VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TRELLO MULTIBOARD

TRELLO MULTIBOARD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB ŠTEFANIŠIN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Štefanišin Jakub**
Program: Informační technologie
Název: **Trello multiboard**
Trello Multiboard
Kategorie: Softwarové inženýrství

Zadání:

1. Seznamte se s API k Trello.com, nastudujte teorii ke kanban filosofii řízení projektů.
2. Navrhněte službu, která umožní uživateli agregovat lístečky z několika Trello nástěnek do jedné. Proces by měl fungovat obousměrně (tedy změny v agregované nástěnce se promítnou zpět do původních).
3. Implementujte a otestujte navrženou službu.
4. Na základě zpětné vazby, navrhněte a implementujte úpravy a vylepšení.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího vývoje.
6. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část bodu 3 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cieľom tejto práce je vytvoriť službu, ktorá umožňuje prácu s viacerými Trello nástenkami na jednom mieste. Dosiahnutie požavoného cieľa nám umožňuje využitie Trello API. Služba je implementovaná využitím moderných technológií pre tvorbu webových aplikácií. Ponúka jednoduché prihlásenie a obojsmernú synchronizáciu zmien medzi našou a Trello aplikáciou. Aplikáciu je možné jednoducho rozširovať o novú funkcionalitu.

Abstract

The goal of this thesis is to create a service that allows users to work with multiple Trello boards in one place. Trello API is used to achieve desired result. The service is implemented using modern technologies for creating web applications. It offers easy login and two-way synchronization of changes between our and Trello application. The application can be easily extended with new functionality.

Kľúčové slová

Kanban, Trello, API, webová aplikácia, React, Redux, JavaScript

Keywords

Kanban, Trello, API, web application, React, Redux, JavaScript

Citácia

ŠTEFANIŠIN, Jakub. *Trello multiboard*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Trello multiboard

Prehlásenie

Prehlasuje, že som tuto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Igora Szőkeho, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jakub Štefanišin
28. mája 2020

Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce, Ing. Igorovi Szőke, Ph.D., ktorý ma odborne viedol.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je vytvoriť službu poskytujúcu užívateľom možnosť agregovať karty z viacerých Trello násteniek do jednej. Služba umožňuje okrem zobrazenia agregovanej nástenky a možnosť vykonávať nad objektami nástenky zmeny, ktoré budú prenášané do aplikácie Trello a rovnako zmeny vykonané v aplikácii Trello budú prenášané späť využitím Trello API. Služba bude implementovaná vo forme webovej aplikácie s architektúrou klient-server. Klient bude poskytovať grafické užívateľské rozhranie vo forme React aplikácie, bude priamo komunikovať so službou Trello využitím Trello API. Server bude spracováva prijaté informácie o zmenách prijatých z aplikácie Trello využitím funkcionality nazývanej Trello Webhook a zasielať tieto dáta klientovi. Služba bude ponúkať jednoduché prihlásenie použitím už existujúceho účtu zo služby Trello.

Aplikácia vzniká z potreby jednoduchej správy kariet viacerých Trello násteniek na jedno mieste. Pri hľadaní riešenia požadovaného problému došlo k zisteniu, že na trhu neexistuje produkčne nasadená služba, ktorá by vyhovovala požadovaným vlastnostiam.

Prvá kapitola práce je venovaná Kanban metodológii, je spomenutá jej história, princípy a praktiky. Druhá kapitola je venovaná službe Trello, ktorá implementuje Kanban metodológiu, na tejto službe stavia aj praktická časť tejto práce využitím Trello API, ktorého popis je nachádza v rovnakej kapitole. Tretia kapitola je venovaná návrhu riešenia. Popisuje hlavne cieľ práce a existujúce riešenia. Vo štvrtej kapitole sú popísané všetky dôležité technológie použité pri implementácii výslednej služby. Piatka kapitola sa venuje samotnej implementácii výslednej služby a testovaniu. Záver práce je venovaný zhrnutiu aktuálnemu stavu aplikácie a jej ďalší možný smer vývoja.

Kapitola 2

Kanban metodológia

Kanban (slovo pochádza z japončiny, v preklade karta, štítok alebo ceduľa) je vizuálny systém na riadenie práce. Vizualizuje proces (pracovný postup) a skutočnú prácu, ktorá daným procesom prechádza. Jedným z cieľov tejto metodológie je identifikovať potenciálne prekážky, či problémy v procese a opraviť ich. [1]

2.1 História Kanbanu

Začiatok sa datuje do neskorých 40. rokoch minulého storočia. Prvý Kanban systém vyvinul Taiichi Ohno pre automobilku Toyota v Japonsku. Bol vytvorený ako jednoduchý plánovací systém, ktorého cieľom bolo optimálne a efektívne kontrolovať a riadiť prácu a zásoby v každej fáze výroby.

Hlavným dôvodom rozvoja Kanbanu bola nedostatočná efektivita a produktivita spoločnosti Toyota v porovnaní s jej americkými automobilovými súpermi. V spojení s Kanban metodológiou, dosiahla Toyota flexibilný a efektívny systém riadenia výroby **JIT (Just-in-time)**[1], ktorý zvýšil produktivitu a zároveň znížil vysoko nákladové zásoby surovín, polotovarov a hotových výrobkov.

Systém Kanban v ideálnom prípade riadi celý hodnotový reťazec od dodávateľa až po konečného spotrebiteľa. Pomáha predchádzať prerušeniu dodávok a nadmernému zásobovaniu tovarom v jednotlivých fázach výrobného procesu. Vyžaduje nepretržité monitorovanie procesu. Pozornosť sa musí venovať zabráneniu prekážok, ktoré by mohli spomaliť výrobný proces. Cieľom je dosiahnuť vyššiu priechodnosť pri kratších dodacích lehotách. Kanban sa stal efektívnym spôsobom v rôznych výrobných systémoch. [1]

2.2 Kanban metóda v IT

Zatiaľ čo bol Kanban predstavený firmou Toyota vo výrobnom priemysle, David J. Anderson bol prvý, kto aplikoval tento koncept na IT a vývoj softvéru v roku 2004. David J. Anderson staval na dielach, ktoré vytvorili Taiichi Ohno, Eli Goldratt, Edward Demmings, Peter Drucker a iní, aby definoval **Kanban metódu**, ktorá uplatňuje koncepty ako **ťahový systém (pull system)**, **teória hromadnej obsluhy (queuing theory)** a **tok (flow)**.

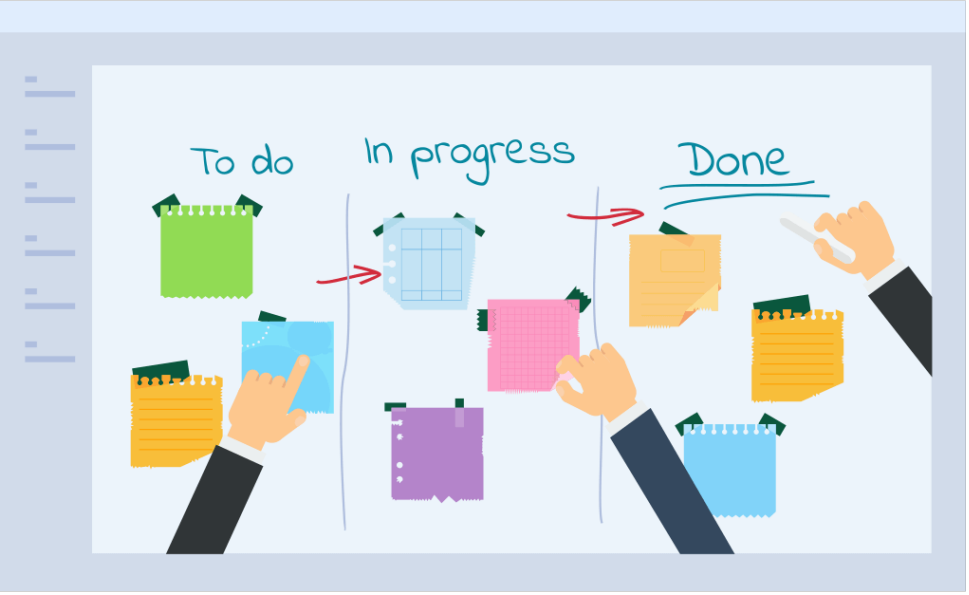
[1] JIT (Just-in-time) - neprekladá sa, prístup k výrobe, ktorý umožňuje podniku vyrábať výrobky v určenom množstve a čase podľa požiadavkov klienta

Kanban nebol vytvorený ako metodológia na vývoj softvéru, ani na správu projektu. Nehovorí, ako má byť softvér vyvíjaný. Namiesto toho, cieľom je neustále zlepšovanie vlastný pracovného procesu.

Táto metóda sa uplatnila ako spôsob, ktorým sa implementujú Lean a Agile princípy. Ponúka princípy pre vizualizáciu práce, dodanie produktov a služieb, a získavanie spätnej väzby častejšie. [1]

2.3 Kanban nástenka

Nástroj použitý na vizualizáciu práce a optimalizáciu toku práce v tíme. Bez rozdielu, či je nástenka tímu fyzická alebo digitálna, jej úlohou je zabezpečiť, že práca je vizualizovaná, pracovný tok (workflow) je štandardizovaný a všetky problémy a závislosti sú ihneď identifikované a vyriešené. Základná Kanban nástenka obsahuje 3-krokový pracovný tok: **To Do**, **In Progress** a **Done**. Závisiac od veľkosti tímu, jeho štruktúry a cieľov, môže byť upravená, aby spĺňala špecifické požiadavky daného tímu.



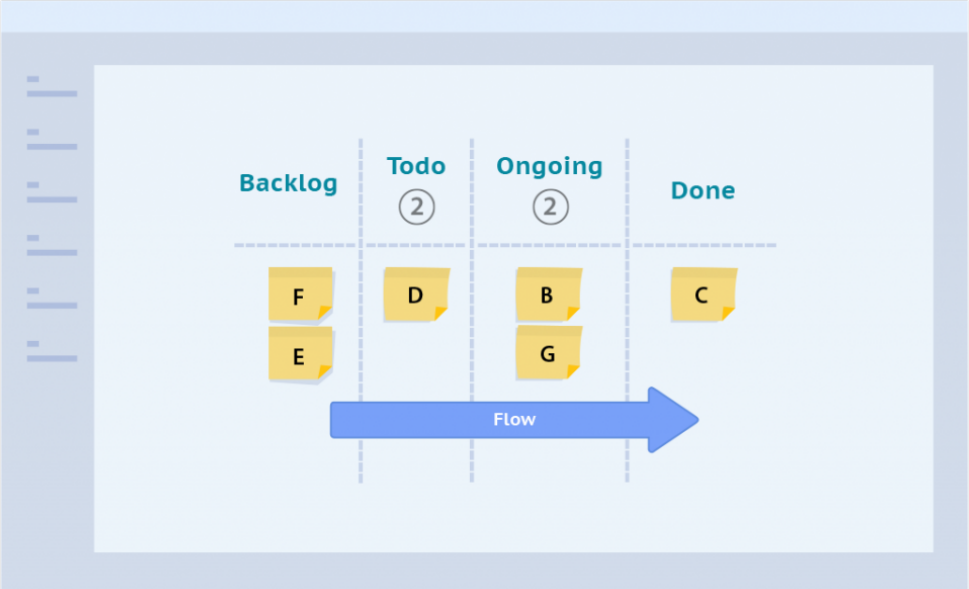
Obr. 2.1: Kanban nástenka, prevzaté z[2]

Kanban spolieha na úplnú transparentnosť práce a komunikáciu v reálnom čase, preto by mala byť Kanban nástenka videná ako jediný zdroj pre prácu tímu. [10]

[2] https://www.digite.com/kanban/what-is-kanban/

2.3.1 Princíp toku (Flow)

Jadrom Kanbanu je princíp nazývaný **„tok“ (Flow)**. Znamená to, že karty (štítky) by mali pretekať systémom čo najrovnomernejšie, bez dlhých čakacích dôb alebo problémov. Všetko, čo brzdí tok, by malo byť hneď preskúmané. Kanban má rôzne techniky, metriky a modely, ktoré ak sú uplatňované, mali by viesť ku kultúre neustáleho zlepšovania (Kaizen).



Obr. 2.2: Princíp Flow, prevzaté z[3]

Koncept toku je veľmi dôležitý, meraním metrík a ich vylepšovaním je možné dramaticky zlepšiť rýchlosť procesov, kým sa redukuje čas cyklu a zlepšuje kvalita produktov a služieb získavaním spätnej väzby od zákazníka. [1]

Aby sa mohol princíp Flow uplatniť, boli identifikované 4 prerekvizity: [9]

- Aktivita musí mať dobre definované ciele.
- Priebeh činnosti musí byť dobre definovaný.
- Aktivita si vyžaduje jednoznačnú a okamžitú spätnú väzbu.
- Úroveň schopností človeka musí byť v rovnováhe s obtiažnosťou danej aktivity.

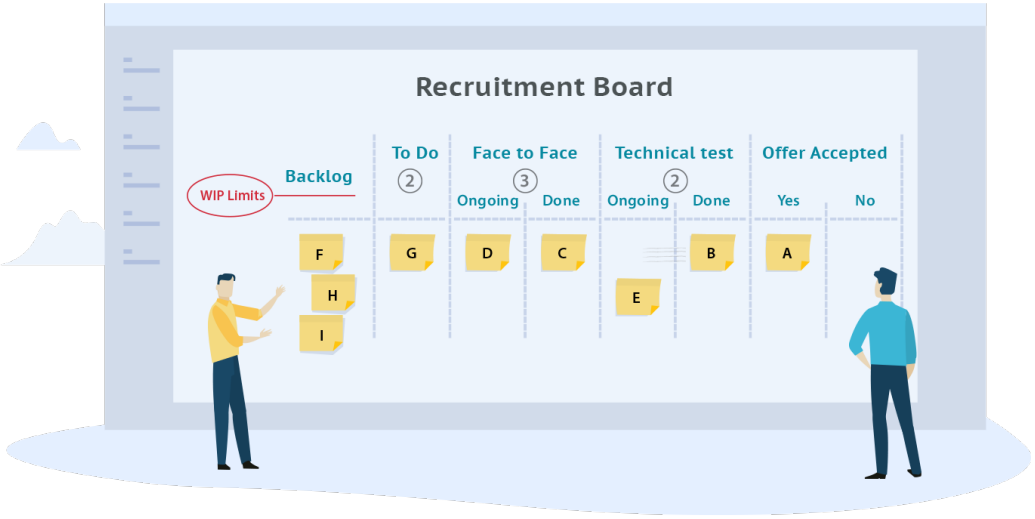
Kanban poskytuje možnosti a nástroje, aby boli naplnená väčšina z týchto prerekvizít: [9]

- Možnosť dobre definovať ciele.
- Sledovanie priebehu činnosti vďaka kanban nástenke.
- Kanban síce priamo nerieši spätnú väzbu alebo rovnováhu schopností človeka a obtiažnosti danej aktivity, no poskytuje priestor a nástroje, napríklad spravovaním veľkosti jednotlivých položiek alebo pomocou WIP limitu 2.3.2.

[3] https://www.digite.com/kanban/what-is-kanban/

2.3.2 WIP limit

WIP (Work-in-progress) limit definuje maximálny počet práce, ktorý môže existovať v každom stave toku. Týmto limitovaním je možné jednoduchšie identifikovať neefektívnosť v pracovnom postupe tímu. Redukuje počet „takmer hotovej“ práce tým, že je tím nútený zamerať sa na menšiu skupinu úloh. Dôležitejšie však je, že sa zvýrazňujú problémy a nedostatky. [1]



Obr. 2.3: Kanban nástenka s naznačenými WIP limitmi, prevzaté z[4]

2.4 Kanban karta

Kanban karta (štítok) je nevyhnutnou súčasťou Kanbanu. Každá karta reprezentuje jednu pracovnú položku, sleduje ako sa pohybuje cez jednotlivé etapy reprezentované na Kanban nástenke. Dovoľuje tímu: [10]

- **Vidieť dôležité detaily o pracovných položkách na prvý pohľad.** Každá karta typicky obsahuje stručný popis položky, jej majiteľa, stav a termín, kedy má byť hotová. Karta taktiež môže obsahovať informácie o dokumentácii, či zoznam položiek, ktoré môžu blokovať jej priebeh.
- **Odovzdanie výstupu hladko a efektívne.** Kanban povzbudzuje tímy, aby stanovili jasné a konzistentné očakávania pre každú oblasť. Keď príde čas odovzdania položky z jedného stavu do nasledujúceho, definované postupy určia kto prevezme vlastníctvo a aké sú ďalšie kroky.
- **Zlepšenie efektívnosti.** Karta uľahčuje sledovať potrebný na to, aby položka prešla od začiatku do konca. Kanban karty spolu s Kanban nástenkami uľahčujú tímom identifikovať problémy v ich pracovnom toku a zefektívniť ich proces.

[4] https://www.digite.com/kanban/what-is-kanban/

2.5 Princípy a praktiky

Kanban metóda nasleduje sadu princípov a praktík pre správu a vylepšenie toku práce. Medzi 4 základné princípy patria: [1]

- **Začni s tým, na čom pracuješ teraz.** Kanban kladie dôraz na nevykonanie zmeny v existujúcom procese hneď. Je potrebné ho aplikovať do aktuálneho pracovného postupu v priebehu časového obdobia, ktorý vyhovuje tímu.
- **Dohodnutie sa na postupných, evolučných zmenách.** Je lepšie vykonávať postupné malé zmeny.
- **Rešpektovanie súčasných úloh, zodpovedností a pracovných miest.** Nie je potrebné vykonávať žiadne organizačné zmeny. Tím má spoločne identifikovať a implementovať potrebné zmeny.
- **Podporovanie riadiacich schopností na každej úrovni.** Kanban podporuje neustále zlepšovanie na všetkých úrovniach v organizácii. Ľudia na všetkých pozíciách môžu poskytnúť nové nápady a riešenia, a tým neustále zlepšovať spôsob, akým sa dodávajú ich výrobky a služby.

Uplatňuje sa 6 hlavných praktík: [1]

- **Vizualizácia toku práce.** Použitie fyzickej, či digitálnej nástenky. Komplexnosť podľa potreby. Rôzne formy kariet, či štítkov rozličnej farby, ktoré rozlišujú typ položky.
- **WIP limit.** Limitovanie „takmer hotovej“ práce je základom implementácie tzv. ťahového systému. Povzbudzuje tím k dokončeniu rozpracovanej položky pred začatím práce na novej. Typicky tím začína s 1 až 1,5 násobkom počtu ľudí pracujúcich na danej fáze.
- **Riadenie toku.** V závislosti od dobre definovaného pracovného toku a nastavených limitov WIP je možné sledovať buď hladký tok v rámci WIP limitov alebo hromadenie práce. Všetko toto ovplyvňuje, ako rýchlo sa práca hýbe od začiatku do konca pracovného procesu. Kanban pomáha analyzovať tento systém a robiť úpravy na zlepšenie toku.
- **Vytvorenie jasných postupov.** Vytvorením postupov sa utvorí spoločný základ pre všetkých účastníkov, aby pochopili, ako vykonávať v systéme akúkoľvek prácu. Môže to byť napríklad zoznam krokov pre každý typ pracovnej položky.
- **Implementácia cyklu spätnej väzby.**
- **Spoločné zlepšovanie, experimentálne vyvíjanie.** Prijímanie malých zmien a postupné zlepšovanie tempom, ktoré vyhovuje tímu.

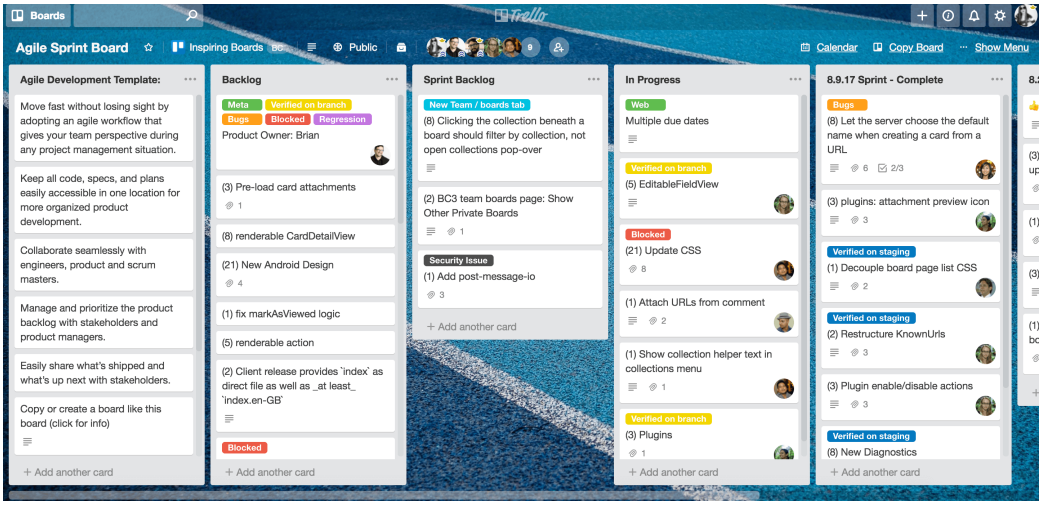
Kapitola 3

Trello

Trello je jedným z mnohých nástrojov, ktoré implementujú metodológiu Kanban. Jedná sa o jednoduchý, flexibilný a vizuálny nástroj na správu projektov, či personálnych agend.

Prvý prototyp tejto aplikácie bol vytvorený v roku 2011 spoločnosťou Fog Creek Software pod názvom Trellis, neskôr po uzavretom beta testovaní vydáva firma v septembri 2011 webovú aplikáciu a aplikáciu pre iPhone. V roku 2012 dosahuje aplikácia pol milióna používateľov a vydáva aplikáciu pre Android. V júli 2014 sa z Fog Creek Software stáva Trello, Inc. a má viac než 4,76 milióna používateľov. Trello sa stáva celosvetovým v máji 2015, kedy pridáva podporu pre viacero cudzích jazykov. Začiatkom roka 2017 je Trello odkúpené firmou **Atlassian**, ktorá ho vlastní dodnes. Aplikácia je dnes dostupná vo viac ako 20 jazykoch s aplikáciami pre web, iPhone, Android, macOS, či Windows.

Tak ako metodológia Kanban, ktorú aplikácia implementuje, využíva Trello nástenky (Boards) reprezentujúce určité celky (projekty, oddelenia, či tímy). Trello nie je určené len na vývoj softvéru, ale je ním možné organizovať aj osobný život alebo školu. Jednotlivé nástenky obsahujú stĺpce, ktoré k sledovaniu priebehu položiek na nástenke, či k triedeniu vecí. Na každej takejto nástenke sú umiestnené karty, ktoré predstavujú jednotlivé úlohy.



Obr. 3.1: Príklad Trello nástenky, prevzaté z[1]

[1] https://freeup.net/blog/freelance-tips/how-to-use-trello-organized/

Trello obohacuje tradičný systém Kanban o nové funkcie, ako je napríklad pridávanie príloh ku kartám v podobe dokumentov, rôzne filtrovanie pomocou termínu dokončenia alebo priradenej osoby alebo zasielanie upozornení e-mailom. Nevýhodou aplikácie Trello však je, že neposkytuje pokročilejšie funkcie ako je radenie úloh podľa priority, hodinovej náročnosti, či automatické presúvanie kariet a pod. Základné funkcie sa však dajú rozšíriť využitím rôznych prepojení s aplikáciami tretích strán (napr. Google Drive, Dropbox, Slack). Trello ponúka možnosť vytvárania vlastných aplikácií a rozšírení. Práve táto možnosť je použitá aj pri tvorbe praktickej časti tejto bakalárskej práce.

Cieľom práce je vytvoriť aplikáciu/službu, ktorá umožní agregovať karty z viacerých Trello nástenieк do jednej a vykonávať nad nimi obojsmerne zmeny. K dosiahnutiu požadovaného výsledku nám pomôže **Trello API**. Samotnému API sa venuje nasledujúca kapitola 3.1

3.1 Trello API

Časti nasledujúcej podkapitoly boli prevzaté z [5]

Trello API[2] je oficiálne aplikačné rozhranie pre tvorbu vlastným aplikácií a rozšírení pre Trello. Obsahuje sadu nástrojov pre prístup k užívateľom, nástenkám, zoznamom, kartám a viac.

Táto podkapitola obsahuje prehľad funkcionalít, ktoré Trello API poskytuje a ktoré boli využité pri tvorbe praktickej časti tejto bakalárskej práce. Prvým krokom pre prácu s API je **autorizácia a autentifikácia** 3.1.1, nasleduje ukážka API volaní v časti 3.1.2 a nakoniec funkcionalita nazývaná **Trello webhook** 3.1.3, ktorá umožňuje našej aplikácií prijímať informácie o zmenách vykonaných v aplikácii Trello.

3.1.1 Autorizácia a autentifikácia

Trello využíva tzv. delegovaný autorizačný a autentifikačný tok za účelom, aby aplikácie využívajúce API nepotrebovali riešiť ukladanie a spracovanie užívateľských mien a hesiel. Namiesto toho, aplikácie predajú kontrolu Trellu, kde sa identifikujú pomocou API kľúča a potom, čo Trello dá užívateľovi možnosť vybrať svoj účet a prihlásiť sa, predá kontrolu späť aplikácii spolu s API tokenom.

Prvým krokom je práve získane **API kľúča**. Jedná sa o veľmi jednoduchý krok, kedy postačuje existencia účtu, prihlásenie a navštívenie nasledujúcej webovej adresy `https://trello.com/app-key`. Kľúč sa nachádza vo vrchnej časti stránky a jedná sa o 32 znakový reťazec obsahujúci náhodne alfanumerické znaky.

Samotný API kľúč nepovoľuje prístup k dátam Trello užívateľa, preto môže byť verejne dostupný. Naopak, **API token** umožňuje prístup k dátam užívateľa, ten by mal ostať tajný.

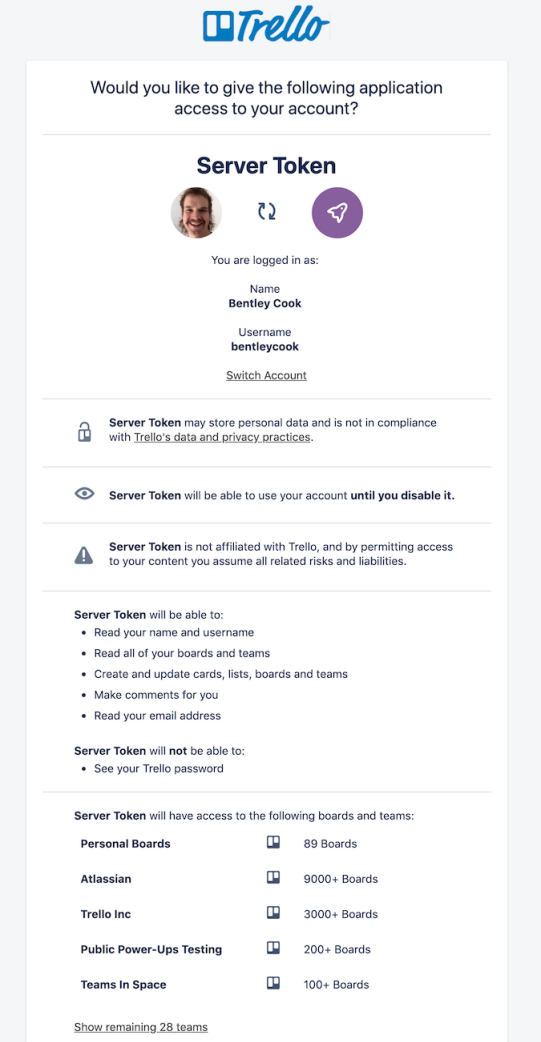
Trello API využíva autentifikáciu založenú na tokenoch, aby umožnila aplikáciám tretích strán pristupovať k API. Po tom, čo užívateľ povolí prístup aplikácii prístup k jeho účtu a dátam, aplikácii je udelený token, ktorý je možné použiť na zasielanie požiadavkov na API v mene užívateľa.

Existujú dva spôsoby, ako autorizovať klienta a tým získať token. Prvým spôsobom je využitím ich vlastnej autorizačnej routy, druhým je využitím OAuth1.0. V našej aplikácii bol

[2] **API** - aplikačné programovanie rozhranie, rozhranie, ktorým si dve aplikácie medzi sebou vymieňajú dáta

využitý prvý spôsob za pomoci Javascript knižnice **Client.js**. Knižnici Client.js je venovaná časť 5.2.4.

Pretože je API kľúč viazaný na užívateľa, je odporúčané vytvoriť samostatný Trello účet čisto určený pre tvorbu aplikácie alebo rozšírenia. Po obdržaní API kľúča, je tento kľúč použitý na získanie prístupu Trello užívateľov k aplikácii. Stačí presmerovanie užívateľa na špecifikovanú URL adresu s vyplnenými GET parameterami. Užívateľovi sa zobrazí okno s žiadosťou o udelenie prístupu k aplikácii.



Obr. 3.2: Ukážka autorizačného vyskakovacieho okna, prevzaté z[3]

Trello užívatelia môžu sledovať metadáta k aplikáciám, ktoré autorizovali a udelili token navštívením stránky `https://trello.com/{username}/account`, kde môžu sledovať zoznam aplikácii, aké práva im pridelili, dátum autorizácie a dátum vypršania platnosti tokenu.

Užívatelia rovnako môžu kedykoľvek anulovať niektorú z autorizácií, čím zneplatnia token udelený danej aplikácii a tá už nebude môcť vykonávať požiadavky v mene užívateľa.

[3] `https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/`

3.1.2 Ukážka práce s API

Po úspešnom získaní API kľúča a tokenu pre účet, je možné začať pracovať s API. Jedným z najčastejšie používanými objektami v Trello sú nástenky. Informácie o nich je možné získať jednoduchým GET požiadavkom.

```
curl 'https://api.trello.com/1/members/me/boards?key={yourKey}&token={yourToken}'
```

Výpis 3.1: Príklad požiadavku na získane informácií o nástenkách užívateľa

Po uskutočnení požiadavku vyššie, je získaná odpoveď, ktorá obsahuje objekty vo formáte **JSON**[4] o všetkých nástenkách, ku ktorým je daný užívateľ priradený.

```
[
    {
        "name": "Board A",
        "id": "5b6893f01cb3228998cf13e",
        "url": "https://trello.com/b/pLc9vEa1/board-a",
        ...
    }
    {
        "name": "Board B",
        "id": "5b6893f01cb3228998cf25g",
        "url": "https://trello.com/b/ev0Scr3C/board-b",
        ...
    }
    ...
]
```

Výpis 3.2: Príklad zjednodušenej odpovede o nástenkách užívateľa

Pomocou Trello API je možné vykonávať veľké množstvo požiadavkov nad rôznymi dátami z Trella, kompletný prehľad je možné nájsť na adrese
https://developer.atlassian.com/cloud/trello/rest/

3.1.3 Trello Webhook

Trello Webhook je funkcionalita, ktorá umožňuje aplikáciám dostávať upozornenia o zmenách, ktoré v Trelle nastali. Je to spôsob, ktorým získava upozornenia aj naša aplikácia.

Webhook je priradený tokenu udelenému aplikácii a môže sledovať len objekty, ku ktorým má token prístup.

Prvou podmienkou k vytvoreniu Webhooku je získanie prístupového tokenu. Tejto časti je venovaná podkapitola 3.1.1. Druhou podmienkou je URL adresa v **callbackURL** parametri. Keď nastane zmena nad modelom špecifikovaným vo Webhooku, práve na adresu špecifikovanú v tomto parametri sú zaslané dáta o zmene formou POST požiadavku. Telo požiadavku obsahuje dáta vo formáte JSON. Špecifikovaná URL adresa v callbackURL parametri musí byť platná v čase vytvorenia Webhooku. Trello overí platnosť adresy zaslaním HEAD požiadavku a očakávaním návratového kódu 200 OK ako odpovede. Poslednou požiadavkou je **identifikátor modelu**, ktorý chce aplikácia sledovať. Môže sa jednať o identifikátor nástenky, zoznamu, karty alebo užívateľa. Rôzne objekty sledujú rôzne typy zmien.

[4] **JSON (Javascript Object Notation)** - jednoduchý textový formát slúžiaci na prenos dát

```
\$.POST("https://api.trello.com/1/tokens/{yourToken}/webhooks?key={yourKey}", {
    callbackURL: "http://anywebsite.com/webhookCallback",
    description: "Webhook",
    idModel: "6f3g3cg6jzc3a11330200532d"
});
```

Výpis 3.3: Príklad vytvorenia Webhooku sledujúceho zmeny nad nástenkou

```
{
    // Informacie o~vykonanej zmene
   "action": {
      "id": "51f9424bcd6e040f3c002412",
      "idMemberCreator": "4fc78a59a885233f4b349bd9",
      "data": {
         "board": {
            "name": "Board A",
            "id": "6f3g3cg6jzc3a11330200532d"
         },
         "old": {
            "name": "Old Name"
         }
         "card": {
            "idShort": 1239,
            "name": "New Name",
            "id": "51a79e72dbb7e23c7c003778"
         },
      },
      "type": "updateCard",
      "date": "2020-02-17T11:21:34.031Z",
      ...
   },
   // Informacie o~modeli, nad ktorym bola vykonana zmena
   "model": {
      "id":"6f3g3cg6jzc3a11330200532d",
        ...
   }
}
```

Výpis 3.4: Príklad prijatého POST požiadavku o zmene z Trello Webhook

Kapitola 4

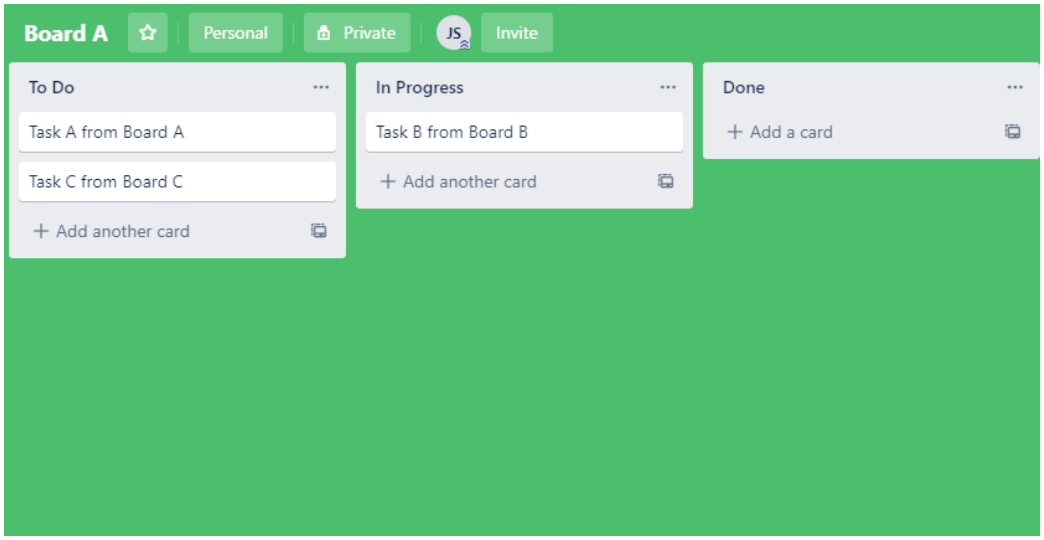
Návrh aplikácie

Kapitola je venovaná návrhu aplikácie, ktorá je výsledkom tejto bakalárskej práce. Je potrebné špecifikovanie cieľov a požiadaviek, aký účel má výsledná aplikácia plniť. Ďalej sa kapitola venuje návrhu riešenia a už existujúcim riešeniam.

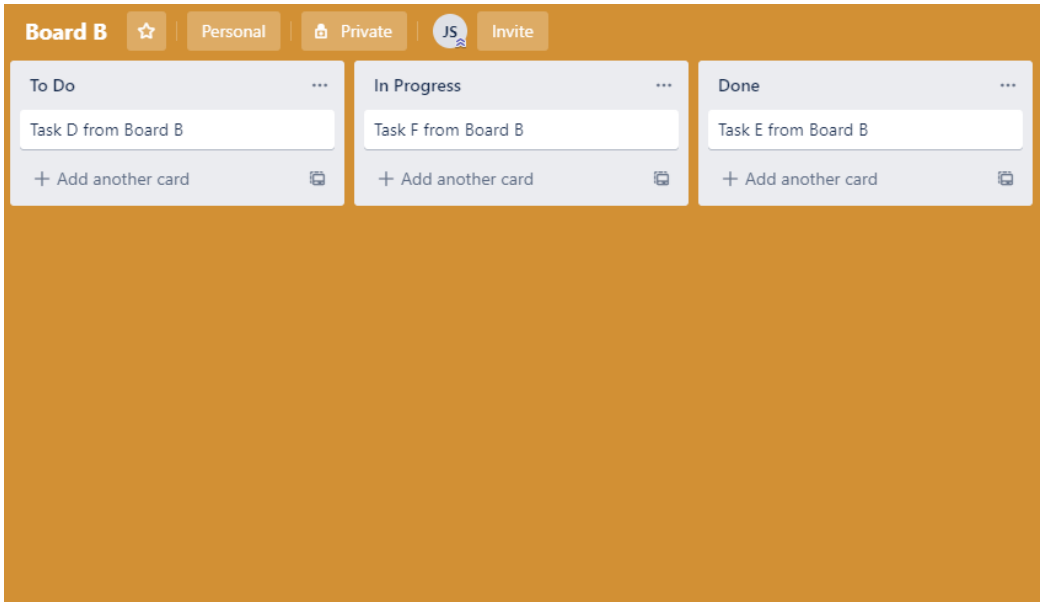
4.1 Cieľ práce a špecifikácia riešenia

Hlavným cieľom tejto práce je navrhnúť a implementovať službu, ktorá umožňuje agregáciu lístkov (kariet) z viacerých Trello nástenieck do jednej. Taktiež má umožňovať obojsmernú synchronizáciu zmien, to znamená, že zmena vykonaná v našej aplikácii je prenesená do Trella a zmena vykonaná v Trelle je prenesená do našej aplikácie.

Aplikácia by teda mala ponúknuť užívateľovi jednu nástenku, ktorá obsahuje zoznamy a karty z viacerých jeho nástenieck.

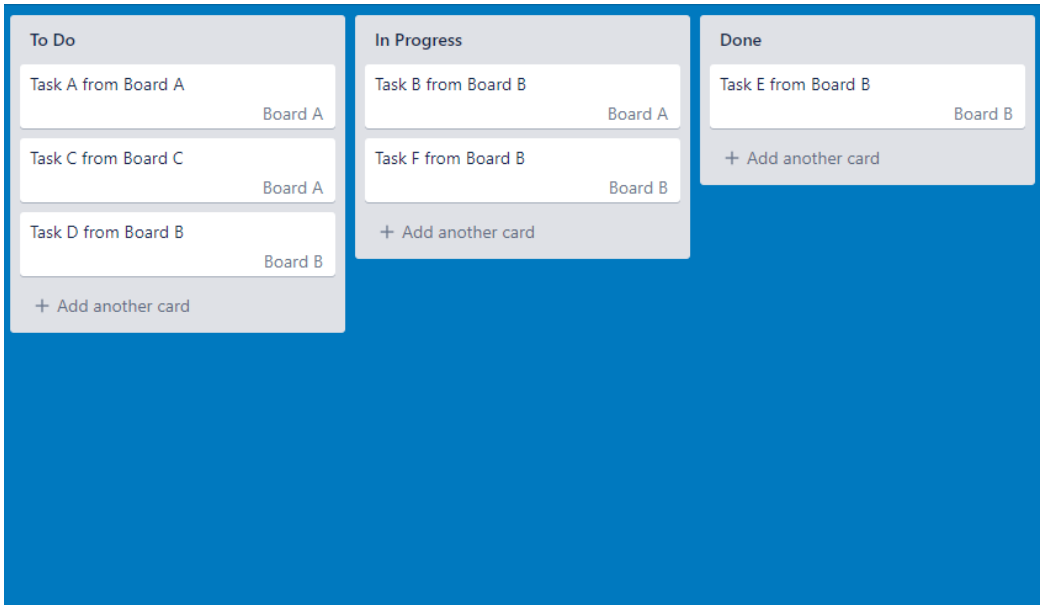


Obr. 4.1: Príklad jednoduchej nástenky č. 1



Obr. 4.2: Príklad jednoduchej nástenky č. 2

Obrázky 4.1 a 4.2 znázorňujú dve jednoduché nástenky, ktoré obe obsahujú zoznamy s rovnakým názvom. V prípade, že nástenky obsahujú zoznamy s rovnakým názvom, bude sa zoznam s daným názvom vyskytovať v našej aplikácii len raz. Výsledok agregácie kariet v našej aplikácii je znázornený na obrázku 4.3.



Obr. 4.3: Agregovaná nástenka z príkladov 4.1 a 4.2

Okrem samotného zobrazenia by mala aplikácia ponúknuť vykonávanie zmien nad objektami tejto nástenky. Jedná sa o manipuláciu so zoznamami a kartami, ako ich presúvanie, ich editácia, možnosť vytvárať nové, či objekty mazať. Cieľom aplikácie je ponúknutie minimálne základných a najviac používaných možností, ktoré ponúka samotné Trello.

Aplikácia by mala ponúknuť užívateľom jednoduché prihlásenie využitím ich Trello účtu.

4.2 Existujúce riešenia

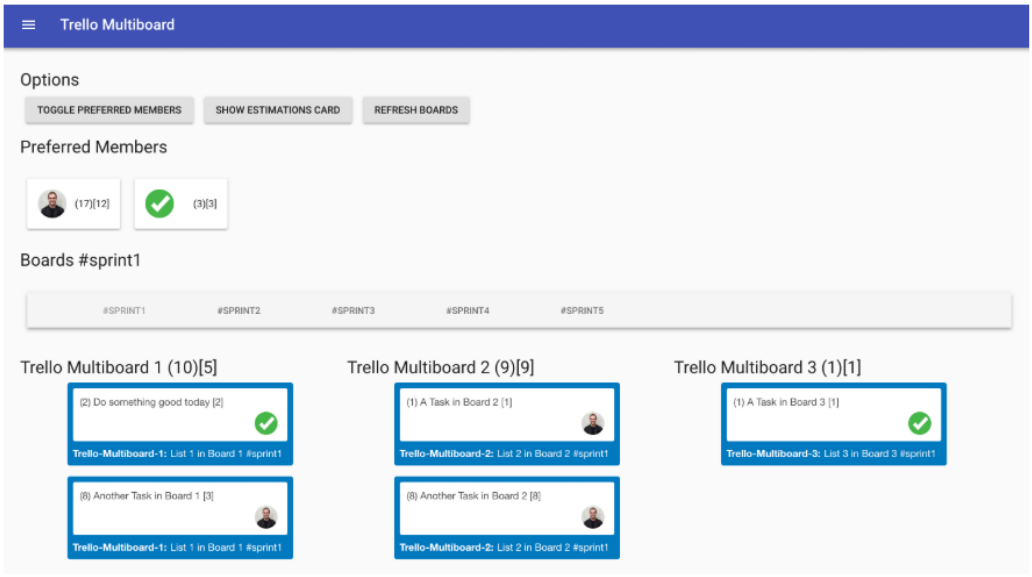
Dôležitým krokom pri návrhu a implementácii riešenia určitého problému je prehľad už existujúcich riešení. Takýto prehľad je dôležitý k získaniu informácií o obtiažnosti implementácie riešeného problému, odporúčaným, či neodporúčaným technológiám, pomocou ktorých bude prebiehať implementácia, či k získaniu inšpirácie.

Výsledkom prieskumu existujúcich riešení bola jediná webová aplikácia.

React-Trello-Multiboard

Aplikácia **React-Trello-Multiboard** je aplikácia typu SPA vytvorená pomocou Javascriptovej knižnice React. Umožňuje užívateľom vidieť jednotlivé nástenky užívateľa ako záložky, ktoré obsahujú vybrané zoznamy a karty danej nástenky. Autorizácia prebieha vyplnením údajov v konfiguračnom súbore. V čase práce na tejto bakalárskej práci, aplikácia nie je nasadená na žiadnom hostingu, a teda jedinou možnosťou je stiahnutie aplikácie z jej GitHub repozitára[1] a spustením aplikácie lokálne.

Aplikáciu nebolo možné riadne otestovať a zistiť všetku funkcionalitu, ktorú ponúka, nakoľko nebolo možné nainštalovať všetky potrebné balíčky z dôvodu problémov so závislosťami niektorých balíčkov. V popisu aplikácie a dema vyplýva, že aplikácia slúži len na prezeranie zoznamov a kariet a neponúka možnosť synchronizácie zmien s aplikáciou Trello.



Obr. 4.4: Screenshot z aplikácie React-Trello-Multiboard, prevzaté z GitHub repozitára

4.3 Návrh riešenia

Rozhodol som sa implementovať požadovanú službu formou webovej aplikácie typu **SPA (Single Page Application)**. Aplikácie tohoto typu sú charakteristické tým, že sa snažia spojiť výhody desktopových aplikácii a dostupnosť webových aplikácii. Tento prístup je odlišný od tradičného, kedy pri webových aplikáciách je primárnou otázkou predovšetkým

[1] **React-Trello-Multiboard** - https://github.com/natterstefan/react-trello-multiboard

server a klient slúži iba ako sprostredkovateľ. U SPA sa vývoj sústredí na klientskú časť a server zvyčajne slúži iba ako zdroj dát.

Vzhľadom na očakávanú komplexnosť chovania aplikácie a užívateľského rozhrania som sa rozhodol nevyužiť spôsob tvorby webových stránok pomocou čistého HTML, CSS a Javascriptu a siahol som po Javascriptom frameworku zameranom presne na tento účel.

Najpoužívanejšími frameworkami sú: React, Angular.js, Vue.js, Ember.js, či Backbone.js. Rozhodol som sa použiť React, ktorý ponúka kvalitnú dokumentáciu a veľké množstvo knižníc.

Nakoľko je cieľom práce poskytnúť užívateľovi obojsmernú synchronizáciu zmien, je potrebná existencia aj serverovej časti našej aplikácie. To znamená, že naša aplikácia bude tvorená architektúrou klient-server.

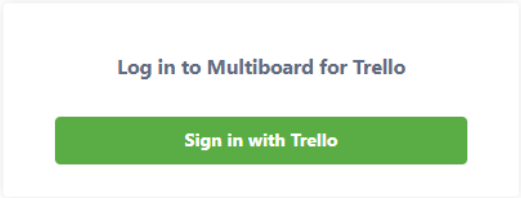
Server bude plniť dve činnosti. Prvou je prezentácia súborov klientskej aplikácie užívateľovi a druhou je zasielanie dát o zmenách, ktoré sa udiali v aplikácii Trello. Pre serverovú časť aplikácie som sa rozhodol pre Node.js pre konzistenciu použitia rovnakého jazyka pre klientskú aj serverovú časť a pre povahu serverovej časti aplikácie.

Server bude klienta kontaktovať použitím knižnice socket.io, ktorá plní požadovaný účel zasielania dát klientovi bez toho, aby si o ne samotný klient žiadal. Táto funkcionalita je potrebná k prijímaniu dát o zmenách v aplikácii Trello, ktoré budú našej aplikácii zasielané využitím funkcionality nazývanej Trello Webhook 3.1.3.

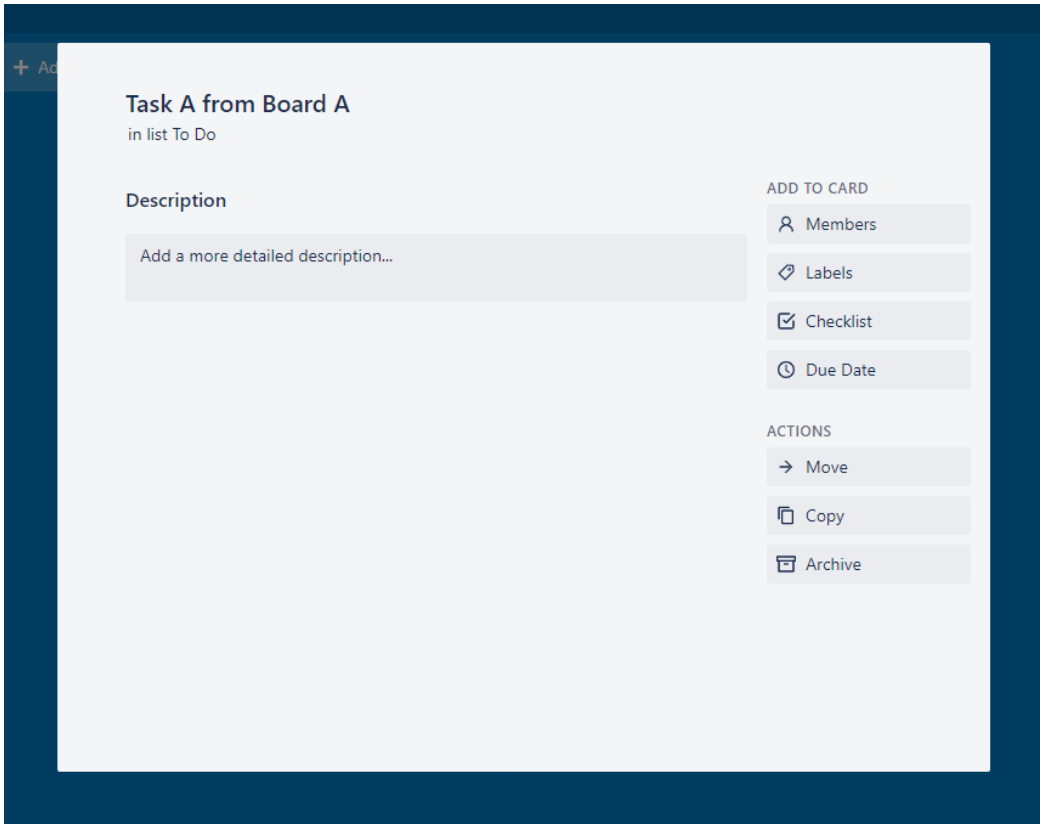
Všetky hlavné technológie použité pri tvorbe našej webovej aplikácie sú popísané v kapitole 5.

4.4 Návrh užívateľského rozhrania

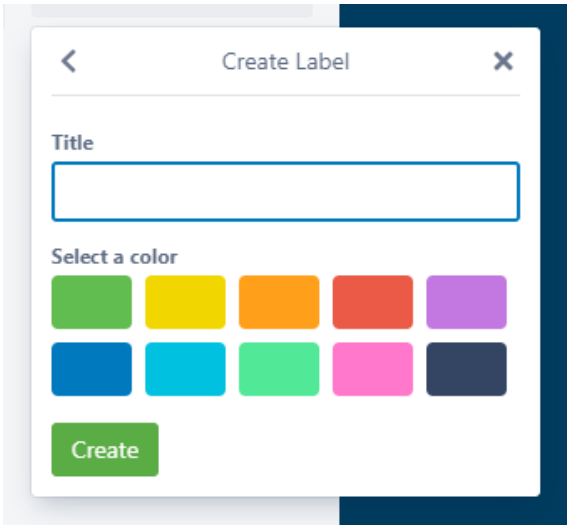
Primárnou cieľovou skupinou našej aplikácie sú používatelia samotnej Trello aplikácie. Z tohoto dôvodu som sa rozhodol využiť a implementovať užívateľské rozhranie, ktoré je skoro identické s tým, ktoré poskytuje Trello. Užívatelia tak budú ihneď s užívateľským rozhraním oboznámený. Obrázok 4.3 zobrazuje užívateľské rozhranie nástenky, spolu so zoznamami a kartami. Nasledujúce obrázky znázorňujú niektoré časti implementovaného užívateľského rozhrania.



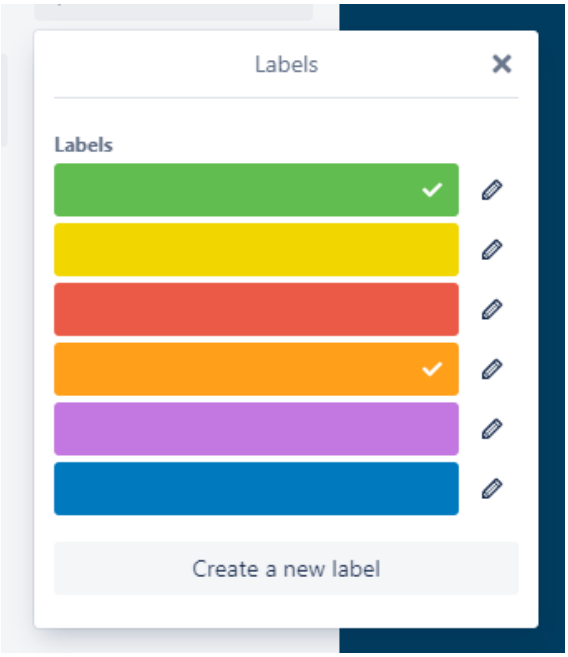
Obr. 4.5: Prihlasovacia obrazovka



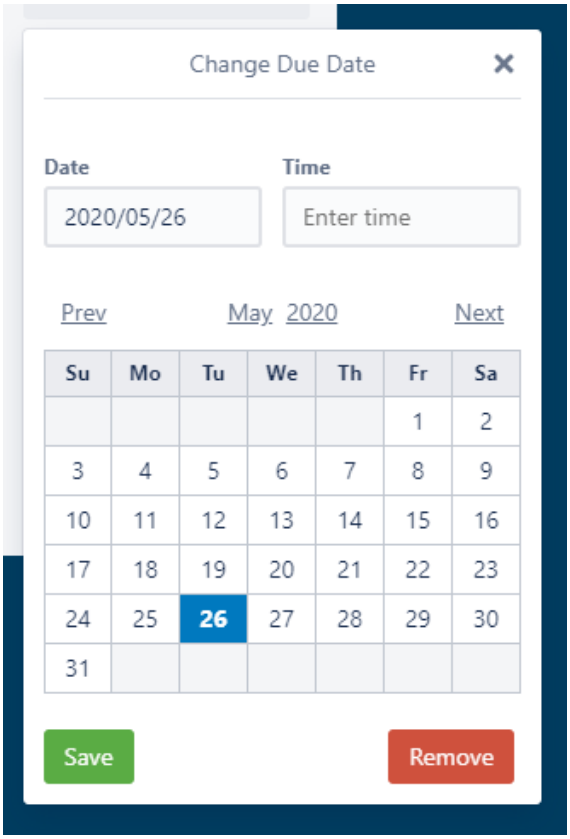
Obr. 4.6: Detail karty



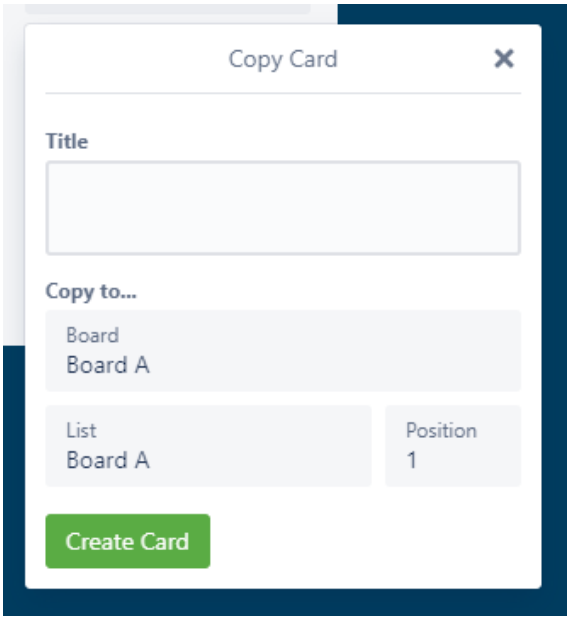
(a) Vytvorenie nového štítku



(b) Voľba štítkov pre karty



(c) Nastavenie termínu



(d) Vytvorenie kópie karty

Obr. 4.7: Popover pre rôzne akcie, ktoré je možné vykonať na karte

Kapitola 5

Použité technológie

Nasledujúca kapitola sa zaoberá prehľadom dôležitých technológií využitých pri tvorbe našej aplikácie. Kapitola je rozdelená do dvoch hlavných častí podľa architektúry aplikácie - serverové a klientské.

5.1 Serverové technológie

Serverová časť našej aplikácie je tvorená jednoduchým **Node.js** serverom využívajúcim primárne framework **Express.js**, knižnicu **socket.io** a jednoduchú JSON databázu **lowdb**.

5.1.1 Node.js

Platforma Node.js[1] je open-source, multiplatformové Javascriptové prostredie bežiace na V8 Javascriptovom interpréte od spoločnosť Google. Vyvinul ju Ryan Dahl v roku 2009. Využíva sa na tvorbu vysoko škálovateľných serverových a sieťových aplikácii. Ide o prostredie umožňujúce spúšťať Javascript kód mimo webový prehliadač.

Node.js má niekoľko výhod, vďaka ktorým sa stal jednou z veľmi populárnych a často používaných technológii [7]:

- **Asynchrónnosť a udalosťami riadená architektúra** - všetky API Node.js knižnice sú asynchrónne, tj. neblokujúce. To znamená, že napríklad pri načítaní dát sa nečaká na dokončenie operácie, ale aplikácia pokračuje ďalej. Keď sa načítavanie dokončí, výsledok je spracovaná v cykle, ktorý spracováva prichádzajúce udalosti
- **Rýchlosť** - aplikácie postavené na Node.js sú veľmi rýchle.
- **Jednovláknové, no vysoko škálovateľné** - Node.js aplikácie bežia na jednom vlákne a využívajú tzv. cykle udalostí (angl. event loop). Tento cyklus spracováva prijaté udalosti alebo čaká, kým nie je nejaká udalosť vyvolaná a potom pre ňu volá jej obslužnú funkciu (angl. callback).
- **Žiaden buffering** - aplikácie nikdy neukladajú údaje do vyrovnávacej pamäte, vydávajú informácie v blokoch.
- **Licencia** - Node.js je vydaný pod MIT licenciou.

[1] https://nodejs.org/en/

5.1.2 Express.js

Express.js[2] je jednoduchý webový framework založený na moduli http v Node.js a komponentoch frameworku Connect.js. Tieto komponenty sa nazývajú middleware a slúžia ako základ Express.js frameworku. Middleware umožňujú využitie už existujúceho kódu pre svoj projekt, ktorý rieši väčšinu základných problémov, ako napr. spracovanie HTTP požiadavkov, práca s cookies, či session, práca s prihlasovaní, smerovanie, či spracovanie chýb.

Middleware sú funkcie, ktoré sprístupňujú objekty request (req), response (res) a funkciu next() slúžiaca k prípadnímu spusteniu ďalšieho middleware. Express.js obsahuje tzv. zásobník middleware funkcií, ktoré sa postupne spúšťajú pri prijatí každého HTTP požiadavku. [4] Je možné ich pripojiť k aplikácii pomocou metód app.use() alebo app.VERB().

Aplikácie tvorené v Express.js sú obvykle tvorené niekoľkými po sebe idúcimi krokmi:

- Definovanie závislostí, napríklad pripojenie knižníc tretích strán.
- Vytvorenie samotného objektu Express.js.
- Konfigurácia aplikácie, nastavenie premenných.
- Pripojenie k databáze.
- Definovanie funkcií middleware.
- Tvorba smerovania (routing).
- Samotný štart aplikácie a nastavenie portu, na ktorom aplikácia počúva.

5.1.3 Lowdb

Lowdb[3] je veľmi jednoduchú JSON databázu vytvorenú firmou Lodash. Umožňuje jednoduché ukladanie a čítanie dát bez potreby tvorby databázového servera.

5.1.4 Socket.io

Socket.io[4] je knižnica umožňujúca obojsmernú, udalosťami založenú komunikáciu v reálnom čase medzi klientom a serverom.

5.1.5 Ngrok

Jedná sa o nástroj využitý pri vývoji našej aplikácie. Pri produkčnom nasadení nie je potrebný. Poskytuje možnosť „zviditeľniť“ localhost pre verejnú sieť. Podstatou aplikácie je preposielanie požiadavkov prijatých z verejnej siete na službu bežiacu lokálne. Nástroj bol využitý pre prijímanie Trello Webhook dát.

[2] https://expressjs.com/
[3] https://openbase.io/js/lowdb
[4] https://socket.io/

5.2 Klientské technológie

Klientská časť našej aplikácie je tvorená využitím Javascriptovej knižnice React s využitím knižnice Redux pre správu stavu aplikácie, knižnicou client.js a axios pre autorizáciu, autentifikáciu a komunikáciu s Trello API.

5.2.1 React

React[5] je Javascriptová knižnica využívaná na tvorbu znovu-použiteľných komponentov, ktoré slúžia ako základ pri tvorbe užívateľského rozhrania aplikácií. Bol vyvinutý spoločnosťou Facebook v roku 2011 a neskôr v roku 2013 sa stáva verejne dostupnou knižnicou. V kontexte softvérovej architektúry MVC (Model-View-Controller), reprezentuje React úlohu V (View).

Srdcom každej React aplikácie sú komponenty. Komponent je nezávislá časť užívateľského rozhrania. Každá aplikácia obsahuje minimálne jeden komponent, ktorý sa nazýva koreňový komponent. Príkladom komponentu môže byť menu, ktoré obsahuje položky (znova komponenty). Komponenty vytvárané Reactom sa nazývajú prezentačné komponenty. Rozdeľujú sa na funkcionálne a triedne, pričom varianta triednych komponentov je dostupná od **ECMAScriptu 6 (ES6)**[6]. Obe varianty sú ekvivalentné a ich použitie je možné zameniť. Líšia sa však prístupom k svojím elementom. Pri funkcionálnych komponentoch sa premenné predávajú ako parametre, pri triednych tieto parametre obdrží komponent automaticky.

Príklady 5.1 a 5.2 ilustrujú rozdiel spomínaný vyššie.

```
function Welcome(props) {
    return (
        <h1> Hello, { props.name } </h1>
    );
}

const welcome = <Welcome name="John" />;
```

Výpis 5.1: Predávanie hodnôt do funkcionálnych komponentov

```
class Welcome extends React.Component {
    render() {
        return (
            <h1> Hello, { this.props.name } </h1>
        );
    }
}

const user = <User name="John" />;
```

Výpis 5.2: Predávanie hodnôt do triednych komponentov

Jedným z hlavných prvkov komponent je ich**stav (angl. Component State)**. Slúži ku kontrole nutnosti prekreslenia daného komponentu a prípadne ako zdroj jeho dát. Pokiaľ dôjde k zmene stavu komponentu, je vyvolaná funkcia, ktorá zabezpečí znovu vykreslenie daného komponentu. Implementácia komponentov využívajúcich stav bola pred verziou

[5] https://reactjs.org/

[6] **ES6** - taktiež nazývaný Javascript 6, štandard podľa špecifikácie http://www.ecma-international.org/ecma-262/6.0/

knižnice React 16.8.0 možná len pomocou triedneho komponentu, ktorý dedí od triedy **React.Component**, kde sa stav inicializuje pomocou premennej `this.state` v konštruktore triedy. Od tejto verzie je možné písať funkcionálne komponenty využívajúce stav za pomoci **React Hooks API**. Tento typ komponentov inicializuje stav funkciou `React.useState()`.

Príklady 5.3 a 5.4 ilustrujú rozdiel v inicializácii stavu pri funkcionálnom a triednom komponente.

```
function User({ name }) {
    const [name] = React.useState(name);
}

const welcome = <Welcome name="John" />;
```

Výpis 5.3: Inicializácia stavu pri funkcionálnych komponentoch využitím React.useState()

```
class User extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            name: this.props.name;
        }
    }
}

const user = <User name="John" />;
```

Výpis 5.4: Inicializácia stavu pri triednych komponentoch v konštruktore triedy

Syntax, ktorou sú komponenty zapisované sa nazýva **JSX**[7]. JSX alebo aj Javascript XML je rozšírenie XML/HTML pre Javascript. **Babel**[8] prekladá JSX do `React.createElement()` volaní [6]. Nasledujúce príklady 5.5 a 5.6 sú preto zameniteľné.

```
const element = <h1 className="greeting"> Hello, world! </1>;
```

Výpis 5.5: Vytvorenie jednoduchého elementu v JSX

```
const element = React.createElement(
    'h1',
    { className: 'greeting' },
    'Hello, world!'
);
```

Výpis 5.6: Vytvorenie jednoduchého elementu pomocou React.createElement()

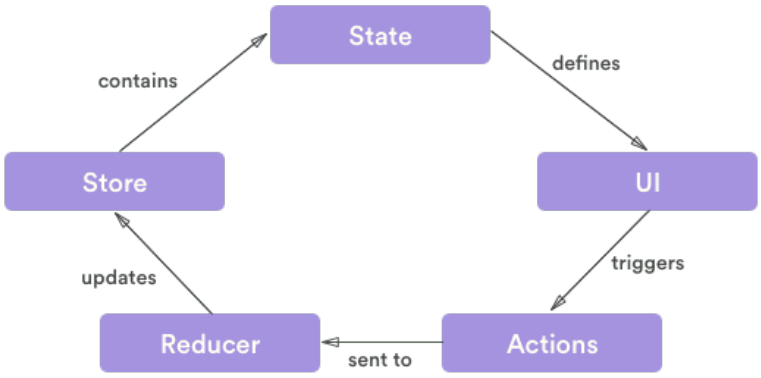
[7] https://facebook.github.io/jsx/
[8] **Babel** - Javascript kompilátor

5.2.2 Redux

Ako bolo v predchádzajúcej podkapitole o Reacte 5.2.1 spomenuté, jedným z najdôležitejších prvkov komponentov v React aplikáciách je ich stav. Knižnica Redux rieši práve tento problém. Jedná sa o knižnicu starajúcu sa o globálny stav aplikácie. Navrhnutá bola pre React, ale jej obdoby je možné použiť aj s ostatnými knižnicami/frameworkami na tvorbu užívateľských rozhraní ako sú Angular, Vue.js alebo React Native.

Redux si zakladá na troch princípoch [3]:

- Stav celej aplikácie je uložený v strome objektov v rámci objektu nazývaného aj store. Z pohľadu implementácie sa jedná o objekt, ktorý obsahuje objekty reprezentujúce jednotlivé časti aplikácie.
- Stav je určený len na čítanie. Jediným spôsobom ako možno zmeniť stav aplikácie je volaním metód `dispatch()` s parametrami akcie, ktorá popisuje aká zmena stavu má byť vykonaná.
- Zmeny sú vykonávané tzv. čistou funkciou (angl. Pure function) [8], čo znamená, že jej návratová hodnota je ovplyvnená iba jej vstupmi a nemá žiadne vedľajšie účinky. Akcia, ktorá definuje, aká zmena sa má vykonať, je ďalej predaná funkcii, ktorá sa nazýva **reducer**, a tá vykoná konkrétnu zmenu stavu.



Obr. 5.1: Tok riadenia stavu v React aplikácii využívajúcej Redux, prevzaté z[9]

[9] https://dev.to/radiumsharma06/abc-of-redux-5461

Redux tvoria tri časti [2]:

Akcie a Actions Creators

Akcie sú reprezentované objektami, ktoré sú vytvárané funkciami, ktoré nazývame **Action Creators**. Tieto funkcie umožňujú parametrizáciu akcií, ktoré okrem typu akcie nesú aj ľubovoľné dáta, ktoré sú potrebné pre ďalšiu transformáciu stavu aplikácie. Akcie však nepopisujú proces stavu, len definujú aká zmena má byť vykonaná.

```
const ADD_ITEM = 'ADD_ITEM';

const actionAddItem = (item) => ({
    type: ADD_ITEM,
    item,
});
```

Výpis 5.7: Príklad jednoduchej Redux akcie

Reducers

Reducers sú funkcie, ktoré vykonajú konkrétnu zmenu stavu aplikácie popísanú akciou. Zmenou sa v tomto prípade rozumie vytvorenie kópie stavu, jej úpravy podľa danej akcie a následne nahradenie starého stavu novým.

```
const itemReducer = (state, action) => {
    switch(action.type) {
        case ADD_ITEM:
            return {
                ...state,
                items: state.items.push(action.item)
            };
        default:
            return state;
    }
}
```

Výpis 5.8: Príklad jednoduchej Redux reducer funkcie

Pri práci so stavom aplikácie sa využíva **princíp nemenných dát (angl. Immutable Data)**. Tento princíp určuje, že štruktúra (pole, objekt, pole objektov atď.), ktorá bola vytvorená pomocou nemenných dát nemôže byť po vytvorení priamo menená. Pri každej chcenej zmene je potrebné vytvoriť kópiu tejto štruktúry a zmeny vykonať na nej.

Store

Store je objekt, v ktorom je uložený stav celej aplikácie. Definuje niekoľko funkcií na prácu s jeho stavom. Funkcia `getState()` je používaná k získaniu momentálneho stavu. Zmeny sa vykonávajú volaním `dispatch()`.

5.2.3 Prepojenie React a Redux

Aby mohli React komponenty pristupovať k stavu aplikácie spravovaného pomocou knižnice Redux, je nutné využiť knižnicu **react-redux**, ktorá prepája obe knižnice. Tá poskytuje

komponent nazývaný `Provider` a funkciu `connect()`. Provider najčastejšie býva tzv. koreňový komponent aplikácie a prijíma inštanciu Redux store ako parameter.

```
import React from react ;
import ReactDOM from react-dom ;
import { Provider } from react-redux ;
import { createStore } from redux ;
import { rootReducer } from ./reducers ;
import App from ./App ;

const store = createStore(rootReducer);

ReactDOM.render(
    <Provider store={ store }>
        <App/>
    </Provider>,
    document.getElementById(root)
);
```

Výpis 5.9: Prepojenie React s Redux

Na zabezpečenie prístupu komponentu k stavu aplikácie, je potrebné ho obaliť pomocou funkcie `connect()`, ktorá obalí požadovaný komponent komponentom vyššieho radu. Funkcia očakáva dva parametre:

- **mapStateToProps()** - slúži k namapovaniu stavu aplikácie do komponentu. Najčastejšie vraciame iba určitú časť stavu aplikácie.
- **mapDispatchToProps()** - slúži k namapovaniu dispatch funkcie, tj. možnosť volať akciu priamo, bez nutnosti obalenia volania do dispatch().

Dáta a akcie sú potom predané komponentu cez parametre `props`.

```
import React from 'react';
import { connect } from 'react-redux';
import Item from './Item';

class List extends React.Component {
    render() {
        return this.props.items.map((item) => {
            return <Item data={ item } updateItem={ handleUpdateItem } />;
        }
    }
}

const mapStateToProps = (state) => ({
    items: state.items,
});

const mapDispatchToProps = (dispatch) => ({
    handleUpdateItem: (id, text) => dispatch(updateItem(id, text)),
});

export default connect(mapStateToProps, mapDispatchToProps)(List);
```

Výpis 5.10: Pripojenie komponentu na Redux pomocou connect()

5.2.4 Trello Client.js

Trello Client.js je malá Javascriptová knižnica, ktorá umožňuje jednoduchú interakciu aplikácie s Trello API. Poskytuje globálny Trello objekt, ktorý obsahuje pomocné metódy pre akcie s Trello API.

Jej pripojenie a použitie je veľmi jednoduché, je potrebné pridanie nasledujúceho riadku do **<head> tagu HTML súboru** aplikácie a prístupní sa pre aplikáciu globálny Trello objekt.

```
<script src="https://trello.com/1/client.js?key=[key]"></script>
```

Výpis 5.11: Pripojenie knižnice Trello client.js

Pomocou tejto knižnice je možná aj autorizácia/deautorizácia aplikácie pre Trello API.

```
Trello.authorize({
    type: redirect | popup,
    name: '', // Identifikator aplikacie, ktora ziada o~autorizaciu
    persist: true | false, // Ulozenie tokenu lokalne
    interactive: true | false, // Ak false, pouzie sa len lokalne ulozeny token
    scope: { read: true | false, write: true | false, account: true | false }, // Prava, o~
    ktore ziada aplikacia
    expiration: 1hour | 1day | 30days | never, // Expiracia tokenu
    success: callback, // Funkcia volana pri povoleni
    error: callback, // Funkcia volana pri zamietnuti
});
```

Výpis 5.12: Autorizácia aplikácie pre Trello API

Poskytuje taktiež množstvo pomocných metód, napríklad jednu pre každú HTTP metódu:

```
Trello.get(path[, params], success, error)
Trello.post(path[, params], success, error)
Trello.put(path[, params], success, error)
Trello.delete(path[, params], success, error)
```

Výpis 5.13: Metódy pre HTTP požiadavky na API

Kapitola 6

Implementácia a testovanie

Nasledujúca kapitola je venovaná samotnej implementácii praktickej časti tejto bakalárskej práce. Kapitola je rozdelená do niekoľkých častí. Začína popisom štruktúry aplikácie, ďalej sa kapitola venuje nastaveniu servera a konfigurácii, spusteniu a nasadeniu aplikácie. Neskôr je popísaná implementácia niektorých časti klientskej aplikácie a prepojenie klientskej a serverovej časti knižnicou socket.io.

6.1 Štruktúra aplikácie

Koreňový adresár projektu obsahuje samostatné zložky pre klient a server, a súbory správca balíčkov.

```
trello-multiboard /
├── build (Obsahuje vytvorené statické súbory React aplikácie)
├── client
│   ├── public (Verejný adresár)
│   └── src (Zdrojové súbory React aplikácie)
│       ├── actions (Definície Redux akcií)
│       ├── api (Primárne pomocné funkcie pri API volaniach)
│       ├── components (React komponenty)
│       ├── reducers
│       ├── store
│       ├── styles (CSS)
│       ├── config.js
│       └── index.js
├── server
│   ├── db (Obsahuje súbor db.json, ktorý slúži ako lokálna databáza)
│   ├── routes (Obsahuje definíciu routingu)
│   ├── index.js
│   └── server.js
├── package-lock.json
├── README.md
└── package.json
```

Obr. 6.1: Adresárová štruktúra projektu

Jednotlivé zložky a súbory budú podrobnejšie popísané v nasledujúcich podkapitolách.

6.2 Nastavenie servera, konfiguračný súbor

Po spustení príkazu `npm run start` sa spustí kód súboru `/server/index.js`, ktorý zaobaľuje a prekladá pomocou kompilátoru **Babel** súbor samotného servera `/server/server.js`. Najprv je nastavený server. K nastaveniu sa využívajú súbory `/client/src/config.js`, súbor `index.js` v adresári `/server/routes` a súbory nainštalovaných balíčkov.

6.2.1 Server

Hlavným súborom servera je súbor `/server/server.js`. Pri spustení aplikácie sa sa začne interpretovať využitím Node.js interpreta kód tohoto súboru. Súbor plní nasledujúce úlohy:

- Načíta potrebné moduly.
- Inicializuje inštanciu frameworku Express.js.
- Inicializuje jednoduchú JSON databázu použitím modulu lowdb.
- Inicializuje ďalšie potrebné moduly, ako napríklad spracovanie tiel HTTP požiadavkov vo formáte JSON.
- Nastaví prezentovanie statickej React aplikácie klientovi.
- Nastaví router nachádzajúci sa v súbore /routes/index.js
- Vytvorí inštanciu HTTP servera využitím modulu http.
- Špecifikuje spracovanie udalostí knižnice socket.io.
- Nastaví server, aby počúval na špecifikovanom porte.

Vyššie popísané činnosti sú vykonané pri spustení servera. Keď už je server spustený, automaticky pri HTTP požiadavku typu GET prezentuje klientovi React aplikáciu, pri iných požiadavkách sú predávané routeru.

6.2.2 Config.js

Súbor `/client/src/config.js` je potrebné upraviť pred spustením aplikácie, či už lokálne alebo pri nasadení aplikácie do produkčného prostredia. Nasledujúce dva parametre je potrebné nastaviť podľa prostredia, v ktorom je plánované nasadenie:

- **server** - doménové meno spolu s portom, na ktorom bude server spustený. Pri lokálnom spustení „http://localhost:3001“.
- **onlyClient** - true/false hodnota špecifikujúca, či má byť využitá aj Trello Webhook funkcionalita. Ak false, nie je potrebné vypĺňať nasledujúci parameter
- **webhookCallbackURL** - špecifikuje /update route pre server. V prípade produkčného nasadenia sa sem vyplní doménové meno, pre lokálne spustenie je potrebné použiť aplikáciu typu ngrok 5.1.5, ktorá umožňuje smerovanie požiadavkov z verejnej siete na localhost.

Konfiguračný súbor ďalej obsahuje parametre **appName, read, write a expiration**, ktoré špecifikujú informácie, ktoré sú použité pri autorizácii aplikácie pre Trello API.

6.2.3 Smerovanie

Smerovane vykonávané na serveri je definované v súbore `/server/routes/index.js`. Tento súbor špecifikuje iba dve routy:

- HEAD /update - Po zaslaní žiadosti na vytvorenie Trello Webhooku 3.1.3 je na túto routu zaslaný požiadavok na overenie platnosti adresy. Server odpovedá zaslaním 200 OK.
- POST /update - Na túto routu sú zasielané dáta o zmenách, ktoré nastanú v aplikácii Trello. Pri prijatí je vyvolaná `update` udalosť knižnice socket.io a tá zabezpečí zaslanie dát príslušnému klientovi.

Všetky požiadavky typu GET sú predané klientskej časti aplikácie, ktorá rieši zobrazenie určitej časti stránky (dokumentu) pomocou knižnice **react-router-dom**. Smerovanie v React aplikácii je umožnené jednoduchým obalením, v našom prípade koreňového komponentu, komponentom knižnice s názvom `Router`. Koreňový komponent potom vracia komponent `Switch`, v ktorom sú obalené jednotlive routy.

```
class App extends React.Component {
    render = () => {
        return (
            <Switch>
                <Route exact path="/" component={Board} />
                <Route path="/user/login" component={LoginForm} />
                <Route path="/" component={Board} />
            </Switch>
        );
    };
};
```

Výpis 6.1: Koreňový komponent App s niekoľkými routami

Znamená to, že v prípade GET požiadavku na routu /user/login je užívateľovi vykreslený komponent `LoginForm`, ktorý reprezentuje prihlasovaciu obrazovku, v prípade akejkoľvek inej routy je vykreslený komponent `Board`, ktorý reprezentuje samotnú Kanban nástenku aplikácia. Tam je overované, či je užívateľ autorizovaný.

6.2.4 npm

V súboroch `package.json` a `package-lock.json` sú uložené informácie o projekte. Tieto súbory sú využívané správcom balíčkov **npm**. Súbor `package.json` obsahuje informácie ako názov, či verzia projektu, informácie o autorovi a licencia. Ďalej súbor obsahuje názov hlavného spúšťacieho súboru aplikácie a skripty. Pomocou skriptov je možné vykonávať operácie nad aplikáciou. Naša aplikácia obsahuje nasledujúce scripty:

- `npm run client-build` - prebehne presun do adresára `/client` a zavolanie funkcie `react-scripts build`, ktorá vytvorí statické súbory React aplikácie.

- `npm run server` - volanie funkcie `nodemon -r esm server/server.js`, ktorá spustí server na pozadí.
- `npm run start` - zavolanie oboch predchádzajúcich scriptov, najprv `npm run client-build` a potom `npm run server`.

Najdôležitejšou častou súboru `package.json` je zoznam závislostí, tj. zoznam balíčkov nainštalovaných pre daný projekt. Pokiaľ je potrebné pridať určitý modul/knižnicu na požadovanú funkcionality a aplikácia je vyvíjaná v jazyku Javascript, je možné použiť práve správcu balíčkov `npm`. Jednoduchým použitím príkazu `npm install <nazov_balicka> -save` bude daný balíček nainštalovaný do projektu a bude pridaná jeho závislosť do súboru `package.json`.

Súbor `package-lock.json` obsahuje informácie o stave nainštalovaných balíčkov. Súbor nie je pre funkčnosť aplikácie povinný a ak neexistuje, tak si ho `npm` vytvorí. Súbor `package.json` obsahuje balíčky, ktoré majú byť nainštalované s ich presnou alebo minimálnou verziou, `package-lock.json` obsahuje informácie o práve nainštalovaných balíčkoch. Pokiaľ závislosť určuje minimálnu požadovanú verziu, reálne môže byť nainštalovaná aj verzia novšia, pokiaľ je spätne kompatibilná.

6.3 Spustenie aplikácie a jej nasadenie

Pre spustenie našej aplikácie a jej nasadenie je potrebné mať nainštalované `Node.js`. Správca balíčkov `npm` je nainštalovaný spolu s `Node.js`.

Prvým krokom je skopírovanie projektu alebo jeho stiahnutie z GitHub repozitára[1].

Závislosti nie sú súčasťou projektu a je ich potrebné doinštalovať využitím správcu balíčkov `npm`. Zoznam závislostí a ich verzie sa nachádza v súbore `package.json`. Závislosti sa nainštalujú zavolaním `npm install` v koreňovom adresári projektu.

Ďalším krokom je potreba konfigurácie projektu. Popis konfiguračného súboru je uvedený v podkapitole 6.2.2.

Spustenie aplikácie prebehne zavolaním `npm run start`.

Pre nasadenie tejto aplikácie bolo využitá **Google Cloud Platform**[2]. Aplikácia je verejne dostupná na stránke `http://trello-multiboard.com`.

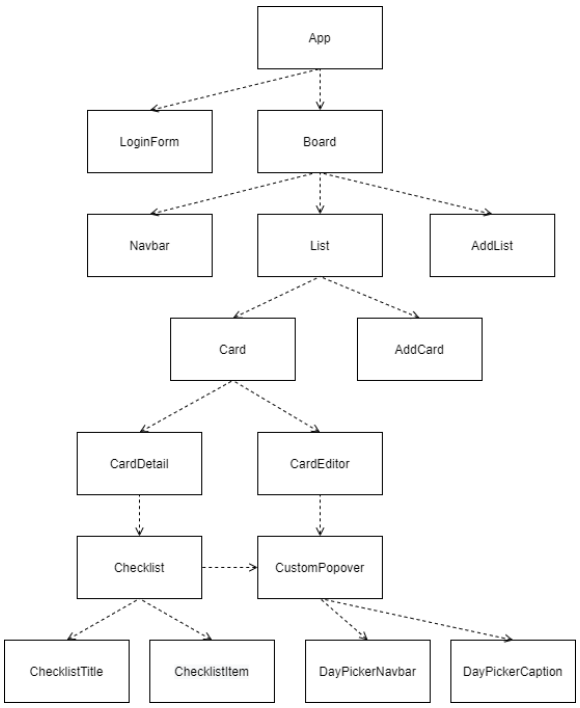
[1] `https://github.com/xstefa22/multiboard-for-trello`
[2] `https://cloud.google.com/`

6.4 Klientská aplikácia

Nasledujúca podkapitola je venovaná implementácii klientskej časti aplikácie. Ako prvé popisuje štruktúru aplikácie vzhľadom na React komponenty, implementáciu autorizácie, prepojenie klientskej a serverovej časti a prácu s API.

6.4.1 React komponenty

Komponenty sa nachádzajú v zložke `/client/src/components`. Koreňovým komponentom je trieda `App`. Prehľad komponentov a ich závislostí je znázornený na obrázku 6.2.



Obr. 6.2: Diagram komponentov našej React aplikácie

Nasleduje popis niektorých dôležitých komponent aplikácie:

- `App` je koreňovým komponentom aplikácie. Slúži zároveň ako router na strane klienta. Smerovaniu je venovaná časť 6.2.3.
- `LoginForm` reprezentuje prihlasovaciu obrazovku. Rieši autorizáciu aplikácie k používaniu Trello API.
- `Board` reprezentuje samotnú Multiboard nástenku. Chovanie, ktoré v nej prebieha je popísané v časti 6.4.3.
- `List` vykresľuje jednotlivé zoznamy Multiboard nástenky. Nereprezentuje priamo zoznamy z násteniek prihláseného užívateľa, ale určité agregované pseudo-zoznamy.
- `Card` vykresľuje karty prihláseného užívateľa. Vykresľuje sa názov karty, štítky, ktoré má karta pridelené a z ktorej nástenky pochádza.

- `CardDetail` je modal okno, ktoré reprezentuje detail karty. Obsahuje názov, popis karty, štítky, termín a to-do zoznamy. Obsahuje tiež tlačítka, ktoré reprezentujú rôzne akcie, ktoré je možné nad kartami vykonávať.
- `CardEditor` je veľmi podobný komponentu CardDetail. Vykresľuje editor karty, ktorý sa objaví po kliku na ikonu ceruzky, ktorá sa zobrazí pri ukázaní na kartu.
- `CustomPopover` je popover okno, ktoré obsahuje rôzne polia na základe požadovanej akcie pri kliknutí na niektoré z tlačidiel akcií v komponentoch CardDetail a CardEditor. Príkladom týchto polí môže byť voľba dátumu a času pri nastavení termínu karte, voľba zoznamu, nástenky a pozície pri presúvaní karty a podobne.
- `Settings` vykresľuje modal okno, ktoré umožňuje užívateľovi vybrať, ktoré nástenky chce v našej aplikácii synchronizovať. Jeho preferencie sa potom uložia do session.

6.4.2 Prihlasovanie

Prihlasovanie prebieha v komponente `LoginForm`. Pri návšteve užívateľa sa aplikácia najprv pokúsi prihlásiť z tokenu uloženého v `localStorage`, ak nejaký uložený je. Aplikácia poskytuje veľmi jednoduché prihlásenie využitím existujúceho Trello účtu. Pri prihlasovaní aplikácia požiada užívateľa po potvrdenie autorizácie aplikácie o prístup k Trello účtu užívateľa. K prihlasovaniu je využitá knižnica `Client.js` 5.2.4.

6.4.3 Nástenka

Po úspešnom prihlásení aplikácia presmeruje užívateľa na samotnú Multiboard nástenku. Pred jej vykreslením sú vykonané GET požiadavky na získanie násteniek, zoznamov, kariet užívateľa a ďalších potrebných objektov. Tieto objekty sú uložené pomocou knižnice Redux. V tomto kroku sa taktiež zašle požiadavok o vytvorení Trello Webhooku pre sledovanie zmien v Trello. Aplikácia si vytvorí pseudo-zoznamy na základe unikátnych názvov zoznamov užívateľa a vykreslí tie, aby sa nezobrazovali zoznamy s rovnakým názvom viackrát. V nich sa následne vykreslia karty, ktorých zoznamy majú daný názov.

6.4.4 Prepojenie klient-server pomocou socket.io

Pre synchronizáciu zmien z Trello do našej aplikácie je okrem funkcionality Trello Webhook využitá aj knižnica socket.io. Tá nám poslúži k zasielaniu prijatých udalostí z Trella klientovi.

Po prihlásení užívateľa klientská časť nadviaže spojenie so serverom využitím funkcie `socketIOClient`. Klientská časť sleduje dve udalosti:

- `connect` - udalosť spojenia klienta so serverom. Po nadviazaní spojenia so serverom vyvolá klient udalosť `authorization` a zašle serveru dáta v podobe identifikátoru prihláseného užívateľa.
- `update` - udalosť prijatia dát o zmene z Trella, ktoré klientovi zaslal server. Overuje sa, či je zmena naozaj adresovaná prihlásenému užívateľovi, ak áno je vyvolaná Redux akcia, ktorá dáta spracuje.

Server sleduje tri udalosti:

- `authorization` - klient zasiela identifikátor prihláseného užívateľa, ten sa dočasne uloží spolu s identifikátorom pridelenej socket.io schránky pomocou knižnice lowdb do súboru vo formáte JSON.
- `update` - je prijatá informácia o zmene z Trella. Na základe identifikátoru užívateľa, ktorému je zmena adresovaná z prijatých dát sa v dočasnom úložisku dohľadá identifikátor socket.io schránky. Ak je nájdený, tak sú dáta zaslané príslušnému klientovi.
- `disconnect` - klient ukončil spojenie. V dočasnom úložisku sa dohľadá záznam na základe identifikátoru schránky a je odstránený.

6.4.5 Synchronizácia zmien z Multiboard do Trello

V prípade vykonania zmeny v našej aplikácii je vyvolaná Redux akcia reprezentujúca danú zmenu, v ktorej je potom vykonaný požiadavok na úpravu v Trelle pomocou API. Požiadavky na API sú vykonávané pomocou globálneho objektu Trello sprístupneného cez `Client.js`. Nasledujúca ukážka demonštruje akcie volané pri zmene názvu karty v našej aplikácii.

```
export const actionCardUpdate = (card, data, updateLocally = false, dataToUpdateOnlyLocally
     = {}) => {
// Parametere:
//  card - objekt karty, ktor ma byt aktualizovany
//  data - objekt reperezentujuci dta, ktore sa zmenili
//  updateLocally - znaci, ci je potrebna okamzita aktualizacia aj lokalne ulozeneho
    objektu karty
//  dataToUpdateOnlyLocally - data navyse k~aktualizacii lokalne

    return (dispatch) => {
        if (updateLocally) {
            store.dispatch({
                type: CARD_UPDATE,
                payload: {
                      card,
                      data: { ...data ..dataToUpdateOnlyLocally }
                }
            });
            // Vykonanie zmeny lokalne
        }

        return window.Trello.put('/cards/' + card.id, { ...data },
            (response) => dispatch({ type: CARD_UPDATE_SUCCESS, payload: { response } }),
            // Aktualizacia cez API sa podarila
            (error) => dispatch({ type: CARD_UPDATE_FAILURE, payload: { card, error } }));
            // Nastala chyba, vratenie zmeny lokalne
    };
};
```

Výpis 6.2: Redux akcia vyvolaná pri zmene karty

Zmeny sú často vykonávané lokálne z dôvodu, aby užívateľ nemusel čakať na výsledok požiadavku na API a zmena bola hneď viditeľná. V prípade, že vznikne chyba, tak je objekt vrátený do stavu pred zmenou. Nakoľko Redux akcie majú štandardne vracať len objekty, pre komplexnejšie akcie ako v ukážke 6.2 bola použitá knižnica `redux-thunk`.

Umožňuje vykonávať asynchrónne Redux akcie. Dovoľuje taktiež vyvolávať v akcii iné akcie, či pristupovať k stavu aplikácie, čo štandardne nie je možné.

6.4.6 Synchronizácia zmien z Trello do Multiboard

Ako je popísané v podkapitole 6.4.4, klient po prijatí dát o zmene vyvolá Redux akciu, ktorá zmeny spracováva. Tá podľa typu prijatej akcie dáta spracováva. Ukážka 3.4 znázorňuje zjednodušenú štruktúru prijatých dát. Nasledujúca ukážka demonštruje spracovanie akcie typu `updateCard`, ktorá aktualizuje kartu.

```
export const actionUpdateReceived = (update) => {
// Parameter update obsahuje prijate data o~zmene
    switch (update.action.type) {
        case "updateData": {
            return {
                type: WEBHOOK_UPDATE_CARD,
                payload: {
                    old: update.action.data.old,
                    // Obsahuje kluce a hodnoty, ktore boli zmenene
                    card: update.action.data.card,
                    // Obsahuje zjednoduseny objekt karty spolu s~novymi hodnotami
                }
            }
        }
        ...
    }
}
```

Výpis 6.3: Spracovanie akcie typu updateCard

Následne je vyvolaný Redux reducer:

```
const dataReducer = (state = initialState, action) => {
// Parametre:
//  state - obsahuje momentalny stav aplikacie
//  action - typ spracovavanej Redux akcie
    const payload = action.payload;

    switch (action.type): {
        case WEBHOOK_UPDATE_CARD: {
            const { old, card } = payload;

            const cards = [...state.cards];

            cards.map((c) => {
                if (c.id === card.id) {
                // Najdenie upravovanej karty
                    Object.keys(old).forEach((key) => {
                    // Iteracia cez kluce, ktorch hodnoty boli zmenene a nahradenie hodnot
    novymi
                        c[key] = card[key];
                    }
                }
            });

            return {
                ...state,
                cards
            };
        }
        ...
    }
}
```

Výpis 6.4: Reducer vykonávajúci zmenu updateCard

Podobný systém je aplikovaný pre všetky prijaté akcie. To umožňuje vykonanie zmien ihneď po ich prijatí bez nutnosti a prekreslenie len súvisiacich častí aplikácie bez nutnosti znova načítania aplikácie.

6.4.7 Implementácia užívateľského rozhrania

Prvky užívateľského rozhrania boli implementované využitím viacerých knižníc:

- `react-beautiful-dnd` - knižnica pre `drag  drop`, využitá pri implementácii presúvania kariet a zoznamov.
- `react-day-picker` - knižnica pre kalendár využitý pri voľbe termínu karty.
- `react-icons` - knižnica obsahujúca množstvo ikon zadarmo.
- `styled-components` - zjednodušuje písanie štýlov CSS pre komponenty a JSX elementy.
- `material-ui` - framework s už hotovými komponentmi pre užívateľské rozhrania.

6.5 Testovanie

Testovanie slúži k odhaleniu chýb a získaniu spätnej väzby od požívateľov aplikácie. Podkapitola sa najprv venuje vyhodnoteniu testovania funkcionality aplikácie a neskôr užívateľského rozhrania.

6.5.1 Priebeh a vyhodnotenie

Po vytvorení prototypu aplikácie obsahujúceho len niekoľko málo funkcií, bola aplikácia nasadená a sprístupnená verejne. V prvej iterácii, aplikácia poskytovala užívateľom len časť funkcionality:

- Prihlásenie.
- Zobrazenie zoznamov a kariet.
- Pridávanie kariet a zoznamov.
- Presúvanie zoznamov v rámci nástenky.
- Presúvanie kariet v rámci zoznamu alebo medzi nimi.
- Zobrazenie detailu karty.
- Implementácia len niektorých Webhook akcií.

Aplikácia a informácie o nej boli zdieľané na vybraných sociálnych sieťach a niekoľkým známym. Testovanie prebehlo voľnou formou. Spätná väzba bola podaná vo forme komentárov a súkromných správ. Týkali sa primárne opravy chýb všeobecných, či nezrovnalostí v užívateľskom rozhraní a funkcionality, ktorá by mala byť do ďalšej verzie pridaná.

Na základe spätnej väzby bolo opravené množstvo chýb a pridaná nová funkcionalita:

- Zjednodušené prihlásenie kliknutím na jedno tlačítko.
- Pridaná možnosť editácie kariet, pridanie štítkov a ich úprava, termínov, pridanie To-Do zoznamov a ich úprava.
- Možnosť karty kopírovať, archivovať a mazať.
- Implementácia väčšiny Webhook akcií.
- Možnosť výberu, ktoré nástenky sa agregujú.

Rozloženie a prístupnosť prvkov užívateľského rozhrania bola užívateľom známa zo samotného Trella a preto boli s užívateľským rozhraním spokojní.

Spätná väzba obsahovala aj podnety k pridaniu extra funkcionality, ktorú samotné Trello zatiaľ neponúka.

Kapitola 7

Záver

Cieľom práce bolo vytvoriť službu, ktorá agreguje viacero Trello násteniek do jednej. Služba mala taktiež ponúkať obojsmernú synchronizáciu zmien. Pre realizáciu som si vybral službu implementovať vo forme webovej aplikácie využitím knižníc a frameworkov React pre klientskú časť aplikácie, ktorý ponúka možnosti modernej tvorby užívateľských rozhraní a pre serverovú časť Node.js z dôvodu povahy požadovanej funkcionality.

7.1 Zhodnotenie výsledku práce

Podľa môjho osobného názoru aplikácia splnila požiadavky stanovené na začiatku. Jediným väčším problémom, ktorý sa počas implementácie vyskytol bol spôsob, akým môže naša aplikácia prijímať a spracovávať dáta o zmenách z aplikácie Trello. Riešenie nakoniec poskytla knižnica socket.io.

Som rád, že som mohol vytvoriť aplikáciu, ktorá má potenciál skutočného využitia pre určitú skupinu ľudí. Technológie využité pri tvorbe finálneho produktu ako aj Trello API, ktoré umožnilo existenciu výslednej aplikácie poskytovali množstvo zdrojov a skvelú dokumentáciu, čo značne uľahčilo prácu na aplikácii.

7.2 Návrh na ďalší vývoj

Myslím si, že výsledná aplikácia plní požadovanú funkcionalitu. Kladie sa však otázka, či aplikácia poskytuje dostatok funkcionality samotnej služby Trello, aby mohla byť používaná aj ako náhrada a nie len ako rozšírenie, či doplnok.

Ponúka sa taktiež návrh a implementácia nového grafického užívateľského rozhrania, či pridanie úplne novej funkcionality, ktorú Trello v súčasnosti neposkytuje.

Aplikácia bola vyvíjaná vo forme webovej aplikácie, no bolo by možné jej nasadenie aj formou mobilnej aplikácie, napríklad využitím frameworku React-Native.

Literatúra

[1] *What is Kanban?* [online]. [cit. 17.2.2020]. Dostupné z: https://www.digite.com/kanban/what-is-kanban/.

[2] *Redux and React: An Introduction* [online]. 18.11.2017 [cit. 25.3.2020]. Dostupné z: https://jakesidsmith.com/blog/post/2017-11-18-redux-and-react-an-introduction/.

[3] *Redux - Three Principles* [online]. 2015-2020 [cit. 24.3.2020]. Dostupné z: https://redux.js.org/introduction/three-principles.

[4] *Using middleware* [online]. 2017 [cit. 20.3.2020]. Dostupné z: https://expressjs.com/en/guide/using-middleware.html.

[5] *API Introduction* [online]. 2019 [cit. 12.3.2020]. Dostupné z: https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/.

[6] *Introducing JSX* [online]. 2020 [cit. 21.3.2020]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[7] *Node.js - Introduction* [online]. 2020 [cit. 20.3.2020]. Dostupné z: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm.

[8] *Master the JavaScript Interview: What is a Pure Function?* [online]. 25.3.2016 [cit. 24.3.2020]. Dostupné z: https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-pure-function-d1c076bec976.

[9] Falkowitz, R. *Psychology, Flow and Kanban* [online]. 2014 [cit. 17.2.2020]. Dostupné z: https://www.3cs.ch/psychology_flow_and_kanban/.

[10] Radigan, D. *Kanban - How the kanban methodology applies to software development* [online]. [cit. 17.2.2020]. Dostupné z: https://www.atlassian.com/agile/kanban.

Príloha A

Obsah priloženého CD

CD obsahuje nasledujúce položky:

- `multiboard-for-trello` - zdrojové kódy
- `video.mp4` - demonštračné video
- `plagat.pdf` - plagát formátu A2
- `bp-source` - zdrojový kód práce
- `bd.pdf` - PDF súbor práce

Súbor `README.md` v adresári `multiboard-for-trello` obsahuje informácie o aplikácii a jej spustení.