

**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: Informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci záverečnej práce: Ing. Oto Haffner, PhD.
Konzultant: Ing. Erich Stark, PhD.

Bratislava 2020

Bc. Karol Krist



ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Karol Krist**

ID študenta: 79995

Študijný program: aplikovaná informatika

Študijný odbor: informatika

Vedúci práce: Ing. Oto Haffner, PhD.

Konzultant: Ing. Erich Stark, PhD.

Miesto vypracovania: Ústav automobilovej mechatroniky

Názov práce: **Co-working aplikácia pre teamové projekty**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Témou diplomovej práce je vytvorenie co-working aplikácie, ktorá umožňuje komunikáciu a manažovanie práce členom tímu.

Úlohy:

1. Spracujte prehľad podobných aplikácií.
2. Analyzujte potencionálne technické problémy.
3. Analyzujte a naštudovať vybrané frameworky ako Angular/Ionic, Electron alebo Vue.
4. Navrhnite aplikáciu so zvolenými požiadavkami.
5. Implementujte aplikáciu.
6. Otestujte implementovanú aplikáciu a zhodnoťte výsledky.

Zoznam odbornej literatúry:

1. Kostova, S. – Vranić, V. Applying aspect-oriented change realization in the mobile application domain. In *Proceeding Programming’18 Companion Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France — April 09 – 12, 2018*. New York: ACM, 2018, s. 87–92. ISBN 978-1-4503-5513-1.
2. Trúchly, P. – Petrík, T. Mobile application supporting an engage in sport and social activities. In *ICETA 2014*. Danvers: IEEE, 2014, s. 477–482. ISBN 978-1-4799-7738-3.
3. Angular

Riešenie zadania práce od: 23. 09. 2019

Dátum odovzdania práce: 15. 05. 2020

Bc. Karol Krist

študent

Dr. rer. nat. Martin Drozda

vedúci pracoviska

prof. Dr. Ing. Miloš Oravec

garant študijného programu

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

Cieľom tejto diplomovej práce je prieskum aktuálne existujúcich co-working aplikácií a analýza ich nedostatkov. Následne na základe tejto analýzy navrhnut co-working aplikáciu, ktorá zahŕňa fukcionalitu 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 designed application using chosen technologies for implementing. Finally test implemented application and analyze test results.

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

Vyhľásenie 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 11.5.2020

.....

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 podakovať môjmu konzultantom 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 podakovanie 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	5
2.1.3 Doplňky	6
2.1.4 Výber najznámejších importov	6
2.1.5 Cena	7
2.2 Facebook Workplace	7
2.2.1 História	8
2.2.2 Používateľské rozhranie	8
2.2.3 Cena	9
2.3 Microsoft Teams	9
2.3.1 História	9
2.3.2 Používateľské rozhranie	10
2.3.3 Doplňky	11
2.3.4 Cena	11
2.4 Trello	11
2.4.1 Kanbanská nástenka	12
2.4.2 História	12
2.4.3 Používateľské rozhranie	13
2.4.4 Doplňky	14
2.4.5 Cena	14
2.5 Nedostatky aplikácií	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	Architekúra	29
4.6	Apache CouchDB	30
4.6.1	Výhody použitia CouchDB	30
4.7	PouchDB	32
5	Návrh aplikácie	33
5.1	Špecifikácia požiadaviek	34
5.1.1	Používateľské požiadavky	34
5.2	Diagramy	35
5.2.1	Diagramy prípadov použitia	36
5.2.2	Sekvenčné diagramy	38
5.3	Databázový model CouchDB	39
6	Implementácia aplikácie	45
6.1	Prihlasovanie a registrácia	45
6.2	Stránka prihláseného používateľa	49
6.3	Stránka vybraného tímu	51
6.4	Chat	53
6.5	Dizajn aplikácie a odstránenie problémov	53

6.6 Export na desktopovú aplikáciu použitím Electronu	54
Záver	55
Zoznam použitej literatúry	56
Prílohy	I
A Náhľad dizajnu Co-lab aplikácie	II

Zoznam obrázkov a tabuliek

Obrázok 1	Y-base študentský co-working space na internáte STU ŠD Mladost Bratislava	2
Obrázok 2	Co-working centrum v San Franciscu[5]	3
Obrázok 3	Slack aplikácia[6]	5
Obrázok 4	Stránka Facebook Workplace[8]	7
Obrázok 5	Aplikácia MS Teams[10]	10
Obrázok 6	Jednoduchá kanbasnká nástenka[12]	12
Obrázok 7	Náhľad nástenky v aplikácii Trello[11]	13
Obrázok 8	Stránka profilu v aplikácii Trello[11]	14
Obrázok 9	Jednoduchý program Hello World v Node.js[14]	19
Obrázok 10	Architektúra platformy Node.js[15]	21
Obrázok 11	Zjednodušená verzia fungovania Node.js architektúry[17]	22
Obrázok 12	Jednoduchý <i>package.json</i> súbor[18]	23
Obrázok 13	Vývojové prostredie Ionic studio	25
Obrázok 14	Princíp fungovania Angularu[24]	27
Obrázok 15	Firmy využívajúce vo svojich aplikáciách Electron[25]	29
Obrázok 16	JSON dokument v aplikácii na správu CouchDB[26]	30
Obrázok 17	Náhľad PouchDB v prehliadači Google Chrome[27]	32
Obrázok 18	Návrh komunikácie medzi komponentami	33
Obrázok 19	Use case diagram pre používateľské prostredie	36
Obrázok 20	Use case diagram pre administrátorské prostredie	37
Obrázok 21	Use case diagram prostredia člena tímu	37
Obrázok 22	Sekvenčný diagram prihlásenia a registrácie	38
Obrázok 23	Sekvenčný diagram komunikácie komponentov v aplikácii	39
Obrázok 24	JSON dokument používateľa v Couchdb	40
Obrázok 25	JSON dokument udalosti v Couchdb	41
Obrázok 26	JSON dokument tímu v Couchdb	42
Obrázok 27	JSON dokument miestnosti v Couchdb	42
Obrázok 28	JSON dokument príspevku v miestnosti v Couchdb	43
Obrázok 29	JSON dokument správy v Couchdb	43
Obrázok 30	JSON dokument úlohy v Couchdb	44
Obrázok 31	JSON dokument komentáru úlohy v Couchdb	44

Obrázok 32	Funkcia register na strane back-endu	46
Obrázok 33	Funkcia login na strane back-endu	47
Obrázok 34	Vytvorenie jwt tokenu	47
Obrázok 35	Prihlásenie a registrácia na strane front-endu	48
Obrázok 36	Overenie platnosti jwt tokenu	48
Obrázok 37	Nastavenie na live počúvanie zmien databázy	49
Obrázok 38	Načítanie dát prihláseného používateľa	50
Obrázok 39	Zobrazenie používateľových tímov	50
Obrázok 40	Použitie html elementu calendar	51
Obrázok 41	Implementácia vyskakovacieho okna na pridanie člena tímu	52
Obrázok A.1	Úvodná stránka aplikácie	II
Obrázok A.2	Prihlasovacia stránka aplikácie	II
Obrázok A.3	Profil používateľa	III
Obrázok A.4	Kalendár používateľa	III
Obrázok A.5	Tímy používateľa	IV
Obrázok A.6	Úlohy používateľa	IV
Obrázok A.7	Kontakty používateľa	V
Obrázok A.8	Chat s iným používateľom	V
Obrázok A.9	Stránka členov tímu	VI
Obrázok A.10	Miestnosť v tíme	VI
Obrázok A.11	Kalendár tímu	VII
Obrázok A.12	Stránka úlohy	VII
Obrázok A.13	Editovanie úlohy adminom	VIII

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 ľudi je v dnešnej dobe neodmysliteľnou 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 bud platená alebo ich funkcia 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 reakcii 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 si veľa firm vytvorilo vlastnú aplikáciu, ktorá je používana len v rámci danej firmy. Preto sme sa rozhodli vytvoriť použitím moderných technológií na tvorbu aplikácie 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 zdielanie jedných priestorov umožňuje tejto skupine ľudí zdieľať hodnoty, skúsenosti, nápady a profitovať zo synergického efektu, ktorý prináša sústredenie talentovaných pracovníkov na jednom mieste. Tiež umožňuje spoznávanie nových ľudí, ktorí môžu byť v budúcnosti pre človeka dôležití. Najčastejšími využívateľmi co-workingu sú napríklad obchodníci, umelci, programátori, konzultanti, dizajnéri, makléri, agenti, študenti, malé a začínajúce firmy[1, 2, 3, 4].

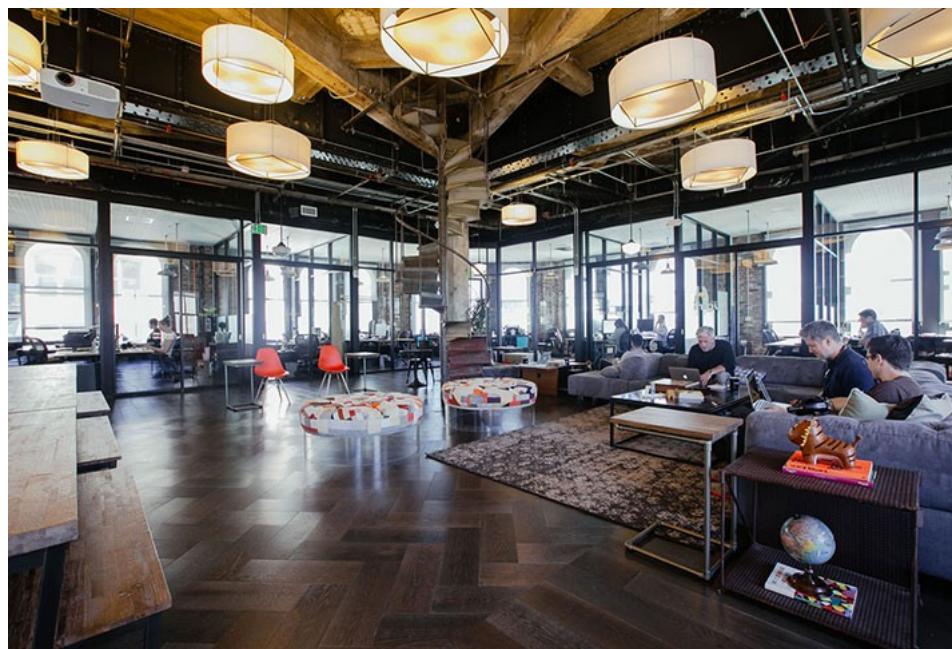


Obr. 1: Y-base študentský co-working space na internáte STU ŠD Mladost Bratislava

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, meetingové priestory, prípadne aj sietová tlačiareň. To znamená, že pre bežného používateľa nevyžaduje začatie práce v takomto centre žiadne počiatočné alebo investičné náklady. Jediným poplatkom je zaplatenie členstva v centre[1, 2, 3, 4].

1.1 História co-workingu

Slovo co-working v zmysle popisu ľudí, ktorí pracujú v akomkoľvek prostredí prvýkrát, použila Bernie DeKoven v roku 1995. K prvému vzniku co-workingového 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 co-working bol známy, ale nikto ho nepočul pod významom ako co-working miesto alebo centrum[5].



Obr. 2: Co-working centrum v San Franciscu[5]

Dnes sa vytváranie co-workingových centier pokladá za globálny fenomén, ktorý v niektorých mestách dosahuje ročný nárast až 24,2%. Predpokladom je, že do roku 2022 bude celosvetovo vytvorených vyše 30500 centier alebo priestorov a viac ako 5,1 milióna členov. Co-working je nová cesta trvalo udržiavateľného spájania pracovného a osobného života. Je to globálny pilier, ktorý bude formovať spôsob našej práce v budúcnosti[5].

2 Co-working aplikácie

Co-working aplikácie sú aplikácie, ktoré slúžia na komunikáciu a manažovanie tímu pri práci viac ľudí na dosiahnutí určitého ciela alebo výsledku. Týchto aplikácií 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 co-workingové centrum v online svete. Keď si zoberieme služby, ktoré ponúkajú co-workingové centrá, tak sa dajú prirovnáť k niektorým službám, ktoré ponúkajú aplikácie. Či už sa jedná o meetingové miestnosti - v aplikácii sú to 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 používateľmi[6].

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 volnom preklade znamená: „Protokol prehľadávania všetkých konverzácií a znalostí.“

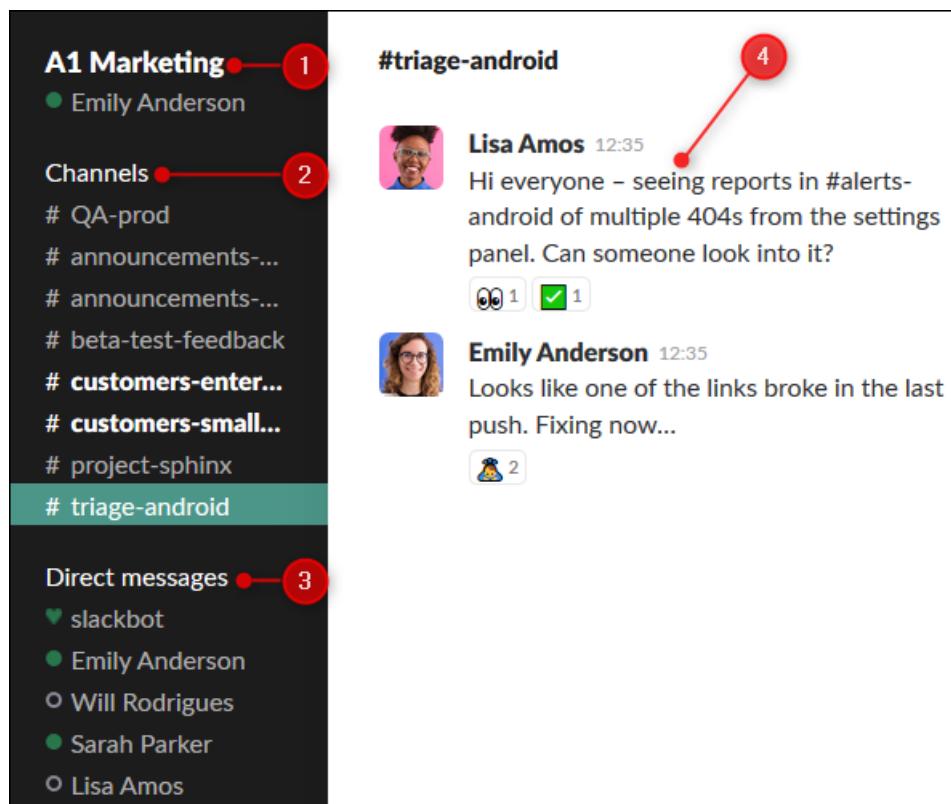
V marci 2015 spoločnosť Slack oznámila, že bola vo februári 2015 opakovane napadnutá hackermi počas 4 dní. Počas tohto útoku boli ohrozené údaje používateľov. Medzi tieto údaje patrili e-mailové adresy, používateľské mená, heslá, telefónne čísla a Skype mená, ktoré boli priradené k ich účtom. Po tomto útoku Slack pridal do svojej aplikácie dvojfaktorovú autentifikáciu[6].

2.1.2 Používateľské rozhranie

Ked' 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 vidľ. 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 pridaní alebo pozvaní. Priame správy sú vždy súkromné, ale môžu obsahovať až 8 ľudí.



Obr. 3: Slack aplikácia[6]

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

2.1.3 Doplňky

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 a prípadne video používania doplnku. Doplnok je možné pridať priamo cez aplikácie Slack alebo cez stránku Slacku.

Doplňky sú na stránke rozdelené do kategórií, do ktorých viac menej patria. Tieto kategórie sú rôzne od botov, ktorí automaticky ako je niečo zmenené na strane doplnkovej aplikácie, pridajú do daného kanála upozornenie alebo správu, až po priame importy, ktoré pridávajú funkciu 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

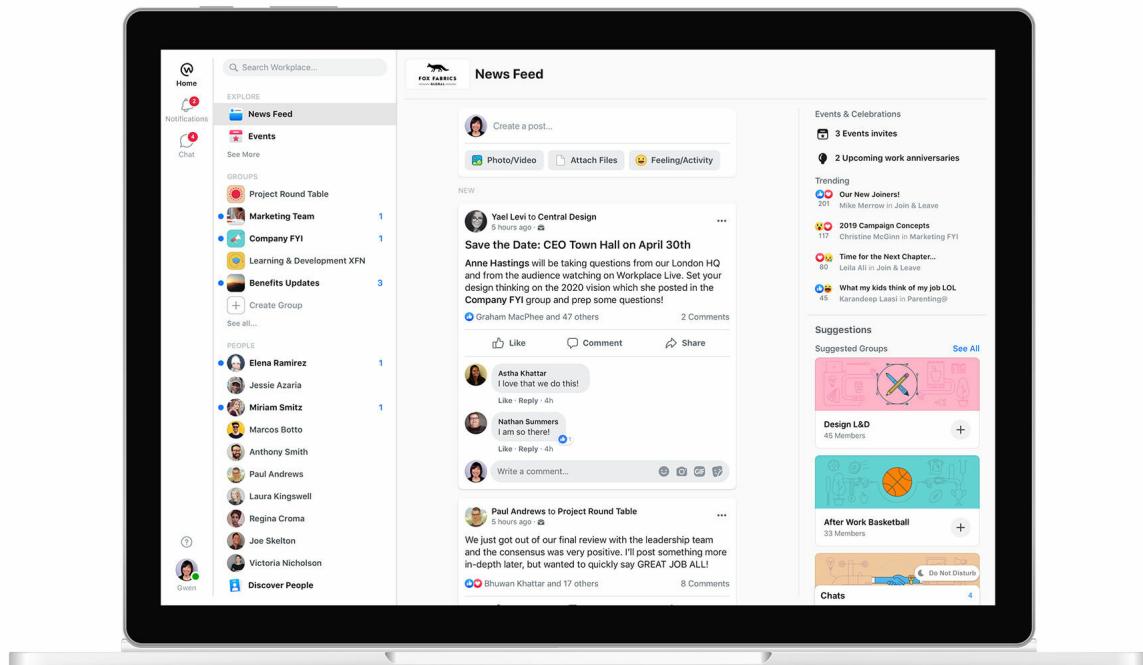
2.1.5 Cena

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 žiadni jedno-kanálový alebo viac-kanálový hostia a limitované možnosti administrácie.

Ak chce používateľ sprístupniť celú funkcia, tak je to dosť drahé. Vychádza to približne 12 dolárov na používateľa mesačne pri ročnej platbe alebo približne 15 dolárov pri mesačnej platbe. Ak teda napríklad máme 1000 člennú skupinu, ktorá chce používať Slack, vychádza to približne 144000 dolárov pri ročnej platbe[6].

2.2 Facebook Workplace

Čas, ktorý ľudia strávili v práci chatovaním, prezeraním a likovaním príspevkov na Facebooku, znižoval produktivitu ľudí v hodnote cca 3,5 bilióna dolárov. Táto téma sa pretriásala cez seriózne diskusie, až po vytváranie vtipných obrázkov. Preto sa ľudia v spoločnosti Facebook začali zaoberať touto otázkou, či sa funkcia Facebooku nedá preniesť aj na pracovnú aplikáciu. Toto dalo počiatok Facebook Workplace, ktorý podľa slov jeho stvoriteľov má podporiť kolaboráciu tým, že ju urobí zábavnou a jednoduchšou pre ľudí[7, 8].



Obr. 4: Stránka Facebook Workplace[8]

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 zavádzat beta verzia s názvom Facebook at Work. Kedže 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 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 vyvájanú platformu od manažérov firiem, kde bol Facebook at Work zavedený. Bolo to hlavne zapríčinené tím, 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[7, 8].

2.2.2 Používateľské rozhranie

Rozhranie Workplacu je podľa slov používateľov na 95% rovnaké ako 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 písat so svojimi kolegami

Pre používateľov je toto sice výhoda, kedže 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, nakolko donedávna 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[7, 8].

2.3 Microsoft Teams

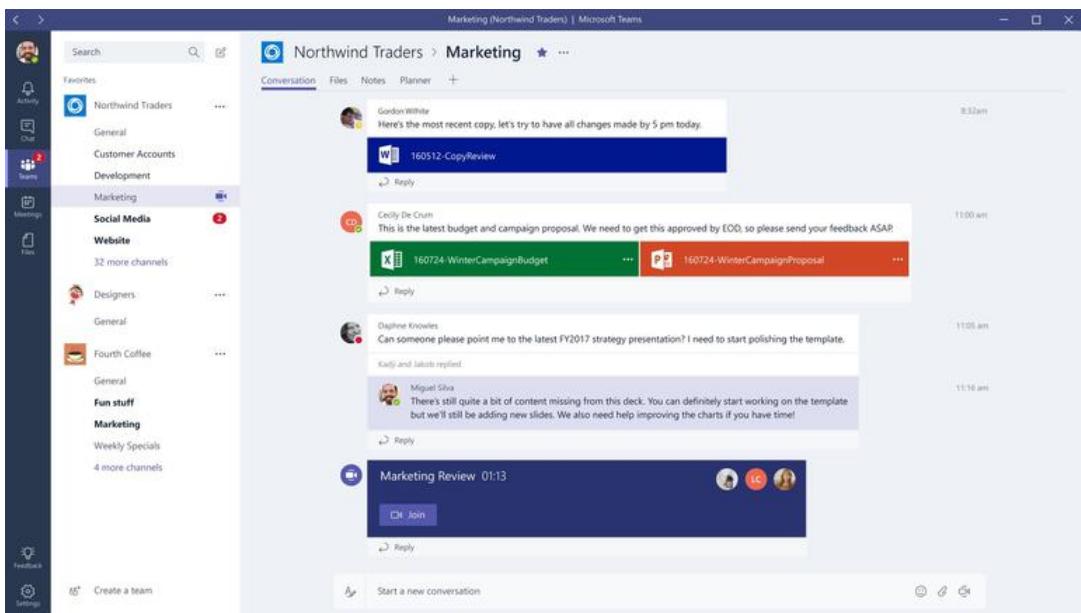
Microsoft Teams je aplikácia na pracovnú komunikáciu a kolaboráciu od spoločnosti Microsoft, ktorá bola vyvinutá, aby konkurovala aplikáciám Slack, Workplace, HipChat. Vo svojej najjednoduchšej podobe je to aplikácia umožňujúca komunikáciu medzi členmi vytvorennej skupiny na báze miestností - kanálov. Microsoft sa už pred vydaním Teams pokúšal presadiť na trhu pracovných aplikácií pomocou svojho Skype for Business. Táto aplikácia však nenaplnila očakávania tak, ako jej lepšia verzia Teams[9].

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 ale 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 využívajúce Slack, začali používať platenú službu Office, ktorej Teams bol súčasťou. Neskôr však Teams rozšíril o 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[10]

V júli 2018 oznámil Microsoft bezplatnú verziu služby Teams s obmedzeniami typu počet používateľov a kapacity ukladania súborov. V novembri 2019 dosiahol Teams 20 miliónov používateľov, čo bol nárast o 7 miliónov od júla 2019[9].

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 pozývanie používateľov. Po prihlásení sa pomocou Microsoft účtu, môžu potom 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 hlasová aj video komunikácia Skype štýlom.

Mimo klasických miestností-kanálov je možne vytváranie aj takzvaných meeting miestností, ktoré slúžia na komunikáciu počas pracovných poriad, meetingov a stretnutí. Tieto miestnosti umožňujú priame pridanie času a dátumu meetingu do kalendára Outlooku. Na náhlade nástenky 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 pracovníci prideľovať a sledovať prácu používateľov. Tak isto sa jednotlivé úlohy na tejto „nástenke úloh“ dajú komentovať, presúvať alebo hodnotiť.

Samořejmostou je veľmi dobrá kompatibilita s inými aplikáciami Microsoftu ako

je Word, Excel, PowerPoint, OneDrive. Jednotlivé dokumenty týchto aplikácií sa dajú priamo posielat, upravovať v aplikácii Teams a tak majú jednotliví používatelia vždy najaktuálnejšiu verziu dokumentu k dispozícii. Microsoft pridal 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 aplikácie prišlo k nejakej zmene – napríklad príde upozornenie do miestnosti, že niektorý používateľ pushol na git niečo nové[9].

2.3.3 Doplňky

Do Teams podobne ako do Slacku je možné pridanie rôznych doplnkov od iných vývojárov. Tieto doplnky sú však 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.

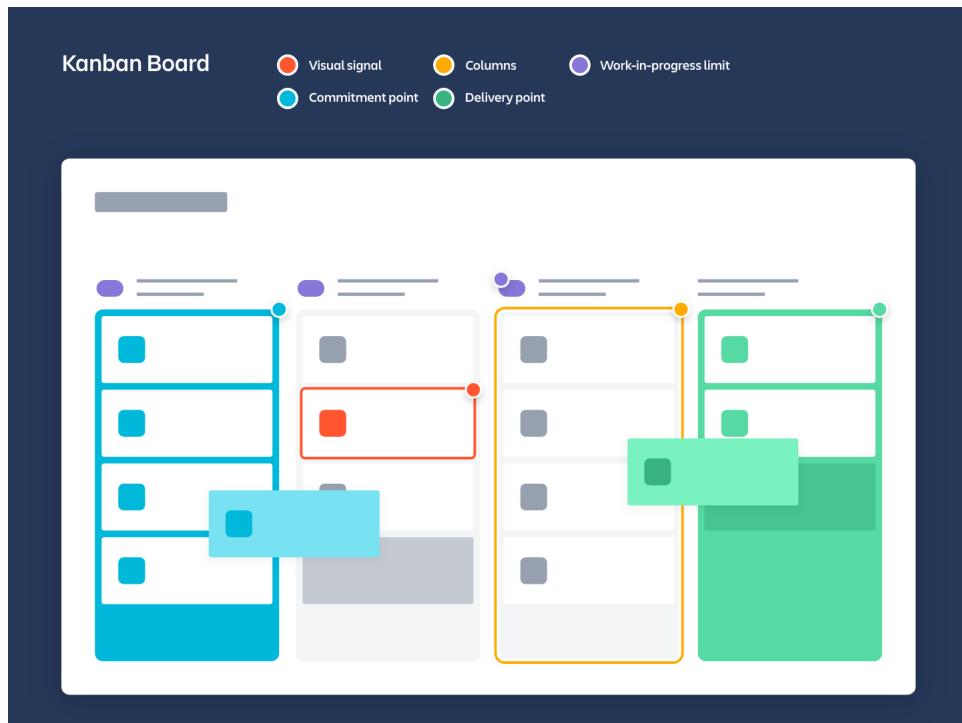
Platených verzii je viac. Tieto verzie sú viazané na balík Office 365, ktorý má používateľ/firma zaplatený. Za balík Office 365 Business Essentials pýta Microsoft 4,20€ za používateľa, mesačne, s ročnou viazanosťou a za balík Office 365 Business Premium 10,50€ za používateľa, mesačne, s viazanosťou na rok. V balíku Premium je zahrnutá kompletná funkcionality aplikácie Teams. Pri balíčku Essentials sú tam stále isté obmedzenia[9].

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[11].

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[12].



Obr. 6: Jednoduchá kanbanská nástenka[12]

2.4.2 História

Trello bolo spustené v roku 2011 spoločnosťou Fog Creek. Za celou myšlienkovou 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 rapídny nárast používateľov, kedy ešte v marci malo Trello 35 miliónov používateľov, ale už v októbri to bolo 50 miliónov[11].

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.

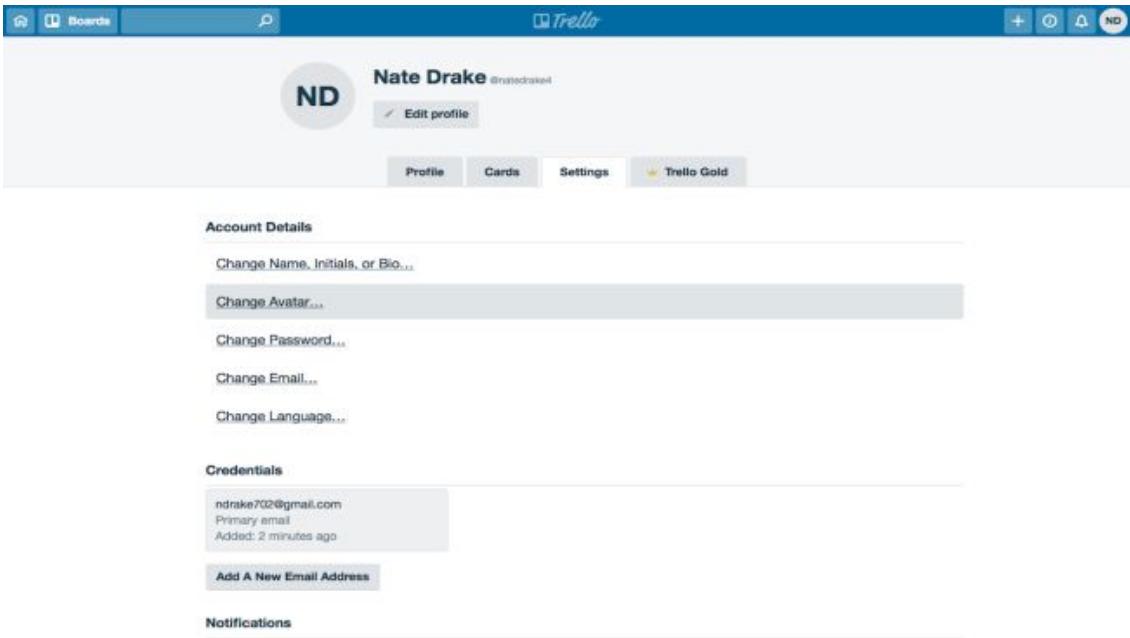
Hned po prihlásení sa používateľovi zobrazí nástenka, kde vidí tímy, v ktorých je pridaný. Tak isto môže vytvoriť novú kartu tímu. Po vytvorení sa otvorí okno tímu, ktoré zväčša pozostáva z 3 stĺpcov – To Do, Doing a Done, plus možnosť pridať ďalšie stĺpce. Tri dopredu vytvorené stĺpce sa samozrejme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



Obr. 7: Náhľad nástenky v aplikácii Trello[11]

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 možne zmeniť meno, pridať fotku profilu, zmeniť iniciály, zmeniť avatara z iniciálok na fotku, nastaviť politiku upozornení alebo pre farboslepých používateľov je možnosť zapnutia módu pre nich[13].



Obr. 8: Stránka profilu v aplikácii Trello[11]

2.4.4 Doplnky

Do jednotlivých násteniek alebo tímov sa dajú pridať rôzne doplnky, ktoré rozširujú možnosti použitia Trella. 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 násteniek, kariet, členov tímu alebo príloh. Je možné pridať jeden doplnok na nástenku a tiež pridanie prílohy o veľkosti do 10MB alebo prepojiť akýkoľvek súbor z Google Drive, Dropbox alebo OneDrive účtom.

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 nástenku a tiež pridať 250MB prílohy. Ďalšími rozširujúcimi vecami je možnosť usporiadania násteniek do kolekcií, nastaviť, kto tieto kolekcie môže vidieť alebo nahratie vlastných pozadí pre nástenky.

Enterprise balík stojí 20,83\$ mesačne, pri ročnom predplatnom. Balík zahŕňa funkcionality Business balíka, plus dvojfaktorová autentifikácia, šifrovanie súborov, vlastná kontrola zabezpečenia, vylepšené SLA[11].

2.5 Nedostatky aplikácií

Pri každej aplikácii, okrem Facebook Workplace, sme mohli vidieť, že využívajú rôzne doplnky. Pre použitie týchto doplnkov musí používateľ zväčša disponovať účtom založeným v danej platforme doplnku. 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 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 ako doplnky.

Druhým veľkým nedostatkom, ktorý by sme chceli našou aplikáciou odstrániť je nutnosť platenia. Je samozrejmé, že pre veľké firmy nie je problém si radšej priplatíť za stabilnú a 100% funkčnú aplikáciu, ktorá poskytuje veľké množstvo služieb aj za cenu pridania veľkého množstva doplnkov, ale pre študentov je odstránenie potreby platenia za aplikáciu veľkým prínosom. Jedná sa hlavne o odstránenie limitu používateľov bez poplatku za aplikáciu. Pri niektorých aplikáciách je 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 funkčionalitu ako má Slack bez doplnkov, čiže vytvorenie tímov a komunikácia v nich. K tomu pridáme funkčionalitu doplnkov, ako je kalendár s udalosťami a jednoduchú nástennu úlohu 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 vyvýjaného softvéru problém. Preto musí vývojár už počas vyvýjania 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, najskôr si musíme povedať, ako bude naša aplikácia vyvýjaná. Po zvažení sme sa rozhodli, že aplikácia bude vyvýjaná 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-aplikácií sa vývojár nemusel zamýšľať nad tým, na akom zariadení bude používateľ spúštať web-aplikáciu (smartfóny neexistovali) a jediné zariadenie, na ktorom mohol používateľ otvoriť web-aplikáciu, bol počitač cez prehliadač. Dnes treba myslieť na to, že používateľ môže aplikáciu otvárať na smartfóne, tablete alebo inom zariadení. Preto musia byť web-aplikácie vyvýjané spôsobom, aby sa zobrazovaná stránka vedela natívne prispôsobiť veľkosti obrazovky, na ktorej je zobrazovaná.

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

Na odstránenie toho potencionálneho problému sme sa rozhodli našu aplikáciu vyvýjať v prostredi Ionic, ktoré nám umožňuje aplikáciu vyvýjať zároveň pre viac platform. V Ionicu je možné aplikáciu vyvýjať pre štandardný webový prehliadač, zároveň je aplikácia vyvýjaná aj pre webové prehliadače v smartfónoch a tabletoch. Samozrejme, že je ale potrebné využívanie responzívnych prvkov a elementov, aby sa vedeli automaticky prispôsobiť veľkosti obrazovky. Pred začatím vývoja sme sa dohodli web-aplikáciu po implementovaní vyexportovať pomocou frameworku Electron do štandardnej aplikácie pre platformu Windows. Pomocou Electronu je 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 aplikácií zameraných na určitú potrebu používateľa veľmi podobá. Je to hlavne preto, aby používateľ mohol hned s aplikáciou pracovať a nemusel sa učiť, ako ju používať. Tento problém bol už čiatočne spomenutý pri probléme s používateľským rozhraním, napokoľko je to s ním úzko späté. Na odstránenie tohto problému sme sa preto rozhodli vyvýjať našu aplikáciu tak, aby bola podobná aplikácii Slack, čo sa týka vizuálnej stránky. Používateľ, ktorý používal už v minulosti aplikáciu Slack bude hned vedieť, ako využívať aj našu aplikáciu a tak isto bude design našej aplikácie prispôsobený tak, aby aj používateľ, ktorý nevyužíval podobnú aplikáciu hned 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 vyvýjať 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 písat 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 ktorémukolvek používateľovi. Ako sme už vyššie spomnuli, naša aplikácia bude vyexportovaná ako desktopová aplikácia, čiže ju pôjde spustiť bez pripojenia na internet, ale nastáva tu problém už pri prihlásení. Na prihlásenie bude potrebné byť pripojený na internet, napokoľko sa prihlásovacie údaje musia overiť s online databázou. Potom už bude možné stratit pripojenie na internet, napokoľko využijeme replikáciu databázy na lokálnu databázu PouchDB. Táto databáza preberie všetky dátá pri prihlásení z online databázy a budeme ich môcť zobrazovať. Tak isto, hned pri akomkoľvek zápisе do online databázy, je táto lokálna databáza hned aktualizovaná, ak je zariadenie opäťovne pripojené na internet. Ak sa pripojenie stratí, bude zobrazovať len tie dátá,

ktoré boli v databáze do straty pripojenia.

3.5 Bezpečnosť

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

V našej aplikácii sme sa hlavne zamýšlali nad zabezpečením dát pri práci s nimi. Tu sme sa tiež rozhodovali, ktoré dátá bude treba zabezpečiť a ktoré nie. Za najdôležitejšie dátá sme zvolili prihlásovacie heslo používateľa. Ostatné dátá nie sú až také dôležité a aj s prihliadnutím na náročnosť aplikácie sme sa rozhodli zabezpečiť len heslo. Heslo bude hned pri registrácii na strane servera zašifrované pomocou prípadnej knižnice do node.js bcrypt a tak uložené do databázy. Po zašifrovaní už heslo nikdy nebude rozšifrovávané, čiže pri prihlásovaní sa zadané heslo pomocou funkcie knižnice bcrypt overí so zašifrovaným heslom a na základe toho bude overená správnosť zadaného hesla.

Zabezpečovanie servera nebudeme realizovať, keďže aplikácia bude testovaná len na lokálnom stroji. Ak by sa ďalej uvažovalo o distribúcii aplikácie, bude 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. Jedným 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 funkcionalita.

4 Použité technológie

V nasledujúcej kapitole zhrnieme technológie, ktoré sme sa rozhodli použiť na implemetovanie našej aplikácie. Ku každej technológií popíšeme možnosti, ktoré plánujeme využiť, nakoľko rozpísaať 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 sietových aplikácií. Na nasledovnom príklade jednoduchej aplikácie Hello World môžeme vidieť, že sa mnoho spojení dá zvládnut súčasne - Obr. 9. Po každom spojení sa spustí spätné volanie - callback, ale ak nie je potrebné vykonat 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[14]

Toto je v rozpore s dnešným bežnejším a modernejším prístupom založeným na vláknoch operačného systému, ktoré bežia súčasne. Vytváranie vlákov pri sietovej 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[14].

4.1.1 História

Node.js bol vytvorený Ryanom Dahlom v roku 2009 a bol podporovaný len pre platfromy Linux a MacOS. Pre porovanie 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 JavaScriptovej enginu V8 od spoločnosti Google, udalostnej služky (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 zdielanie a publikovanie zdrojových kódov doplnkových modulov. Zároveň uľahčuje ich stahovanie, 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 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 ES6 z V8 a zároveň umožnila dlhodobo trvajúci vývoj platformy.

4.1.2 Architektúra Node.js

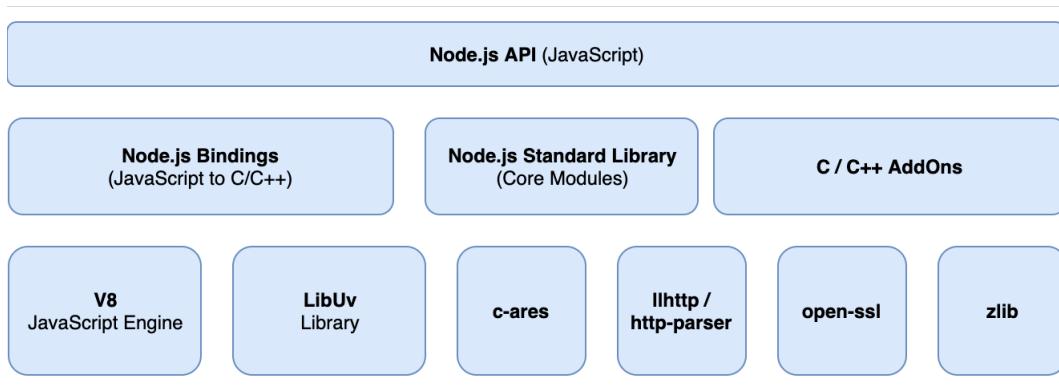
Každá platforma má svoju architektúru, tak isto aj Node.js. V tejto kapitole rozoberieme jej hlavné vlastnosti a časti. Nie všetky budeme potrebovať pri implementácii, ale je dobré ich poznať.

Základnou vlastnosťou je, že Node.js má asynchronné a udalostne riadené API. Toto zaručuje to, že nasledujúce volania nie sú blokované 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 služky. Vďaka tomu je jednovlákновý 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, posielané na výstup, sú zoskupované do malých blokov.

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



Obr. 10: Architektúra platformy Node.js[15]

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 sietových 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[15, 16].

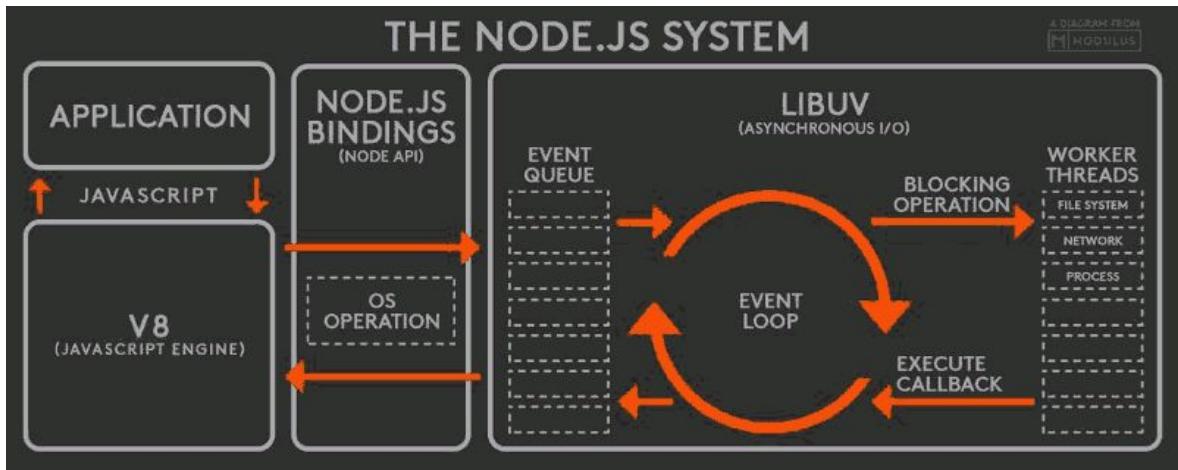
Na spodku sa náhádzajú knižnice C/C++:

- V8 JavaScript Engine - konvertuje JavaScript do strojového kódu daného operačného systému
- LibUV - multiplatformová C knižnica zameriavajúca sa na asynchronné I/O operácie
- c-ares - C knižnica pre asynchronné DNS požiadavky a odpovede
- http-parser - C knižnica pre HTTP požiadavky a odpovede
- open-ssl - kryptografické funkcie
- zlib - C knižnica pre synchronnu aj asynchronnu 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 jednovláknový. Kedže 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 architektúry[17]

Ked sa Node.js spustí tak inicializuje slučku udalostí a spustí poskytnutý vstupný skript, ktorý môže uskutočňovať asynchronné 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 zaradí na spodok fronty udalostí a jej výsledok je teda znova spracovaný, keď sa k nemu slučka dostane. Toto sa opakuje kým nie je fronta udalostí vyčerpaná alebo nie je dosiahnutý maximálny počet spätných volaní[17].

4.1.4 Nevýhody Node.js

Ako sme vyššie popísali, Node.js má veľké množstvo výhod, medzi ktoré hlavne patrí asynchronné I/O, škalovateľnosť, 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...
- nepochopenie 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 vyvýjame. 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 tisícky voľných balíkov na stiahnutie a používanie. NPM program sa automaticky nainštaluje pri inštalácii Node.js. NPM sa následne dá spúštať z príkazového riadku. Tak isto je možné vytvorenie vlastného balíka a nahratie ho do centrálneho 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 definové vlastnosti, meno a verzia balíka. Na Obr. 12 môžeme vidieť, ako takýto súbor vyzerá[18, 19].

```
{  
  "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[18]

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 Bradleym 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 ale tiež možné zvoliť natívne Ionic componenty 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 perifériá potom môže aplikácia využívať pre svoju funkčnosť.

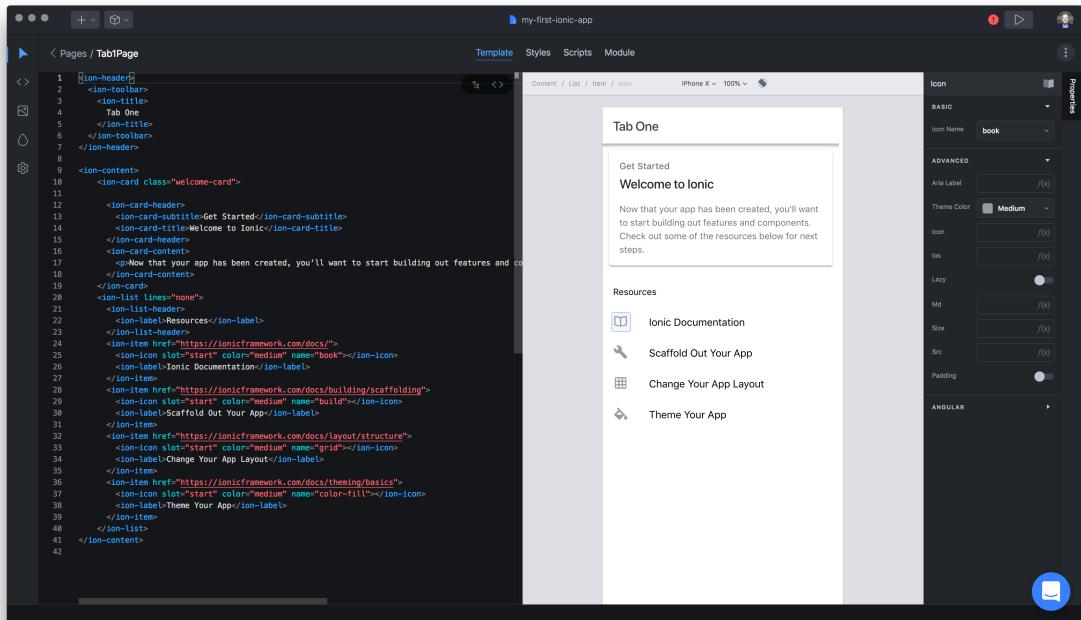
Capacitor

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 odomkné celú natívnu funkčnosť každej platfromy prostredníctvom konzistentných API medzi platformami. S použitím Capacitora nemusí vývojár spravovať viac rozhraní API pre každú platformu, ale môže namiesto toho využívať aplikáciu ako celok, bez nutnosti uvažovať na akej platforme bude aplikácia spustená[20].

Cordova

Podobne ako Capacitor je Cordova open-source projekt umožňujúci spustenie web aplikácie na rôznych platformách. Narozenie od Capacitoru nie je Cordova prispôsobená na využitie Electronu pre export na desktopovú aplikáciu. Preto sa dnes už viacej využíva novší Capacitor[20]. Ionic využíva všetky doteraz dostupné webové komponenty s tým, že ich funkčnosť zrýchluje 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 pro-

stredie 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 využívať 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 využívaná aplikácia nemusí dobre fungovať. Využívanie aplikácií pomocou Ionicu je možné na ktoromoľvek operačnom systéme - Windows, Linux, MacOS.

Prvým krokom pri inštalácii je nainšalovanie 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 Cordova respektíve Capacitor 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 zadá *angular*, čiže aplikácia bude využívaná pomocou frameworku Angular a bude sa používať Capacitor. Kedže Ionic je NPM modul, je samozrejmé, že do aplikácie je možné doinštalovať tak-

mer akýkoľvek ďalší NPM modul na rozšírenie funkčnosti aplikácie. Pri vývoji aplikácie je ešte potrebné pomocou doplnku Cordova alebo Capacitor špecifikovať na ktorú platforem má aplikácia fungovať. Toto sa urobí príkazom *ionic cordova/capacitor platform add ios/android* zadaným do príkazového riadku. Je možné pridať aj obidve platfromy.

4.4 Angular

Angular je framework na vývývanie webových aplikácií vyvinutý a spravovaný spoločnosťou Google. Googlem 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 používanejšia jeho druhá verzia Angular 2+, označovaná tiež ako Angular, čo označuje všetky verzie od verzie 2, ktoré sú narozenie od prvej verzie založené už na TypeScripte. Hned 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 narozenie iných frameworkov doby, keď Angular vyšiel, je Angular výlučne objektovo-orientovaný a jeho syntax je prekvapivo blízky Java 8, aj keď je založený na TypeScripte[21, 22, 23].

4.4.1 História

Prvá verzia AngularJS bola vydaná ešte v roku 2010. Ako bolo vyše spomenuté, táto verzia bola založená na JavaScripte. Aj keď po vydaní novšej 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 vytvájaného jazyka ArtScript. Po ohlásení však Microsoft súhlásil s pridaním podpory pre nimi vytvájaný TypeScript a tak bol Angular 2 založený ako TypeScriptový framework. Zaujímavosťou je, že hneď od vydania Angularu 2 bolo možné v ňom vývíjať 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 ohľásený 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 oficálne vydaný 23. marca 2017 a priniesol novinky ako HttpClient - vylepšená knižnica pre vytváranie HTTP požiadaviek, nový životný cyklus udalostí a iné menšie vylepšenia.

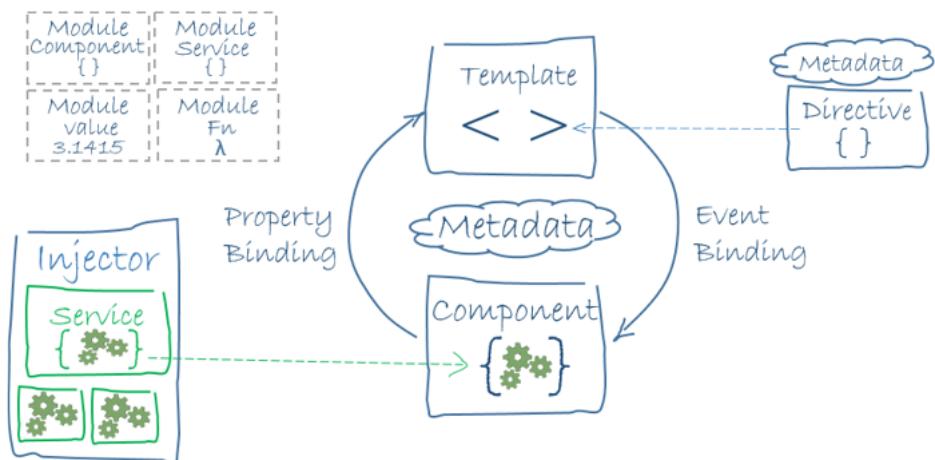
Dnes je už vydaná verzia 9, pričom každá verzia (5,6,7,8) priniesla vždy hlavne podporu nových Material Dizajnov, podporu nových knižníc a podporu tvorby progresívnych

web aplikácií. Google sa pri vydávaní nových aktualizácií zaručil, že každá nová verzia bude späť kompatibilná a nové verzie chce vydávať dva krát ročne[21, 22].

4.4.2 Architektúra

Základ Angularu tvoria NgModuly. Tieto moduly poskytujú komplikáciu komponentov, čo sú druhá základná zložka architektúry Angularu. NgModuly zhlukujú súvisiace kódy do 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 usporiadane ďalšie komponenty. Komponenty v sebe používajú aj služby, ktoré poskytujú špecifické funkcie, ktoré nie vždy priamo súvisia so zobrazením. Poskytovatelia týchto služieb sú vkladaní do komponentov ako závislosti. Týmto sa stáva kód modulárny, opakovateľne použiteľný a efektívny. Komponenty aj služby do nich vkladané sú označované dekorátormi, ktoré určujú, o aký typ komponentu a služby ide a poskytujú takiež 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[24].



Obr. 14: Princíp fungovania Angularu[24]

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 Angulare 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 zaistuje to vyššiu opäťovnú použiteľnosť komponentov v celej aplikácii. To umožňuje vytváranie používateľských rozhraní s mnohými pohyblivými časťami a zároveň zrýchluje priebeh vývoja aplikácie[22].

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 zistovaní 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, vysvetluje, že prechod z jazyka JavaScript do jazyka TypeScript je opodstatnený nástrojmi pre veľké projekty v podnikovom meradle. TypeScript má lepšie služby navigácie, automatického doplňania a refaktoring[22].

RxJS: efektívne asynchronné programovanie

RxJS je knižnica bežne používaná v Angular na spracovanie asynchronných dátových volaní. Umožňuje nezávisle a paralelne spracovať 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 asynchronné programovanie, ale táto knižnica uľahčila vela vecí[22].

Dlhodobá podpora

Niektoří 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[22].

Silná komunita

Kedže 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[22].

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[25].



Obr. 15: Firmy využívajúce vo svojich aplikáciách Electron[25]

4.5.1 Architekúra

Elektronové aplikácie sa skladajú z viacerých procesov. Existuje proces „prehliadača“ a niekoľko procesov „vykreslenia“. Proces prehliadača spúšta aplikačnú logiku a potom môže spustiť viacero procesov vykreslovania, 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.

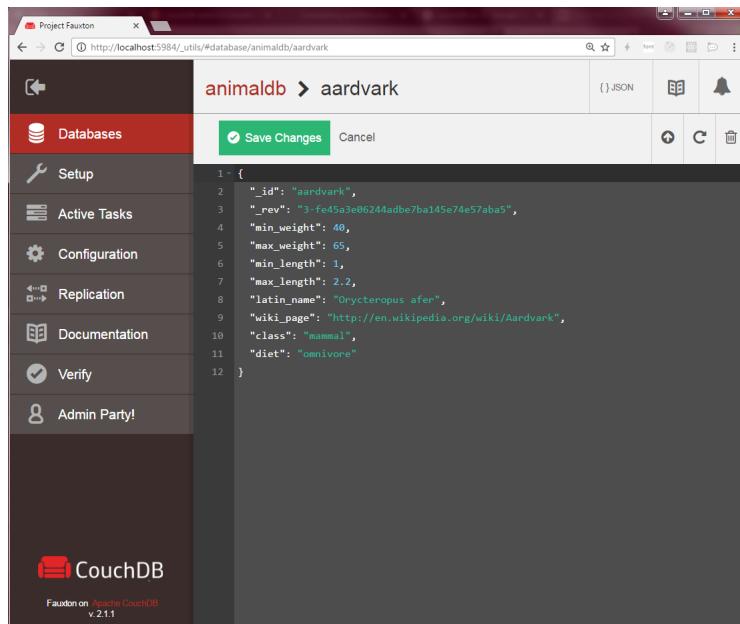
Väčšina rozhraní Electron

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 osobitne zaoberať[25].

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 rodzaj 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 aktívou komunitou vývojárov, ktorí neustále vylepšujú softvér so zameraním na jednoduché používanie webu[26].



Obr. 16: JSON dokument v aplikácii na správu CouchDB[26]

4.6.1 Výhody použitia CouchDB

CouchDB predstavuje celý rad výhod zameraných na používateľov a vývojárov. Motívaciu vývoja CouchDB možno definovať jedným slovom: relax. CouchDB prichádza s množstvom výhod navrhnutých na zníženie námahy pri prevádzkovaní pružného distri-

buovaného 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 rozdeľenie 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 údajov alebo počet používateľov[26].

Žiadne read-locks

Vo väčšine relačných databáz - kde sú údaje uložené v tabuľkách - ak sa niekedy potrebuje aktualizovať alebo upraviť tabuľka, 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 v 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 verziované 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[26].

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 zjednodušíť 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[26].

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, rýchlu a výkonnou na použitie pri zachovaní vysokej dostupnosti[26].

Stavaná pre offline používanie

Ked' 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[26].

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[27].

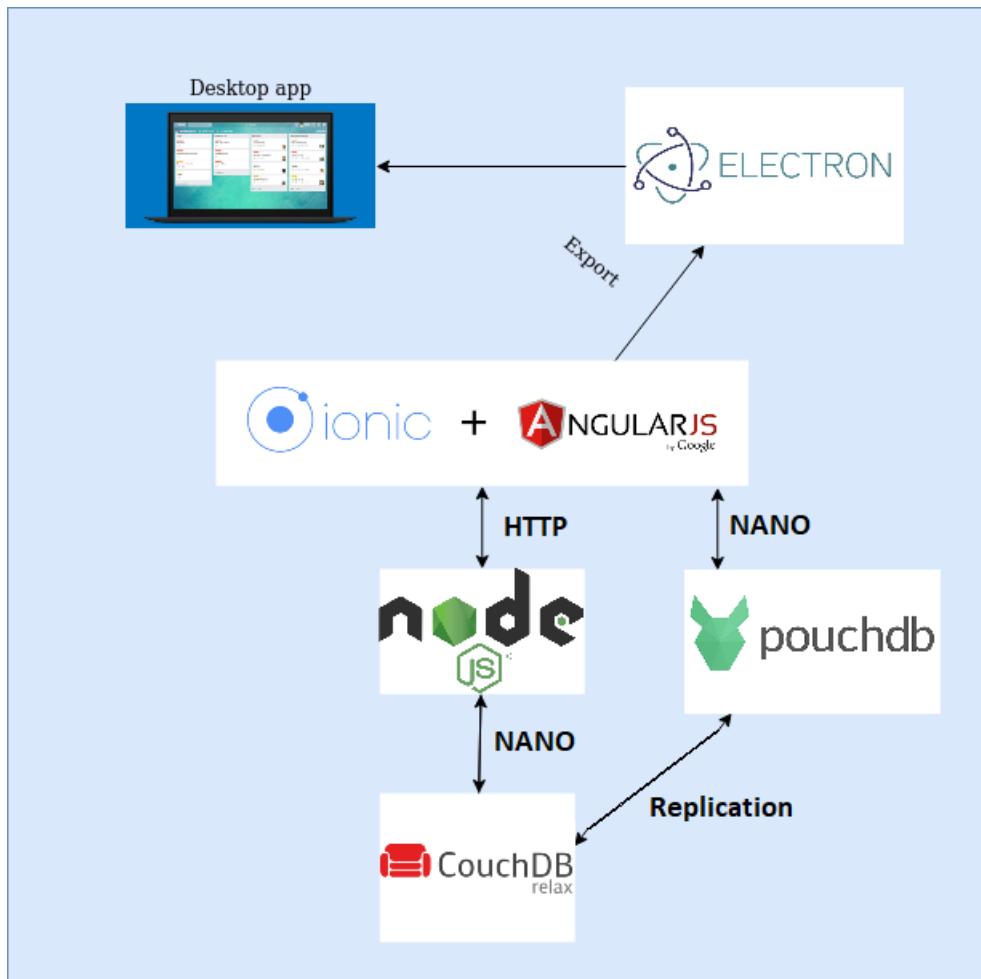
The screenshot shows the Google Chrome DevTools Application tab. On the left, there's a tree view under 'Storage' with nodes for Local Storage, Session Storage, IndexedDB, and a specific instance of _pouch_users at http://127.0.0.1. This instance has sub-nodes for attach-store, local-store, by-sequence, and detect-blob-support. The 'by-sequence' node is expanded, showing a table with three rows. The table has columns for '#', 'Key (Key path: "_doc_id_rev")', and 'Primary key'. The rows are numbered 0, 1, and 2, corresponding to document IDs paul54@gmail.com, peter43@gmail.com, and sofia23@gmail.com respectively. The 'Primary key' column contains values 3, 1, and 2.

#	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[27]

5 Návrh aplikácie

Témou diplomovej práce je implementovať 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 dátá a zároveň posielat žiadane dátá na frontend implementovaný pomocou typscriptovo založeného frameworku Ionic. Ionic sa, 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 Win-

dows/Linux/MacOS.

5.1 Špecifikácia požiadaviek

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

5.1.1 Používateľské požiadavky

Používateľské požiadavky sa stanovili na základe prehľadu co-working aplikácií, ktoré už existujú. Za základ sa zobrali hlavné funkcionality Slacku a k nim sa pridali buď doplnky Slacku alebo funkcie iných aplikácií. 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 prepínat. Administrátorské prostredie bude umožňovať vytváranie tímov, pozývanie a pridávanie členov tímu a manažovanie taskov. Prostredie člena tímu bude umožňovať vytváranie miestností na stránke tímu, vytváranie stretnutí, pridávanie príspevkov do miestností, 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 aministrátor a používateľ

Člen tímu:

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

Nefunkcionálne požiadavky:

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

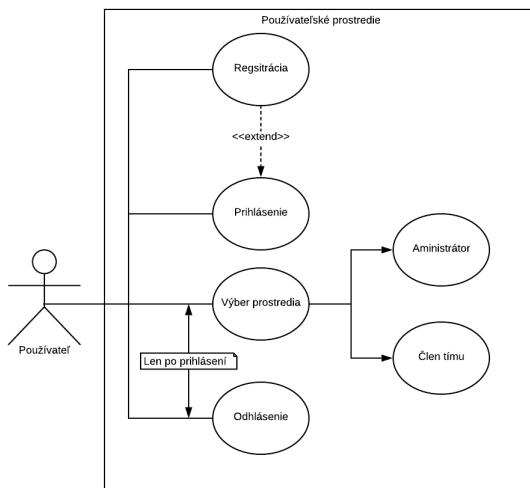
5.2 Diagramy

Po zadefinovaní požiadaviek môžeme pristúpiť 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 jednotlivý 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 nebudem, nakoľko nami používaný JavaScript ani TypeScript nie sú klasické objektovo orientované jazyky. Diagramy, ktoré ale budeme určiť vytvárať sú sekvenčné. Tie nám vizualizujú interakciu medzi komponentami alebo časťami kódu v aplikácii.

5.2.1 Diagramy prípadov použitia

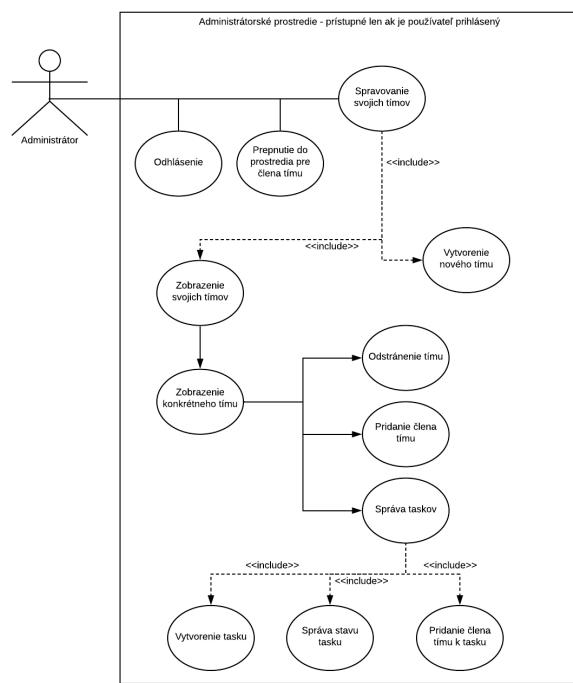
Diagramy prípadov použitia sa používajú na popísanie správania systému z hľadiska jeho používania. Ukazujú aký rôzny používatelia môžu systém používať a aké činnosti môžu v tomto systéme vykonávať.

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.



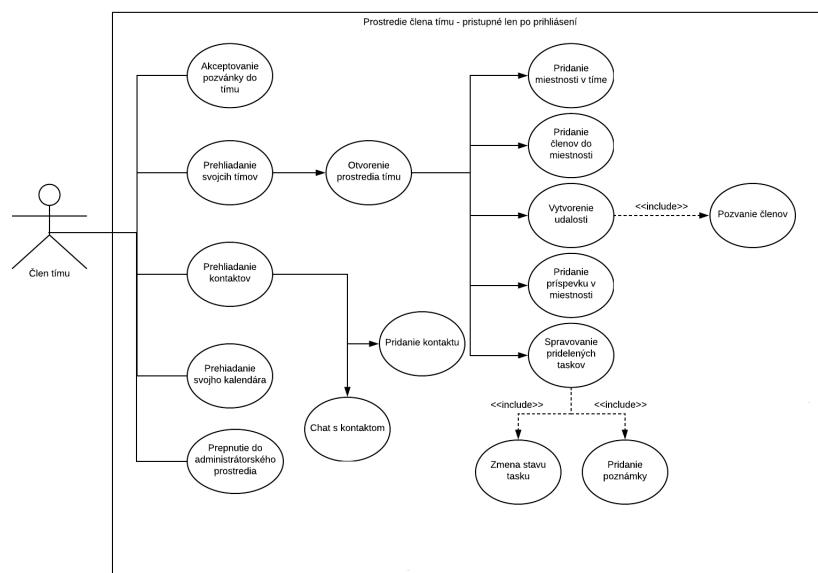
Obr. 19: Use case diagram pre používateľské prostredie

Druhý diagram - Obr. 20 nám ilustruje prípady použitia ak používateľ vybral možnosť prihlásiť sa ako administrátor, prípadne ak sa člen tímu prepol do admnistrátorského rozhrania. Administrátor teraz môže buť 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.



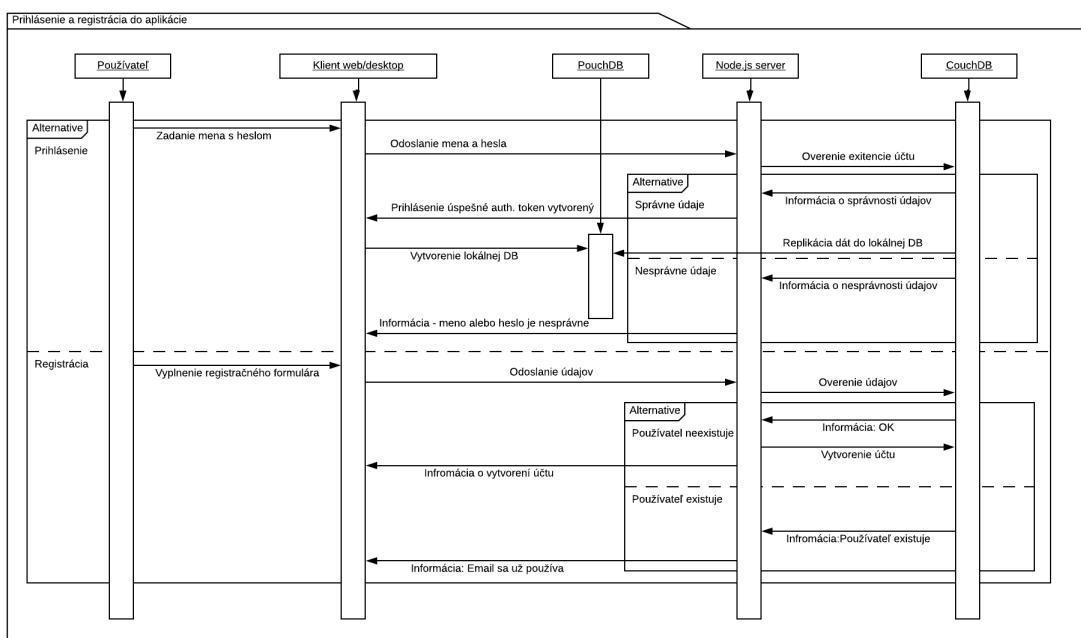
Obr. 21: Use case diagram prostredia člena tímu

5.2.2 Sekvenčné diagrame

Nasledujúce diagrame budú asi najpoužívanejšie pri implementácii aplikácie, nakoľko sekvenčné diagrame 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 a po vyplnení údajov sa 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 kientovi. Na strane klienta sa potom vytvorí lokálna databáza, ktorá si hned z online databázy stiahne všetky dátá a aplikácia tu začína pracovať len s lokálnou databázou.

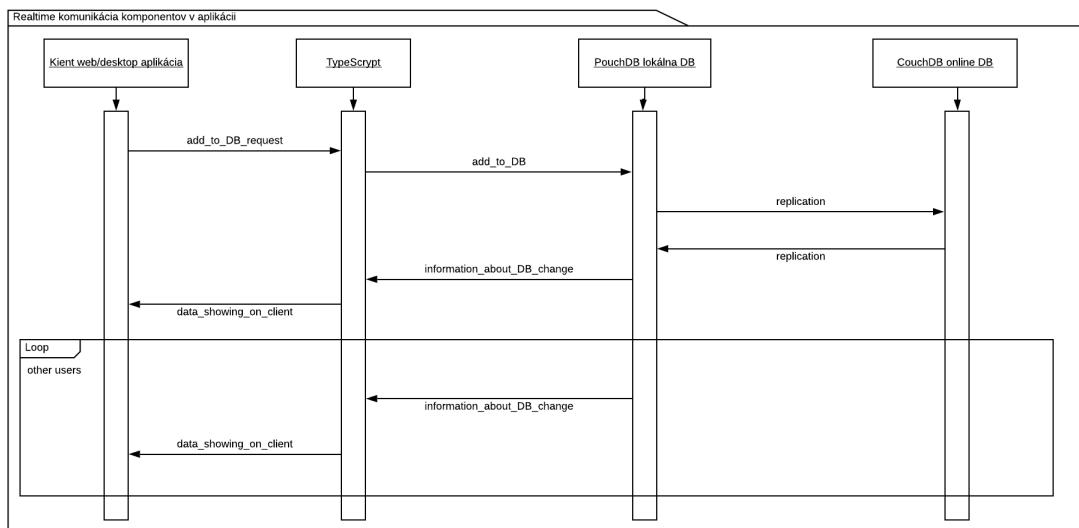


Obr. 22: Sekvenčný diagram prihlásenia a registrácie

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 úspešnejej 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 dátá pridané do lokálnej databázy PouchDB a ak má zariadenie, na ktorom beží aplikácia prístup na internet, sú tieto dátá hned zreplikované do online databázy. Ak prístup na internet nie je, tak sú dátá zatial 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 dátá, 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 jednoduchých 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 spolu s ich neskoršou reprezentáciou v JSON dokumentoch na strane CouchDB. Rozdiely oproti návrhu a ich neskoršou reprezentáciou sú spôsobené potrebou zmeny pri implementácii aplikácie kvôli zjednodušeniu jej behu.

Používateľ:

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

```
{  
    "_id": "5327c7a2d0b9a3c0dcfe57f57100a8cd",  
    "_rev": "53-6de4c75fd6bc408976133674e85957e0",  
    "name": "Karol Krist",  
    "email": "karolkrist1@gmail.com",  
    "password": "$2b$10$bFpSsSYfOWtC4Tr/7cZoiOpn0JNm6p5gaLzVo3YZfY01StgmgIZ2",  
    "teams": [  
        "Karol team",  
        "37e03387-731d-485b-9c36-037dce1cf589",  
        "Karol 2 team",  
        "eecd3944-139f-4e0f-9b7a-3598e6cb3368",  
        "Karol 3 team",  
        "dda0467b-f046-4102-96fa-c9f793234750"  
    ],  
    "calendar": [],  
    "contacts": [  
        "wildfireyesie@gmail.com",  
        "60760917-15e5-4684-b9e9-d05d8fcbe72"  
    ],  
    "tasks": [  
        "Task 1",  
        "e530fb0b-921c-46f6-9000-1c7cfb53131c"  
    ],  
    "info": "My name is Karol Krist and I am student of STU FEI Bratislava."  
}
```

Obr. 24: JSON dokument používateľa v Couchdb

Udalosť:

- názov
- tím

- začiatok
- koniec
- popis

```
{  
    "_id": "06d4c4f6-d3d4-41c5-a273-a9d19a2b1d50",  
    "_rev": "1-a3d430fe25e4e406c97457091df8ddff",  
    "title": "Test Event 1",  
    "startTime": "2020-04-23T14:41:56.932Z",  
    "endTime": "2020-04-23T15:41:56.932Z",  
    "allDay": false,  
    "desc": "Karol team test event 1",  
    "owner": "37e03387-731d-485b-9c36-037dce1cf589"  
}|
```

Obr. 25: JSON dokument udalosti v Couchdb

Tím:

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

```
{  
    "_id": "37e03387-731d-485b-9c36-037dce1cf589",  
    "_rev": "3-74c584b4544794938c9f70d269536c90",  
    "team_name": "Karol team",  
    "admin": [  
        "karolkrist1@gmail.com",  
        "Karol Krist"  
    ],  
    "users": [  
        "karolkrist1@gmail.com",  
        "Karol Krist",  
        "wildfireyesie@gmail.com",  
        "Jozko Mrkvicka"  
    ]  
}
```

Obr. 26: JSON dokument tímu v Couchdb

Miestnosť:

- názov
- členovia
- príspevky

```
{  
    "_id": "e3aceee8d-6fb0-4921-9c65-1ee42d7339e5",  
    "_rev": "1-d8e465e7fac824a7548c13d4f38ad619",  
    "room_name": "Room1",  
    "team_id": "37e03387-731d-485b-9c36-037dce1cf589"  
}
```

Obr. 27: JSON dokument miestnosti v Couchdb

Príspevok:

- text
- autor
- čas pridania
- miestnosť

```
{  
  "_id": "132a2669-d912-4c42-a1ad-f95c7874d4b2",  
  "_rev": "1-5e82d51bf44d68e18f68d02225cd50b2",  
  "text": "Lets go ",  
  "user_name": "Karol Krist",  
  "time": "2020-04-19T14:37:21.615Z",  
  "room_id": "e3acee8d-6fb0-4921-9c65-1ee42d7339e5"  
}
```

Obr. 28: JSON dokument príspevku v miestnosti v Couchdb

Správa:

- text
- autor
- komu
- čas odoslania

```
{  
  "_id": "4bb7c97a-611c-48ed-94eb-7c8c205ba530",  
  "_rev": "1-50b9af3715d9686d89c93505b2392b0c",  
  "chat_id": "60760917-15e5-4684-b9e9-d05d8fcbe72",  
  "text": "Ahoj",  
  "from": "wildfireyesie@gmail.com",  
  "to": "karolkristi@gmail.com",  
  "time": "2020-04-26T16:09:16.345Z",  
  "msg": "message",  
  "from_name": "Jozko Mrkvicka",  
  "to_name": "Karol Krist"  
}
```

Obr. 29: JSON dokument správy v Couchdb

Úloha:

- názov
- vedúci úlohy
- text
- tím
- pridelení používateľa

- stav úlohy
- komentáre

```
{
  "_id": "e530fb0b-921c-46f6-9000-1c7cfb53131c",
  "_rev": "7-d048e82068ca0cf5b30b3e8f2d17942",
  "task_team_id": "37e03387-731d-485b-9c36-037dce1cf589",
  "task_name": "Task 1",
  "task_admin": "karolkristi@gmail.com",
  "task_admin_name": "Karol Krist",
  "task_desc": "Finish final thesis with documentation and all other stuff",
  "team_name": "Karol team",
  "status": "Done",
  "participants": [
    "Jozko Mrkvicka",
    "wildfireyesie@gmail.com"
  ],
  "start_date": "2020-05-01T11:15:19.810Z",
  "finish_date": "2020-05-01T14:43:53.406Z"
}
```

Obr. 30: JSON dokument úlohy v Couchdb

Komentár:

- text
- autor
- čas pridania
- úloha

```
{
  "_id": "c9e55ad4-8a03-4775-be28-24c02a5a8273",
  "_rev": "1-202e9e038c93c67e048ab7d44fb9db0",
  "text": "Koment 1",
  "author": "Karol Krist",
  "time": "2020-05-01T14:27:30.295Z",
  "task_id": "e530fb0b-921c-46f6-9000-1c7cfb53131c"
}
```

Obr. 31: JSON dokument komentáru úlohy v Couchdb

6 Implementácia aplikácie

Po vypracovaní kompletného návrhu sme začali s implementáciou. Tá sa dala rozvrhnúť na viac fáz podľa toho, ako sme postupne implementovali aplikáciu. Začali sme s implementáciou prihlásovacieho a regisitračného 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 implementáciu 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 zaobrali bolo vytvorenie chatu medzi používateľmi, ktorých máme v kontaktoch. Poslednou časťou bolo vyladenie potencionálnych problémov a celkový dizajn aplikácie nakoľko doteraz sme implementovali hlavné 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 offline režime. Táto časť aplikácie bola ale implemetovaná tak, aby ako jediná vyžadovala 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 prípája na online databázu. Na Obr. 32 môžeme vidieť implementáciu funkcie na registráciu používateľa. Po odoslaní vyžadovaných dát používateľom z regisitračnej stránky sa z requestu - req vytiahnu dátá email, meno a heslo. Prvé, čo sa vo funkciu 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 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 prípadnej knižnice do Node.js bcrypt. Následne sa už len vytvoria potrebné dodatočné záznamy, ktoré JSON dokument používateľa potrebuje - polia tímov, udalostí, kontaktov, úloh a prázdný popis používateľa a tento údaj sa pomocou prípadnej 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. 32: Funkcia register na strane back-endu

Na Obr. 33 je kód funkcie na prihlásenie. Podobne, ako pri regiszračnej funkcií, 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 zadané 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. 33: Funkcia login na strane back-endu

Ak je všetko v poriadku, vytvorí sa pomocou funkcie `createToken jsonwebtoken`. Kód tejto funkcie môžeme vidieť na Obr. 34. 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 tokenu 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. 34: 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 adries 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. 35.

```

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. 35: Prihlásenie a registrácia na strane front-endu

Na strane font-endu sme implementovali ešte niekoľko ďalších potrebných funkcií. Samozrejmá je funkcia na odhlásenie používateľa, kde sa len zmažal jwt token. Tiež dôležitou funkciou bola funkcia na overenie platnosti tokena. Pri prihlásení sa token uložil do lokálnej pamäte. Pri každom načítaní, niektoľ z ďalej vytvorených stránok sa volá funkcia checkToken (Obr. 36). Tu sa pomocou vyššie spomenutej knižnice na prácu s jwt tokenom overí platnosť tokena. Ak nie je platný, tak sa nastaví 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. 36: 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. Hned po načítaní tejto stránky sa online databáza CouchDB zreplikuje do lokálnej databázy PouchDB a nastaví sa takzvané počúvanie databázy na live zmeny, čo umožní v reálnom čase meniť zobrazené dátá hneď po ich editácii, pridaní alebo zmazaní z databázy. Toto nastavenie sa realizuje pomocou vstavanej funkcie knižnice pouchdb changes() (Obr. 37).

```
· this.db.changes({
·   since: 'now',
·   live: true,
·   include_docs: true
· }).on('change', async (change) => {
·   if (change.deleted) {
·     this.load();
·   } else {
·     this.load();
·   }
· }).on('error', function (err) {
· });
·});
```

Obr. 37: Nastavenie na live počúvanie zmien databázy

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 dátá 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 premenných, do ktorých ukladáme dátá z DB a z ich getterov a setterov. Toto načítanie dát z databázy a uloženie do sharreddata je na Obr. 38.

```

constructor(private authService: AuthService, private router: Router, private _shareddata: SharedDataService) {
  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.logedemail = res;
    this.load();
  });
  this.router.events.subscribe((event: RouterEvent) => {
    this.selectedPath = event.url;
  })
}

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

```

Obr. 38: Načítanie dát prihláseného používateľa

Zobrazenie dát na stránke sme následne realizovali podľa potreby. Stránka profilu len zobrala dopredu vytiahnuté dáta z DB o používateľovi a zobrazila ich. Na zobrazenie tímov, ktorých členom je prihlásený používateľ sme použili tabuľkové rozloženie a funkciu Angularu ngFor. Táto funkcia prechádza polom tímov a vykresluje dáta každého prvku v poli podľa predlohy. Podobným princípom sme vytvorili aj zobrazenie kontaktov prihláseného používateľa. Ukážka kódu tohto zobrazenia je na Obr. 39

```

<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. 39: Zobrazenie používateľových tímov

Na vykreslenie kalendára s udalosťami sme použili príavný modul Ionicu ionic2-calendar. Tento modul nám umožnil použiť html element calendar (Obr. 40), 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 dátá z databázy upravovať. Jediný údaj potrebný upraviť bol dátum a čas, nakoľko ten sa do databázy ukladal 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. 40: Použitie html elementu calendar

6.3 Stránka vybraného tímu

Ked' používateľ na stránke po prihlásení otvoril stránku s jeho tímami, má možnosť na niektorý z jeho tímov kliknúť. Po kliknutí sa mu otvorí stránka tímu, ktorá pozostáva z viacerých možností. Základom stránky je postranné menu, kde sú zobrazené vytvorené miestnosti v teame a vytvorené úlohy v teame. Taktiež tu je možnosť prezrieť si kalendár tímu a stránku s členmi tímu, kde sú zobrazené profily členov tímu a je tu taktiež možnosť pridania používateľa do kontaktov. Administrátor, respektíve vedúci tímu má navyše možnosť pridávania členov do tímu a vytváranie prípadne odstránenie udalostí v tímovom kalendári. Vytváranie miestností a úloh je umožnené všetkým členom tímu.

Implementáciu tejto časti sme začali implementovaním funkcie pre administrátora tímu na pridanie členov do tímu a následne ich zobrazenie na stránke členov. Pridávanie členov sme zrealizovali pomocou vyskakovacích alert okien, kde administrátor zadá email používateľa, ktorého chce pridať. Kód implementácie vyskakovacieho okna môžeme vidieť na Obr. 41. Následne sa tento email použije na vyhľadanie v databáze. Ak je používateľ s tímto emailom nájdený, je pridaný do tímu a administrátor je o tom informovaný vyskakovacím oknom. Ak sa tento email nenašiel so žiadnym používateľom, opäť vyskočí informácia o tom, že daný používateľ neexistuje. Samotnú stránku členov tímu sme implementovali rovnako ako stránku kontaktov pri úvodnej stránke po prihlásení.

```

async openAddMember() {
  let prompt = await this.alertCtrl.create({
    header: "Add Team Member",
    inputs: [
      {
        name: 'email',
        placeholder: 'User email'
      },
    ],
    buttons: [
      {
        text: 'Cancel',
        handler: data => {}
      },
      {
        text: 'Add',
        handler: data => {
          this.adduser(data);
        }
      }
    ]
  });
  await prompt.present();
}

```

Obr. 41: Implementácia vyskakovacieho okna na pridanie člena tímu

Ďalšou časťou, ktorú sme implementovali bol kalendár a pridávanie udalostí. Kalendár sme implementovali rovnako ako pri kalendári používateľa. Jediným rozdielom bolo načítanie dát. Tu sme načítavali udalosti len tímu, v ktorom sa nachádzame. Pridávanie udalostí sme implementovali pomocou rozbalovacieho okna, kde po kliknutí na tlačidlo pridanie udalosti sa nám rozbalí formulár, do ktorého treba zadať názov udalosti, jej popis, čas od kedy do kedy bude udalosť prebiehať, prípadne vybrať možnosť udalosť na celý deň. V tomto prípade sa do času začiatku uloží vybraný deň a čas 7:00 ráno a do času konca sa uloží vybraný deň konca a čas 22:00.

V implementácii tímovej časti sme ďalej pokračovali vytvorením funkcionality miestností. Tu sme najkôr implementovali možnosť vytvorenia miestnosti. Toto sme realizovali podobne ako pridanie člena do tímu pomocou vyskakovacieho okna, kde sa zadal názov miestnosti. Následne sa vytvorená miestnosť zobrazila v bočnom menu. Po kliknutí na názov miestnosti v menu sa otvorí stránka miestnosti, na ktorej sa zobrazia pridané príspevky a možnosť pridať nový príspevok. Zobrazenie príspevkov sme realizovali pomocou vyššie spomínanej funkcie Angularu ngFor, ktorá pre každý príspevok vytiahnutý z databázy vytvorila zobrazenie dát príspevku podľa predlohy. Na spodku stránky sme implementovali pridanie príspevku. To pozostáva z jednoduchého elementu textarea, ktorý sa po kliknutí naň jemne zväčší a používateľ môže napísať správu príspevku. Po stla-

čení tlačidla enter sa potom príspevok vytvorí pod menom prihláseného používateľa a čas pridania sa nastaví na aktuálny čas. Takto vytvorený príspevok sa pošle do databázy. Pomocou na začiatku spomínanej funkcie changes() sa hned príspevok zobrazí všetkým online používateľom.

Poslednou funkcionalitou, ktorú sme v tejto časti implemenovali bolo vytváranie, zobrazenie a správa úloh. V postrannom menu sme implementovali tlačidlo na pridanie úlohy. Toto sme tiež implementovali pomocou vyskakovacieho okna, kde používateľ zadá názov a popis úlohy. Následne je úloha vytvorená a zobrazí sa v postrannom menu, kde je možné na ňu kliknúť. Po kliknutí sa zobrazí stránka úlohy, kde sú zobrazené jej údaje, komentáre od používateľov a možnosť pridať komentár. Údaje o úlohe zahŕňajú status úlohy - zadaná, pracuje sa na nej a spravená. Tento status je možné meniť po kliknutí naň, dátum zadania úlohy, dátum dokončenia úlohy (kým status nie je nastavený na spravená zobrazuje sa tu nápis prebieha), meno vedúceho úlohy a mená členov tímu, ktorí na úlohe pracujú. Vedúci úlohy má možnosť pridania člena tímu na spolupodieľanie sa na úlohe. Pridávanie komentárov sme implementovali rovnako ako pridávanie príspevkov v miestnosti.

6.4 Chat

Poslednou funkcionálnou časťou, ktorú sme implementovali bol chat medzi používateľmi, ktorí sa navzájom majú v kontaktoch. Používateľ po prihlásení ma v postrannom menu možnosť kliknúť na kontakty. Tu sa mu zobrazí stránka kontaktov, na ktoré má možnosť kliknúť. Následne sa zobrazí stránka chatu s používateľom, na ktorého bolo kliknuté. Samotný chat sa implementoval podobne, ako pridávanie príspevkov do miestností alebo komentárov k úlohám.

6.5 Dizajn aplikácie a odstránenie problémov

Dizajn aplikácie sme prispôsobovali tak, aby pripomínil podobné aplikácie a aby bolo ľahké si zvyknúť na používanie aplikácie novým používateľom. Všetky možnosti využitia aplikácie sú spravené prehľadne a používateelia intuitívne vedia ako aplikáciu využívať. Jednotlivé stránky aplikácie a ich dizajn môžeme vidieť v prílohe A.

Po dokončení dizajnu sme začali s hľadaním a odstraňovaním problémov, ktoré sa mohli vyskytnúť. Jedným z častých problémov bolo opakované zobrazovanie niektorých príspevkov, komentárov alebo udalostí. Toto bolo zapríčinené nevyprázdnením poľa dáných dát pri zaznamenaní zmeny v databáze. Keď nastala zmena v databáze, bolo nutné buď zmazať celé pole a jeho vytvorenie na novo alebo pridať do poľa len nové dátá, ktoré boli do databázy pridané. Riešili sme to prvým spôsobom, nakolko nám prišiel efektívnejší,

ako vždy prehľadávať pole a pridávať, prípadne odstraňovať z neho dátu podľa zmeny v databáze. Ďalším problémom, ktorý sa objavil, bolo miznutie postranného menu po zmenšení okna webového prehliadača alebo po exporte na desktopovú aplikáciu po zmenšovaní okna aplikácie. Z tohto dôvodu sme do každej stránky implementovali tlačidlo, ktoré sa zobrazí len vtedy, ak sa okno zmenší tak, že postranné menu zmizne. Pomocou tohto tlačidla sa postranné menu zobrazí o úroveň vyššie ako stránka, na ktorej sa nachádzame. Tým pádom je v tomto momente možné pracovať len s menu, ale po kliknutí mimo menu sa opäť vrátime na otvorenú stránku.

6.6 Export na desktopovú aplikáciu použitím Electronu

Posledným krokom implementácie bolo vyexportovanie nami naprogramovanej webovej aplikácie na desktopovú aplikáciu. Kedže sme celú aplikáciu vyvíjali pod operačným systémom Linux, tak sme aj export aplikácie robili pre tento operačný systém. Samozrejme je možné aplikáciu vyexportovať aj pre iné operačné systémy. Táto časť pozostávala najskôr s nainštalovaním knižnice Electronu do našej aplikácie použitím npm. Do príkazového riadku v našej aplikácii sme zadali príkaz `npm install electron --save-dev`, čo nám nainšalovalo najnovšiu verziu Electronu do našej aplikácie. Následne sme už mohli pracovať s exportovaním. Najskôr sme ale upravili potrebné parametre aplikácie - názov, autor, verzia. Následne sme už mohli začať s exportovaním. Prvým krokom bolo vytvorenie zdrojových súborov pre inštaláciu. Toto sme zrealizovali pomocou príkazu `electron-packager . --platform=linux`. Tu si môžeme všimnúť, že sme za platformu vybrali práve spomínaný Linux. Tu by sa dalo zmeniť pre aký systém chcem aplikáciu exportovať. Po tomto príkaze sme vytvorili inštalačný súbor pre systém Linux pomocou príkazu `electron-installer-debian --src Co-lab-linux-x64/ --arch amd64 --config debian.json`. Tiež si môžeme všimnúť, že niektoré dodatočné parametre príkazu sa týkajú verzie pre aký Linux chceme inštalačný súbor. Pri parametri src si treba dať pozor na správnu cestu k vytvoreným zdrojovým súborom, ktoré sme vytvorili pri predchádzajúcim príkaze `electron-packager`. Následne nám už Electron vytvoril inštalačný súbor, ktorý sme už len spustili a aplikácia sa nainštalovala pod systémom Linux. Pri simultánnom používaní aplikácie cez web aj cez nainštalovanú aplikáciu sme otestovali, či je správanie a funkčnosť aplikácie rovnaká. Po tomto teste sme mohli skonštatovať, že aplikácia funguje rovnako aj cez webový prehliadač, aj ako desktopová aplikácia čo bol náš cieľ.

Záver

Podľa zadaných cieľov tejto práce sme zanalyzovali teraz dostupné a populárne co-working aplikácie. Na základe tejto analýzy sme zistili nedostatky a problémy týchto aplikácií a prišli s návrhom vlastnej aplikácie. Následne sme spravili prieskum aktuálnych trendov a technológií používaných pri tvorbe web aplikácií a vybrali sme technológie, ktoré sme neskôr použili na implementovanie našej aplikácie. Ďalším krokom, ktorý sme splnili bolo navrhnuť celú aplikáciu s prihliadnutím na technológie, ktoré sme sa rozhodli použiť. Následne sme podľa návrhu aplikáciu implementovali ako webovú aplikáciu. Podľa ciela, ktorý sme si stanovili, aby aplikácia bola implementovaná formou desktopovej aplikácie, sme webovú aplikáciu exportovali na desktopovú. Po tomto kroku sme aplikáciu podrobili testu na fukcionalitu, či aplikácia splňa funkciu funkciu, ktorú sme jej stanovili. Z výsledku týchto testov môžeme skonštatovať, že aplikácia splňa nami stanovené kritériá.

Aplikácia umožňuje spolupracovanie a komunikáciu skupín ludí pracujúcich na dosiahnutí nimi stanoveného ciela. Cez aplikáciu je možné komunikovať v uzavretej skupine ludí a manažovať túto skupinu podľa potreby.

Cieľom tejto aplikácie je zjednodušiť komunikáciu a spoluprácu tímov pracujúcich na projektoch, či už v rámci školy alebo aj mimo nej. Jej hlavným cieľom je mať pod jedným účtom možnosti, na ktoré v iných aplikáciách treba mať viacero účtov u rôznych poskytovateľov služieb a je nutné niektoré možnosti importovať ako doplnkové služby. Ďalším cieľom aplikácie je priniesť študentom bezplatnú aplikáciu na spolupracovanie a komunikáciu, nakoľko väčšina iných aplikácií si vyžaduje poplatky buď za celkové využívanie aplikácie alebo za využívanie jej časti, prípadne pridanie doplnkov.

Práca na aplikácii tu zdaleka nemusí končiť. Aplikácia bola navrhovaná aj implementovaná tak, aby sa ďalej dala rozširovať a dali sa do nej doplnať ďalšie funkcie a možnosti. Možnými rozšíreniami, ktoré je možné ľahko implementovať pri pokračovaní sú napríklad notifikácie v aplikácii, pridanie možnosti posielat emoticony, fotky, obrázky, možnosť implementovať prevzatie udalostí z kalendárov iných aplikácií, ako je napríklad Google kalendár a iné. Aplikácia môže veľmi dobre poslúžiť ako odrazový mostík iným študentom pri záujme pokračovať vo vývoji tejto aplikácie alebo pri tvorbe vlastnej, podobnej aplikácie.

Zoznam použitej literatúry

1. COWORKER. *What Is Coworking?* 2018. [online]. [cit: 2020-04-06]. Dostupné na internete: <<https://www.coworker.com/mag/what-is-coworking>>.
2. ROUSE, Margaret. *What Is Coworking?* [online]. [cit: 2020-04-06]. Dostupné na internete: <<https://whatis.techtarget.com/definition/coworking>>.
3. CAMPUS. *Čo je to coworking? Alebo 9 vyjadrení členov coworku.* 2019. [online]. [cit: 2020-04-06]. Dostupné na internete: <<https://www.campus-cowork.com/sk/blog/70/co-je-to-coworking-alebo-9-vyjadreni-clenov-coworku.html>>.
4. BISNOW. *Everything You Need To Know About Coworking.* 2014. [online]. [cit: 2020-04-07]. Dostupné na internete: <<https://www.bisnow.com/national/news/commercial-real-estate/Everything-You-Need-To-Know-About-Coworking-40306>>.
5. FOERTSCH, Carsten. *The birth of coworking spaces.* 2011. [online]. [cit: 2020-04-06]. Dostupné na internete: <<http://www.deskmag.com/en/the-birth-of-coworking-spaces-global-survey-176>>.
6. WOODGATE, Rob. *What Is Slack, and Why Do People Love It?* 2019. [online]. [cit: 2019-12-24]. Dostupné na internete: <<https://www.howtogeek.com/428046/what-is-slack-and-why-do-people-love-it/>>.
7. MENON, Abhilash. *Facebook Workplace: The GOOD and the BAD.* [online]. [cit: 2019-12-24]. Dostupné na internete: <<https://hiverhq.com/blog/facebook-workplace/>>.
8. SUMMERS, Nick. *Facebook's Workplace redesign looks nothing like Slack.* [online]. [cit: 2019-12-24]. Dostupné na internete: <<https://www.engadget.com/2019-08-07-facebook-workplace-redesign-interview.html>>.
9. NICK HEATH, Susan Harkins. *Microsoft Teams: A cheat sheet.* 2016. [online]. [cit: 2019-12-24]. Dostupné na internete: <<https://www.techrepublic.com/article/microsoft-teams-the-smart-persons-guide/>>.
10. BEDRICH, Vaclav. *Microsoft Teams míří na pozici jednicky mezi firemními komunikacními nástroji. V počtu firem překonal Slack.* 2018. [online]. [cit: 2019-12-24]. Dostupné na internete: <<https://www.czechcrunch.cz/2018/12/microsoft-teams-miri-na-pozici-jednicky-mezi-firemnimi-komunikacnimi-nastroji-v-poctu-uzivatelu-prekonal-slack/>>.
11. DRAKE, Nate. *Trello review.* 2018. [online]. [cit: 2020-01-03]. Dostupné na internete: <<https://www.techradar.com/reviews/trello/>>.

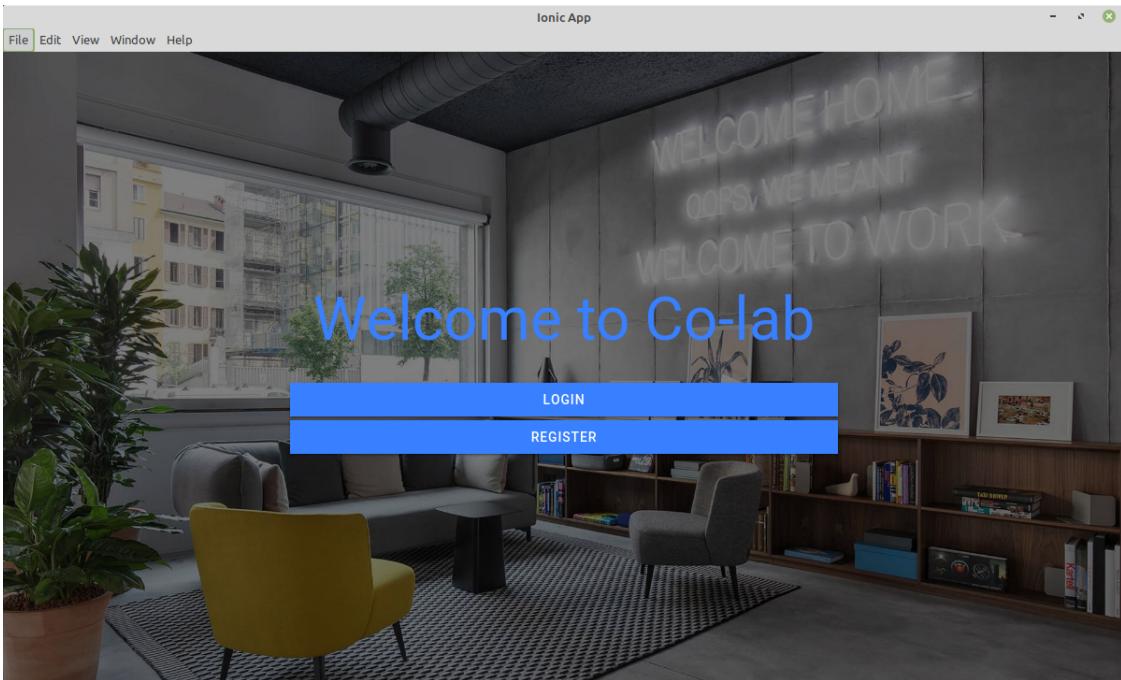
12. REHKOPF, Max. *What is a kanban board?* [online]. [cit: 2020-01-03]. Dostupné na internete: <<https://www.atlassian.com/agile/kanban/boards>>.
13. WIRED. *How to use Trello like a pro.* 2019. [online]. [cit: 2020-01-03]. Dostupné na internete: <<https://www.wired.co.uk/article/how-to-use-trello>>.
14. NODEJS. *Node.js.* [online]. [cit: 2020-04-09]. Dostupné na internete: <<https://nodejs.org/>>.
15. BIBILE, Udara. *NodeJS Architecture and Concurrency Model.* 2020. [online]. [cit: 2020-04-11]. Dostupné na internete: <<https://medium.com/chathuranga94/nodejs-architecture-concurrency-model-f71da5f53d1d>>.
16. SKILLS, V. *Node.JS Architecture.* [online]. [cit: 2020-04-11]. Dostupné na internete: <<https://www.vskills.in/certification/tutorial/web-development/node-js>>.
17. NODEJS. *The Node.js Event Loop, Timers, and process.nextTick().* [online]. [cit: 2020-04-09]. Dostupné na internete: <<https://nodejs.org/uk/docs/guides/event-loop-timers-and-nexttick>>.
18. W3SCHOOLS. *What is npm?* [online]. [cit: 2020-04-11]. Dostupné na internete: <<https://www.w3schools.com/whatis/whatisnpm.asp>>.
19. W3SCHOOLS. *Node.js and NPM.* [online]. [cit: 2020-04-11]. Dostupné na internete: <<https://www.w3schools.com/nodejs/nodejsnpm.asp>>.
20. LYNCH, Max. *Cordova vs Capacitor.* [online]. [cit: 2020-04-12]. Dostupné na internete: <<https://ionicframework.com/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development>>.
21. KRIST, Karol. *Bakalárska práca - Manažérsky web front-end pre mobilnú aplikáciu.* FEI STU, 2018. ISBN FEI-5382-79995.
22. ALTEXSOFT. *The Good and the Bad of Angular Development.* 2020. [online]. [cit: 2020-04-15]. Dostupné na internete: <<https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development>>.
23. FAIN, Yakov. *Angular 2 and TypeScript - A High Level Overview.* 2016. [online]. [cit: 2020-04-15]. Dostupné na internete: <<https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview>>.
24. GOOGLE. *Introduction to Angular concepts.* [online]. [cit: 2020-04-15]. Dostupné na internete: <<https://angular.io/guide/architecture>>.
25. ROBERTS, Tyler. *Getting Started with Angular and Electron.* [online]. [cit: 2020-04-12]. Dostupné na internete: <<https://alligator.io/angular/electron>>.

26. EDUCATION, IBM Cloud. *Apache CouchDB*. 2019. [online]. [cit: 2020-04-13]. Dostupné na internete: <<https://www.ibm.com/cloud/learn/couchdb>>.
27. BODNAR, Jan. *PouchDB tutorial*. 2019. [online]. [cit: 2020-04-13]. Dostupné na internete: <<http://zetcode.com/javascript/pouchdb/>>.
28. 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.
29. 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.
30. CODECONDO. *Top 8 Challenges in Web Application Development in 2018*. 2018. [online]. [cit: 2020-04-07]. Dostupné na internete: <<https://codecondo.com/top-8-challenges-in-web-application-development-in-2018>>.
31. LITTLE, Chantelle. *7 challenges in web application design and development*. 2019. [online]. [cit: 2020-04-07]. Dostupné na internete: <<https://tillerdigital.com/blog/7-challenges-in-web-application-development>>.
32. MARUTITECHLABS. *5 Challenges in Web Application Development*. [online]. [cit: 2020-04-07]. Dostupné na internete: <<https://tillerdigital.com/blog/7-challenges-in-web-application-development>>.

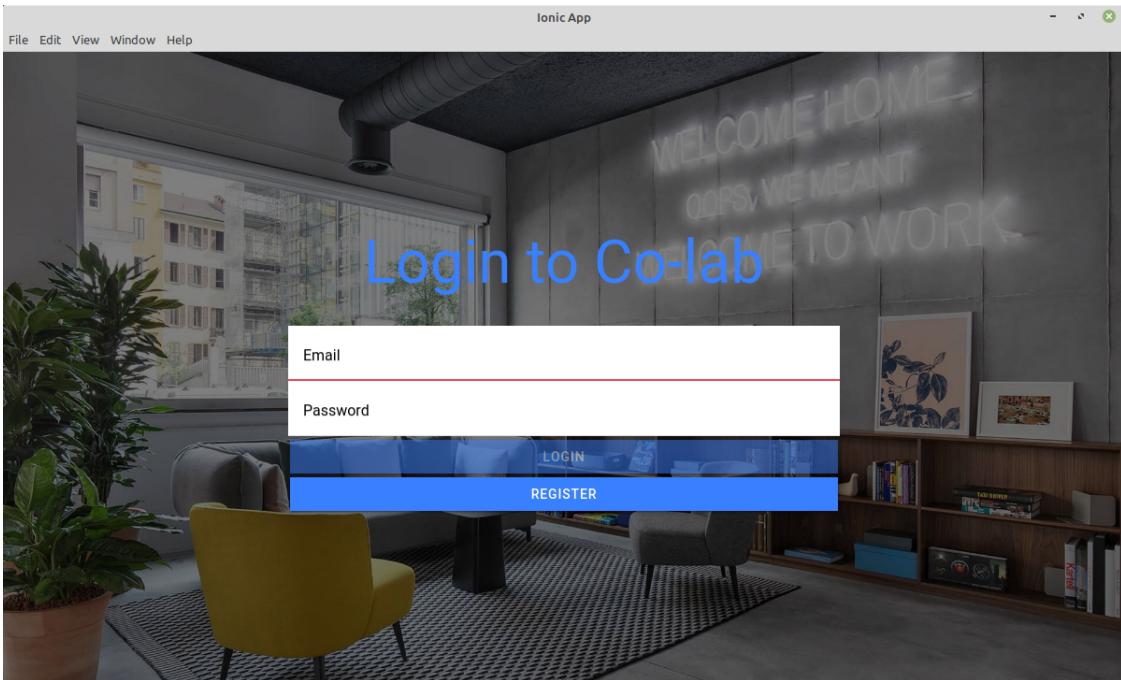
Prílohy

A Náhľad dizajnu Co-lab aplikácie	II
---	----

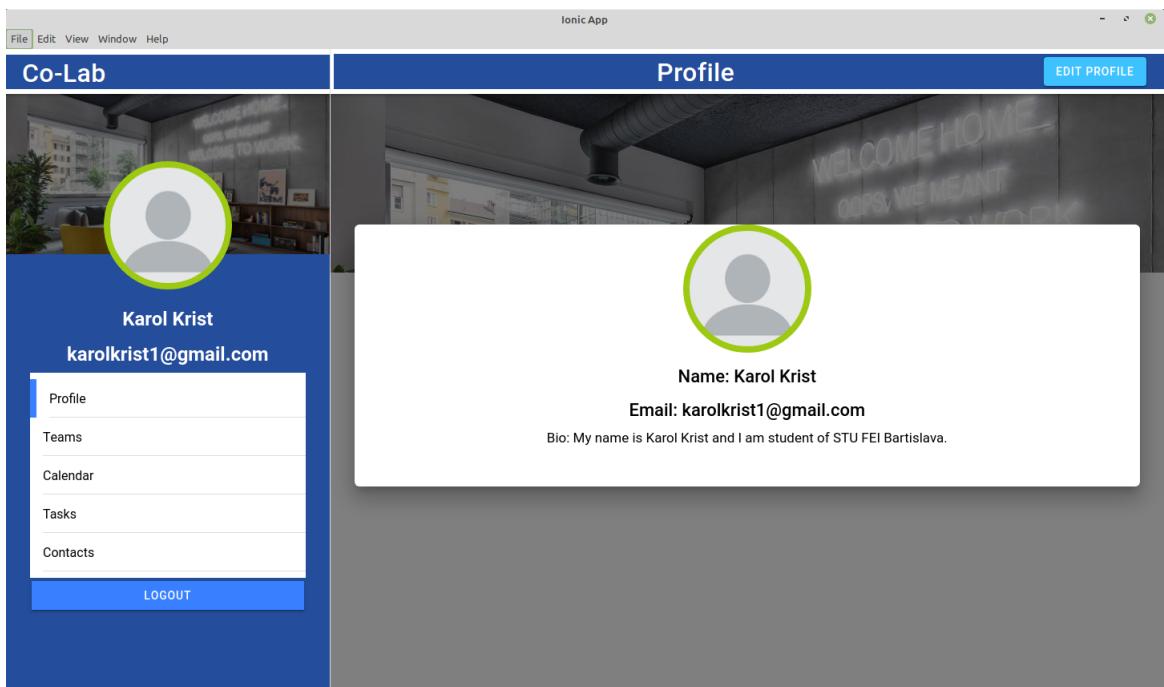
A Náhľad dizajnu Co-lab aplikácie



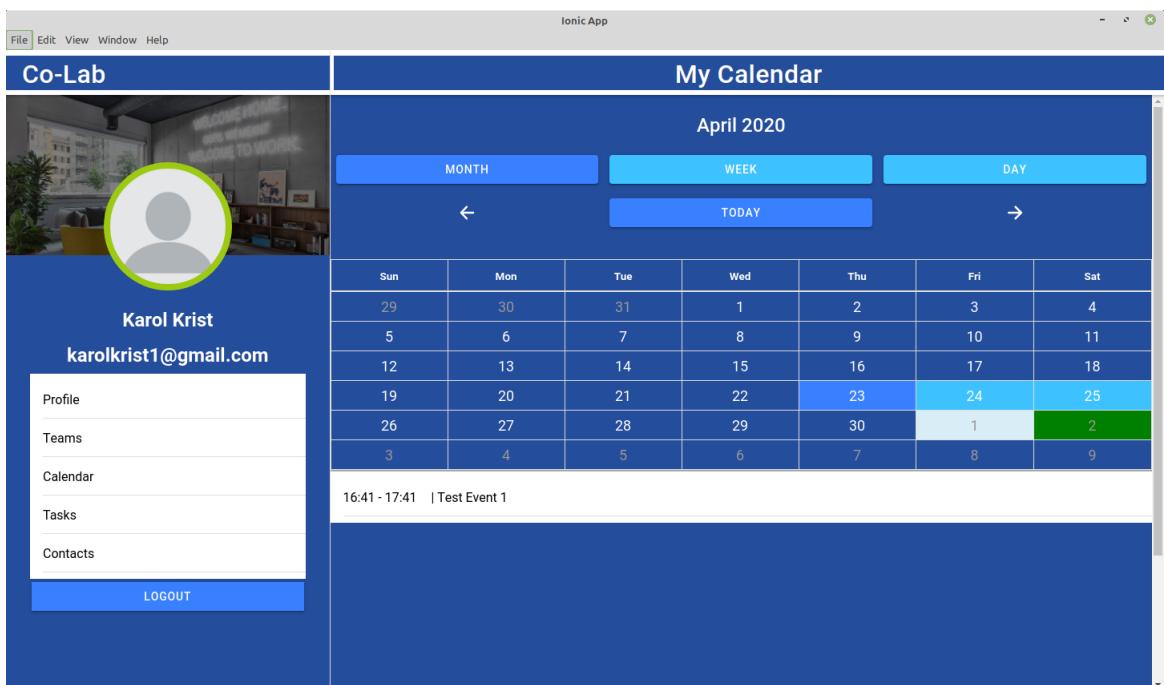
Obr. A.1: Úvodná stránka aplikácie



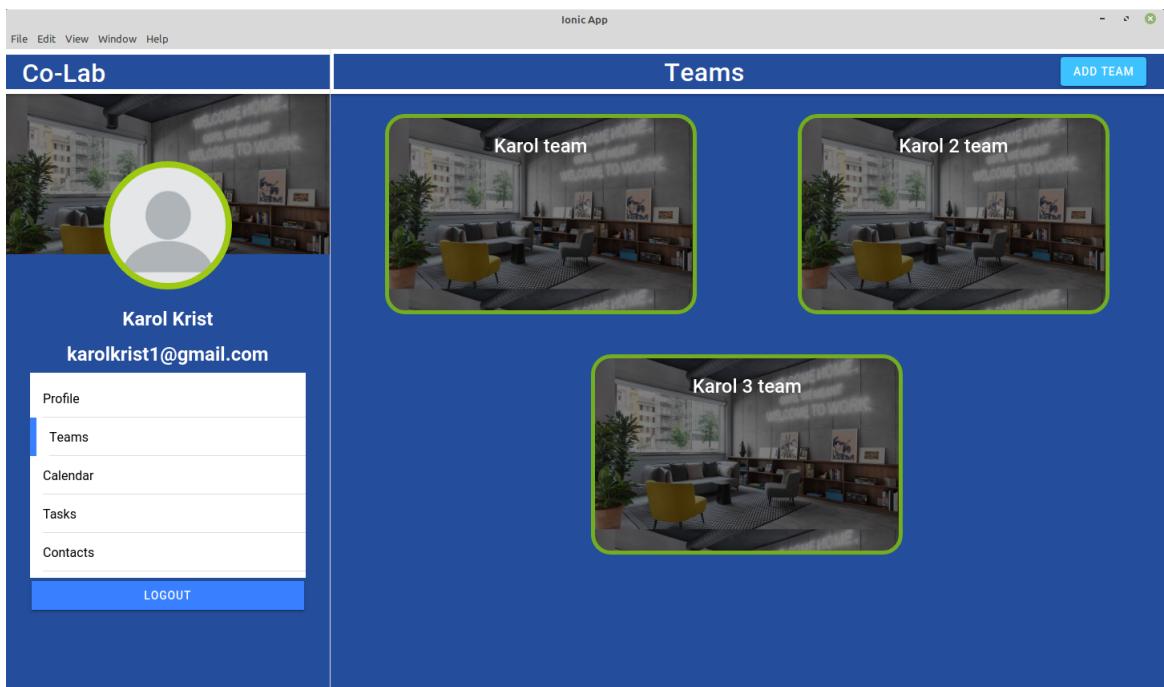
Obr. A.2: Prihlásovacia stránka aplikácie



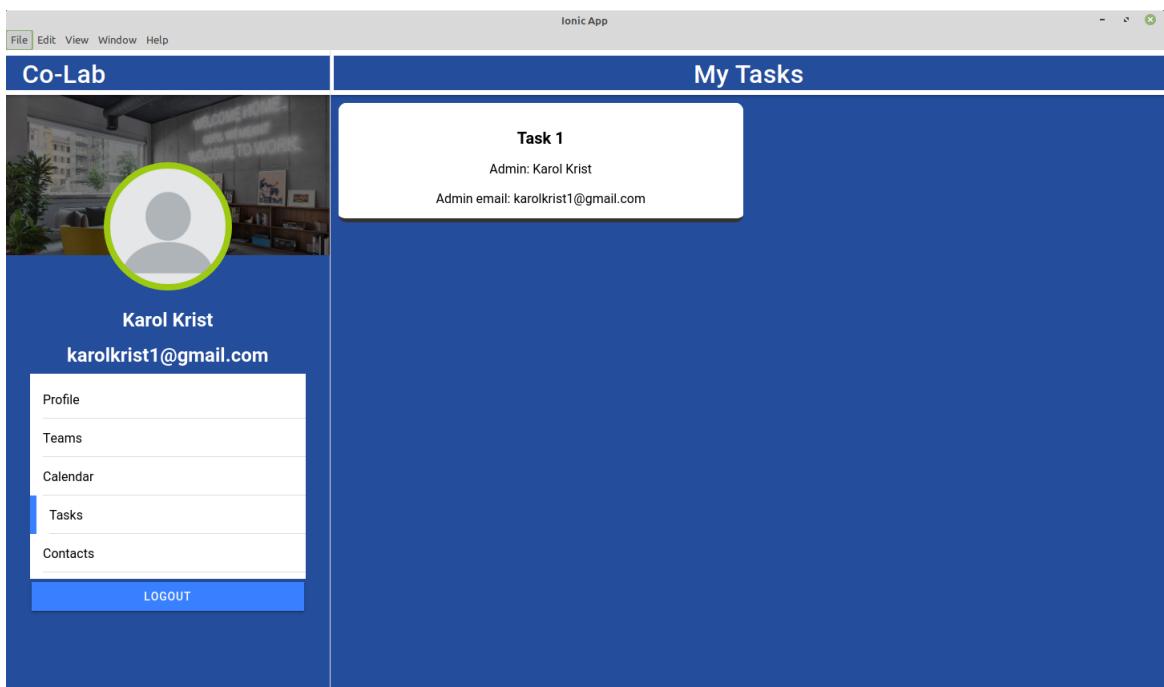
Obr. A.3: Profil používateľa



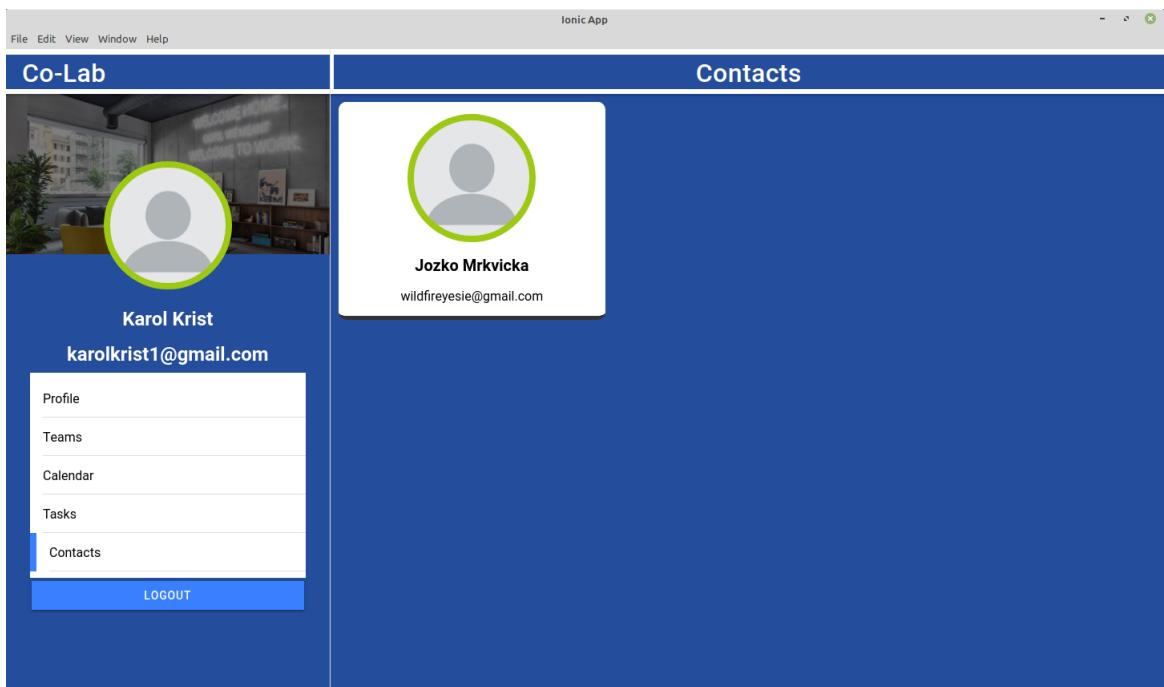
Obr. A.4: Kalendár používateľa



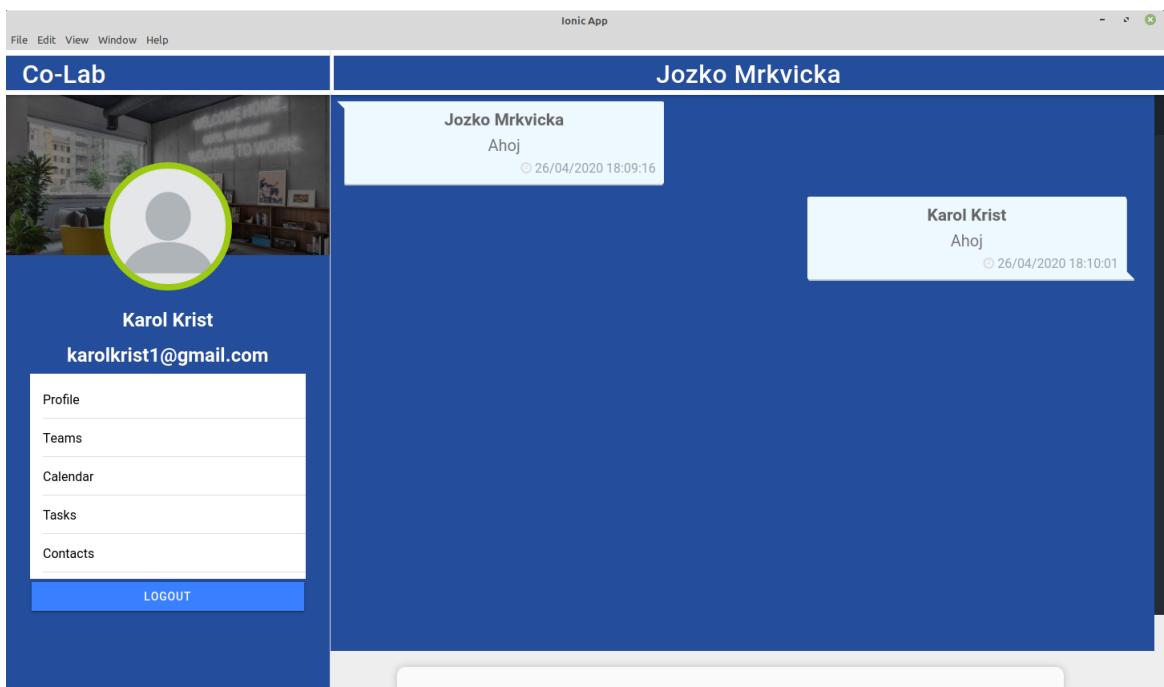
Obr. A.5: Tímy používateľa



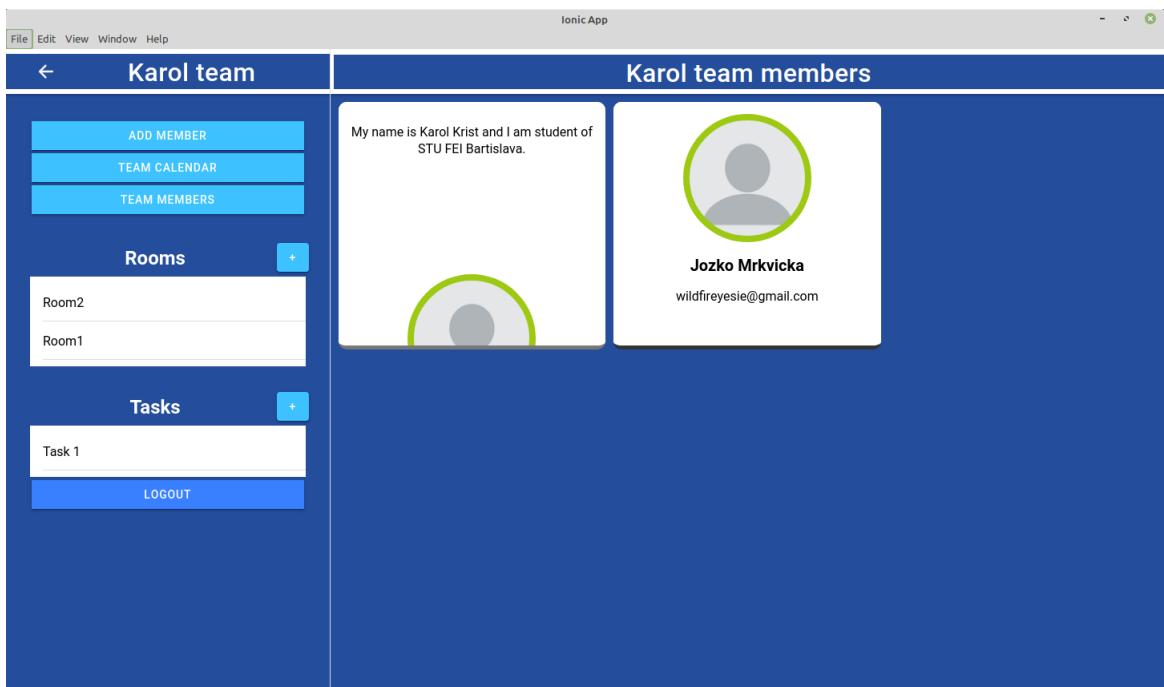
Obr. A.6: Úlohy používateľa



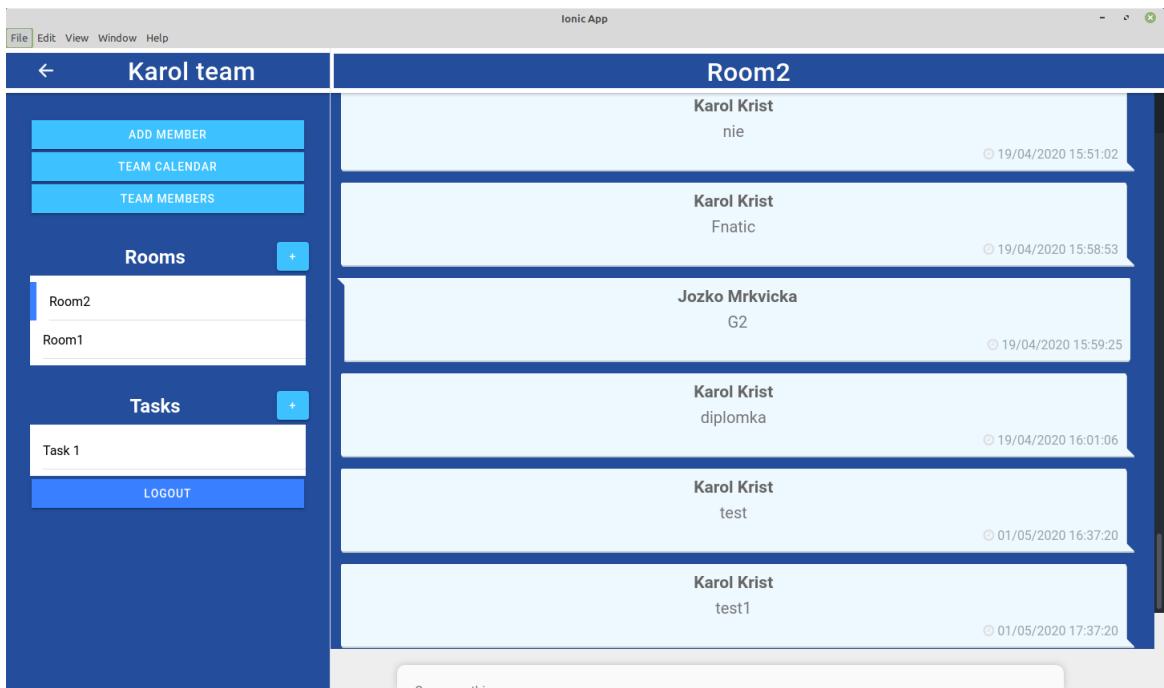
Obr. A.7: Kontakty používateľa



Obr. A.8: Chat s iným používateľom



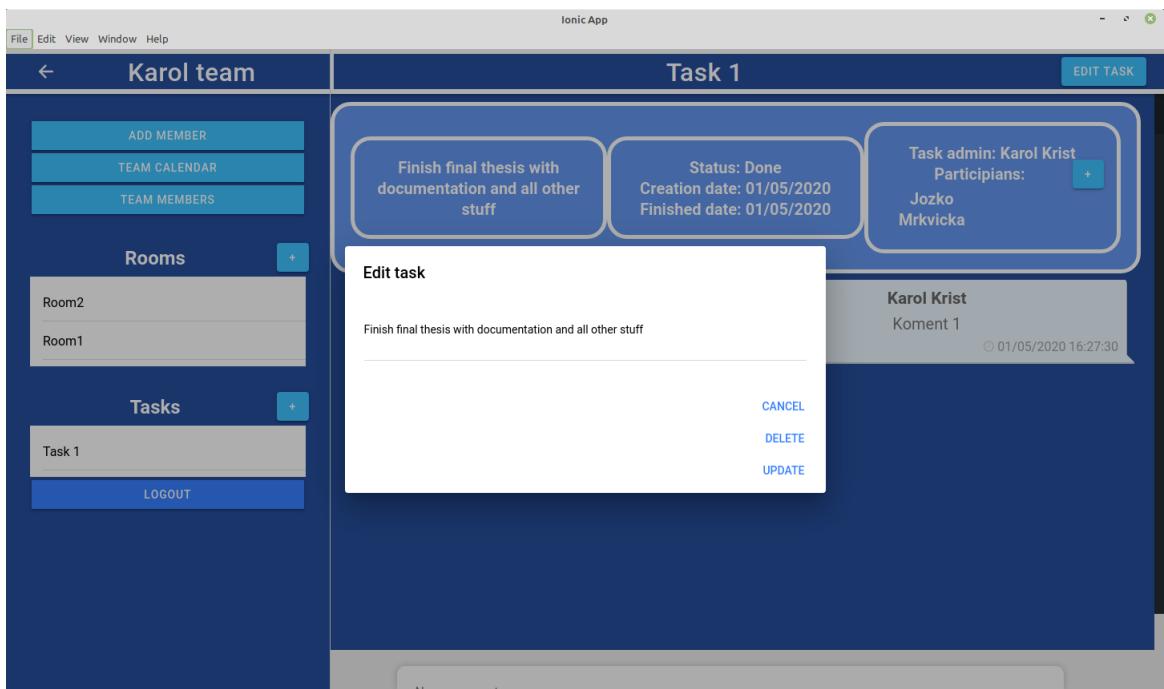
Obr. A.9: Stránka členov tímu



Obr. A.10: Miestnosť v tíme

Obr. A.11: Kalendár tímu

Obr. A.12: Stránka úlohy



Obr. A.13: Editovanie úlohy adminom