co-working application for team projects
Supervisor:	Ing. Oto Haffner, PhD.
Consultant:	Ing. Erich Stark, PhD.
Place and year of submission:	Bratislava 2020

The main goal of this diploma thesis is to research currently existing co-working applications and to analyze their shortcomings. Subsequently, based on this analysis to design a co-working applications that includes chosen functionality of existing applications but at the same time removes the shortcomings found. Then implement this application using chosen technologies for implementing similar applications. Finally test implemented application and analyze test results.

Keywords: Co-working, co-working applications, team projects, framework, FEI STU

Vyhlásenie autora

Podpísaný Bc. Karol Krist čestne vyhlasujem, že som diplomovú prácu Co-working aplikácia pre teamové projekty vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol Ing. Oto Haffner, PhD.

Bratislava, dňa 28.4.2020

.....
podpis autora

Podakovanie

Ďakujem môjmu vedúcemu práce Ing. Otovi Haffnerovi, PhD. za vedenie, zhovievavosť, pripomienky a cenné rady, ktoré mi poskytol pri vypracovaní tejto diplomovej práce. Tiež sa chcem poďakovať môjmu konzultatntovi Ing. Erichovi Starkovi, PhD. za jeho odbornú pomoc, rady, pripomienky a nápady, ktoré mi veľmi pomohli pri vypracovaní a pomohli mi aj pri rozvíjaní mojej osobnosti. Moje veľké poďakovanie patrí tiež rodičom za ich neustálu podporu počas celého štúdia na vysokej škole.

Obsah

Úvod	1
1 Co-working	2
1.1 História co-workingu	3
2 Co-working aplikácie	4
2.1 Slack	4
2.1.1 História	4
2.1.2 Používateľské rozhranie	4
2.1.3 Doplnky	5
2.1.4 Výber najznámejších importov	6
2.1.5 Cena	6
2.2 Facebook Workplace	7
2.2.1 História	7
2.2.2 Používateľské rozhranie	8
2.2.3 Cena	8
2.3 Microsoft Teams	8
2.3.1 História	9
2.3.2 Používateľské rozhranie	10
2.3.3 Doplnky	10
2.3.4 Cena	10
2.4 Trello	11
2.4.1 Kanbanská nástenka	11
2.4.2 História	12
2.4.3 Používateľské rozhranie	12
2.4.4 Doplnky	14
2.4.5 Cena	14
2.5 Nedostatky aplikácii	15
3 Možné technické problémy	16
3.1 Používateľské rozhranie	16
3.2 Používateľské skúsenosti	17
3.3 Výkon	17
3.4 Strata pripojenia na internet	17
3.5 Bezpečnosť	18

3.6	Zhrnutie	18
4	Použité technológie	19
4.1	Node.js	19
4.1.1	História	19
4.1.2	Architektúra Node.js	20
4.1.3	Event loop	21
4.1.4	Nevýhody Node.js	22
4.2	Node Package Manager	23
4.3	Ionic framework	24
4.3.1	Služby a vlastnosti	24
4.3.2	Inštalácia	25
4.4	Angular	26
4.4.1	História	26
4.4.2	Architektúra	27
4.4.3	Výhody Angularu	28
4.5	Electron	29
4.5.1	Architektúra	30
4.6	Apache CouchDB	30
4.6.1	Výhody použitia CouchDB	31
4.7	PouchDB	33
5	Návrh aplikácie	34
5.1	Špecifikácia požiadaviek	35
5.1.1	Používateľské požiadavky	35
5.2	Diagramy	36
5.2.1	Diagramy prípadov použitia	37
5.2.2	Sekvenčné diagramy	39
5.3	Databázový model CouchDB	40
6	Implementácia aplikácie	43
6.1	Prihlasovanie a registrácia	43
6.2	Stránka prihláseného používateľa	47
	Záver	49
	Zoznam použitej literatúry	50

Prílohy	I
A Štruktúra elektronického nosiča	II

Zoznam obrázkov a tabuliek

Obrázok 1	Y-base študentký co-working space na internáte STU ŠD Mladost Bratislava	2
Obrázok 2	Co-working centrum v San Franciscu	3
Obrázok 3	Slack aplikácia	5
Obrázok 4	Stránka Facebook Workplace	7
Obrázok 5	Aplikácia MS Teams	9
Obrázok 6	Jednoduchá kanbasnká nástenka	12
Obrázok 7	Náhľad nástenky v aplikácii Trello	13
Obrázok 8	Stránka profilu v aplikácii Trello	14
Obrázok 9	Jednoduchý program Hello World v Node.js	19
Obrázok 10	Architektúra platformy Node.js	21
Obrázok 11	Zjednodušená verzia fungovania Node.js architektúri	22
Obrázok 12	Jednoduchý <i>package.json</i> súbor	23
Obrázok 13	Vývojové prostredie Ionic studio	25
Obrázok 14	Princíp fungovania Angularu	27
Obrázok 15	Firmy využívajúce vo svojich aplikáciách Electron	29
Obrázok 16	JSON dokument v aplikácii na správu CouchDB	31
Obrázok 17	Náhľad PouchDB v prehliadači Google Chrome	33
Obrázok 18	Návrh komunikácie medzi komponentami	34
Obrázok 19	Use case diagram pre používateľské prostredie	37
Obrázok 20	Use case diagram pre administrátorské prostredie	38
Obrázok 21	Use case diagram prostredia člena tímu	38
Obrázok 22	Sekvenčný diagram prihlásenia a registrácie	39
Obrázok 23	Sekvenčný diagram komunikácie komponentov v aplikácii	40
Obrázok 24	Funkcia register na strane back-endu	44
Obrázok 25	Funkcia login na strane back-endu	45
Obrázok 26	Vytvorenie jwt tokenu	45
Obrázok 27	Prihlásenia a registrácia na strane front-endu	46
Obrázok 28	Overenie platnosti jwt tokenu	46
Obrázok 29	Načítanie dát prihláseného používateľa	47
Obrázok 30	Zobrazenie používateľových tímov	48
Obrázok 31	Použitie html elementu calendar	48

Zoznam skratiek a značiek

NPM - Node package manager

API - Application programming interface

I/O - input and output

JSON - JavaScript Object Notation

HTTP - HyperText Transfer Protocol

URL - Uniform Resource Locator

SDK - Software development kit

CLI - Command-line interface

JS - JavaScript

TS - TypeScript

NoSQL - non SQL alebo non relational

MVCC - Multi-Version Concurrency Control

jwt - Json web token

Úvod

Spolupráca, komunikovanie alebo tiež co-working skupiny ľudí je v dnešnej dobe nedomyšliteľnou súčasťou práce na dosiahnutí spoločného cieľa alebo výsledku. So spoluprácou sa stretávame denne v práci, v škole alebo aj pri voľnočasových aktivitách. Preto existuje v dnešnej dobe veľmi veľa aplikácií, programov, webových stránok alebo nástrojov, ktoré nám pomáhajú uľahčiť komunikáciu, spoluprácu alebo manažovanie v rámci skupiny ľudí alebo tímu. Avšak väčšina týchto pomocníkov je buď platená alebo ich funkcionálnosť je nedostatočná pre daný tím. Tak isto z praxe vieme povedať, že častokrát je potrebné si založiť viacero účtov u rôznych poskytovateľov určitých služieb aby sme si nakoniec vyskladali ideálneho pomocníka pre co-working.

Z reakcií ľudí používajúcich rôzne programy, aplikácie, nástroje uľahčujúce co-working - hlavne z radov študentov vieme povedať, že by ocenili novú co-working aplikáciu, ktorá by im uľahčila prácu na rôznych tímových projektoch do školy. Pre študentov je hlavným nedostatkom terajších aplikácií hlavne ich cena. Väčšina aplikácií si vyžaduje poplatok za dlhodobé používanie prípadne sa vyžaduje poplatok za dodatočné služby, ktoré by študenti ocenili.

Vývoj takýchto aplikácií v dnešnej dobe ale nie je až tak náročný preto veľa firiem si vytvorilo vlastnú aplikáciu, ktorá je používaná len v rámci danej firmy. Preto sme sa rozhodli vytvoriť použitím moderných technológií na tvorbu aplikácií co-working aplikáciu zameriavajúcu sa hlavne na potreby študentov pri práci na tímových projektoch.

1 Co-working

Co-working je v samotnej podstate model poskytovania obchodných služieb, ktorý zahŕňa jednotlivcov, ktorí pracujú nezávisle alebo spolupracujú v spoločných kancelárskych priestoroch. Toto zdieľanie jedných priestorov umožňuje tejto skupine ľudí zdieľať hodnoty, skúsenosti, nápady a profitovať zo synergického efektu, ktorý prináša sústreďenie talentovaných pracovníkov na jednom mieste. Tiež umožňuje spoznávanie nových ľudí, ktorí môžu byť v budúcnosti pre človeka dôležitý. Najčastejšími využívatelmi co-workingu sú napríklad obchodníci, umelci, programátori, konzultanti, dizajnéri, makléri, agenti, malé a začínajúce firmy.



Obr. 1: Y-base študentký co-working space na internáte STU ŠD Mladosť Bratislava

Pre účely co-workingu sa po celom svete začali budovať tzv. co-working centrá, ktorých počet každým rokom rastie. Vznik týchto centier je výsledkom hľadania stratégií, ako sa vyrovnáť s rizikami a problémami nových, flexibilných štýlov práce. Veľa centier bolo založených internetovými podnikateľmi, ktorí chceli nájsť alternatívu k práci z kaviarní alebo k izolácii spôsobenej prácou z domu. Tieto centrá sú zväčša ihneď pripravené a vybavené potrebnými vecami k práci ako sú stoly, stoličky, tabule na písanie, kuchynka, internetové pripojenie, mítingové priestory prípadne aj sieťová tlačiareň. To znamená, že pre bežného používateľa nevyžaduje začatie práce v takomto centre žiadne počiatkové alebo investičné náklady. Jediným poplatkom je zaplatenie členstva v centre.

1.1 História co-workingu

Slovo coworking v zmysle popisu ľudí, ktorí pracujú v akomkoľvek prostredí prvý krát použila Bernie DeKoven v roku 1995. K prvému vzniku coworkingového priestoru respektíve centra prišlo ale až v roku 2006 v San Franciscu pod vedením Brada Neuberga. Toto centrum sa nazývalo *San Francisco Coworking Space* a bolo otvorené len 2 dni v týždni. Kuriozitou je, že prvý mesiac ho nikto nevyužíval lebo hoci termín coworking bol známy ale nikto ho nepočul pod významom ako coworking miesto alebo centrum.



Obr. 2: Co-working centrum v San Franciscu

Dnes sa vytváranie coworkingových centier pokladá za globáln fenomén, ktorý v niektorých mestách dosahuje ročný nárast až 24,2%. Predpokladom je, že do roku 2022 bude celostvetovo vytvorených vyše 30500 centier alebo priestorov a viac ako 5,1 milióna členov. Coworking je nová cesta trvalo udržiavateľného spájanie pracovného a osobného života. Je to globálny pilier, ktorý bude formovať spôsob našej práce v budúcnosti.

2 Co-working aplikácie

Co-working aplikácie sú aplikácie, ktoré slúžia na komunikáciu a manažovanie tímu pri práci viac ľudí na dosiahnutie určitého cieľa alebo výsledku. Týchto aplikácii na trhu existuje veľa pričom niektoré sú bezplatné, niektoré sú platené a niektoré sú vyvinuté priamo vo firme kde sa používajú a nemá k nim teda prístup nikto okrem danej firmy. Medzi najznámejšie patrí napríklad Slack, Facebook workplace, Trello, Microsoft Teams a iné. Tieto aplikácie majú slúžiť ako náhrada za coworkingové centrum v online svete. Keď si zoberieme služby, ktoré ponúkajú coworkingové centrá tak sa dajú prirovnať k niektorým službám, ktoré ponúkajú aplikácie. Či už sa jedná o mítingové miestnosti - v aplikácii online chatové miestnosti v rámci tímu alebo o možnosť spoznávania nových ľudí osobne - v aplikácii online.

2.1 Slack

Slack je komunikačný nástroj vyvinutý pre pracovné využitie spoločnosťou Slack Technologies – „jedno miesto pre posielanie správ, nástroje a súbory“. To znamená, že Slack je softvér na posielanie „okamžitých“ správ s možnosťou pridania ďalších doplnkov podľa potreby používateľa. Tieto doplnky ale nie sú potrebné na plynulý chod aplikácie pretože základnou funkciou Slacku je len posielanie správ. Na Slacku existujú 2 spôsoby komunikácie: 1. spôsob sú tzv. kanály čo je v podstate skupinový rozhovor a 2. spôsob sú priame správy medzi dvoma ľuďmi.

2.1.1 História

Slack začal ako aplikácia na internú komunikáciu v spoločnosti Stewarda Butterfielda Tiny Speck pri vývoji online hry Glitch – hra bola vydaná v septembri 2011. Pre širšiu verejnosť bol Slack vydaný v auguste 2013. Slack je skratka pre: „Searchable Log of All Conversation and Knowledge“ čo vo voľnom preklade znamená „Protokol prehľadávania všetkých konverzácií a znalostí.“

V marci 2015 spoločnosť Slack oznámila, že bola vo februári 2015 napadaná hackermi počas 4 dní. Počas tohto útoku boli ohrozené údaje používateľov. Medzi tieto údaje patrili e-mailové adresy, používateľské mená, heslá, telefónne čísla a Skype mená ktoré boli priradené k ich účtom. Po tomto útoku Slack pridal do svojej aplikácie dvojfaktorovú autentifikáciu.

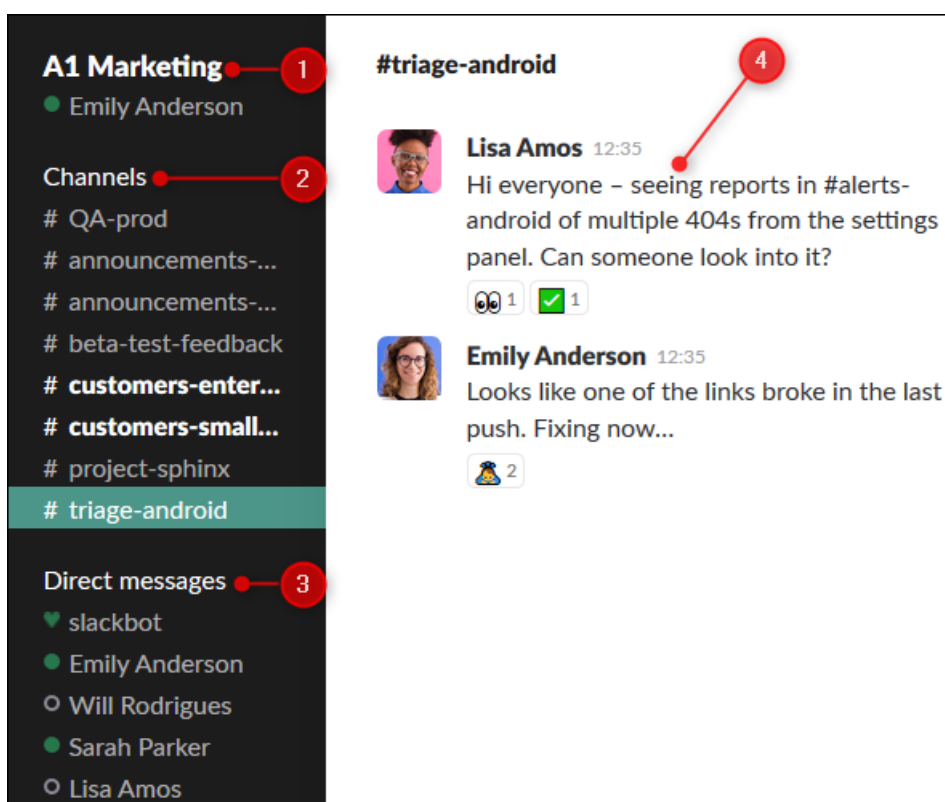
2.1.2 Používateľské rozhranie

Keď chce používateľ začať používať Slack musí najskôr cez stránku Slacku vytvoriť názov svojej „inštancie“ Slacku. Tento názov sa potom stane súčasťou URL adresy,

ktorá slúži ako pozvánka. Potom buď cez stránku Slacku alebo poslaním URL pozve ľudí, ktorých chce mať na svojom Slacku.

Po akceptovaní tejto pozvánky si používateľ vytvorí účet. Po prihlásení pod týmto účtom sa používateľovi otvorí stránka „inštancie“ Slacku alebo ak má nainštalovanú aplikáciu tak aplikácia Slacku vid. Obr. 3

Kanály na Slacku môžu byť verejné čo znamená, že každý člen skupiny ho vidí a môže sa k nemu pripojiť alebo súkromné čo znamená, že ich vidia len ľudia pridaný alebo pozvaný. Priame správy sú vždy súkromné ale môžu obsahovať až 8 ľudí.



Obr. 3: Slack aplikácia

1. Názov skupiny/inštancie
2. Zoznam kanálov
3. Zoznam kontaktov
4. Okno chatu

2.1.3 Doplnky

Do aplikácie Slack je možné pridať rôzne doplnky či už priamo vytvorené firmou Slack Technologies alebo inými firmami ako je Google, Jira, Trello a iné. Celkový prehľad

doplnkov Slack uvádza na svojej stránke kde sa vie používateľ priamo prekliknúť aj na stránku daného doplnku kde je napísané čo doplnok prináša do Slacku plus tutoriál ako ho používať prípadne video používania doplnku. Doplnok je možné pridať priamo cez aplikácie Slack alebo cez stránku Slacku.

Doplnky sú na stránke rozdelené do kategórii, do ktorých viac menej patria. Tieto kategórie sú rôzne od botov, ktorý automaticky ako je niečo zmenené na strane doplnkovej aplikácie, pridajú do daného kanála upozornenie alebo správu, až po priame importy, ktoré pridávajú funkcionality priamo do Slacku.

2.1.4 Výber najznámejších importov

- Google Drive
- Google kalendár
- Trello
- Twitter
- Outlook kalendár
- OneDrive
- GitHub
- Polly(hlasovania a prieskumy)
- Jira Cloud
- TimeBot
- WorkBot
- Giphy
- Doodle Bot

2.1.5 Cena

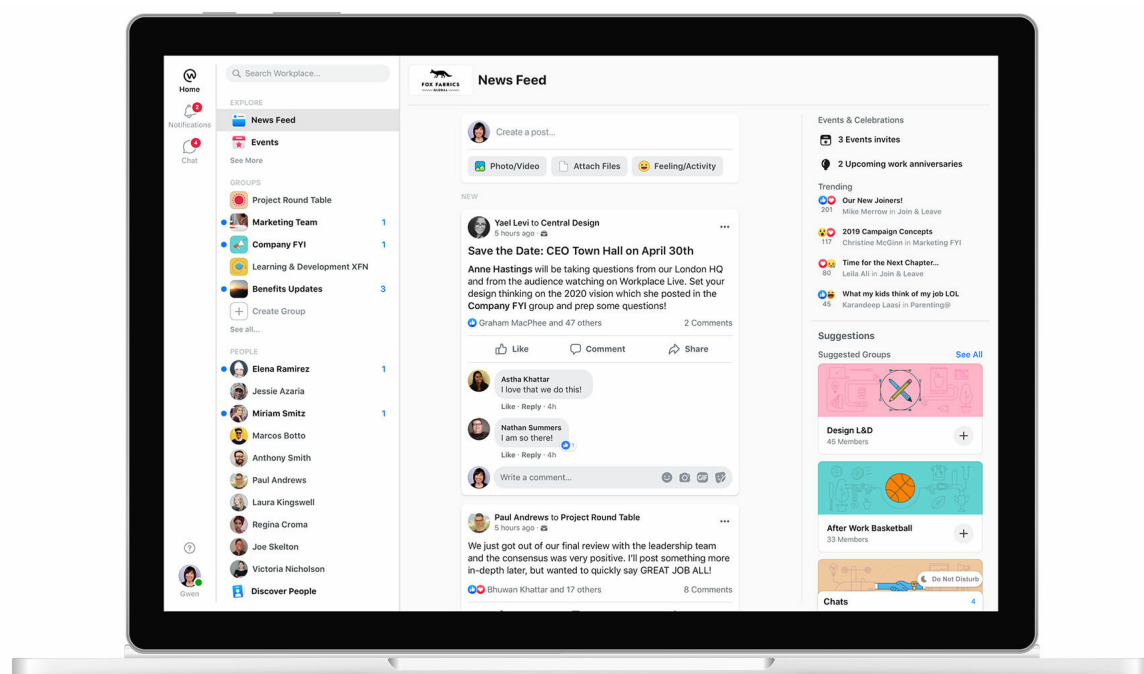
Slack ako taký je zadarmo ale sú tam isté obmedzenia. Hlavným obmedzením je prístup len k 10000 najnovším správam a používateľ môže pridať len 10 doplnkov na inštanciu. Ďalšími obmedzeniami sú napríklad žiadny jedno-kanálový alebo viac-kanálový hostia a limitované možnosti administrácie.

Ak chce používateľ sprístupniť celú funkcionality je to dosť drahé. Vychádza to približne 12 dolárov na používateľa mesačne pri ročnej platbe alebo približne 15 pri mesačnej

platbe. Ak teda napríklad máme 1000 člennú skupinu, ktorá chce používať Slack vychádza to približne 144000 dolárov pri ročnej platbe.

2.2 Facebook Workplace

Čas, ktorý ľudia strávili v práci chatovaním, prezeraním a likovaním príspevkov na Facebooku znižoval produktivitu ľudí cca v hodnote 3,5 bilióna dolárov. Táto téma sa pretriasala cez seriózne diskusie až po vytváranie vtipných obrázkov. Preto sa ľudia vo Facebooku začali zaoberať touto otázkou či sa funkcionality Facebooku nedá preniesť aj na pracovnú aplikáciu. Toto dalo počiatok Facebook Workplacu, ktorý podľa slov jeho tvorcov má podporiť kolaboráciu tým, že ju urobí zábavnou a jednoduchšou pre ľudí.



Obr. 4: Stránka Facebook Workplace

2.2.1 História

Myšlienka sa začala rozvíjať v roku 2014 keď prišla myšlienka nasmerovať tendenciu členov tímu prechádzať Facebook počas pracovnej doby k lepšej produktivite a spolupráci. Začiatkom roku 2015 sa začala do vybraných spoločností zavádzať beta verzia s názvom Facebook at Work. Keďže Facebook týmto vstupoval do neznámych vôd bolo samozrejmé, že beta verzia sa veľmi často menila. V polovici roka 2015 Facebook získal veľmi silného spojenca pre vývoj tejto aplikácie - The Royal Bank of Scotland – Škótska národná banka. Tá u seba presadila masívne využitie tejto beta verzie Facebook at Work kedy sa vytvorilo až 100000 nových účtov.

Ku koncu roka 2015 mal Facebook veľa kladných ohlasov na túto svoju vyvíjanú platformu od manažérov firiem kde bol Facebook at Work zavedený. Bolo to hlavne zapríčinené tým, že zamestnanci mali veľmi jednoduchý prístup k pracovným profilom svojich kolegov a podriadených. Tak isto kvôli prehľadnosti sa názov zmenil na Facebook Workplace.

V októbri 2016 boli oficiálne na trh uvedené pracovné a mobilné verzie aplikácie Facebook Workplace. Týmto Facebook začal konkurovať gigantom ako Slack a Microsoft Teams (vtedy Skype for Business) na poli podnikového softvéru.

2.2.2 Používateľské rozhranie

Rozhranie Workplacu je podľa slov používateľov na 95% rovnaké ako to na Facebooku. Väčšina populárnych funkcií Facebooku bola pridaná tiež do Workplacu:

- Newsfeed – zobrazujú sa tu všetky tímové aktivity ako napríklad príspevky členov tímu, firemné akcie a všetky informácie týkajúce sa práce prihláseného používateľa
- Live tools – funkcie na živý prenos (Live streaming)
- Skupiny – manažéri môžu vytvárať skupiny na pracovisku a tým udržiavať aktivity jednotlivých skupín/tímov na jednom mieste
- Správy – používatelia si môžu písať so svojimi kolegami

Pre používateľov je toto síce výhoda keďže sa nemuseli učiť používať nový nástroj avšak pre firmy je problém prijať v podstate napodobeninu Facebooku ako pracovnú aplikáciu. Kvôli tomuto Facebook aj oddelil profily Facebooku a Workplacu nakoľko do nedávna boli tieto profily prepojené a aj vytváranie účtu sa robilo cez profil Facebooku.

2.2.3 Cena

Workplace je pre firmy na prvé tri mesiace zadarmo. Potom sa jeho cena odvíja od počtu aktívnych účtov. Do 1000 používateľov sa platia 3 doláre za každého, do 10000 používateľov sa platí 2 doláre za každého a nad 10000 používateľov sa platí 1 dolár za každého. Pre neziskové organizácie a akademické účely je úplne zadarmo aj po 3 mesiacoch.

2.3 Microsoft Teams

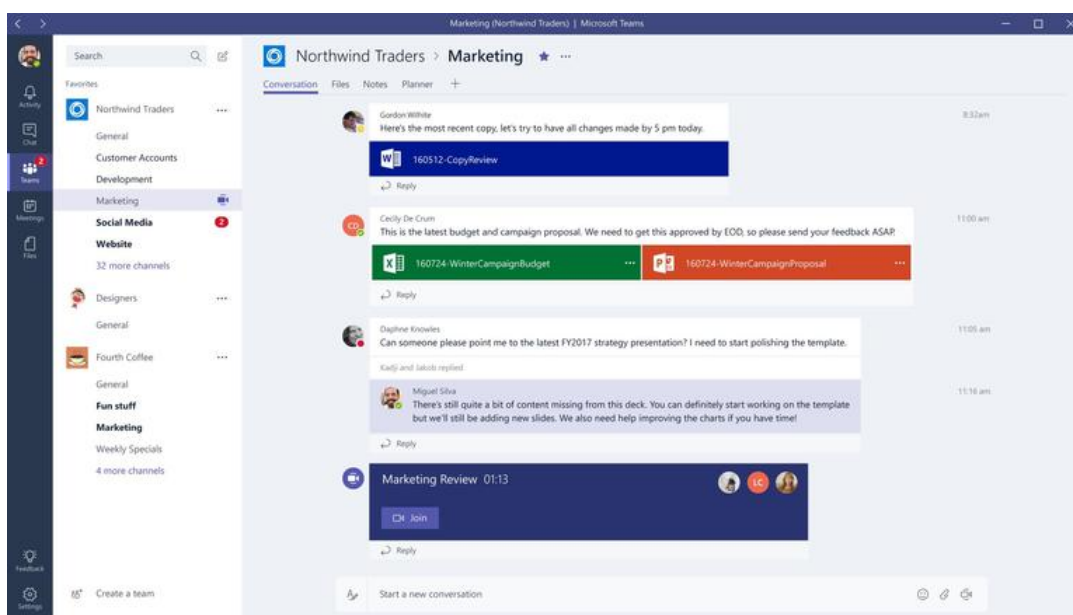
Microsoft Teams je aplikácia na pracovnú komunikáciu a kolaboráciu od spoločnosti Microsoft, ktorá bola vyvinutá aby konkurovala aplikáciám Slack, Workplace, HipChat. Vo svojej najjednoduchšej podobe je to aplikácia umožňujúca komunikáciu medzi členmi vytvorenej skupiny na báze miestností - kanálov. Microsoft sa už pred vydaním Teams pokúšal presadiť na trhu pracovných aplikácií pomocou svojho Skype for Business. Táto aplikácia avšak nenaplnila očakávania tak ako jej lepšia verzia Teams.

2.3.1 História

V marci 2016 chcel Microsoft kúpiť Slack za 8 miliárd dolárov avšak Bill Gates bol proti a skôr presadzoval zlepšenie ich aplikácie Skype for Business. Tento nákup presadzoval hlavne vysoko postavený pracovník Microsoftu Qi Lu. Ten avšak ešte v roku 2016 opustil spoločnosť a v Novembri toho istého roku Microsoft oznámil prácu na aplikácii Teams ako na aplikácii, ktorá má konkurovať Slacku.

Slack uznal Teams ako konkurenčnú službu avšak uviedol, že nekonkurujú rovnakému publiku nakoľko Teams v tej dobe neumožňoval ľuďom bez predplateného balíka Office 365 vstup do aplikácie Teams. Slack to zdôvodňoval aj tým, že neprepokladá sa, že by malé a stredné podniky, ktoré už Slack používajú nezačnú používať platenú službu Office, ktorej Teams bol súčasťou. Neskôr však Teams pridal možnosť pridania nových ľudí aj bez predplateného balíka Office 365. Slack na toto reagoval integráciou služieb od Google (Drive, Kalendár, Gmail).

V roku 2017 sa udiali 2 udalosti. Prvou bolo, že Teams nahradili MS Classroom v balíku Office 365 for Education. Druhou udalosťou bola správa v septembri, že Teams nahradzujú Skype for Business. Tým bola aj ukončená služba Skype for Business.



Obr. 5: Aplikácia MS Teams

V júli 2018 oznámil Microsoft bezplatnú verziu služby Teams s obmedzeniami typu počet používateľov a kapacitu ukladania súborov. V novembri 2019 dosiahol Teams 20 miliónov používateľov čo bol nárast o 7 miliónov od júla 2019.

2.3.2 Používateľské rozhranie

Rozhranie Teams je veľmi podobné tomu na Slacku. Na vytvorenie tímu treba vytvoriť URL, ktorá potom slúži rovnako ako na Slacku na pozívanie používateľov. Po prihlásení sa pomocou Microsoft účtu potom môžu už jednotliví používatelia vytvárať miestnosti, do ktorých môžu pridávať príspevky. Tak isto je možná komunikácia medzi používateľmi bez miestnosti štýlom ako je v aplikácii Skype. Tu je povolená aj hlasová a video komunikácia Skype štýlom.

Mimo klasických miestností-kanálov je možné vytváranie aj takzvaných meeting miestností, ktoré slúžia na komunikáciu počas pracovných porád, meetingov a stretnutí. Tieto miestnosti umožňujú priame pridanie času a dátumu meetingu do kalendára Outlooku. Náhľad nástienky po otvorení miestnosti je možné vidieť v akom stave stretnutie je: ešte nezačalo, prebieha, ukončené.

Microsoft do Teams pridal aj funkciu podobnú funkcionalite Trella – dashboard. Pomocou tejto funkcie môžu manažéri, učitelia, vedúci prideľovať a sledovať prácu používateľov. Tak isto sa jednotlivé úlohy na tejto „nástenke úloh“ dajú komentovať, presúvať alebo hodnotiť.

Samozrejmosťou je veľmi dobrá kompatibilita s inými aplikáciami Microsoftu ako je Word, Excel, PowerPoint, OneDrive. Jednotlivé dokumenty týchto aplikácií sa dajú priamo posilať, upravovať v aplikácii Teams a tak majú jednotliví používatelia vždy najaktuálnejšiu verziu dokumentu k dispozícii. Microsoft pridal ale aj komunikáciu s aplikáciami od iných vývojárov ako je GitHub, Evernote, Zendesk pomocou konektorov. Táto komunikácia slúži najmä na posielanie upozornení, že na strane druhej aplikácii prišlo k nejakej zmene – napríklad príde upozornenie do miestnosti, že niektorý používateľ pusol na git niečo nové.

2.3.3 Doplnky

Do Teams podobne ako do Slacku je možné pridanie rôznych doplnkov od iných vývojárov. Tieto doplnky sú avšak po väčšine len Boti, ktorých úlohou je posielanie upozornení, že na strane druhej aplikácie prišlo k nejakej zmene – pridanie nových vecí na github, zmena v Evernote a iné. Týchto Botov je potvrdených zatiaľ 85 a 70 konektorov – aplikácie, ktoré slúžia nielen na posielanie upozornení.

2.3.4 Cena

Microsoft Teams je v svojej jednoduchšej podobe zadarmo. Obmedzeniami bezplatnej verzie sú napríklad prístup na OneDrive, plánovania schôdze, nahrávanie schôdze pomocou Microsoft Stream, technická podpora, viacfaktorová autentifikácia pre všetkých

používateľov.

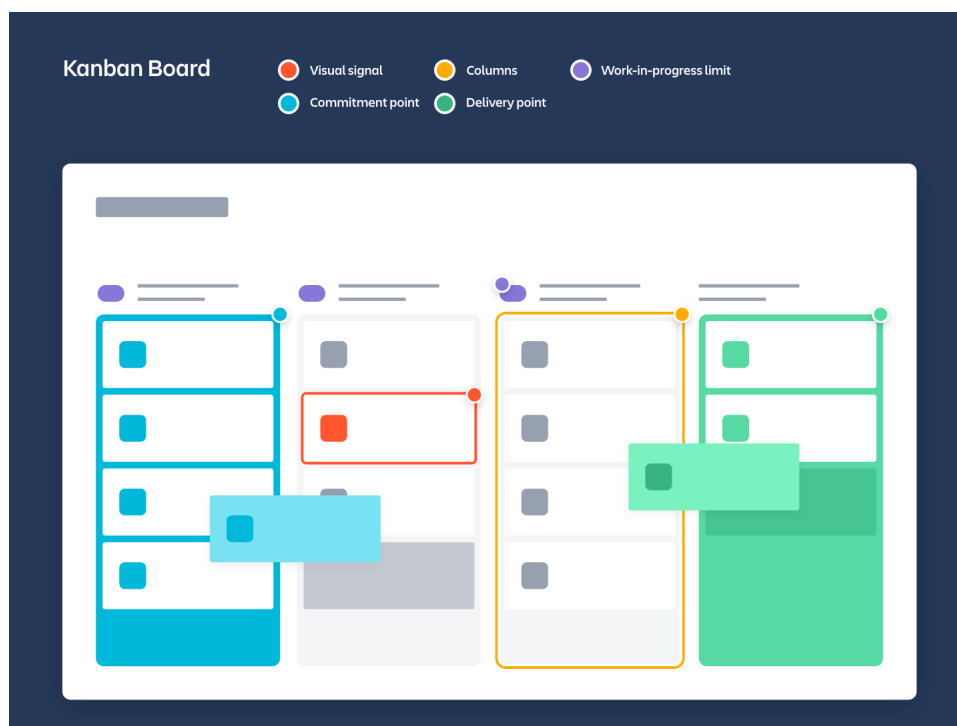
Platených verzii je viac. Tieto verzie sú viazané na balík Office 365, ktorý má používateľ/firma zaplatený. Za balík Office 365 Business Essentials pýta Microsoft 4,20€ za používateľa mesačne s ročnou viazanosťou a za balík Office 365 Business Premium 10,50€ za používateľa mesačne tak isto s viazanosťou na rok. V balíku Premium je zahrnutá všetka funkcionálna aplikácia Teams. Pri balíčku Essentials sú tam stále isté obmedzenia.

2.4 Trello

Trello je aplikácia na vytváranie pracovnej nástenky v kanbanskom štýle. Aplikácia od roku 2017 patrí spoločnosti Atlassian avšak vydaná bola v roku 2011 spoločnosťou Fog Creek Software. Aplikácia obsahuje virtuálnu nástenku kde členovia tímu môžu vytvárať, organizovať a priradovať úlohy v rámci projektu. Použitý kanbanský/kartový štýl umožňuje členom tímu vzájomne spolupracovať a komunikovať pri práci na projekte. Používatelia môžu k projektovým kartám pridávať komentáre, odkazy, súbory a fotografie. Trello existuje v rôznych podobách. Existuje webová aplikácia, desktopová aplikácia či už pre Windows alebo MacOS a tak isto existujú aj mobilné aplikácie pre Android a iOS. Existuje aj import aplikácie do Slacku.

2.4.1 Kanbanská nástenka

Kanbanská nástenka je agilný nástroj na riadenie projektov navrhnutý tak, aby pomohol vizualizovať priebeh projektu a maximalizovať efektívnosť. Na kanbanskej nástenke sa používajú karty, stĺpce a neustále zlepšovanie. Toto má za následok jednoduchú vizualizáciu prebiehajúcej práce a tým vedúcim tímov pomáhať lepšie manažovať prácu tímu.



Obr. 6: Jednoduchá kanbasnká nástenka

2.4.2 História

Trello bolo spustené v roku 2011 spoločnosťou Fog Creek. Za celou myšlienkou bol hlavne zakladateľ spoločnosti Joel Spolsky. Magazín Wired zaradil aplikáciu do „The 7 Coolest Startups You Haven´t Heard of Yet“ – Najlepších 7 startupov, o ktorých ste nepočuli. Tak isto sa v článku objavil názor, že Trello uľahčuje a svojim spôsobom spríjemňuje prácu na projektoch.

V máji 2016 ohlásilo Trello, že dosiahlo 1,1 milióna aktívnych používateľov denne a 14 miliónov celkovo založených účtov. V januári 2017 spoločnosť Atlassian kúpila Trello za 425 miliónov dolárov s tým, že 22% podielu stále ostávala v rukách investorov a zakladateľa Joela Spolskeho. V roku 2019 nastal rapidný nárast používateľov kedy ešte v marci malo Trello 35 miliónov používateľov ale už v októbri to bolo 50 miliónov.

2.4.3 Používateľské rozhranie

Prostredie aplikácie je veľmi jednoduché na používanie a nemá v sebe veľa zbytočných vecí. Tak isto navigácia v prostredí je veľmi jednoduchá a intuitívna. Registrácia je veľmi jednoduchá. Jediné čo nový používateľ potrebuje je meno, heslo a email.

Hneď po prihlásení sa používateľovi zobrazí nástenka kde vidí tímy, v ktorých je pridaný. Tak isto môže vytvoriť novú kartu tímu. Po vytvorení sa otvorí okno tímu, ktoré zväčša pozostáva z 3 stĺpcov – To Do, Doing a Done plus možnosť pridať ďalšie stĺpce.

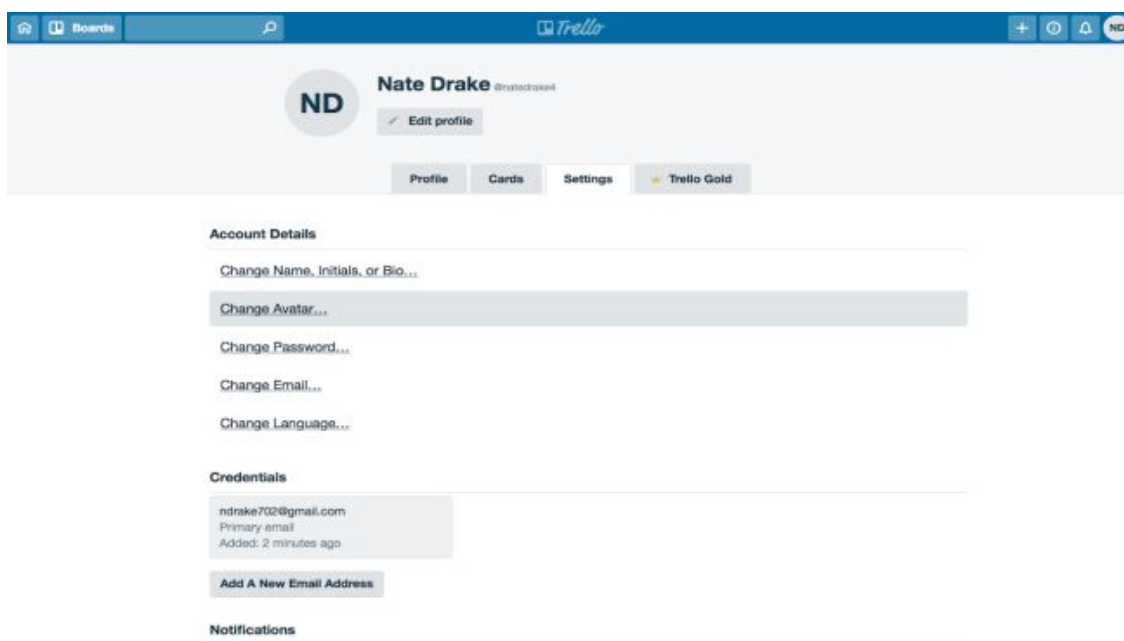
Tri dopredu vytvorené stĺpce sa samozrejme dajú vymazať alebo premenovať.



Obr. 7: Náhľad nástenky v aplikácii Trello

V jednotlivých stĺpcoch je potom možné vytvárať nové karty, ktoré reprezentujú nejakú úlohu, komunikáciu alebo čokoľvek čo tím potrebuje. Po kliknutí na kartu sa zobrazí okno kde sa nachádza popis karty, komentáre a možnosti čo sa dá s kartou robiť – priradiť kartu používateľom aby pri zmene na karte dostali upozornenie. Upozornenia sa dajú samozrejme aj vypnúť. Na jednotlivé nástenky je možné pridanie rôznych doplnkov ako je napríklad kalendár, Google drive, Slack, Mapy alebo špecifické doplnky.

Po kliknutí na profil sa otvorí stránka používateľského profilu. Tu je možné zmeniť meno, pridať fotku profilu, zmeniť iniciály, zmeniť avatara z iniciálok na fotku, nastaviť politiku upozornení alebo pre farebne slepých používateľov je tu možnosť zapnutia módu pre farebne slepých.



Obr. 8: Stránka profilu v aplikácii Trello

2.4.4 Doplnky

Do jednotlivých nástieniek alebo tímov sa dajú pridať rôzne doplnky, ktoré rozširujú možnosti použitia Trelly. Okrem rôznych Botov, ktorí len posielajú preddefinované správy alebo upozornenia je ponuka doplnkov veľmi veľká – od štandardných doplnkov ako je kalendár, Dropbox, Google Drive, Slack, GitHub, až po špecifické určené nie len na co-working ako napríklad Card Family, Prize Checker, Prize Tag a iné.

2.4.5 Cena

Trello ponúka 3 typy účtov. Bezplatný, ktorý zahŕňa neobmedzený počet nástieniek, kariet, členov tímu alebo príloh. Je možné pridať jeden doplnok na nástienku a je možné pridať prílohu o veľkosti do 10MB alebo prepojiť akýkoľvek súbor z Google Drive, Dropbox alebo OneDrive účtu.

Business balík stojí 9,99\$ mesačne ale platí sa ročné predplatné. Balík ponúka to isté čo bezplatný balík s tým, že je možné pridať neobmedzený počet doplnkov na nástienku a je možné pridať 250MB prílohy. Ďalšími rozširujúcimi vecami je možnosť usporiadania nástieniek do kolekcií, nastaviť kto tieto kolekcie môže vidieť alebo nahrať vlastných pozadí pre nástienky.

Enterprise balík stojí 20,83\$ mesačne pri ročnom predplatnom. Balík zahŕňa funkcionality Business balíka plus dvojfaktorová autentifikácia, šifrovanie súborov, vlastná kontrola zabezpečenia, vylepšené SLA.

2.5 Nedostatky aplikácii

Pri každej aplikácii okrem Facebook Workplace sme mohli vidieť, že využívajú rôzne doplnky. Pre použitie týchto doplnkov avšak musí používateľ zväčša disponovať účtom založeným v danej platforme. Napríklad ak chce používateľ v Slacku využívať Google kalendár musí mať založený účet na platforme Google. Táto skutočnosť je častokrát medzi používateľmi považovaná za otravnú a častokrát má používateľ potom zbytočne vytvorené účty na veľkom množstve platform, ktoré zvyknú používateľa aj spamovať na email. Preto do našej aplikácie chceme importovať služby, ktoré sú zväčša najžiadanejšie medzi službami ponúkanými doplnkami.

Druhým veľkým nedostatkom, ktorý by sme chceli našou aplikáciou odstrániť je nutnosť platenia. Je samozrejmé, že pre veľké firmy nie je problém si radšej priplatiť za stabilnú a 100% funkčnú aplikáciu, ktorá poskytuje veľké množstvo služieb aj za cenu pridania veľkého množstva doplnkov. Avšak pre študentov je odstránenie potreby platenia za aplikáciu veľkým prínosom. Jedná sa hlavne o odstránenie limitu používateľov bez poplatku za aplikáciu. Pri niektorých aplikáciach je avšak poplatok aj za určité služby v podobe doplnkov. Toto sa v našej aplikácii tak isto nachádzať nebude. Každý používateľ bude mať plný a rovnaký prístup k aplikácii. Za základ našej aplikácie sme si zvolili rovnakú funkcionality ako má Slack bez doplnkov čiže vytvorenie tímov a komunikácia v nich. K tomu pridáme funkcionality doplnkov ako je kalendár s udalosťami, možnosť hlasovania v tíme a jednoduchú nástenku úloh v tíme podobnú ako má Trello.

3 Možné technické problémy

Pri tvorbe akéhokoľvek softvéru sa vývojár stretne s množstvom situácií, ktoré môžu vyvolať pri používaní vyvíjaného softvéru problém. Preto musí vývojár už počas vyvíjania tohto softvéru tieto možné problémy odstrániť. V tejto kapitole sa preto budeme venovať možným technickým problémom, ktoré sa môžu vyskytnúť v našej pripravovanej aplikácii a načrtneme ako tieto problémy plánujeme odstrániť.

Preto aby sme správne identifikovali možné problémy si najskôr musíme ale povedať ako bude naša aplikácia vyvíjaná. Po zvážení sme sa rozhodli, že aplikácia bude vyvíjaná princípom web-aplikácie s tým, že bude mať na pozadí aj serverovú časť, ktorá bude zodpovedná za komunikáciu s databázou.

3.1 Používateľské rozhranie

Pred pár rokmi pri vývoji web-aplikácii sa vývojár nemusel zamýšľať nad tým na akom zariadení bude používateľ spúšťať web-aplikáciu - smartfóny neexistovali a jediné zariadenie, na ktorom mohol používateľ otvoriť web-aplikáciu bol počítač cez prehliadač. Dnes avšak treba myslieť na to, že používateľ môže aplikáciu otvárať na smartfóne, tablete alebo inom zariadení. Preto musia byť web-aplikácie vyvíjané spôsobom aby sa zobrazovaná stránka vedela natívne prispôbiť veľkosti obrazovky, na ktorej je zobrazovaná.

Ďalším faktorom je intuitívnosť využívania aplikácie. V dnešnej dobe sú v oblube jednoduché aplikácie, ktoré vie používateľ intuitívne používať. Či už sa jedná o prehľadnosť aplikácie alebo o navigáciu v aplikácii. Pokiaľ si využívanie aplikácie žiada najskôr si preštudovať dopodrobna ako aplikáciu používať, aplikácia stratí lojalitu používateľov.

Na odstránenie toho potencionálneho problému sme sa rozhodli našu aplikáciu vyvíjať v prostredí Ionic, ktoré nám umožňuje aplikáciu vyvíjať zároveň pre viac platform. V Ionicu je možné aplikáciu vyvíjať pre štandardný webový prehliadač, avšak zároveň je aplikácia vyvíjaná aj pre webové prehliadače v smartfónoch a tabletoch. Samozrejme je ale potrebné využívanie responzívnych prvkov a elemntov aby sa vedeli automaticky prispôbiť veľkosti obrazovky. Pred začatím vývoja sme sa ale dohodli web-aplikáciu po implementovaní vyexportovať pomocou frameworku Electron do štandarnej aplikácie pre platformu Windows. Pomocou Electronu je ale možné aplikáciu vyexportovať na akúkoľvek platfomu či už Linux, MacOS, Android alebo aj iOS len zmenou príkazu na exportovanie. Týmto by sa mal tento potencionálny problém s používateľským rozhraním odstrániť.

3.2 Používateľské skúsenosti

Používatelia sa pri prechode na novú aplikáciu obávajú hlavne o to či ju budú vedieť používať. Preto v dnešnej dobe sa častokrát veľa aplikácii zameraných na určitú potrebu používateľa veľmi podobá. Je to hlavne preto aby používateľ mohol hneď s aplikáciou pracovať a nemusel sa učiť ako ju používať. Tento problém bol už čiastočne spomenutý pri probléme s používateľským rozhraním nakoľko je to s ním úzko späté. Na odstránenie tohto problému sme sa preto rozhodli vyvíjať našu aplikáciu tak aby bola podobná aplikácii Slack čo sa týka vizuálnej stránky. Používateľ, ktorý používal už v minulosti aplikáciu Slack bude hneď vedieť ako využívať aj našu aplikáciu avšak tak isto bude design našej aplikácie prispôsobený tak aby ja používateľ, ktorý nevyužíval podobnú aplikáciu hneď vedel ako našu aplikáciu využívať.

3.3 Výkon

Rýchlosť načítavania aplikácie a celková rýchlosť jednotlivých služieb, ktoré aplikácia ponúka je jedným z hlavných faktorov na prilákanie používateľov. Preto je potrebné aplikáciu vyvíjať tak aby jednotlivé služby neboli náročné na či už výpočtový čas alebo na hardvér zariadenia, na ktorom aplikácia beží.

Na odstránenie tohto problému bude preto potrebné aplikáciu implementovať bez zbytočných chýb v kóde a písať kód čo najjednoduchšie. Tak isto bude potrebné aby sme nevyužívali príliš náročné doplnky. Pri tomto probléme bude tiež potrebné dbať na rýchlosť načítavania a ukladania do databázy. Po preštudovaní viac potencionálne využiteľných databáz sme sa rozhodli využiť databázu CouchDB s replikovaním na lokálnu databázu PouchDB. Toto využitie dvoch databáz rozoberieme pri probléme so stratou pripojenia na internet.

3.4 Strata pripojenia na internet

Tento problém môže nastať ktorémukoľvek používateľovi. Ake sme už vyššie spomenuly naša aplikácia bude vyexportovaná ako desktopová aplikácia čiže spustiť bez pripojenia na internej ju pôjde avšak tu nastáva problém už pri prihlásení. Na prihlásenie bude potrebné byť pripojený na internet nakoľko sa prihlasovacie údaje musia overiť s online databázou. Potom avšak bude možné stratiť pripojenie na internet nakoľko využijeme replikáciu databázy na lokálnu databázu PouchDB. Táto databáza preberie všetky dáta pri prihlásení z online databázy a budeme môcť ich zobrazovať. Tak isto hneď pri akomkoľvek zápise je do online databázy je táto lokálne hneď aktualizovaná ak je teda zariadenie pripojené na internet. Ak pripojenie stratí bude zobrazovať len tie dáta, ktoré

boli v databáze do straty pripojenia.

3.5 Bezpečnosť

Bezpečnosť dát, ktoré používateľ zadá na stránke je samozrejmosťou každej stránky. Je samozrejmé, že v tomto bode je dosť obtiažne zabrániť všetkým potencionálnym hrozbám. Tu je možné aj zvážiť, ktoré dáta sú dôležitejšie a ktoré menej. Za ochranu dát v databáze samozrejme zodpovedá čiastočne už aj samotný softvér použitej databázy ale treba zabezpečiť aj zariadenie, na ktorom databáza beží a dáta keď sú vytiahnuté z databázy a pracuje sa s nimi.

V našej aplikácii sme sa hlavne zamýšľali na zabezpečení dát pri práci s nimi. Tu sme sa tiež rozhodovali, ktoré dáta bude treba zabezpečiť a ktoré nie. Tu sme za najdôležitejšie dáta zvolili prihlasovacie heslo používateľa. Ostatné dáta nie sú až také dôležité a aj s prihladením na náročnosť aplikácie sme sa rozhodli zabezpečiť len heslo. Heslo bude hneď pri registrácii na strane servera zašifrované pomocou prídavnej knižnice do node.js bcrypt a tak uložené do databázy. Po zašifrovaní už heslo nikdy nebude rozšifrované čiže pri prihlasovaní sa zadané heslo pomocou funkcie knižnice bcrypt overí so zašifrovaným heslom a na základe toho bude overená správnosť zadaného hesla.

Zabezpečovanie servera nebudeme relaizovať keďže aplikácia bude testovaná len na lokálnom stroji. Ak by sa ďalej uvažovalo o distribúciu aplikácie bude samozrejme možné zabezpečenie servera ako aj samotnej aplikácie rozšíriť.

3.6 Zhrnutie

Tieto problémy sú asi najzásadnejšie a najväčšie, s ktorými sa stretneme pri vývoji aplikácie. Jendým menej zásadným problémom, ktorý ale ešte stojí za zmienku je problém možnosti rozšírenia aplikácie. Niektoré aplikácie sú implementované tak, že je skoro nemožné k nim po vydaní dorobiť nejaké rozšírenie alebo dodatok. Pri implementovaní našej aplikácie na to budeme myslieť a bude implementovaná tak aby sa k nej jednoducho dala dorobiť ďalšia funkcionality.

4 Použité technológie

V nasledujúcej kapitole zhrnieme technológie, ktoré sme sa rozhodli použiť na implementovanie našej aplikácie. Ku každej technológii popíšeme možnosti, ktoré plánujeme využiť nakoľko rozpísať všetky možnosti je pre nás zbytočné.

4.1 Node.js

Node.js je na svojich stránkach popísaný ako JavaScript zameraný na udalosti pre tvorbu škálovateľných sieťových aplikácií. Na nasledovnom príklade jednoduchšej aplikácie Hello World môžeme vidieť, že sa mnoho spojení dá zvládnuť súčasne - Obr. 9. Po každom spojení sa spustí spätné volanie - callback, ale ak nie je potrebné vykonať žiadnu akciu program bude spať kým ho niekto nezavolá.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Obr. 9: Jednoduchý program Hello World v Node.js

Toto je v rozpore s dnešným bežnejším a modernejším prístupom založeným na vláknach operačného systému, ktoré bežia súčasne. Vytváranie vlákien pri sieťovej komunikácii je relatívne neefektívne a veľmi náročné na používanie. Mimo toho sú používatelia servera Node.js bez obáv takzvaného mŕtveho zablokovania procesu, pretože neexistujú žiadne zámky. Takmer žiadna akcia v Node.js priamo nevykonáva I/O, takže proces sa nikdy neblokuje. Preto sa odporúča na škálovateľné aplikácie používať Node.js.

4.1.1 História

Node.js bol vytvorený Ryanom Dahlom v roku 2009 a bol podporovaný len pre platformy Linux a MacOS. Pre porovnanie prvé prostredie pre JavaScript na strane servera LiveWire Pro Web bolo uvedené na trh v roku 1996. Údržba a ďalší vývoj bol najskôr

vedený len samotným Dahlom, ale neskôr bol sponzorovaný firmou Joyent. Dahl prezentoval Node.js 8. Novembra 2009 na konferencii European JSConf. Node sa vtedy skladal z JavaScriptového enginu V8 od spoločnosti Google, udalostnej slučky (event loop) a nízko-úrovňového I/O API.

V januári 2010 prišiel na trh NPM - správca balíkov pre Node.js. Tento správca umožňuje používateľom Node.js zdieľanie a publikovanie zdrojové kódy doplnkových modulov. Zároveň uľahčuje ich sťahovanie, inštaláciu a aktualizáciu. Neskôr v lete 2011 Microsoft a Joyent spolupracovali na vytvorení podpory Node.js pre Windows, ktorú v tom roku aj vydali.

V roku 2014 po nezhodách pri vývoji sa časť komunity odštiepila a vytvorila io.js. O rok neskôr vznikla, ale Node.js Foundation, ktorá zjednotila roztrieštenú komunitu a zjednotila Node.js v0.12 a io.js v3.3 do Node v4.0. Toto zjednotenie prinieslo novinky z ES6 z V8 a zároveň umožnila dlhodobý trvajúci vývoj platformy.

4.1.2 Architektúra Node.js

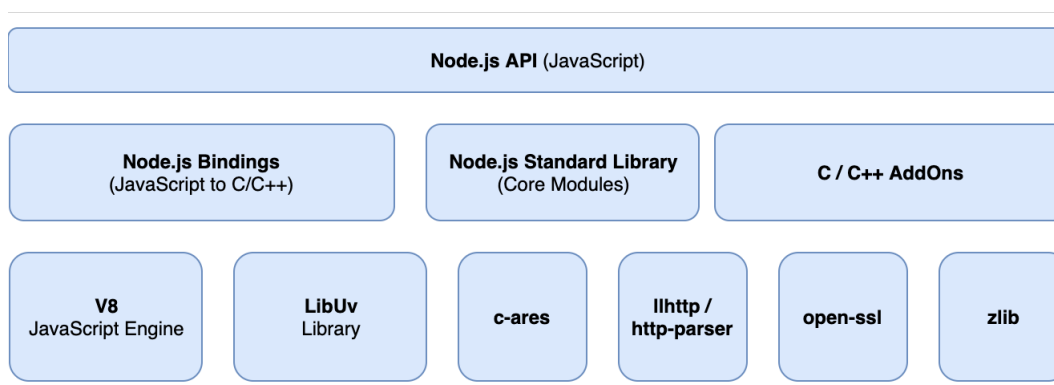
Každá platforma má svoju architektúru, tak isto aj Node.js. V tejto kapitole rozoberieme jej hlavné vlastnosti a časti. Nie všetky budeme potrebovať pri implementácii ale je dobré ich poznať.

Základnou vlastnosťou je, že Node.js má asynchrónne a udalostne riadené API. Toto zaručuje to, že nasledujúce volania nie sú blokové predchádzajúcim volaním. V praxi to znamená, že Node.js nikdy nečaká kým je z volania vrátená hodnota respektíve je volanie dokončené a ukončené. Server sa po zavolaní volania presunie na ďalšie a vnútorný notifikačný mechanizmus udalosti dostane odpoveď z prechádzajúcich volaní vtedy keď sú dokončené a dostane sa tým aj k výsledku.

Druhou veľkou výhodou je existencia udalostnej slučky. Vďaka tomu je jednovláknový a vysoko škálovateľný. Vyššie spomínaný udalostný mechanizmus napomáha serveru vrátiť odpoveď bez blokovania. Tým sa server stáva vysoko škálovateľný v porovnaní s tradičnými riešeniami na strane servera, ktoré majú obmedzený počet vlákien a tým sa aj počet požiadaviek, ktoré sú spracované súčasne stáva obmedzený.

Ďalšou výhodou je, že aplikácie založené na platforme Node.js sú bez vyrovnávacej pamäte, čiže údaje posielené na výstup sú zoskupované do malých blokov.

Na Obr. 10 môžeme vidieť základné rozdelenie architektúry platformy Node.js.



Obr. 10: Architektúra platformy Node.js

Na vrchole sa nachádza Node.js API, ktoré je napísané v JavaScripte a je možné k nemu priamo pristupovať za účelom využitia v aplikáciach. Pod API sa nachádza trio Node.js Bindings, Node.js Standard Libraries a C/C++ AddOns. Node.js Standard Libraries sú funkcie súvisiace s knižnicami operačného systému a umožňujú využívanie napríklad časovačov, súborového systému alebo sieťových volaní http. Node.js Bindings je knižnica, ktorá umožňuje komunikáciu JavaScriptu s C/C++ a tým ich viaže dokopy. Poslednou časťou sú C/C++ AddOns čo sú dynamicky linkované doplnky viazané na C/C++ knižnice. Toto umožňuje vytvorenie akejkoľvek C/C++ knižnice a jej využitie v Node.js.

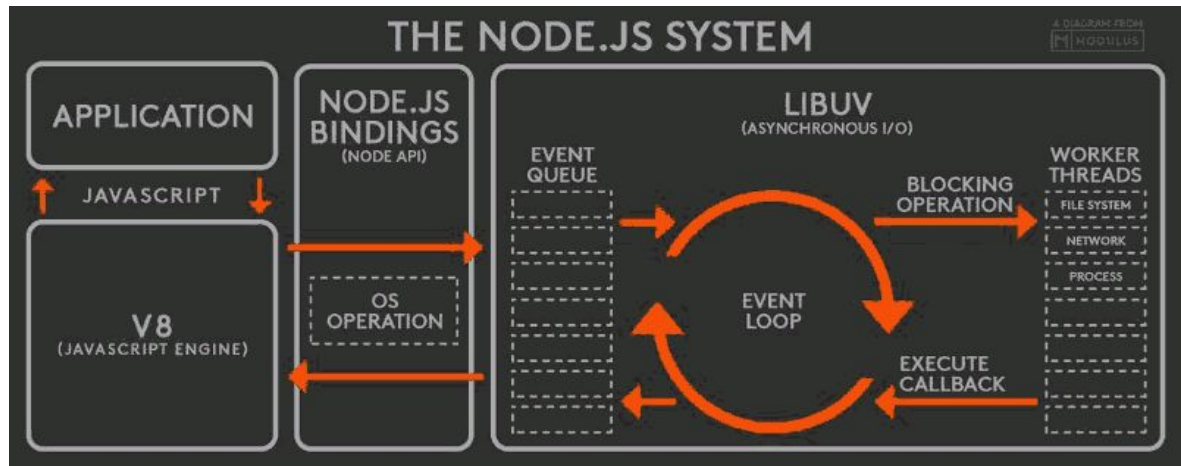
Na spodku sa nachádzajú knižnice C/C++:

- V8 JavaScript Engine - konvertuje JavaScript do strojového kódu daného operačného systému
- LibUv - multiplatformová C knižnica zameriavajúca sa na asynchrónne I/O operácie
- c-ares - C knižnica pre asynchrónne DNS požiadavky a odpovede
- http-parser - C knižnica pre HTTP požiadavky a odpovede
- open-ssl - kryptografické funkcie
- zlib - C knižnica pre synchronnú aj asynchrónnu kompresiu, dekompresiu a pre streamovanie dát

4.1.3 Event loop

Event loop alebo udalostná slučka je to čo umožňuje Node.js vykonávať neblokujúce vstupno-výstupné operácie napriek tomu, že JavaScript je jednvláknový. Keďže väčšina

moderných jadier má viacvláknové spracovanie tak môžu zvládnuť vykonávať viac operácií na pozadí. V Node.js to ale funguje tak, že keď je jedna z operácií dokončená, jadro povie Node.js, aby do fronty na vykonanie pridalo príslušné spätné volanie na výsledok vykonanej operácie. Na Obr. 11 môžeme vidieť zjednodušený diagram ako to funguje.



Obr. 11: Zjednodušená verzia fungovania Node.js architektúri

Keď sa Node.js spustí tak inicializuje slučku udalostí a spustí poskytnutý vstupný skript, ktorý môže uskutočňovať asynchrónne volania API. Následne sa začne spracovávať slučka udalostí. Tá postupne spracováva udalosti z fronty udalostí bez toho aby čakala na dokončenie daných udalostí. Keď je táto udalosť hotová tak sa jej spätné volanie zaradi na spodok fronty udalostí a jej výsledok je teda znova spracvaný keď sa k nemu slučka dostane. Toto sa opakuje kým nie je fronta udalostí vyčerpaná alebo nie je dosiahnutý maximálny počet spätných volaní.

4.1.4 Nevýhody Node.js

Ako sme vyššie popísali Node.js má veľké množstvo výhod, medzi ktoré hlavne patrí asynchrónne I/O, škalovalnosť, veľká a aktívna komunita a veľké množstvo prídavných modulov. Avšak Node.js má aj určité nevýhody. Medzi hlavné patrí:

- neefektívnosť pri náročných procesoch pre CPU - tvorba reportov, analýz, zložité výpočty...
- pri nepochopení práce s callbackmi môže viesť používanie Node.js k zle napísaným kódom
- menej štandardných knižníc v porovnaní s klasickou Javou a .NET platformou

Môžeme teda vidieť, že nie vždy je využitie Node.js možnosťou. Veľmi záleží akú aplikáciu vyvíjame. Ak pracujeme na aplikácii pre real-time komunikáciu s využitím web-socketov, streaming alebo rýchlu prácu so súbormi tak je Node.js veľmi vhodný.

4.2 Node Package Manager

Node package manager, skrátene len NPM je správca prídavných balíkov vyvinutý pre Node.js. Oficiálna stránka <https://www.npmjs.com> obsahuje tisíce voľných balíkov na stiahnutie a používanie. NPM program sa automaticky nainštaluje pri inštalácii Node.js. NPM sa následne dá spúšťať z príkazového riadku. Tak isto je možné vytvorenie vlastného balíka a nahrať ho do centrálného repozitára oficiálnej stránky NPM.

Balík v Node.js je zväčša adresár obsahujúci všetky súbory, ktoré prídavný modul potrebuje. Modul je JavaScriptová knižnica, ktorú programátor môže pridať do svojho projektu. Jedným zo súborov, ktoré sa nachádzajú v adresári musí byť metadátový súbor s názvom *package.json*. V tomto súbore sú definované vlastnosti, meno a verzia balíka. Na Obr. 12 môžeme vidieť ako takýto súbor vyzerá.

```
{
  "name" : "foo",
  "version" : "1.2.3",
  "description" : "A package for fooing things",
  "main" : "foo.js",
  "keywords" : ["foo", "fool", "foolish"],
  "author" : "John Doe",
  "licence" : "ISC"
}
```

Obr. 12: Jednoduchý *package.json* súbor

Inštalácia balíkov sa dá zrealizovať tromi spôsobmi:

- manuálna inštalácia len pre projekt, na ktorom pracujeme - v termináli sa v priečinku projektu zavolá príkaz **npm instal *názov_modulu***. Posledná verzia balíka sa nainštaluje pre projekt a uloží do adresára *node_modules*.
- manuálna inštalácia globálne pre systém - v termináli sa zavolá príkaz **npm instal *názov_modulu* -g**. Táto možnosť sa ale neodporúča.
- poslednou možnosťou je dopísanie do projektového suború *package.json* do časti dependencies názov modulu a jeho verziu a v termináli zadať príkaz **npm install**.

4.3 Ionic framework

Ionic je voľne dostupné, open-source prostredie na vytváranie hybridných aplikácií pre mobilné zariadenia vytvorené Maxom Lynchom a Adamom Bradleyom zo spoločnosti Drifty Co. v roku 2013. Prvá verzia vydaná z roku 2013 bola založená na AngularJS a Apache Cordova, avšak najnovšia verzia bola prepracovaná ako skupina webových komponentov, čo umožňuje programátorom zvoliť pri tvorbe aplikácie ľubovoľný framework ako je napríklad Angular, React alebo Vue.js. Je tiež ale možné zvoliť natívne Ionic komponenty bez použitia akéhokoľvek iného frameworku. Ionic poskytuje nástroje a služby na tvorbu nie len mobilných aplikácií ale aj klasických aplikácií pre Windows, MacOS alebo progresívnych webových aplikácií založených na moderných spôsoboch tvorby webových aplikácií pomocou CSS, HTML a Sass.

4.3.1 Služby a vlastnosti

Ionic používa doplnok Cordovu alebo novší Capacitor na získanie prístupu k operačnému systému zariadenia, na ktorom aplikácia beží a tým aplikácia vie pristupovať k perifériám zariadenia ako je napríklad GPS modul, kamera alebo baterka. Tieto periférie potom môže aplikácia využívať pre svoju funkčnosť.

Capacitor

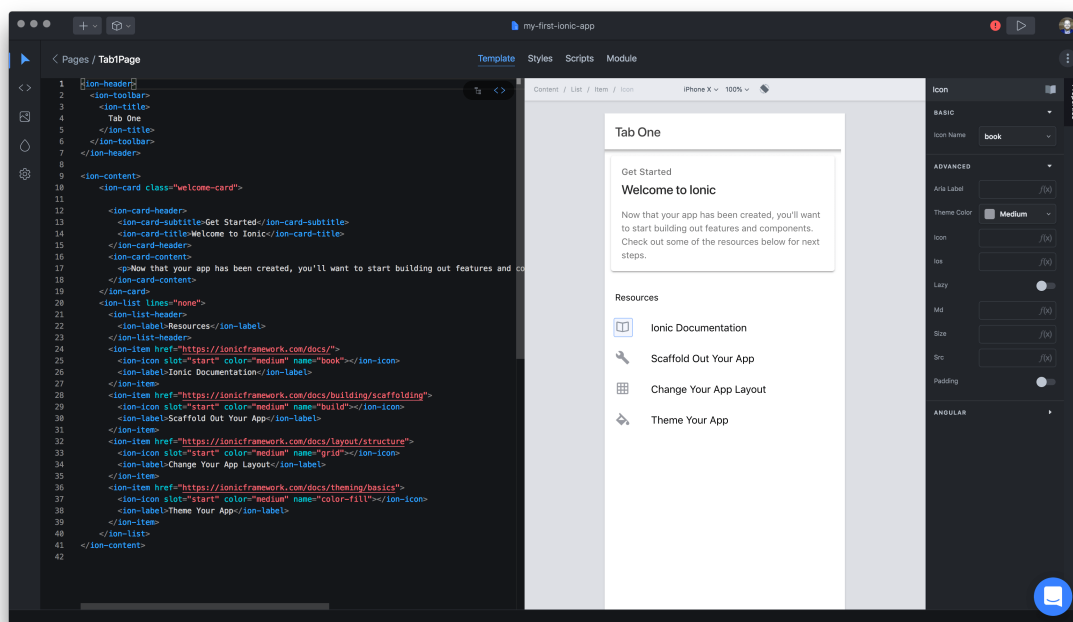
Capacitor je open-source projekt, ktorý umožňuje spúšťanie moderných web aplikácií natívne na operačných systémoch Android alebo iOS, prípadne s využitím Electronu na systémoch Windows, MacOS. Tiež poskytuje výkonné a ľahko použiteľné rozhranie pre prístup k natívnym SDK a API na každej platforme. O Capacitore sa hovorí ako o výkonnom novom prehliadači webových aplikácií, ktorý odomkne celú natívnu funkčnosť každej platformy prostredníctvom konzistentých API medzi platformami. S použitím Capacitora nemusí vývojár spravovať viac rohaní API pre každú platformu ale môže namiesto toho vyvíjať aplikáciu ako celok bez nutnosti uvažovať na akej platforme bude aplikácia spustená.

Cordova

Podobne ako Capacitor je Cordova open-source projekt umožňujúci spustenie web aplikácie na rôznych platformách. Avšak na rozdiel od Capacitoru nie je Cordova prispôbená na využitie Electronu pre export na desktopovú aplikáciu. Preto sa dnes už viacej využíva novší Capacitor.

Ionic využíva všetky do teraz dostupné webové komponenty s tým, že ich funkčnosť zrýchľuje a robí ju menej náročnú pre zariadenie, na ktorom aplikácia beží. Tak isto

ako bolo už vyššie spomenuté umožňuje využitie ktorýchkoľvek moderných frameworkov a ich komponentov. Pre vývojárov je k dispozícii vlastné prostredie na tvorbu Ionic aplikácií Ionic studio (Obr. 13) a taktiež je dostupné CLI pre prácu s projektami cez príkazový riadok.



Obr. 13: Vývojové prostredie Ionic studio

4.3.2 Inštalácia

Pri inštalácii Ionic frameworku je potrebné si povedať, že Ionic je NPM modul čo znamená, že na jeho inštaláciu je potrebné mať nainštalovaný Node.js. Tak isto je potrebné povedať, že pomocou Ionicu je možné vyvíjať aplikácie pre mobilné operačné systémy Android verzia 4.1+ a iOS 7+ pričom treba ešte spomenúť, že pri veľkom množstve existujúcich Android zariadení je možné, že na niektorých zariadeniach vyvíjaná aplikácia nemusí dobre fungovať. Vyvíjanie aplikácií pomocou Ionicu je možné na ktoromkoľvek operačnom systéme - Windows, Linux, MacOS.

Prvým krokom pri inštalácii je nainštalovanie Apache Cordova alebo Capacitor ak nerátame inštaláciu Node.js. Cordova aj Capacitor sú podobne ako Ionic NPM moduly takže ich inštalácia je jednoduchá. Po nainštalovaní Cordovi respektíve Capacitora sa môže nainštalovať samotný NPM modul Ionic. Následne je už možné vytvoriť projekt a začať pracovať na aplikácii. Pri vytváraní projektu je ale ešte možné špecifikovať framework aký chceme na vývoj aplikácie použiť. Toto sa realizuje príkazom *ionic start nazov_aplikacie -type=angular -capacitor*. Tu si všimnime, že do prepínača type sa zadal angular čiže

aplikácia bude vyvíjaný pomocou frameworku Angular a bude sa používať Capacitor. Keďže Ionic je NPM modul je samozrejmé, že do aplikácie je možné doinštalovať takmer akýkoľvek ďalší NPM modul na rozšírenie funkčnosti aplikácie. Pri vývoji aplikácie je ešte ale potrebné pomocou doplnku Cordova alebo Capacitor špecifikovať na akej platforme má aplikácia fungovať. Toto sa urobí príkazom `ionic cordova/capacitor platform add ios/android` zadaným do príkazového riadku. Je možné pridať aj obidve platformy.

4.4 Angular

Angular je framework na vyvíjanie webových aplikácií vyvinutý a spravovaný spoločnosťou Google. Googlom je označovaný ako *"Super framework založený na JavaScripte pre čokoľvek"*. Takto bola označovaná jeho prvá verzia AngularJS, ktorá bola založená na JavaScripte. Dnes je však používanější jeho druhá verzia Angular 2+, označovaná tiež ale ako Angular, čo označuje všetky verzie od verzie 2, ktoré sú narozdiel od prvej verzie založené už na TypeScript. Hneď po svojom vydaní sa Angular stal jedným z najviac používaných frameworkov pre tvorbu web aplikácií na trhu. Tento úspech je založený hlavne na tom, že narozdiel iných frameworkov doby keď Angular vyšiel je Angular výlučne objektovo-orientovaný a jeho syntax je prekvapivo blízka Java 8 aj keď je založený na TypeScript.

4.4.1 História

Prvá verzia AngularJS bola vydaná ešte v roku 2010. Ako bolo vyššie spomenuté táto verzia bola založená na JavaScripte. Aj keď po vydaní novej verzie Angular 2 väčšina vývojárov prešla na novšiu verziu, AngularJS je stále udržiavaný framework so stálou podporou nových aktualizácií a živou komunitou vývojárov.

Na konci roka 2014 Google ohlásil, že pracuje na novej verzii frameworku Angular 2, ktorý bude kompletne prepísaný AngularJS do novo vyvíjaného jazyka TypeScript. Po ohlásení avšak Microsoft súhlasil s pridaním podpory pre nimi vyvíjaný TypeScript a tak bol Angular 2 založený ako TypeScriptový framework. Zaujímavosťou je, že hneď od vydania Angularu 2 bolo možné v ňom vyvíjať aplikácie aj v JavaScripte nakoľko má Angular 2 podporu aj JavaScriptu. Oficiálne bol Angular 2 vydaný 14. septembra 2016 aj keď ako beta verzia fungoval už od roku 2015.

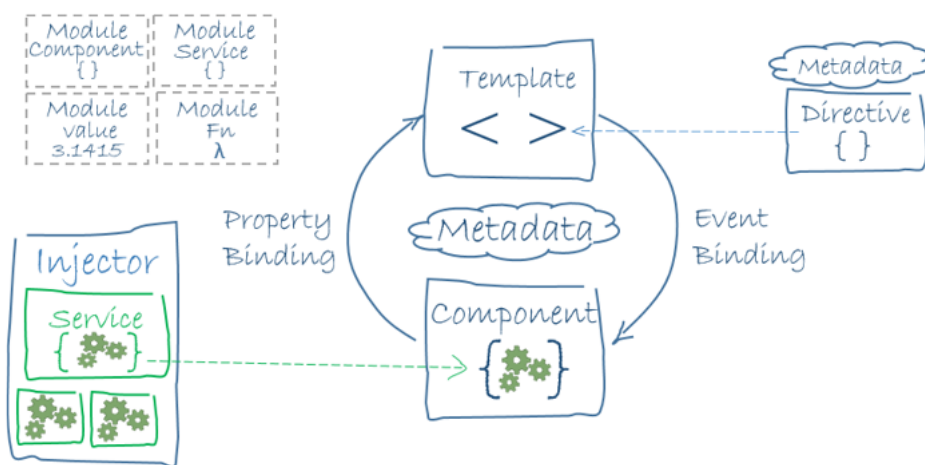
Na konci roka 2016 bol ohlásený Angular 4 ako nástupca Angularu 2. Označenie 3 bolo preskočené aby sa zabránilo nezrovnalostiam z dôvodu, že nové aktualizácie Angularu 2 boli označované ako v3.3.0. Angular 4 bol oficiálne vydaný 23. marca 2017 a priniesol novinky ako HttpClient - vylepšená knižnica pre vytváranie HTTP požiadaviek, nový životný cyklus udalostí a iné menšie vylepšenia.

Dnes je už vydaná verzia 9, pričom každá verzia (5,6,7,8) priniesla vždy hlavne podporu nových Material Dizajnov, podporu nových knižníc a podporu tvorby progresívnych web aplikácii. Google sa pri vydávaní nových aktualizácii zaručil, že každá nová verzia bude spätne kompatibilná a nové verzie chce vydávať dva krát ročne.

4.4.2 Architektúra

Základ Angularu tvoria NgModuly. Tieto moduly poskytujú kompiláciu komponentov, čo sú druhá základná zložka architektúry Angularu. NgModuly zhlukujú súvisiace kódy je množín, ktoré majú spoločnú funkcionality. Skrátene sa dá povedať, že každá aplikácia je definovaná množinou NgModulov. Základom každej aplikácie je jeden koreňový NgModul, ktorý môže mať pod sebou ľubovoľný počet ďalších NgModulov. Tento koreňový modul umožňuje navigáciu v aplikácii a bootstrapping.

Ako bolo vyššie spomenuté druhým stavebným prvkom sú komponenty. Komponenty zodpovedajú za to čo sa zobrazuje na obrazovke používateľa. Rovnako ako pri NgModuloch aj komponenty musia mať jeden koreňový komponent, ktorý potom v sebe má usporiadané ďalšie komponenty. Komponenty v sebe používajú aj služby, ktoré poskytujú špecifické funkcie, ktoré nie vždy priamo súvisia so zobrazením. Poskytovatelia týchto služieb sú vkladajú do komponentov ako závislosti. Týmto sa stáva kód modulárnym, opakovateľne použiteľným a efektívnym. Komponenty aj služby do nich vkladané sú označované dekorátormi, ktoré určujú o aký typ komponentu, služby ide a poskytujú taktiež metadáta, ktoré hovoria o tom ako tieto komponenty a služby používať. Metadáta tiež spájajú komponenty so šablónami, ktoré definujú ako budú komponenty zobrazené na jednotlivých stránkach. Šablóna je kombinácia obyčajných kódov HTML so smernicami Angularu čo umožňuje Angularu úpravu tohto HTML kódu ešte pred jeho zobrazením v okne prehliadača.



Obr. 14: Princíp fungovania Angularu

4.4.3 Výhody Angularu

Komponentová architektúra, ktorá poskytuje vyššiu kvalitu kódu

Architektúra založená na komponentoch je jednou z vecí, ktorá robí rozdiel medzi AngularJS a jeho nástupcom. Angular komponenty možno považovať za malé časti používateľského rozhrania alebo ako časť aplikácie. Aj keď je každý komponent zapuzdrený so svojou funkčnosťou, v Angular existuje prísna hierarchia komponentov. Napríklad v Angular 9 boli predstavené komponenty Mapy Google a YouTube Player.

Zatiaľ čo AngularJS bol postavený hlavne na architektúre Model-View-Controller (MVC), počnúc verziou 2 sa Angular považuje za komponentovo-založený, čo je veľmi podobné MVC, ale zaisťuje to vyššiu opätovnú použiteľnosť komponentov v celej aplikácii. To umožňuje vytváranie používateľských rozhraní s mnohými pohyblivými časťami a zároveň zrýchľuje priebeh vývoja aplikácie.

TypeScript: lepšie nástroje, čistejší kód a vyššia škálovateľnosť

Angular je písaný pomocou TypeScript jazyka, ktorý je v podstate nadradený JavaScriptu. Plne sa skompiluje do JavaScriptu, ale pomáha pri zisťovaní a odstraňovaní bežných chýb pri písaní kódu. Aj keď malé projekty JavaScriptu takéto vylepšenie nevyžadujú, podnikové aplikácie vyzývajú vývojárov, aby vylepšili svoj kód a častejšie overovali jeho kvalitu.

V Angulari píšeme komponenty v TypeScript, šablóny v HTML a rozširujeme ich pomocou Angularu. Takto funguje väčšina JS frameworkov. Šablóny HTML sa potom skompilujú do pokynov jazyka JavaScript, takže TypeScript alebo JS sú naše hlavné nástroje na prácu v Angulari. Victor Savkin, bývalý vývojár tímu Google Angular, vysvetľuje, že prechod z jazyka JavaScript do jazyka TypeScript je opodstatnený nástrojmi pre veľké projekty v podnikovom meradle. TypeScript má lepšie služby navigácie, automatického dopĺňania a refaktoring.

RxJS: efektívne asynchrónne programovanie

RxJS je knižnica bežne používaná v Angular na spracovanie asynchrónnych dátových volaní. Umožňuje nezávisle a paralelne spracovávať udalosti a čakať na vykonanie nejakej udalosti a ponechať webovú stránku nereagujúcu. V zásade to funguje ako montážna linka, kde sa vykonávanie rozdeľuje na jednotlivé a vymeniteľné kusy, a nie je viazaná na jednu osobu. Je samozrejmé, že už pred RxJS existovalo asynchrónne programovanie, ale táto knižnica uľahčila veľa vecí.

Dlhodobá podpora

Niektorí softvéroví inžinieri považujú samotnú skutočnosť, že spoločnosť Google podporuje technológiu Angular, za hlavnú výhodu tejto technológie. Dobrým znamením je, že spoločnosť Google oznámila dlhodobú podporu pre túto technológiu. Inžinieri Igor Minar a Steven Fluin, stojaci za Angularom, potvrdili tento záväzok v hlavnom vystúpení NG-Conf 2017. V zásade to znamená, že spoločnosť Google plánuje držať sa Angularu a ďalej ho rozvíjať a snažiť sa udržať vedúce postavenie medzi front-end inžinierskymi nástrojmi.

Silná komunita

Keďže je tu Angular už zopár rokov je jasné, že za ten čas vznikla veľká a živá komunita, ktorá denodenne prináša nové balíky, vylepšenia, návody, rady a iné užitočné veci pri práci s Angularom. Ak sa priemerný inžinier stratí alebo má problém pri vývoji aplikácie, vždy existuje nástroj, ktorý pomôže vyriešiť problém, ktorý sa objaví.

4.5 Electron

Electron je populárny framework, ktorý umožňuje vytváranie desktopových aplikácií pre MacOS, Linux alebo Windows pomocou známych webových technológií HTML, JavaScript a CSS vytvorený a udržiavaný firmou GitHub. Niektoré veľmi populárne desktopové aplikácie, ako sú Visual Studio Code a Slack, sú vytvorené pomocou frameworku Electron. Electron je založený na prehliadači Chromium pre interpretáciu používateľského rozhrania na Node.js pre prístup k súborovým systémom. Pretože spoločnosť Electron nám poskytuje webové rozhranie pre webové aplikácie, môžeme na vývoj aplikácií pre osobné počítače použiť akýkoľvek druh JavaScriptu prípadne TypeScriptu.



Obr. 15: Firmy využívajúce vo svojich aplikáciách Electron

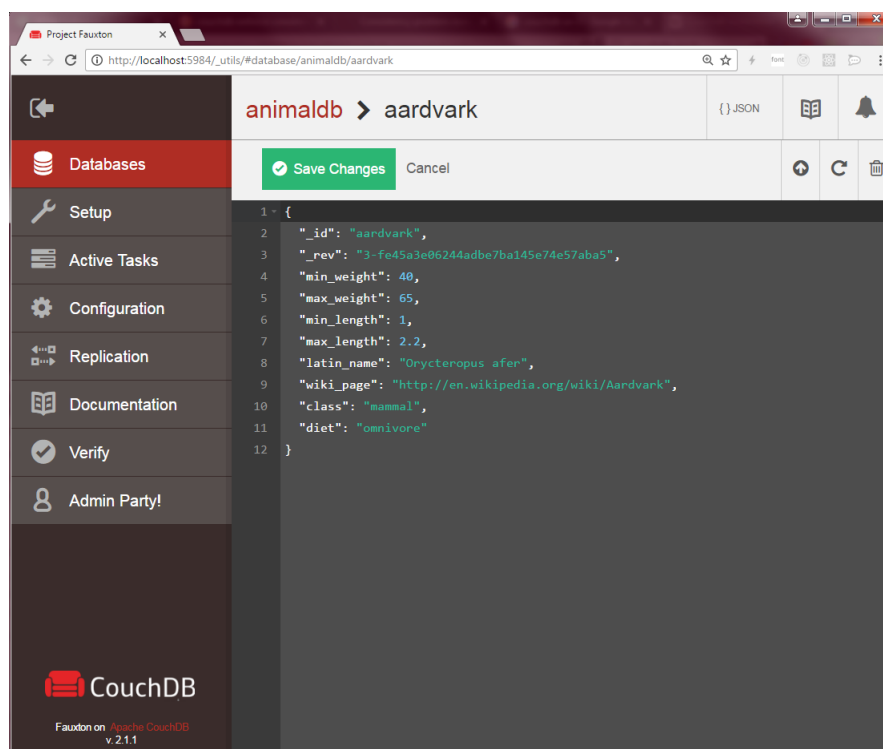
4.5.1 Architektúra

Elektronové aplikácie sa skladajú z viacerých procesov. Existuje proces „prehliadača“ a niekoľko procesov „vykreslenia“. Proces prehliadača spúšťa aplikačnú logiku a potom môže spustiť viacero procesov vykresľovania, ktoré vykresľujú okná, ktoré sa zobrazujú na obrazovke používateľa použitím HTML a CSS. Procesy prehliadača aj vykreslenia sa môžu spustiť s integráciou Node.js, ak je povolená. Väčšina rozhraní Electron API je napísaná v C++ alebo Objective-C a potom je exponovaná priamo aplikačnému kódu prostredníctvom väzieb JS. Prvotné verzie Electronu boli náchylné na webové útoky pomocou cross-site skriptingu nakoľko je Electron založený na engine Chromium, ktorý je náchylný na tento typ útoku. Avšak neskoršími aktualizáciami frameworku Electron bola táto zraniteľnosť odstránená a vývojár sa ňou nemusí osobite zaoberať.

4.6 Apache CouchDB

CouchDB vyvíjaná a udržiavaná firmou Apache je open-source NoSQL databáza, ktorá zhromažďuje a ukladá údaje do dokumentov vo formáte JSON. Na rozdiel od relačných databáz používa CouchDB dátový model bez schém. Tento princíp zjednodušuje správu záznamov v rôznych zariadeniach, mobilných telefónoch a webových prehliadačoch.

CouchDB bola predstavená v roku 2005 a neskôr sa stala projektom firmy Apache Software Foundation v roku 2008. Ako open-source projekt je CouchDB podporovaná veľkou aktívnou komunitou vývojárov, ktorí neustále vylepšujú softvér so zameraním na jednoduché používanie webu.



Obr. 16: JSON dokument v aplikácii na správu CouchDB

4.6.1 Výhody použitia CouchDB

CouchDB predstavuje celý rad výhod zameraných na používateľov a vývojárov. Motiváciu vývoja CouchDB možno definovať jedným slovom: relax. CouchDB prichádza s množstvom výhod navrhnutých na zníženie námahy pri prevádzkovaní pružného distribuovaného systému. Tu sú niektoré kľúčové vlastnosti CouchDB a ako sa líši od iných databáz NoSQL.

Škálovateľnosť

Architektonický návrh CouchDB ju robí mimoriadne prispôsobivou pri rozdeľovaní databáz a meraní údajov na viacerých uzloch. CouchDB podporuje horizontálne rozdelenie a replikáciu na vytvorenie ľahko spravovaného riešenia na vyrovnanie času potrebného na načítanie aj zápis počas zavádzania databázy.

CouchDB je vybavená veľmi odolným a spoľahlivým úložným enginom, ktorý bol vybudovaný od základov pre viackanálové databázové infraštruktúry. Ako NoSQL databáza je CouchDB veľmi prispôsobiteľná a otvára dvere vývoju predvídateľných aplikácií založených na výkone bez ohľadu na objem vašich údajov alebo počet používateľov.

Žiadne read-locks

Vo väčšine relačných databáz - kde sú údaje uložené v tabuľkách - ak niekedy potrebujete aktualizovať alebo upraviť tabuľku tak sa riadok zmenených údajov uzamkne pre ostatných používateľov, až kým sa nespracuje požiadavka na úpravu. Môže to spôsobiť problémy s prístupnosťou pre klientov a celkové prekážky vo vašich procesoch správy údajov.

CouchDB používa MVCC na simultánnu správu prístupu k databázam. To znamená, že bez ohľadu na aktuálne načítanie databázy môže CouchDB bežať na plnú rýchlosť a bez obmedzenia pre svojich používateľov. Pretože dokumenty v CouchDB sú verzované a pripojené v reálnom čase, v požiadavkách na čítanie databázy sa vždy zobrazia najnovšie aktualizované databázové snímky, bez ohľadu na to, kto k dokumentu pristupoval ako prvý.

CouchDB replikácia

Jednou z definujúcich funkcií CouchDB je obojsmerná replikácia, ktorá umožňuje synchronizáciu údajov na viacerých serveroch a zariadeniach prostredníctvom obojsmernej replikácie. Táto replikácia umožňuje podnikom maximalizovať dostupnosť systémov, skrátiť dobu obnovy dát, lokalizovať údaje najbližšie ku koncovým používateľom a zjednodušiť procesy zálohovania.

V CouchDB sa nerozlišuje, či sú dáta uložené na jednom serveri alebo na viacerých serveroch. CouchDB skôr identifikuje zmeny dokumentov, ku ktorým dôjde z akéhokoľvek zdroja, a zabezpečí, aby všetky kópie databázy zostali synchronizované s najaktuálnejšími informáciami. To umožňuje samostatnú správu a správu viacerých replík databázy a zároveň poskytuje presné informácie v reálnom čase vo viacerých počítačových prostrediach.

HTTP API

CouchDB používa RESTful API na prístup do databázy odkiaľkoľvek, s plnou flexibilitou operácií CRUD (vytváranie, čítanie, aktualizácia, mazanie). Tento jednoduchý a efektívny spôsob pripojenia k databáze robí CouchDB flexibilnou, rýchlou a výkonnou na použitie pri zachovaní vysokej dostupnosti.

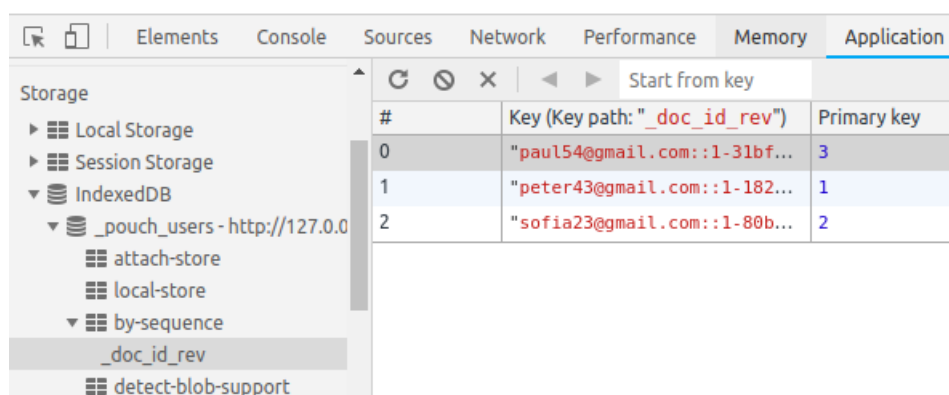
Stavaná pre offline používanie

Keď škálujeme použiteľnosť a prístupnosť databázy, je nevyhnutné vytvárať aplikácie, ktoré fungujú rovnako offline, ako online. CouchDB umožňuje aplikáciám ukladať zozbierané údaje lokálne na mobilných zariadeniach a prehliadačoch a potom ich synchronizovať, keď sa vrátia online.

4.7 PouchDB

PouchDB je open-source databáza JavaScript inšpirovaná Apache CouchDB, ktorá je navrhnutá tak, aby fungovala dobre v prehliadači. PouchDB bola vytvorená s cieľom pomôcť vývojárom webu vytvárať aplikácie, ktoré fungujú rovnako offline, ako online. Aplikáciám umožňuje ukladať údaje lokálne v režime offline, potom ich synchronizovať so servermi CouchDB alebo inými kompatibilnými servermi, keď je aplikácia späť online.

Architektúra PouchDB je rovnaká ako pri CouchDB. Dáta sú ukladané formátom JSON a celá práca s nimi je rovnaká ako pri CouchDB. Preto sa pri online-offline aplikáciách odporúča pri použití CouchDB používať aj PouchDB.



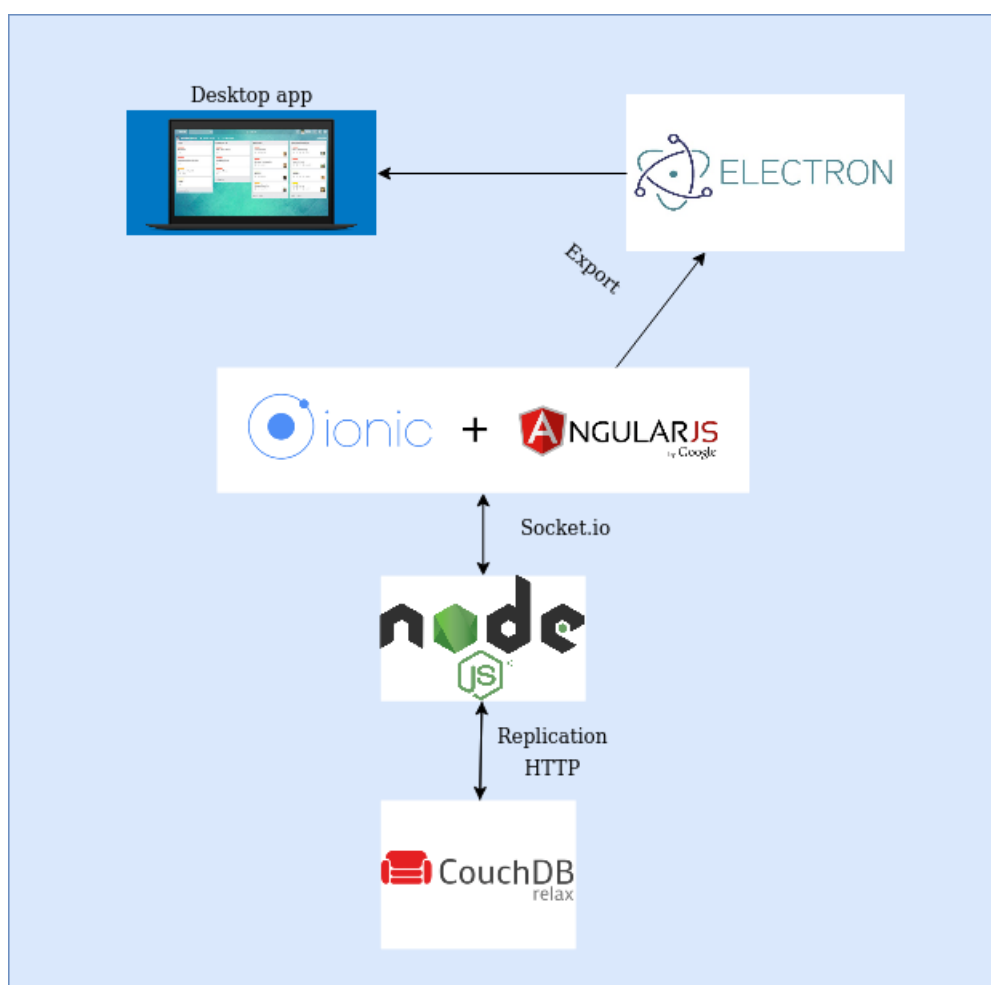
The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. Under 'Storage' > 'IndexedDB', the database '_pouch_users - http://127.0.0.1' is expanded, showing its internal structure. The '_doc_id_rev' store is selected, displaying a table of three records. The table has columns for index, key, and primary key.

#	Key (Key path: "_doc_id_rev")	Primary key
0	"paul54@gmail.com::1-31bf..."	3
1	"peter43@gmail.com::1-182..."	1
2	"sofia23@gmail.com::1-80b..."	2

Obr. 17: Náhľad PouchDB v prehliadači Google Chrome

5 Návrh aplikácie

Témou diplomovej práce je implemetovať co-working aplikáciu s využitím webového frameworku Ionic/Angular s neskorším exportom na desktopovú aplikáciu pomocou Electronu na strane frontendu a s použitím Node.js a CouchDB na strane servera. V tejto kapitole sa budeme zaoberať návrhom celej aplikácie s využitím daných technológií na jednotlivých komponentoch.



Obr. 18: Návrh komunikácie medzi komponentami

Po prieskume aktuálnych technológií sme sa rozhodli využiť Node.js ako náš server, ktorý bude komunikovať s našou NoSql databázou CouchDB, do ktorej bude ukladať zadané dáta a zároveň posielat žiadané dáta na frontend implementovaný pomocou typescriptovo založeného frameworku Ionic. Ionic sa avšak ako bolo spomenuté v kapitole 4.3 využíva na tvorbu webových alebo mobilných aplikácií. Preto finálnu webovú aplikáciu exportujeme nakoniec pomocou Electronu na desktopovú aplikáciu pre platformu

Windows/Linux/macOS.

5.1 Špecifikácia požiadaviek

Prvým krokom pri tvorbe návrhu je špecifikovanie vlastností a celkového správania aplikácie. Definuje sa tu funkcionality, ktorú chceme aby naša aplikácia mala, správanie aplikácie v určitých situáciách a jej kľúčové parametre, ktoré musí podľa požiadaviek spĺňať.

5.1.1 Používateľské požiadavky

Používateľské požiadavky sa stanovili na základe prehľadu co-working aplikácii, ktoré už existujú. Za základ sa zobrali hlavné funkcionality Slacku a k nim sa pridali buď doplnky Slacku alebo funkcie iných aplikácii. Softvér bude mať 2 používateľské rozhrania kedy jedno bude slúžiť ako administrátorské určené pre vedúcich tímov a klasické prostredie určené pre členov tímu. Administrátorské konto bude mať možnosť sa medzi týmito dvoma prostrediami prepínať. Administrátorské prostredie bude umožňovať vytváranie tímov, pozívanie a pridávanie členov tímu a manažovanie taskov. Prostredie člena tímu bude umožňovať vytváranie miestností na stránke tímu, vytváranie stretnutí, zahajovanie hlasovaní, pridávanie príspevkov do miestností, komentovanie príspevkov, komunikáciu medzi členmi tímu, pridanie kontaktu, akceptovanie alebo zamietnutie pozvánky do tímu, správu svojich taskov.

Funkcionálne požiadavky:

Administrátor:

- Prihlásenie a odhlásenie
- Vytvorenie tímu
- Pridanie používateľa do tímu
- Vytvorenie tasku
- Spravovanie tasku
- Pridanie používateľa do tasku
- Nastavenie stavu tasku
- Prepnutie sa medzi rozhraniami administrátor a používateľ

Člen tímu:

- Prihlásenie a odhlásenie
- Pridanie miestnosti v tíme
- Pridanie udalosti v tíme
- Spravovanie svojich taskov
- Pridanie komentára k tasku
- Pridanie kontaktu do svojho zoznamu kontaktov
- Nápisanie správy inému používateľovi
- Komentovanie príspevkov v miestnosti
- Prezeranie svojho kalendára

Nefunkcionálne požiadavky:

- Používateľsky prijateľné prostredie
- Fungovanie aj v offline režime
- Obsluha viacerých používateľov
- Používateľské rozhranie implementované prostredníctvom desktopovej aplikácie
- Dizajn inšpirovaný aplikáciou Slack

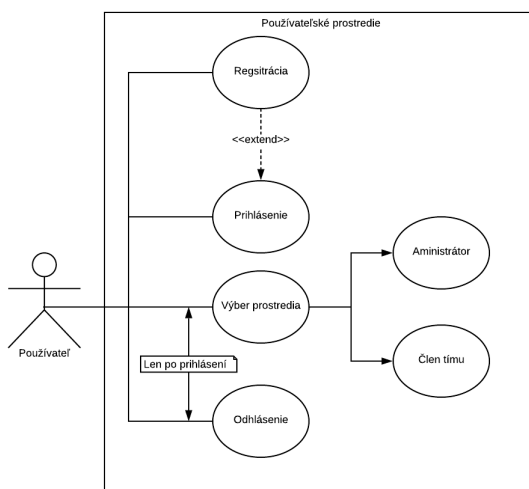
5.2 Diagramy

Po zadeinovaní požiadaviek môžeme prísť k ďalšej časti, z ktorej návrh aplikácie pozostáva a to k diagramom. Podľa týchto diagramov sa potom pri implementácii budeme riadiť nakoľko tie nám presne ukazujú ako sa bude systém správať pri používaní a ako majú jednotlivé komponenti a časti aplikácie medzi sebou komunikovať. Základným diagramom, ktorý si vytvoríme ako prvý bude diagram prípadov použitia, ktorý nám bude ukazovať ako jednotliví herci - typy používateľov môžu používať aplikáciu. Ďalej existujú diagramy tried, ktoré pomáhajú vývojárovi vizualizovať si jednotlivé triedy, ktoré sa tvoria pri objektovo orientovaných jazykoch. V našom prípade ale tento diagram vytvárať nebudeme nakoľko nami používaný JavaScript ani TypeScript nie sú klasické objektovo orientované jazyky. Diagramy, ktoré ale budeme určite vytvárať sú sekvenčné. Tie nám vizualizujú interakciu medzi komponentami alebo časťami kódu v aplikácii.

5.2.1 Diagramy prípadov použitia

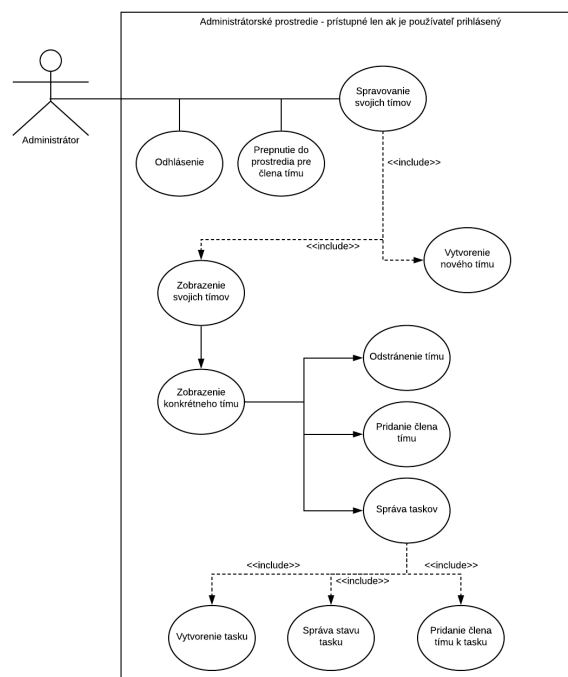
Diagramy prípadov použitia sa používajú na popísanie správania systému z hľadiska jeho používania. Ukazujú aký rôznych používateľov môžu systém používať a aké činnosti môžu v tomto systéme vykonávať.

Obrázok 19 nám ukazuje základné možnosti akéhokoľvek používateľa nášho systému. Tento používateľ sa najskôr musí registrovať do systému. Potom sa môže prihlásiť. Po prihlásení má na výber pod akým typom používateľa sa chce aktuálne prihlásiť - či pod administrátorom alebo ako člen tímu.



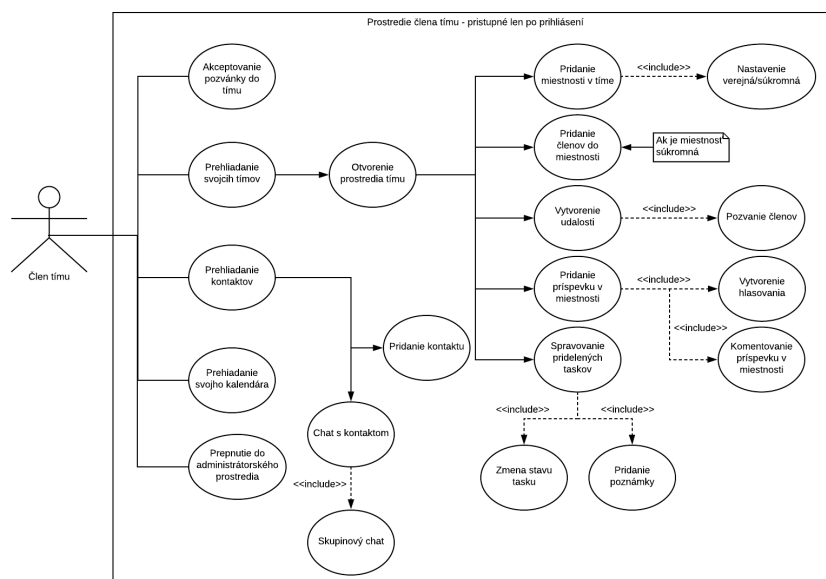
Obr. 19: Use case diagram pre používateľské prostredie

Druhý diagram - Obr. 20 nám ilustruje prípady použitia ak používateľ vybral možnosť prihlásiť sa ako administrátor prípadne ak sa člen tímu prepoľ do administrátorského rozhrania. Administrátor teraz môže byť spravovať svoje tímy alebo vytvoriť nový tím.



Obr. 20: Use case diagram pre administrátorské prostredie

Posledný diagram prípadov použitia - Obr. 21 nám ilustruje ako môže používateľ používať systém ak si vybral možnosť prihlásiť sa ako člen tímu alebo sa prepol z administrátorského rozhrania do rozhrania člena tímu. Tu má používateľ na výber viacero možností čo môže v aplikácii vykonať.



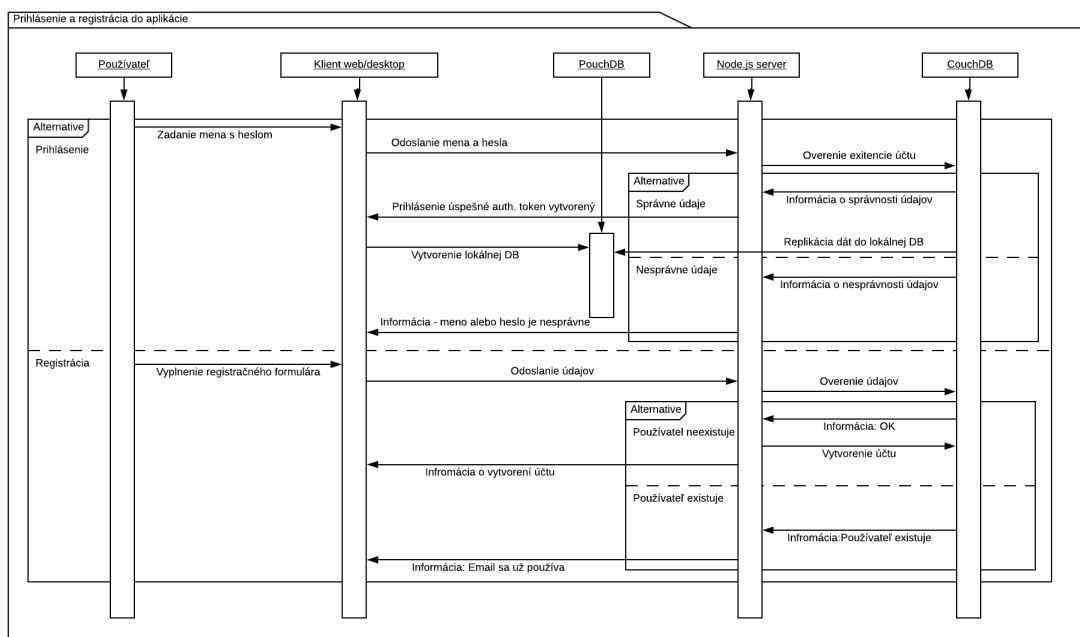
Obr. 21: Use case diagram prostredia člena tímu

5.2.2 Sekvenčné diagramy

Nasledujúce diagramy budú asi najpoužívannejšie pri implemtácii aplikácie nakoľko sekvenčné diagramy nám ukazujú interakcie medzi komponentami v systéme.

Prihlásenie a registrácia

Na Obr. 22 môžeme vidieť komunikáciu jednotlivých komponentov pri prihlásení alebo registrácii do aplikácie. Na základe výberu používateľa či sa chce prihlásiť alebo registrovať sa mu zobrazí príslušná stránka. Pri prihlásení sa po vyplnení údajov, tieto údaje pošlú na server, ktorý v online databáze skontroluje daný email či používateľ existuje. Ak existuje skontroluje aj heslo. Ak sú oba údaje správne, server vytvorí autorizačný token s určitou platnosťou a pošle ho klientovi. Na strane klienta sa potom vytvorí lokálna databáza, ktorá si hneď z online databázy stiahne všetky dáta a aplikáciu tu začína pracovať len s lokálnou databázou.

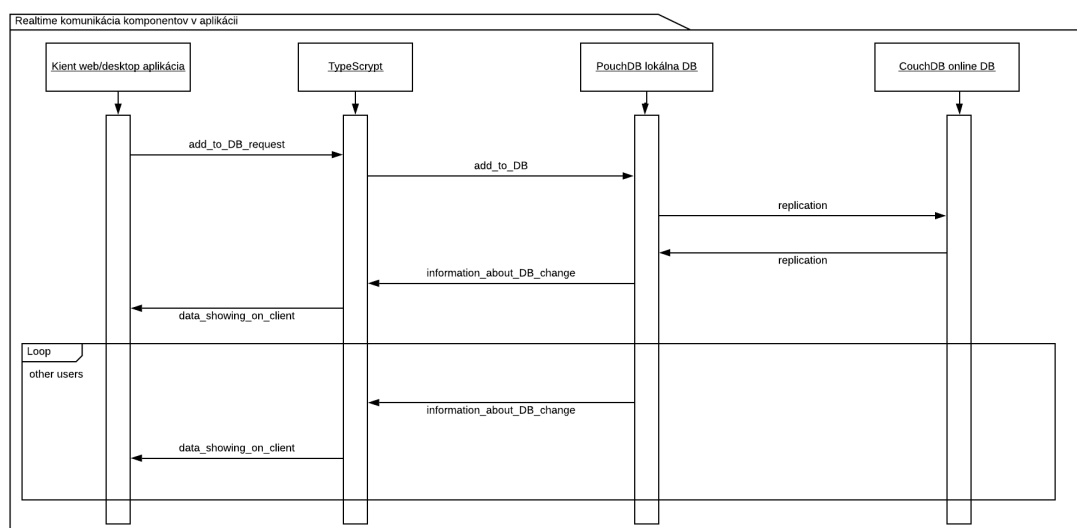


Obr. 22: Sekvenčný diagram prihlásenia a registrácie

Pri registrácii funguje systém podobne. Po vyplnení údajov sa údaje pošlú na server, ktorý overí či daný email sa už nepoužíva. Ak sa nepoužíva tak sa do databázy uloží nový používateľ a na stranu klienta je odoslaná správa o úspešnej registrácii.

Komunikácia komponentov v aplikácii po prihlásení

Diagram Obr. 23 nám znázorňuje ako budú jednotlivé komponenty v aplikácii medzi sebou komunikovať po prihlásení do aplikácie.



Obr. 23: Sekvenčný diagram komunikácie komponentov v aplikácii

Celá komunikácia je založená na http protokole. Na diagrame môžeme vidieť, že ak niektorý používateľ vytvorí tím, udalosť, úlohu alebo napíše správu tak je vytvorená žiadosť na pridanie do databázy. Pridanie do databázy je realizované tak, že najskôr sú dáta pridané do lokálnej databázy PouchDB a ak má zariadenie, na ktorom beží aplikácia prístup na internet sú tieto dáta hneď zreplikované do online databázy. Ak prístup na internet nie je tak sú dáta zatiaľ uložené len v lokálnej databáze do doby kým zariadenie nezíska prístup na internet. Ostatným používateľom je potom odoslaná informácia, že na strane databázy prišlo k zmene a ich dáta, ktoré sa v klientovi zobrazujú sú aktualizované na základe nových dát v databáze.

5.3 Databázový model CouchDB

Po analýze viacerých možností sme sa v našom zadaní rozhodli nepoužiť štandardnú SQL databázu. Toto má za následok to, že sa nedá modelovať databáza cez štandardné ERD diagramy. CouchDB je databáza založená nie na tabuľkách ale na JSON dokumentoch. Hoci si naša aplikácia bude vyžadovať viac špecifických objektov pre jednotlivé záznamy do databázy, tieto objekty nebudú zložité. Jedná sa prevažne o objekty zložené z viacerých jednoduchým dátových hodnôt typu string, integer, date. Tieto objekty bude potrebné vytvoriť ako na strane servera v JavaScripte tak aj na strane frontendu v TypeScripte.

Návrh jednotlivých objektov môžeme vidieť nižšie.

Používateľ:

- základné údaje meno priezvisko heslo mail
- popis používateľa
- kontakty
- miestnosti
- kalendár

Udalosť:

- názov
- tím
- začiatok
- koniec
- popis

Tím:

- názov
- admin
- členovia
- miestnosti
- udalosti
- úlohy

Miestnosť:

- názov
- členovia
- príspevky

Príspevok:

- text
- autor
- čas pridania
- miestnosť

Správa:

- text
- autor
- komu
- čas odoslania

Úloha:

- názov
- vedúci úlohy
- text
- tím
- pridelení používatelia
- stav úlohy
- komentáre

Komentár:

- text
- autor
- čas pridania
- úloha

6 Implementácia aplikácie

Po vypracovaní kompletného návrhu sme začali s implemetáciou. Tá sa dála rozvrhnúť na viac fáz podľa toho ako sme postupne implementovali aplikáciu. Začali sme s implementáciou prihlasovacieho a registračného systému. Po dokončení tejto časti sme prešli do implementácie zobrazenia aplikácie po prihlásení používateľa. Tu sme implementovali časť aplikácie kde sú zobrazené informácie o aktuálne prihlásenom používateľovi, tímy, ktorých je členom, jeho kalendár, v ktorom sú zobrazené udalosti jeho tímov, stránka všetkých jeho úloh zo všetkých tímov a stránka jeho kontaktov. Po implementovaní tejto časti sme prešli na implemetáciu vytvárania a správy tímov. Následne sme implementovali jednotlivé funkcie, ktoré vie vykonávať či už admin tímu alebo člen tímu vo vybranom tíme. Sem patrí vytváranie a správa miestností, úloh a udalostí vo vybranom tíme. Predposlednou časťou, ktorou sme sa zaoberali bolo vytvorenie chatu medzi používateľmi, ktorých máme v kontaktoch. Poslednou časťou bolo vyladenie potencionálnych problémov a celkový dizajn aplikácie nakoľko do teraz sme implementovali hlavne funkcionálne časti aplikácie a dizajn sme nechávali na koniec. Všetky časti až na poslednú sa dajú ešte rozdeliť na implemetáciu back-end časti a front-end časti. Tieto dve časti sme vždy implementovali súčasne aby sme si vždy otestovali požadovanú funkčnosť. Finálnym krokom bol export aplikácie pomocou Elektronu na desktopovú aplikáciu. Nižšie si teraz prejdeme postupne všetky fázy implementácie a podrobne si rozoberieme ako sme jednotlivé časti implementovali.

6.1 Prihlasovanie a registrácia

Medzi nefunkcionálnymi požiadavkami sme spomínali fungovanie aplikácie aj v off-line režime. Táto časť aplikácie bola ale implemetovaná tak aby ako jedina vyžadovali pripojenie na internet. Je to samozrejmé z dôvodu, že na získanie dát z online databázy CouchDB a ich zreplikovanie do lokálnej databázy PouchDB vyžaduje pripojenie na internet. Popíšeme si najskôr ako sme implementovali back-end tejto časti a potom font-end.

Na implementáciu back-endu prihlásenia a registrácie sme využili Node.js server, ktorý sa pripája na online databázu. Na Obr. 24 môžeme vidieť implementáciu funkcie na registráciu používateľa. Po odoslaní vyžadovaných dát používateľom z registračnej stránky sa z requestu - req vytiahnu dáta email, meno a heslo. Prvé čo sa vo funkcii vykoná je overenie či boli zaslané všetky tri údaje. Ak nie server vráti status 400 so správou, že používateľ nezadal všetky potrebné údaje. Ak boli zaslané všetky tri tak sa vytvorí selektor do databázy kde sa overuje zadaný email. Ak sa v databáze nachádza už nachádza vytvorený používateľ so zaslaným emailom tak server vráti opäť status 400

so správou, že používateľ s týmto emailom už existuje. Rozhodli sme sa overovať len na základe emailu pretože sa môže stať, že dvaja používatelia majú rovnaké meno. Ak v databáze takýto používateľ neexistuje tak sa najskôr zašifruje heslo pomocou prídavnej knižnice do node.js bcrypt. Následne sa už len vytvoria potrebné dodatočné záznamy, ktoré json dokument používateľa potrebuje - polia tímov, udalostí, kontaktov, úloh a prázdny popis používateľa a tento údaj sa pomocou prídavnej knižnice na prácu s CouchDB nano pridá do databázy pomocou funkcie insert.

```
...if (!req.body.email || !req.body.password || !req.body.name) {
...  return res.status(400).json({ 'msg': 'You need to send email and password and name' });
...}
...const q = {selector: {email: { '$eq': req.body.email }}};
...users.find(q).then((doc) => {
...  if (doc.docs.length == 0) {
...    let pass = req.body.password;
...    bcrypt.genSalt(10, function (err, salt) {
...      if (err) return next(err);
...      bcrypt.hash(pass, salt, function (err, hash) {
...        if (err) return next(err);
...        pass = hash;
...        let teams = [];
...        let calendar = [];
...        let contacts = [];
...        let tasks = [];
...        let info = "";
...        users.insert({ name: req.body.name, email: req.body.email, info: info,
...          password: pass, teams: teams, calendar: calendar, contacts: contacts,
...          tasks: tasks }).then((body) => {
...            console.log(body)
...          });
...      });
...    });
...  };
...  return res.status(200).json({ 'msg': 'User registered' });
...  return res.status(400).json({ 'msg': 'The user already exists' });
...}
```

Obr. 24: Funkcia register na strane back-endu

Na Obr. 25 je kód funkcie na prihlásenie. Podobne ako pri registračnej funkcii sa najskôr overí či používateľ zadal všetky potrebné údaje. Následne sa overí či zadaný používateľ je v databáze vytvorený. Ak je, tak sa z databázy vytiahne celý jeho json dokument a overí sa či bolo zadane správne heslo.

```

exports.loginUser = (req, res) => {
  if (!req.body.email || !req.body.password) {
    return res.status(400).json({ 'msg': 'You need to send email and password' });
  }
  const q = {
    selector: {
      email: { '$eq': req.body.email }
    }
  };
  users.find(q).then((doc) => {
    if (doc.docs.length !== 0) {
      var match = bcrypt.compareSync(req.body.password, doc.docs[0].password);
      if (match) {
        return res.status(200).json({ token: createToken(doc.docs[0].email, doc.docs[0].name),
          email: doc.docs[0].email,
          name: doc.docs[0].name,
          id: doc.docs[0]._id });
      }
      if (!match) {
        return res.status(400).json({ 'msg': 'The email and password dont match' });
      }
    }
    return res.status(400).json({ 'msg': 'The user does not exist' });
  }, err => {
    return res.status(400).json({ 'msg': err });
  });
};

```

Obr. 25: Funkcia login na strane back-endu

Ak je všetko v poriadku vytvorí sa pomocou funkcie createToken jsonwebtoken. Kód tejto funkcie môžeme vidieť na Obr. 26. Na vytvorenie tokena je potrebná dodatočná knižnica jsonwebtoken. V tokene je uložené meno a email prihláseného používateľa, takzvané "jwt tajomstvo" čo je string podľa ktorého sa token šifruje a dĺžka platnosti tokena - v našom prípade 24 hodín. Po vypršaní tokena sa automaticky používateľ odhlási a musí byť prihlásený na novo.

```

function createToken(email, name) {
  return jwt.sign({ id: name, email: email }, config.jwtSecret, { expiresIn: 86400 })
}

```

Obr. 26: Vytvorenie jwt tokenu

Na strane front-endu sme ako základ implementovali tiež dve funkcie login a register, ktoré len z formulára vytiahnu zadané údaje a pošlú ich na adresu servera s volaním smerovania aké sme nastavili pre login a register. Na volanie adresy servera sme použili dodatočné moduly pre Angular a Ionic HTTPClient a na prácu s jwt tokenom JwtHelperService. Funkcie login a register môžeme vidieť na Obr 27.


```

register(credentials) {
  return this.http.post(`${this.url}/api/register`, credentials).pipe(
    catchError(e => {
      this.showAlert(e.error.msg);
      throw new Error(e);
    })
  );
}

login(credentials) {
  return this.http.post(`${this.url}/api/login`, credentials)
    .pipe(
      tap(res => {
        this.storage.set(TOKEN_KEY, res['token']);
        this.storage.set('email', res['email']);
        console.log(res['email']);
        console.log(res['name']);
        console.log(res['id']);
        this.loggeduser = res['email'];
        this.user = this.helper.decodeToken(res['token']);
        this.data = {
          name: res['name'],
          email: res['email']
        };
        this._shareddata.setData(this.data);
        this.authenticationState.next(true);
      }),
      catchError(e => {

```

Obr. 27: Prihlásenia a registrácia na strane front-endu

Na strane font-endu sme ale implementovali ešte niekoľko ďalších potrebných funkcií. Samozrejماً je funkcia na odhlásenia používateľa kde sa len zmazal jwt token. Tiež dôležitou funkciou bola funkcia na overenie platnosti tokena. Pri prihlásení sa token uložil do lokálnej pamäte. Pri každom načítaní, niektorej z ďalej vytvorených stránok sa volá funkcia checkToken (Obr. 28). Tu sa pomocou vyššie spomenujetej knižnice na prácu s jwt tokenom overí platnosť tokena. Ak nie je platný nastaví sa autorizačný stav na false a používateľ pri otvorení novej stránky alebo obnovení aktuálnej bude automaticky odhlásený a presmerovaný na prihlasovaciu stránku.

```

checkToken() {
  this.storage.get(TOKEN_KEY).then(token => {
    if (token) {
      let decoded = this.helper.decodeToken(token);
      let isExpired = this.helper.isTokenExpired(token);

      if (!isExpired) {
        this.user = decoded;
        this.authenticationState.next(true);
      } else {
        this.storage.remove(TOKEN_KEY);
      }
    }
  });
}

```

Obr. 28: Overenie platnosti jwt tokenu

6.2 Stránka prihláseného používateľa

Po prihlásení sa používateľovi zobrazí stránka s postranným menu kde si vie zvoliť z 5 možností - profil, tímy, kalendár, úlohy a kontakty plus sa tu nachádza možnosť odhlásenia. Hneď po načítaní tejto stránky sa online databáza CouchDB zreplikuje do lokálnej databázy PouchDB.

Táto časť bola jednoduchá. Prvým krokom bolo vytvorenie spojenia s lokálnou databázou. Spojenie sme nadviazali pomocou prídavného modulu pre Angular pouchdb. Po vytvorení spojenia sa načítajú dáta aktuálne prihláseného používateľa. Tieto údaje sú potom podľa výberu v menu distribuované pomocou nami vytvorenej service sharreddata medzi jednotlivými stránkami, ktoré ich zobrazujú. Service sharreddata pozostáva len z premených, do ktorých ukladáme dáta z DB a ich geterov a seterov. Toto načítanie dát z databázy a uloženie do sharreddata je na Obr. 29.

```
constructor(private authService: AuthService, private router: Router, private _sharreddata: SharreddataService) {
  this.db = new PouchDB('appusers');
  this.remote = 'http://admin:admin@localhost:5984/appusers';
  this.db.sync(this.remote, this.options);
  this.authService.getSpecialData().subscribe(res => {
    this.loggedemail = res;
    this.load();
  });
  this.router.events.subscribe((event: RouterEvent) => {
    this.selectedPath = event.url;
  })
}

async load() {
  const q = {
    selector: {
      email: { $eq: this.loggedemail }
    }
  };
  this.db.sync(this.remote, this.options);
  this.data = await this.db.find(q).then((doc) => {
    this._sharreddata.setData(doc.docs[0]);
    return doc.docs;
  });
}
```

Obr. 29: Načítanie dát prihláseného používateľa

Zobrazenie dát na stránke sme následne realizovali podľa potreby. Na zobrazenie tímov, ktorých členom je prihlásený používateľ sme použili tabuľkové rozloženie a funkciu Angularu ngFor. Táto funkcia prechádza polom tímov a vykresľuje dáta každého prvku v poli podľa predlohy. Podobným princípom sme vytvorili aj zobrazenie kontaktov prihláseného používateľa. Ukážka tohto zobrazenia je na Obr. 30

```

<ion-content>
  <ion-grid>
    <ion-row class="row">
      <ion-col menuClose size="5" *ngFor="let num of arr_names" class="teamcol">
        [routerLink]="url" (click)="goToTeam(teams.teams[num],teams.teams[num+1])">
          <div class="teamdiv">
            <h2 class="teamname">{{teams.teams[num]}}</h2>
          </div>
        </ion-col>
      </ion-row>
    </ion-grid>
  </ion-content>

```

Obr. 30: Zobrazenie používateľových tímov

Na vykreslenie kalendára s udalosťami sme použili prídavný modul Ionicu ionic2-calendar. Tento modul nám umožnil použiť html element calendar (Obr. 31), ktorý pekne vykresluje kalendár s možnosťou prepínania zobrazenia na zobrazenie jednotlivých dní alebo zobrazenie mesiaca prípadne týždňa. Tak isto sa veľmi jednoducho pracuje so zobrazením udalostí nakoľko stačí do elementu poslať pole udalostí, ktoré sú vo formáte json čo nám uľahčilo prácu nakoľko nebolo potrebné vytiahnuté dáta z databázy upravovať. Jediné údaje potrebné upraviť bol dátum a čas nakoľko tie sa do databázy ukladali ako string a element kalendár očakával dátový typ Date.

```

<calendar
  [eventSource]="eventSource"
  [calendarMode]="calendar.mode"
  [currentDate]="calendar.currentDate"
  (onEventSelected)="onEventSelected($event)"
  (onTitleChanged)="onViewTitleChanged($event)"
  (onTimeSelected)="onTimeSelected($event)"
  startHour="6"
  endHour="20"
  step="30"
  startingDayWeek="1">
</calendar>

```

Obr. 31: Použitie html elementu calendar

Záver

Zoznam použitej literatúry

1. KOSTOVA S., VRANIĆ V. Applying aspect-oriented change realization in the mobile application domain. *Proceeding Programming'18 Companion Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France*. 2018. ISBN 978-1-4503-5513-1.
2. TRÚCHLY P., PETRÍK T. *Mobile application supporting an engage in sport and social activities*. Danvers: IEEE, 2014. ISBN 978-1-4799-7738-3. Uverejnené v ICETA 2014.

Prílohy

A	Štruktúra elektronického nosiča	II
---	---	----

A Štruktúra elektronického nosiča