VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÝ KANBAN

WEB KANBAN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ FURDA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Furda Jiří**
Program: Informační technologie
Název: **Webový kanban**
Web Kanban
Kategorie: Informační systémy

Zadání:

1. Prostudujte současné technologie pro vývoj webových informačních systémů se zaměřením na klientské aplikace v jazyce JavaScript.
2. Seznamte se s požadavky zadavatele na webovou aplikaci typu kanban. Zmapujte existující infrastrukturu zadavatele a existující řešení v této oblasti.
3. Po konzultaci s vedoucím navrhněte klient-server architekturu aplikace na základě analýzy požadavků.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte testování aplikace ve spolupráci se zadavatelem.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

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

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 21. října 2019

Abstrakt

Tato bakalářská práce se zabývá vytvořením webové aplikace pro řízení projektů za pomocí agilní metodiky Kanban. Aplikace musí být implementovaná dle požadavků společnosti SEACOMP s.r.o., která ji bude využívat. Teoretická část práce se zaměřuje na problematiku projektového řízení a popisuje technologie použité při vývoji aplikace. Praktická část práce obsahuje popis návrhu aplikace a konkrétních částí její implementace. Výsledná aplikace byla otestována zaměstnanci společnosti a zhodnocena v závěru práce.

Abstract

This bachelor thesis deals with the creation of a web application for project management using the agile Kanban methodology. The application must be implemented according to the requirements of SEACOMP s.r.o., which will use it. The theoretical part of the thesis focuses on the issue of project management and describes the technologies used in the development of the application. The practical part of the thesis contains a description of the application design and specific parts of its implementation. The resulting application was tested by the company's employees and evaluated at the end of the thesis.

Klíčová slova

Kanban, řízení projektů, agilní metodiky, web, Angular, ASP.NET, JavaScript, C#

Keywords

Kanban, project management, agile methods, web, Angular, ASP.NET, JavaScript, C#

Citace

FURDA, Jiří. *Webový kanban*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Webový kanban

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Další informace mi poskytl pan Ing. Pavel Vyskočil. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Furda
28. května 2020

Poděkování

Děkuji mému vedoucímu panu Ing. Radkovi Burgetovi, Ph.D. a konzultantovi panu Ing. Pavlovi Vyskočilovi za odborné vedení a cenné rady během řešení této bakalářské práce. Dále bych chtěl poděkovat panu Mgr. Davidu Bryšovi, za umožnění pracovat na firemním zadání diplomové práce a všem svým kolegům, kteří se zapojili do testování aplikace.

Obsah

Kapitola 1

Úvod

V každé výrobě je přirozeně snaha o co největší efektivitu. Konkrétně v oblasti informačních technologií je pro tento účel velice rozšířené použití agilních metodik řízení projektů. Jedním z nástrojů těchto metodik je metodika Kanban.

Cílem této bakalářské práce je vytvořit webovou aplikaci typu Kanban určenou pro řízení projektů na základě požadavků společnosti SEACOMP s.r.o. Za tímto účelem je společností v současné době využíván nástroj třetí strany, který ovšem není plně přizpůsoben potřebám této společnosti, ku příkladu není integrován do dalších interních aplikací.

Téma práce mne zaujalo zejména díky stále rostoucí popularitě interaktivních webových aplikací. Motivující pro mne bylo mít možnost takovou aplikaci implementovat naprosto sám. Také mne lákala možnost firemního zadání bakalářské práce, díky kterému bude mít výsledná aplikace reálné využití v praxi a její existence neskončí dnem odevzdání.

Tato práce částečně navazuje na diplomovou práci mého bývalého kolegy Bc. Adama Teršla, jež nesla název *Webový klient pro nemocniční systém* [49]. Výsledkem této práce je aplikace SSB4Web, do které bude výsledek mé bakalářské práce v budoucnu integrován.

Druhá kapitola práce, zvaná „Rozbor řešené problematiky“, pojednává o historii a současných trendech projektového řízení, zejména pak o metodice Kanban. Poté následuje přehled několika nástrojů určených pro aplikaci této metodiky a jejich srovnání. Dále je pár slov věnováno zadavateli této diplomové práce – společnosti SEACOMP s.r.o., infrastruktuře této společnosti a požadavkům na výslednou aplikaci. Další kapitola, označená „Technologie“, obsahuje stručný přehled technologií a principů využitých při implementaci této aplikace. Následující kapitola s názvem „Návrh aplikace“ se zaměřuje jak na grafický, tak i technický návrh aplikace. Popisuje rozvržení databáze, provázanost částí systému a podobu aplikačního rozhraní. Pátá kapitola jménem „Implementace aplikace“, se pak věnuje zajímavým částem řešení této bakalářské práce. Kapitola „Testování a vyhodnocení“ popisuje formu testování výsledné aplikace a jeho výsledky. Na závěr je poté shrnuto dosažení výsledků této bakalářské práce.

Kapitola 2

Rozbor řešené problematiky

V úvodu této kapitoly jsou stručně shrnuty populární metodiky řízení projektů s větším zaměřením na metodiky agilní a to především na metodiku Kanban. V další části kapitoly je představeno několik již existujících aplikací umožňujících využívat právě tuto zmíněnou metodiku. Dále je text věnován zmapování infrastruktury zadavatele bakalářské práce a jeho požadavkům na výslednou aplikaci. Závěr kapitoly obsahuje popis technologií, které byly využity pro tvorbu požadované aplikace.

2.1 Metodiky řízení projektů

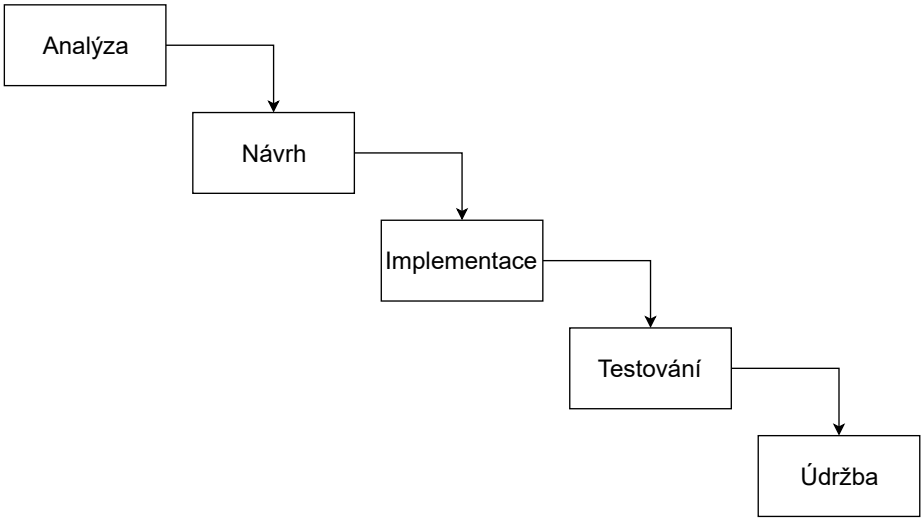
Obecně lze metodiky používané pro řízení softwarových projektů rozdělit do dvou hlavních kategorií. Jedná se o tradiční metodiky a metodiky agilní.

Jak název napovídá, tradiční metodiky mají delší historii. Lidstvo potřebovalo využívat řízení projektů již odpradávna, exemplárním příkladem jsou obrovské stavební projekty jako pyramidy v Gíze nebo Velká čínská zeď. Tyto projekty jistě vyžadovaly důkladné řízení jak pracovních sil, tak i přísunu materiálu a dalších zdrojů. Výsledek musel bezpochyby splňovat určité požadavky. Ohledně metodik podobného plánování ovšem neexistuje příliš mnoho dokumentace. K určité standardizaci řízení projektů totiž začalo docházet až v padesátých letech minulého století. Svou zásluhu zde má i Námořnictvo Spojených států amerických a to konkrétně Projekt Manhattan[1] [45].

Rozdílem mezi dvěma zmíněnými kategoriemi řízení projektů je především strukturace procesů. U tradičních metod probíhají tyto procesy sekvenčně. Tento přístup zpravidla vyžaduje důkladnější počáteční plánování a pozdější změny požadavků mnohdy představují náročný problém. Oproti tomu při použití agilních metod jsou jednotlivé procesy úzce spjaty a pracuje se v iteracích. Díky tomu je do vývoje více zapojen zákazník a takový projekt se snadněji vypořádá s případnými změnami požadavků [10].

Populárním zástupcem tradičních metodik je například vodopádový model (znázorněný diagramem 2.1), na kterém je na první pohled patrné právě sekvenční řazení jednotlivých procesů vývoje. Tento model se objevil koncem šedesátých let minulého století, kdy byly programovací jazyky neefektivní, procesory pomalé a paměť počítačů byla výrazně omezená [25].

[1] Krycí název pro projekt vývoje atomové bomby během druhé světové války



Obrázek 2.1: Grafické znázornění posloupnosti jednotlivých procesů tradiční metodiky řízení projektů označované jako vodopádový model.

U tradičních metodik je předpokladem, že se k dokončeným fázím projektu již nebude nutné vracet. Takový předpoklad lze aplikovat na mnoho odvětví. Příkladem budiž stavba budovy, u které je nutné nejprve řádně celý projekt naplánovat a zdokumentovat. Jakmile samotná stavba započne, větší změny oproti původnímu projektu vznikají velmi zřídka. V oblasti informačních technologií, obzvláště při vývoji softwaru, však odchýlení od původního plánu nebývá žádnou výjimkou. Řízení projektů tohoto typu pomocí striktních tradičních metodik je tedy často obtížné a značně neefektivní [41].

V dnešní době se stává častěji než tomu bylo v minulosti, že se softwarové projekty během vývoje mění, narůstají jejich požadavky, přibývají funkce a to vše i během jejich vývoje. Tento trend se týká především webových systémů. Jako reakce na tyto náhlé změny byly v devadesátých letech stvořeny právě agilní metodiky [25].

2.1.1 Agilní metodiky řízení projektů

Agilní metodiky jsou skupina metod řízení projektů určených především pro vývoj softwaru. Jak název napovídá, jedná se o metodiky, které se vyznačují dynamičností a flexibilitou.

Veřejný zájem o tyto metodiky začal až v pozdních devadesátých letech [18]. V roce 2001 se v Utahu sešlo sedmnáct zastánců agilních metodik a sestavilo Manifest agilního vývoje software[2], který vystihuje čtyři hlavní zásady těchto metod [11].

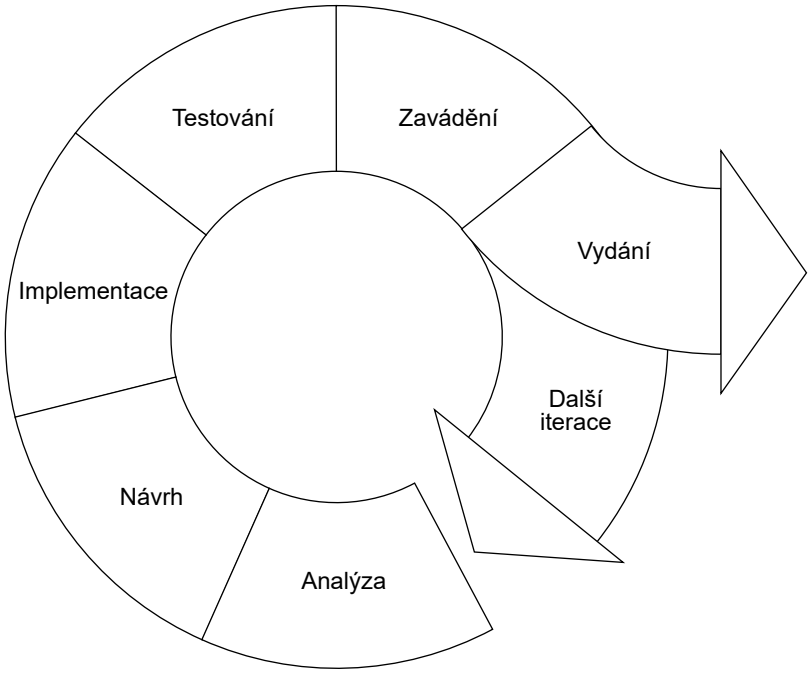
- Jednotlivci a interakce před procesy a nástroji
- Fungující software před vyčerpávající dokumentací
- Spolupráce se zákazníkem před vyjednáváním o smlouvě
- Reagování na změny před dodržováním plánu

Tyto metodiky jsou více zaměřené na lidi a na jejich vzájemnou komunikaci. To je jedním z důvodů, proč jsou agilní metodiky vhodné především pro menší týmy. Díky absenci

[2] Manifest agilního vývoje: https://agilemanifesto.org/iso/cs/manifesto.html

sekvenčního řazení procesů vývoje tyto metody také zvyšují efektivitu nepředvídatelných a stále se měnících projektů. Vypuštěním striktního plánování je také možné projekt mnohdy dokončit rychleji než použitím tradičních metodik.

Vývoj řízený agilními metodikami probíhá iterativně a inkrementálně. Na začátku vývoje se určí základní požadavky a postupnými cykly se tyto požadavky zdokonalují. Jedna iterace takového vývoje trvá zpravidla jeden týden až měsíc. Během této iterace proběhne celý cyklus vývojových procesů (podobný vodopádovému modelu znázorněného diagramem 2.1). Dokončením iterace vzniká prototyp, který může být prezentován zákazníkovi. Díky tomu zákazník získává přesnější představu o podobě produktu a lze včas odhalit nedostatky požadavků, které dříve nebyly známy [29]. Průběh tohoto iterativního vývoje je znázorněný diagramem 2.2.



Obrázek 2.2: Grafické znázornění posloupnosti jednotlivých procesů agilních metodik řízení projektů. Cyklus začíná procesem analýza a pokračuje až po proces zavádění, kdy je projekt buďto hotov, a následuje jeho vydaní, nebo proběhne další iterace a celý cyklus se znovu opakuje.

Mezi populární agilní metodiky patří například Scrum, Lean Developmet, Extrémní programování (angl. *Extreme Programming* nebo také XP), Crystal, Vlastnostmi řízený vývoj (angl. *Feature Driven Development* neboli FDD) a v neposlední řadě také Kanban.

Jedna ze studií na toto téma vydána roku 2015 uvádí, že ze 1386 sledovaných projektů 65% z nich využilo během vývoje agilní metody a 6% projektů bylo dokončeno téměř pouze s použití agilních metod. Závěrem této studie je také pozorování poukazující na fakt, že čím větší poměr agilních metodik byl při vývoji použit, tím úspěšnější dané projekty byly [44].

2.1.2 Metodika Kanban

Kanban je slovo převzaté z japonského jazyka, kde znamená „cedule“. Toto slovo je taktéž spjato s poslední výzvou k objednávce před uzavřením podniku (sundáním cedule). Takovou

objednávku lze označit jako „na poslední chvíli“ nebo také jako objednávku „tak akorát na čas“ [37].

Tak akorát na čas (angl. *Just-in-time*) je filozofie řízení výroby, vyvinuta v sedmdesátých létech japonskou automobilovou společností Toyota Motor Corporation[3]. Hlavní myšlenkou toho přístupu je mít přesné množství materiálu, na správném místě a ve správný čas. Díky tomu dochází k redukci množství materiálu, který aktuálně není zpracováván a nemělo by docházet k jeho přebytečnému hromadění [13]. Jedním z nástrojů této filozofie je právě metodika Kanban, její tehdejší provedení je možné vidět na fotografii 2.3.

Obrázek 2.3: Původní provedení nástěnky Kanban v továrně společnosti Toyota Motor Corporation. *Převzato z [53].*

Metodika Kanban je nástroj pro vizuální kontrolu práce a jejího toku jednotlivými fázemi procesu. Kanban může být realizován jak fyzicky (například tabule s nalepovacími papírky viz fotografie 2.4), tak i digitálně (například pomocí aplikací uvedených v kapitole 2.2).

Ať je nástěnka Kanban digitální či nikoliv, ve většině případů se skládá ze stejných částí, jen v jiném provedení. Nejčastěji se lze setkat s typem, kdy se na horizontální ose nacházejí pojmenované sloupce představující jednotlivé fáze procesu. Do těchto sloupců jsou umístěny nejrůznější karty, reprezentující jednotlivé úkoly. Rozmístění karet mezi sloupci se odvíjí od aktuálního stavu úkolu.

Nejjednodušší variantou takových sloupců může být například trojcie sloupců nazvaných „zásobník“, „v procesu“ a „hotovo“. První ze sloupců nazvaný „zásobník“ je místo, kde se hromadí úkoly, na kterých se ještě nezačalo pracovat. Jakmile práce na daném úkolu započne, jeho karta je přesunuta do dalšího sloupce s označením „v procesu“. Obdobně je

[3]Toyota Motor Corporation: https://global.toyota/en/

tomu při dokončení úkolu, kdy je jeho karta opět přemístěna do jiného sloupce, tentokrát do sloupce „hotovo“. Názorná ukázka této nástěnky je vidět na fotografii 2.4.

Obrázek 2.4: Nástěnka typu Kanban realizována pomocí nalepovacích papírků na magnetické tabuli.

V praxi je takřka nemožné setkat se s takto jednoduchým provedením. Proces výroby či vývoje je třeba rozdělit na jednotlivé části. Při vývoji softwaru bývají často využívány sloupce „návrh“, „vývoj“, „testování“ a další.

Pokud jednu nástěnku využívá více oddělení společnosti, lze se mnohdy setkat i s dalším rozlišením úkolů. Jedná se o takzvané plavecké dráhy (angl. *swimlanes*). Úkoly jsou pak rozděleny nejen do sloupců značících procesy, ale i do vodorovných řádků, které nejčastěji představují jednotlivá oddělení společnosti či týmy.

V dnešní době, jak i u mnoha jiných věcí, převažuje digitální provedení Kanban nástěnek. Její výhodou je především větší přehlednost, vzdálená přístupnost, jednoduší manipulace a hlavně možnost připojovat k jednotlivým kartám podrobnější popis, komentáře, obrázky a další dokumenty. Nicméně i fyzická varianta má své výhody. Jedna z nich je fakt, že je stále na očích a nelze ji skrýt jako například záložku v prohlížeči. Toto provedení může taktéž podněcovat k vetší komunikaci mezi týmy a to jsou důvody proč tuto variantu využívají například ve společnosti Optimizely[4] [2].

Díky využití metodiky Kanban a jejího přehlednému vyobrazení úkolů získávají jednotliví členové týmu lepší přehled o prací svých kolegů. Tento fakt může také vést k lepšímu

[4]Optimizely: https://www.optimizely.com/about/

určení priority úkolů a především odpadá čas ztracený komunikací v momentě, kdy nastane změna úkolu či jeho priority [31]. Kanban svou podstatou také dokáže poukázat na problematické procesy, které zpomalují celkový chod výroby či vývoje [27].

Často se u metodiky Kanban lze potkat s limitováním probíhající práce (anglicky *Work in Progress* či zkráceně WIP). Jedná se o omezení, kdy je dán maximální počet úkolů pro určitou fázi procesu (sloupce nástěnky). Například lze stanovit limit souběžného pracování maximálně na třech grafických návrzích. Toto omezení má pozitivní dopad především na snížení nutnosti věnovat se více úkolům naráz, díky čemuž se lze na jednotlivé úkoly lépe soustředit a tím může být docíleno rychlejšího a kvalitnějšího výsledku [31].

2.2 Aplikace s podobným zaměřením

Jelikož metodika Kanban existuje již několik desítek let, lze v dnešní době nalézt celou řadu programů a webových aplikací, které je možné pro aplikování této metodiky využít. V této podkapitole je několik takových aplikací stručně přiblíženo a jsou vyzdviženy některé jejich klíčové vlastnosti. Pro bližší představu o podobě těchto aplikací je každá doprovázena jejím snímkem. Mezi uvedenými aplikacemi lze nalézt dva populární zástupce z řad produktů společnosti Atlassian – aplikace Trello a Jira. Třetí představenou aplikací je o něco méně známá aplikace Kanboard, která je však v současné době využívaná společností SEACOMP s.r.o. V závěru kapitoly je obsaženo stručné shrnutí rozdílů těchto tří aplikací a možné důvody vedoucí ke zvolení právě jedné z uvedených aplikací.

2.2.1 Trello

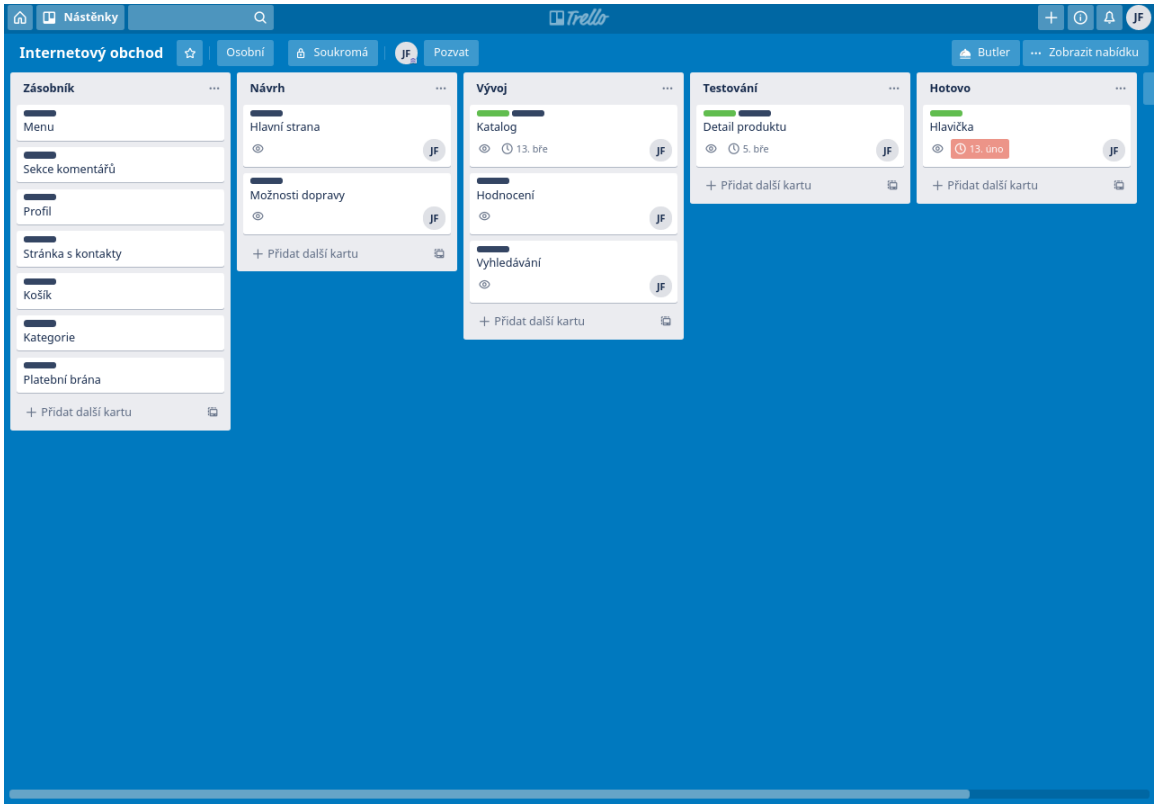
Trello[5] je především webová aplikace provozovaná v současné době společností Atlassian[6]. Historie této aplikace sahá až do roku 2011, kdy byla veřejně spuštěna jako webová aplikace a také jako aplikace pro platformu iOS. K dnešnímu dni je aplikaci Trello možné navíc využít i na mobilní platformě Android nebo jako desktopovou aplikaci pro operační systémy Microsoft Windows či macOS.

Během několika let své existence se aplikace stala velice populární a použilo ji několik milionů uživatelů. K věhlasným společnostem, které tuto aplikaci využívají patří například technologický gigant Google[7], softwarová společnost Red Hat[8] nebo platforma pro skupinové financování Kickstarter[9] [7].

Svou velkou popularitu si aplikace zasloužila především přívětivým uživatelským rozhraním a jednoduchostí použití. Od první návštěvy stránky k práci na své první Kanban nástěnce uživatele dělí doslova minuty. Pro základní použití navíc bez problému vystačí bezplatná verze aplikace. Díky těmto faktům aplikace není využívána pouze softwarovými společnostmi, ale lze ji použít například i jako osobní seznam úkolů [3].

Trello nabízí řadu vylepšení, díky kterým je možné nástěnku rozšířit o funkce aplikací třetích stran. V bezplatné verzi lze však využít pouze jedno takové vylepšení [8]. Nástěnka je umístěna na serverech poskytovatele a službu nelze provozovat na svém vlastním serveru. Nicméně mobilní aplikace umožňují pracovat s nástěnkami i bez přístupu k internetu [36].

[5] Trello: https://trello.com
[6] Atlassian: https://www.atlassian.com
[7] Google: https://about.google/
[8] RedHat: https://www.redhat.com/en/about/company
[9] Atlassian: https://www.kickstarter.com/about



Obrázek 2.5: Vzorová Kanban nástěnka ve webové aplikaci Trello zaměřená na vývoj internetového obchodu.

2.2.2 Jira

Jira[10] je řada produktů společnosti Atlassian, určená ke správě práce v nejrůznějších typech týmů. Nástěnka Kanban je součást například produktu Jira Software, který je, jak název vypovídá, zaměřen především na vývojáře softwaru.

S nástěnkou lze pracovat jak v prohlížeči, tak i pomocí vlastního programu, který je k dispozici pro operační systémy Microsoft Windows, Linux i macOS. K dispozici jsou také mobilní aplikace pro operační systémy iOS i Android.

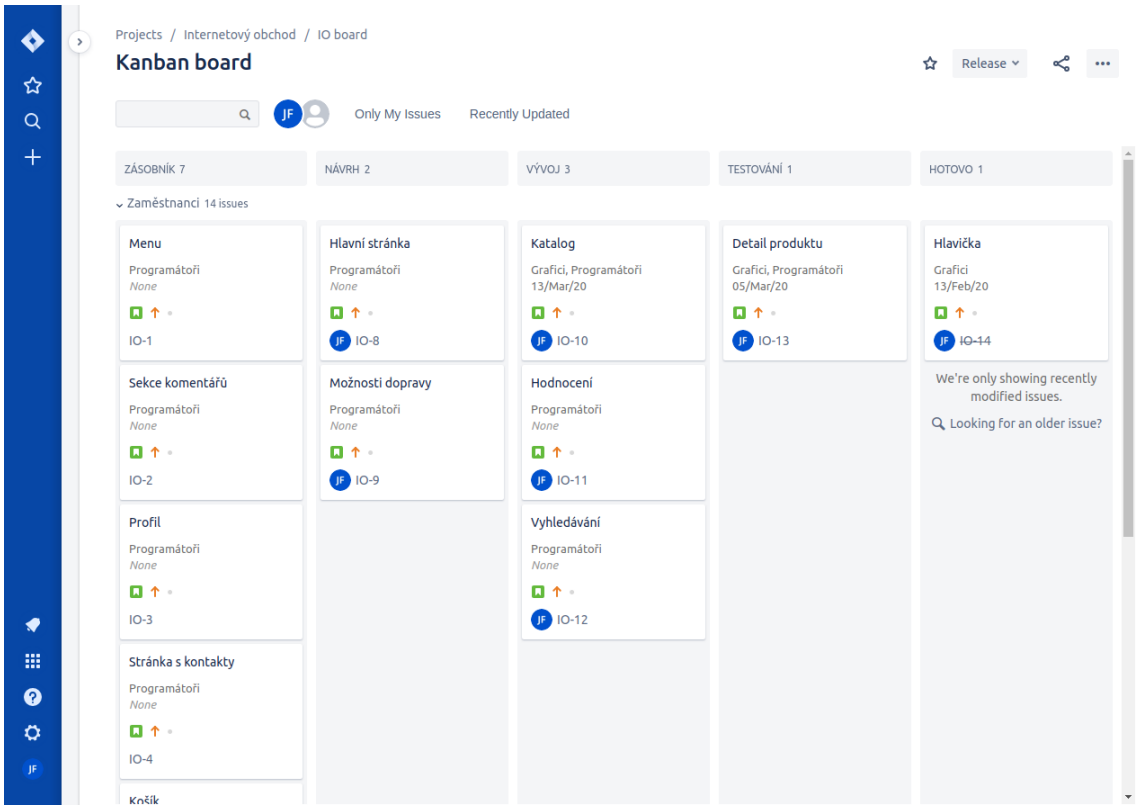
Jednou z funkcí, kterou Jira disponuje, je možnost vygenerování řady grafů a přehledů. V základní verzi bez přídavných modulů lze také tvořit dílčí úkoly a vzájemné závislosti jednotlivých úkolů. Základní funkcionalitu aplikace lze rozšířit využitím placených i bezplatných doplňků, které jsou k dispozici na oficiálním tržišti.

Díky tomu, že je aplikace určená pro softwarové vývojáře je v této aplikaci mimo klasické nástěnky možné také přepnout režim zobrazení na variantu určenou pro agilní metodiku zvanou Scrum, která cílí na dokončení úkolů v rámci iterace cyklu vývoje [28].

Nástěnka může být na rozdíl od aplikace Trello umístěna jak na serverech poskytovatele, tak i na vlastním serveru.

[10] Jira: https://www.atlassian.com/cs/software/jira

Tato aplikace je využívána více něž 65 tisíci zákazníky po celém světě mezi které se řadí například internetovou aukční síň eBay[11], služba pro streamování hudby Spotify[12], společnost vyrabějící síťové prvky Cisco[13] nebo aplikace pro pronájem ubytování Airbnb[14] [4].



Obrázek 2.6: Vzorová Kanban nástěnka ve webové aplikaci Jira Software zaměřená na vývoj internetového obchodu.

2.2.3 Kanboard

Kanboard[15] je bezplatná webová aplikace s otevřeným zdrojovým kódem. Jejím tvůrcem je Frédéric Guillot, nicméně zdrojové kódy jsou volně k dispozici na portálu GitHub[16], kde se k vývoji připojilo dalších 270 lidí [22]. Aplikace je napsaná především v jazyce PHP a JavaScript. K provozování této aplikace je nutné použít vlastní webový server. Kanboard uživatelům nenabízí žádné přepychové uživatelské rozhraní, ale naopak se zaměřuje na jednoduchost a minimalismus [23].

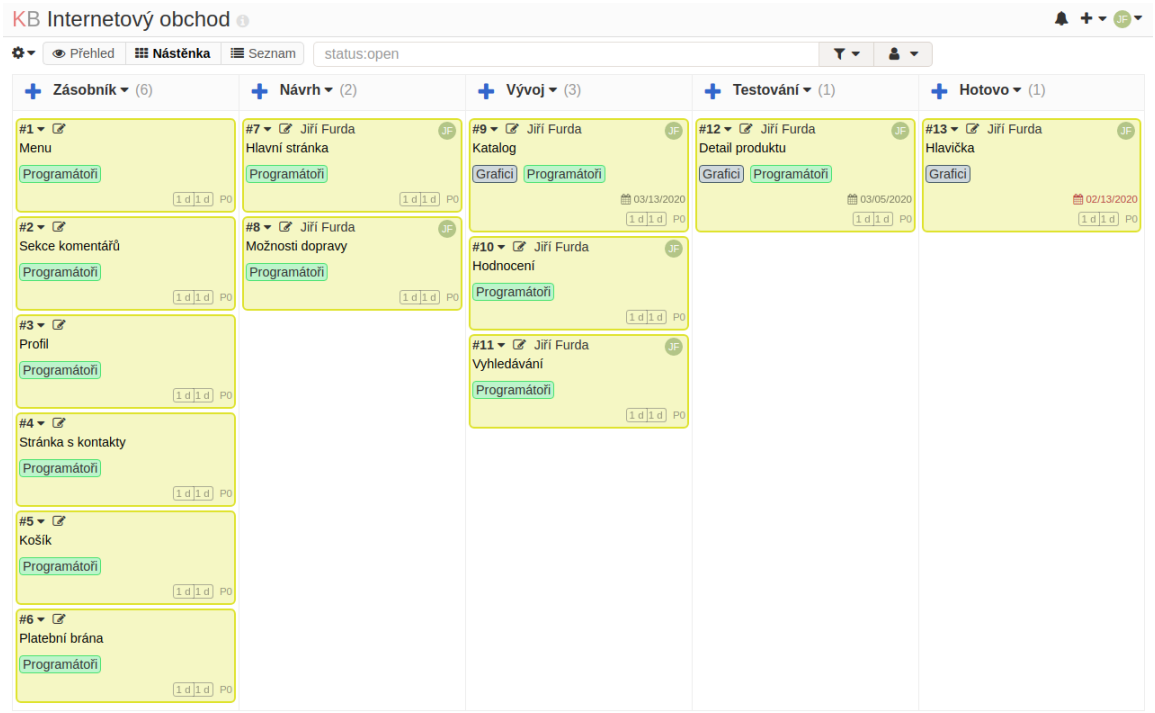
Mimo základní funkce metodiky Kanban tato aplikace umožňuje také vyhledávat úkoly za pomocí vlastního dotazovacího jazyka. Částečně také umožňuje práci s nástěnkou automatizovat pomocí konfigurovatelných akcí.

Vzhledem k otevřenému zdrojovému kódu lze aplikaci v případě potřeby upravit na míru vlastním potřebám. Součástí aplikace je však i rozhraní pro zásuvné moduly. Něko-

[11] eBay: https://www.ebayinc.com/company/
[12] Spotify: https://newsroom.spotify.com/company-info/
[13] Cisco: https://www.cisco.com/c/en_uk/index.html
[14] Airbnb: https://news.airbnb.com/about-us/
[15] Kanboard: https://kanboard.org
[16] Repozitář Kanboard na platformě GitHub: https://github.com/kanboard/kanboard

lik takových modulů lze nalézt i přímo na oficiální stránce[17], avšak nepodléhají žádnému schvalovacímu procesu a není zaručena jejich kompatibilita.

Aplikace však nenabízí žádný program či mobilní aplikaci. Jediný přístup k nástěnce je tedy pomocí internetového prohlížeče.



Obrázek 2.7: Vzorová Kanban nástěnka ve webové aplikaci Kanboard zaměřená na vývoj internetového obchodu.

2.2.4 Souhrn rozdílů

Základní principy nástěnky Kanban jsou součástí všech tří uvedených aplikací. Malou odchylkou je aplikace Trello, která neumožňuje úkoly rozřadit do plaveckých drah. Této funkcionality lze docílit pouze instalací zásuvného modulu do prohlížeče, což zcela jistě není nejvhodnější řešení.

Představené aplikace se mezi sebou liší například cenou, možnostmi provozování, dostupností zdrojového kódu, existencí mobilní aplikace, dostupností technické podpory či přívětivostí uživatelského prostředí. Srovnání několik těchto vlastností je přehledně uvedeno v tabulce 2.1.

Tabulka 2.1: Srovnání rozdílů mezi několika aplikacemi typu Kanban

Aplikace	Cena[18]	Vzdálené úložiště	Vlastní úložiště	Otevřený zdrojový kód
Trello	0–20,83 dolarů [8]	Ano	Ne	Ne
Jira	0–14 dolarů [5]	Ano	Ano	Ne
Kanboard	Zdarma	Ne	Ano	Ano

[17] Zásuvné moduly pro Kanboard: https://kanboard.org/plugins.html
[18] Cena na uživatele za měsíc

Z důvodu ochrany firemního tajemství mohou být preferovány služby Jira a Kanboard, díky možnosti provozovat je přímo na svém vlastním serveru. Konkrétně u aplikace Kanboard může tato vlastnost však představovat pro řadu uživatelů nevýhodu. Aplikaci totiž není možné jednoduše provozovat na serverech poskytovatele. Uživatel musí mít k dispozici server, kde bude aplikace provozována a to se může stát bariérou pro méně pokročilé uživatele počítače.

Kanboard však jako jediná z tří uvedených aplikací disponuje otevřeným zdrojovým kódem. Umožňuje tak větší zásahy do aplikace než je tomu v případě dalších dvou aplikací, kde lze úpravy řešit pouze pomocí doplňků.

Co se ceny týče, všechny aplikace lze provozovat v jistém měřítku zdarma. Trello v bezplatné verzi omezuje počet týmových nástěnek a Jira zase využívá limit uživatelů. Oproti tomu Kanboard, je zcela zdarma. Jediné reálné náklady tedy může představovat provoz serveru. Při provozování aplikace Jira tímto způsobem je třeba uhradit jednorázový poplatek, jehož cena se odvíjí od počtu uživatelů. Použití aplikace Kanboard se tedy může jevit jako nejlepší volba, každopádně v tomto případě je ztracena dostupnost technické podpory. V korporátním prostředí to může být důležitý požadavek, protože výpadek klíčového nástroje pro řízení projektů představuje obrovské riziko pro bezproblémový chod společnosti.

Slabší stránkou aplikace Kanboard je na první pohled uživatelské rozhraní. Nejedná se však pouze o vzhled aplikace. Rozhraní obsahuje i mnoho prvků, které nejsou řešeny pomocí asynchronních požadavků a to může být uživateli vnímáno jako nepohodlné či zastaralé.

Každá z uvedených aplikací tedy obsahuje řadu výhod i nevýhod, nelze tedy jednoznačně určit tu nejlepší. Výběr závisí na konkrétních požadavcích společnosti či týmu, který ji bude využívat. Nicméně za zmínku stojí fakt, že 83% společností ze žebříčku Fortune 500[19], který sdružuje 500 společností ze Spojených států Amerických s nejvyšším hrubým ročním obratem, využívá produkty právě společnosti Atlassian [6].

2.3 Společnost SEACOMP s.r.o.

SEACOMP s.r.o. je česká společnost věnující se již více než dvacet let tvorbě softwarových produktů. Společnost se zabývá především informačními systémy v oblasti logistiky, zdravotnictví, správy budov a řízení projektů [43].

2.3.1 Současný produkt

Jedním z mnoha produktů společnosti SEACOMP s.r.o. je systém SSB[20], určený pro snadnou tvorbu komplexních informačních systémů. Jedná se o počítačový program napsaný za použití technologie .NET.

V loňském roce se v rámci své diplomové práce Bc. Adam Teršl věnoval implementaci webového klienta právě pro tento systém. Výsledkem jeho snažení byla webová aplikace SSB4Web využívající především rámec Angular. Aplikace je schopna komunikovat se serverem systému SSB a díky zásuvným modulům umožňuje prezentaci základních typů dat, které tvoří základ tohoto systému [49]. Desktopová verze klienta však obsahuje celou řadu dalších specializovaných zásuvných modulů, které pro webovou verzi klienta dosud nebyly implementovány. Jedním z těchto modulů je právě nástroj KANBAN board pro optimalizaci spolupráce týmů, jehož implementace je cílem mé práce.

[19] Fortune 500: https://fortune.com/fortune500/
[20] SEACOMP SYSTEM BUILDER

Původní nástroj však svou funkcionalitou pro pokročilejší řízení projektů již nestačí. Z tohoto důvodu se v rámci interního projektového řízení začal ve společnosti SEACOMP s.r.o. používat nástroj Kanboard, který byl upraven a obohacen o některé specializované funkce. Tento nástroj však pracuje mimo ekosystém SSB. Přáním společnosti je mít k dispozici nástroj, který umožňuje pokročilejší práci s nástěnkou typu Kanban a tento nástroj mít plně integrován do produktu SSB4Web. Z tohoto důvodu bylo vypsané toto téma bakalářské práce.

2.3.2 Požadavky na aplikaci

Z důvodu ochrany obchodního tajemství však není možné zveřejnit zdrojové kódy programu SSB. Pro řešení této bakalářské práce je tudíž nejprve nutné vytvořit zjednodušenou verzi toho systému. Ta nese označení SSBLite a dle požadavku zadavatele je psána v jazyce C#, konkrétně za použití ASP.NET Core. Tato část slouží jako server a komunikuje s klienty skrze aplikační rozhraní REST.

Podoba aplikačního rozhraní z důvodu budoucí integrace do systému SSB4Web musí odpovídat podobě používané právě v této zmíněné webové aplikaci. Samotná integrace však na přání společnosti není předmětem této práce. Aplikaci je proto i přes budoucí integraci nutno vytvořit jako samostatnou webovou aplikaci obsahující například i autentizaci uživatelů a další komponenty.

Dále je požadováno, aby webová aplikace používala stejné technologie jako původní SSB4Web, tedy rámec Angular s knihovnou pro správu stavu NgRx. Výjimkou je v tomto případě knihovna použitá pro uživatelské rozhraní, kdy původní diplomová práce využívala knihovnu PrimeNG[21]. Po dokončení této práce však bylo společností požadováno knihovnu nahradit za balík komponent DevExtreme. Tento balík je tedy při implementaci práce rovněž využit.

Od samotné funkcionality aplikace je přirozeně očekáváno standardní chování aplikací toho typu jako je vytváření, mazání, editace a přesuny karet. Nástěnka kromě sloupců musí umět pracovat také s plaveckými drahami a obě tyto struktury musí být v aplikaci konfigurovatelné. Ke kartám je nutné mít možnost přiřadit štítky, které taktéž musejí být přizpůsobitelné. Veškeré akce týkající se karet musí byt zaznamenávány a je třeba umožnit zobrazení historie jednotlivých karet. Serverová část aplikace musí načítat konfiguraci vydefinovanou mimo zdrojový kód aplikace. Pro autentizaci uživatele je požadováno využívat standard JWT.

[21] PrimeNG: https://primefaces.org/primeng/

Kapitola 3

Technologie

V této kapitole jsou ve stručnosti přiblíženy technologie a techniky využité při tvorbě této práce. V dnešní době popularita webových aplikací stále roste a to na úkor klasických desktopových aplikací [26]. Tohoto trendu si je vědoma i společnost SEACOMP s.r.o., a proto je jejím záměrem vytvořit webovou aplikaci typu Kanban. Všechny použité technologie jsou tedy právě webového charakteru.

Úvod podkapitoly je věnován klientské části aplikace. Je zde popsán princip jednostránkových aplikací, programovací jazyk JavaScript, jeho typová nástavba TypeScript, rámec Angular a jeho knihovny NgRx a DevExtreme.

Dále jsou zmíněny technologie zajišťující propojení klientské části aplikace s částí serverovou. Toho je docíleno díky aplikačnímu rozhraní REST a standardu JWT, který zajišťuje autentizaci uživatelů a autorizaci jejich požadavků.

V poslední části podkapitoly je zase popsána serverová část. Ta se skládá z technologie ASP.NET, která zpracovává požadavky zaslané z klientské části aplikace a překládá je na dotazy databáze, která využívá technologii PostgreSQL.

3.1 Jednostránkové aplikace

Tato kapitola čerpá z [24].

V raných dobách internetu se webové stránky skládaly pouze ze statického obsahu. S popularizací elektronického obchodování se však objevila potřeba na stránkách zobrazovat i obsah dynamický.

Této funkcionality lze docílit například díky technologii AJAX[1]. Technologie AJAX umožňuje komunikovat se serverem i po načtení stránky a tato komunikace probíhá na pozadí. Díky tomu není nutné při každé akci znovu načíst celou stránku, čímž je značně zredukován objem přenášených dat.

Při navštívení jednostránkové webové aplikace je ze serveru nejprve načtena počáteční zmenšená část stránky. Další obsah je vyžádán a načten zpravidla až na základě uživatelské interakce. Ve srovnání s vícestránkovými aplikacemi tak tento přístup umožňuje rychleji reagovat na požadavky uživatele.

Často se lze také setkat se změnou adresního řádku prohlížeče při pomyslném přechodu na jinou podstránku. Ve skutečnosti se však uživatel stále nachází na jedné stránce, jen byl její obsah aktualizován. Právě proto se zmíněný princip webových aplikací označuje jako jednostránkový.

[1] Asynchronous JavaScript and XML

Takovéto aplikace se zpravidla skládají z komponent. Pro tvorbu těchto aplikací se využívá jazyk JavaScript, nicméně vzhledem k jejich komplexnosti se v drtivé většině pro implementaci používá některých z rámců tohoto jazyka.

3.2 JavaScript

JavaScript je interpretovaný programovací jazyk s dynamickým typováním, nejčastěji používaný ve webových prohlížečích. Jeho hlavním účelem je umožnit webovým stránkám interagovat s uživatelem a upravovat obsah právě zobrazené stránky [17].

Nejčastěji je kód napsaný v tomto jazyce spuštěn až v internetovém prohlížeči koncového uživatele na rozdíl od jazyků jako například PHP[2], kde je kód zpracován na straně serveru. Nicméně existuje i platforma postavená na tomto jazyce zvaná Node.js[3], která se obdobně jako jazyk PHP zpracovává již na serveru.

3.2.1 TypeScript

Samotný jazyk JavaScript však pro vývojáře může představovat překážku při implementaci rozsáhlých aplikací. Jedním z hlavních problémů je nepřehlednost kódu s dynamickým typováním proměnných.

Řešením tohoto problému je například nástavba TypeScript[4], která tento jazyk obohacuje o prvky běžně používané v jiných programovacích jazycích. Jedná se ku příkladu o moduly, třídy, rozhraní a v neposlední řadě statické typování. Díky tomu je možné jednodušeji vyvíjet robustnější aplikace [12].

Jelikož internetové prohlížeče jazyk TypeScript neznají, je využíván překladač, který kód překládá do jazyka JavaScript.

3.2.2 Rámce jazyka JavaScript

S použitím čistého jazyka JavaScript bez použití knihoven či rámců se dnes setkáváme zřídka. Zejména u větších projektů je psaní kódu v čistém jazyce JavaScript ve srovnáním s použití knihoven a rámců velmi časově náročné [39].

Nejslavnějším rozšířením toho jazyka je bezesporu knihovna jQuery[5]. Ta usnadňuje zejména průchod a manipulaci objektového modulu dokumentu (angl. *Document Object Model* neboli DOM), zpracování událostí, animace a práci s asynchronními požadavky [50].

Knihovna jQuery však není určená pro tvorbu stále populárnějších jednostránkových aplikaci. Na ty se zaměřují zcela jiné rámce, kterých se v poslední době objevuje stále více.

V současné době se mezi tři nejpoužívanější rámce řadí Angular, React a Vue.js [1]. Za dvěma prvními zmíněnými rámci stojí velké společnosti. Rámec Angular je dílem společnosti Google a rámec React zase společnosti Facebook. Autorem rámce Vue.js je však Evan You a komunita vývojářů po celém světě.

Nejmladším z těchto rámců je právě Vue.js. I přesto se však těší větší oblibě než nejstarší uvedený rámec Angular. Angular jako jediný z uvedených rámců využívá nástavbu TypeScript. V ostatních rámcích ji sice také lze využít, není to však vyžadováno. Architektura všech tří rámců si je principem velmi podobná. Liší se opět pouze rámec Angular, který

[2]PHP: https://www.php.net
[3]Node.js: https://nodejs.org/en/
[4]TypeScript: https://www.typescriptlang.org
[5]jQuery: https://jquery.com

komponenty umožňuje rozdělit do jednotlivých modulů. Co se týče obtížností těchto rámců pro programátora, jedná se vždy o subjektivní názor. Nicméně jako nejtěžší z uvedených rámců se obecně považuje Angular a nejlépe je v této kategorii hodnocen rámec Vue.js [40].

3.2.3 Angular

Tato kapitola čerpá z [20].

Angular je webová platforma umožňující tvorbu jednostránkových aplikací s využitím jazyků TypeScript a HTML. Tato platforma si velmi zakládá na modularizaci kódu. Logické celky aplikace jsou proto často rozdělovány do jednotlivých modulů, pro které lze využít metodu líného načítání (angl. *lazy loading*). Obsahem těchto modulů jsou pak jednotlivé komponenty.

Komponenty tvoří pohledy (angl. *views*), které jsou zobrazeny jako elementy ve webové aplikaci. Tyto komponenty využívají služby (angl. *services*), které zajišťují funkcionalitu která není přímo vázaná na pohledy. Služby je tak tedy možné znovu použít na více místech v aplikaci.

Komponenty mají v aplikaci hierarchickou strukturu. Každá aplikace obsahuje alespoň jednu komponentu, která se označuje jako kořenová. Každá z těchto komponent je reprezentována třemi částmi – třídou, která nese veškerou její logiku, šablonou, která určuje její strukturu coby entity jazyka HTML a dále kaskádovými styly, které ovlivňují její vizuální podobu.

Většina prvků tohoto rámce jsou ve skutečnosti pouhé třídy jazyka JavaScript, jejich konkrétní typ tak bývá označován pomocí dekorátorů. Tento rámec poskytuje celou řadu užitečných funkcí jako například oboustrannou datovou vazbu, která umožňuje synchronizaci dat mezi šablonou a samotnou komponentou.

Autorem platformy Angular je Miško Hevery, zaměstnanec společnosti Google. Platformu původně psal pro usnadnění vývoje několika interních projektů. V roce 2010 však byla platforma i její zdrojový kód otevřena široké veřejnosti. V následujících letech se oblast vývoje webových aplikací rapidně změnila a platforma Angular již nedokázala držet krok. Dále se také stala velmi populární, s čímž původní návrh nepočítal. Bylo tedy nutné platformu kompletně přepsat.

Na podzim roku 2016 byla vydána nová verze této platformy s označením Angular 2. K dnešnímu dni se nové verze dostaly až k označení pod číslem 9, nicméně se stále jedná o rozšíření původního jádra verze 2. Na vývoji se stále podílí jak společnost Google, tak i komunita. Pro odlišení původní verze od nových přepsaných verzí se ta původní dnes označuje jako AngularJS [14].

NgRx

NgRx[6] je knihovna pro správu stavu určená pro platformu Angular. Tato knihovna je založena na principu Redux. Ten cílí na zvýšení předvídatelnosti stavu aplikace. Hlavní myšlenkou tohoto principu je koncept jediného zdroje pravdy. Data, sdílená napříč různými komponentami, jsou umístěna na jednom místě, které se označuje jako sklad (angl. *store*). Díky tomu je kdykoliv možné zjistit, v jakém stavu se aplikace zrovna nachází. Mimo to tento princip také zvyšuje přehlednost, který kód ovlivňuje které části aplikace. S narůstající komplexností aplikací dochází v tomto směru často k velké nepřehlednosti. Další výhodou je také dodržování určitého pořadí akcí při jejich paralelním vykonávání [35].

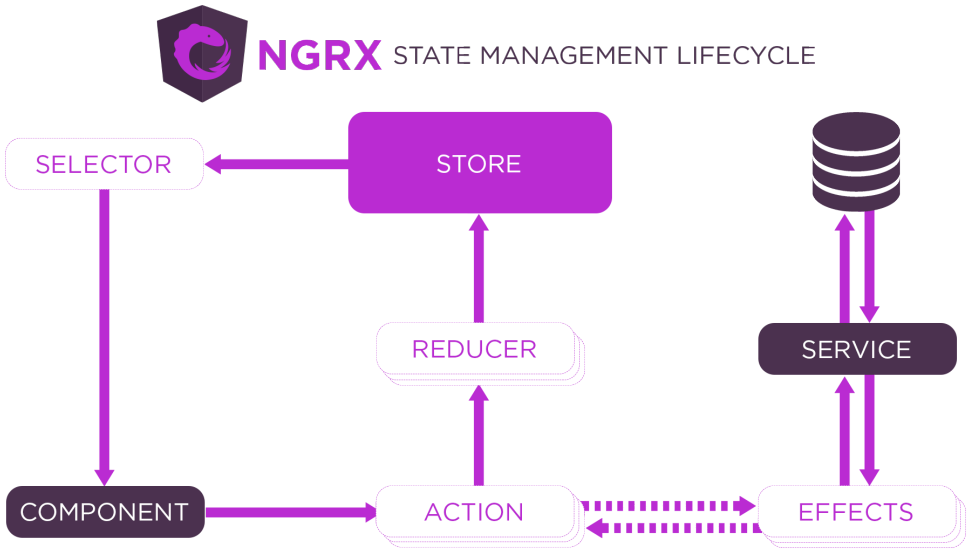
[6] NgRx: https://ngrx.io

Tato knihovna však pro tvorbu aplikací pomocí platformy Angular není nutně potřeba a v některých případech může vývoji i přitížit. I přes veškeré výhody, které využití tohoto systému přináší, je třeba počítat i s negativními následky. Systém má poměrně strmou učební křivku a je třeba znát princip Redux a umět používat knihovnu RxJS [33]. Dalším úskalím je velké množství nutného dodatečného kódu. Přidání zdánlivě jednoduché funkce může trvat delší dobu, protože sebou nese nutnost přidat spoustu kódu napříč různými částmi aplikace. Proto je dobré nejprve důkladně zvážit, zda je vhodné pro daný projekt knihovnu NgRx využít. Vývojáři knihovny proto na jedné z konferencí[7] přišli se zásadou SHARI, které toto rozhodnutí usnadní.

Zásada SHARI [38]:

- Sdílení (angl. *shared*) - Stav, ke kterému přistupuje více komponent a služeb
- Hydratování (angl. *hydrated*) - Stav, který přetrvává a je obnovován z externího zdroje
- Dostupnost (angl. *available*) - Stav, který musí být dostupný při opětovném přístupu na danou podstránku
- Načtení (angl. *retrieved*) - Stav, který musí být načten s vedlejšími účinky
- Ovlivnění (angl. *impacted*) - Stav, který je ovlivněn akcemi jiných zdrojů

Jednotlivé součásti knihovny NgRx, jejich provázanost a tím i celý životní cyklus stavu aplikace v této knihovně je znázorněn na grafu 3.1, kde je také popsán význam jednotlivých součástí.



Obrázek 3.1: Graf znázorňující životní cyklus správy stavu s využitím knihovny NgRx. Stav aplikace je uložen ve skladu (uzel *store*), data jsou z něj získávána pomocí selektorů (uzel *selector*) a jsou použita v komponentách (uzel *component*). Interakce s komponentou vyvolává akci (uzel *action*), která může mít za následek vedlejší efekty (uzel *effects*), což představuje získání dat z databáze za pomocí služeb (uzel *service*). Akce dále vyvolá funkci reduktoru (uzel *reducer*), který vytvoří nový stav dat a opět jej vloží do skladu (uzel *store*). *Převzato z [34].*

[7] ng-conf 2018 - Reducing the Boilerplate with NgRx: https://www.youtube.com/watch?v=t3jx0EC-Y3c

DevExtreme

DevExtreme[8] je balík komponent určený pro tvorbu responzivních webových aplikací. Balík je zaměřen primárně na platformu Angular, nicméně více něž 65 komponent je dostupných také pro platformy jQuery, Vue, React, ASP.NET MVC 5, ASP.NET Core a další [15]. Balík obsahuje mnoho prvků uživatelského rozhraní, které jsou určeny jak pro tradiční webové stránky, tak i moderní mobilní aplikace. Všechny prvky jsou samozřejmě responzivní. Nabídka prvků tohoto balíku je velmi rozmanitá a řadí se mezi ně například datová mřížka, grafy, mapy a editory. Tento balík však lze bezplatně využít jen pro nekomerční využití.

3.3 JSON Web Token

Tato podkapitola vychází z [9].

JSON Web Token (zkráceně JWT) je otevřený standard[9] zajišťující zabezpečenou výměnu informací mezi dvěma stranami. Jedná se o objekt typu JSON[10], který se skládá ze tří částí oddělených tečkou. Jedná se o hlavičku, obsah a podpis.

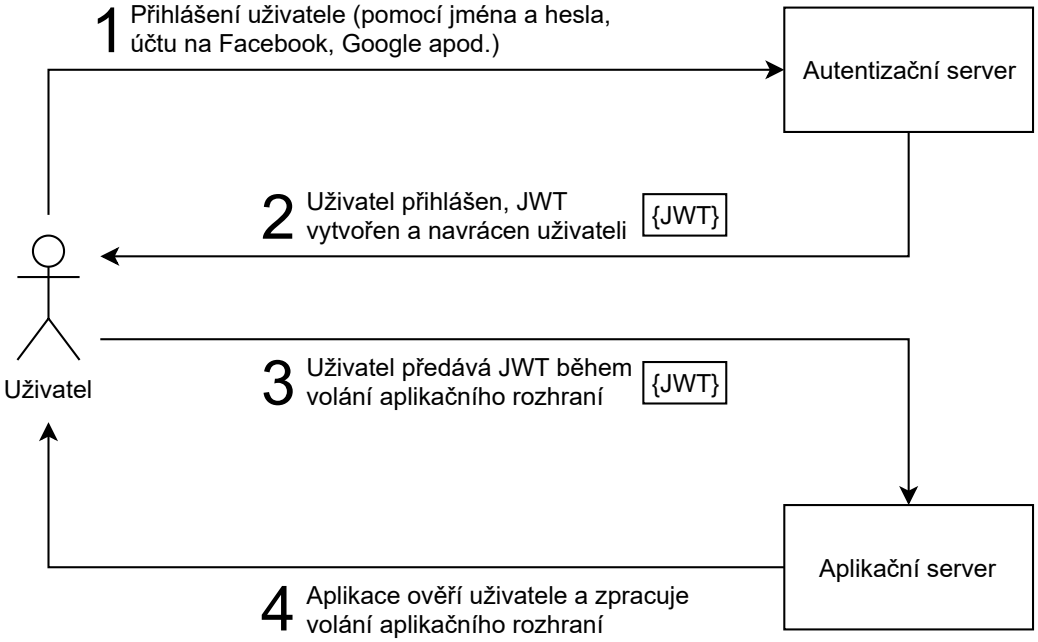
Hlavička je objekt typu JSON, zakódovaný pomocí kódovaní Base64Url[11], který nese informaci o typu žetonu a použitém algoritmu digitálního podpisu.

Obsah je stejně jako hlavička objekt typu JSON zakódovaný pomocí Base64Url. Tentokrát jsou však jeho obsahem tvrzení (angl. *claim*). Ty nesou přenášené informace, které je třeba předávat zabezpečené. Nejčastěji se jedná o informace o uživateli.

Podpis zaručuje, že zpráva při přenosu nebyla pozměněna. Může být také využit pro ověření identity odesílatele. Podpis je vytvořen spojením řetězců dvou předchozích částí, mezi které je vložena tečka. Na konec toho řetězce je dále připojen tajný řetězec. Výsledný řetězec je poté zašifrován algoritmem uvedeným v hlavičce.

V praxi se lze s touto metodou autentizace setkat například na stránkách, které umožňují přihlásit se pomocí účtu u společnosti Google [21]. V takovém případě se jedná o výměnu informací mezi třemi stranami. Posloupnost akcí pro získání žetonu v této situaci a jeho následné použití lze vidět na obrázku 3.2.

[8] DevExtreme: https://js.devexpress.com
[9] RFC 7519: https://tools.ietf.org/html/rfc7519
[10] JavaScript Object Notation
[11] Kódování reprezentující binární data pomocí tisknutelných znaků [46]



Obrázek 3.2: Diagram znázorňující případ užití standardu JWT. *Přeloženo z [47].*

3.4 ASP.NET Core

ASP.NET Core[12] je platforma společnosti Microsoft s otevřeným zdrojovým kódem určena pro tvorbu webových aplikací. Jedná se o rozšíření populární platformy .NET určené především pro programovací jazyk C#, taktéž vyvinutým společností Microsoft.

Přestože tato společnost stojí za dnes nejpopulárnějším operačním systémem Windows, je platformu ASP.NET Core možné provozovat i na jiných operačních systémech. Vydání první verze této platformy proběhlo v roce 2016 [30]. Tehdy však bylo možné platformu využít pouze na operačním systému Windows a tyto dřívější verze nesly název pouze ASP.NET bez označení Core [32].

Mimo funkcionalitu již obsaženou v platformě .NET přináší platforma ASP.NET Core například základní rámec pro zpracování webových požadavků, šablonový systém pro tvorbu dynamických stránek zvaný Razor, celou řadu užitečných knihoven a pokročilý autentizační systém. Platforma také nabízí před-připravenou šablonu pro vytvoření webu využívající architekturu MVC[13]. Platformu lze využít jak pro vývoj plnohodnotné webové aplikace či stránky, tak i pro vývoj služby zpracovávající požadavky aplikačního rozhraní REST[14] [19].

3.5 PostgreSQL

PostgreSQL[15] je objektově-relační databázový systém s historií do roku 1986. Jedná se o vysoce přizpůsobitelné rozšíření jazyka SQL s otevřeným zdrojovým kódem [51]. Tento systém obsahuje všechny běžné vlastnosti, které jsou u podobných systémem zvykem. Jme-

[12] ASP.NET Core: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet
[13] Architektura model-view-controller neboli model-pohled-kontroler
[14] Representational State Transfer neboli Reprezentativní převod stavu
[15] PostgreSQL: https://www.postgresql.org

novitě například transakce, pohledy, referenční integritu cizích klíčů nebo uzamykání [48]. Jednou z předností tohoto systému je rozmanitá možnost volby jazyku pro implementaci procedur. Oficiální distribuce obsahuje programovací jazyky PL/pgSQL, PL/Tcl, PL/Perl a PL/Python. Lze však využít i obdoby dalších jazyků jako jsou Java, PHP nebo Unix shell [52].

Kapitola 4

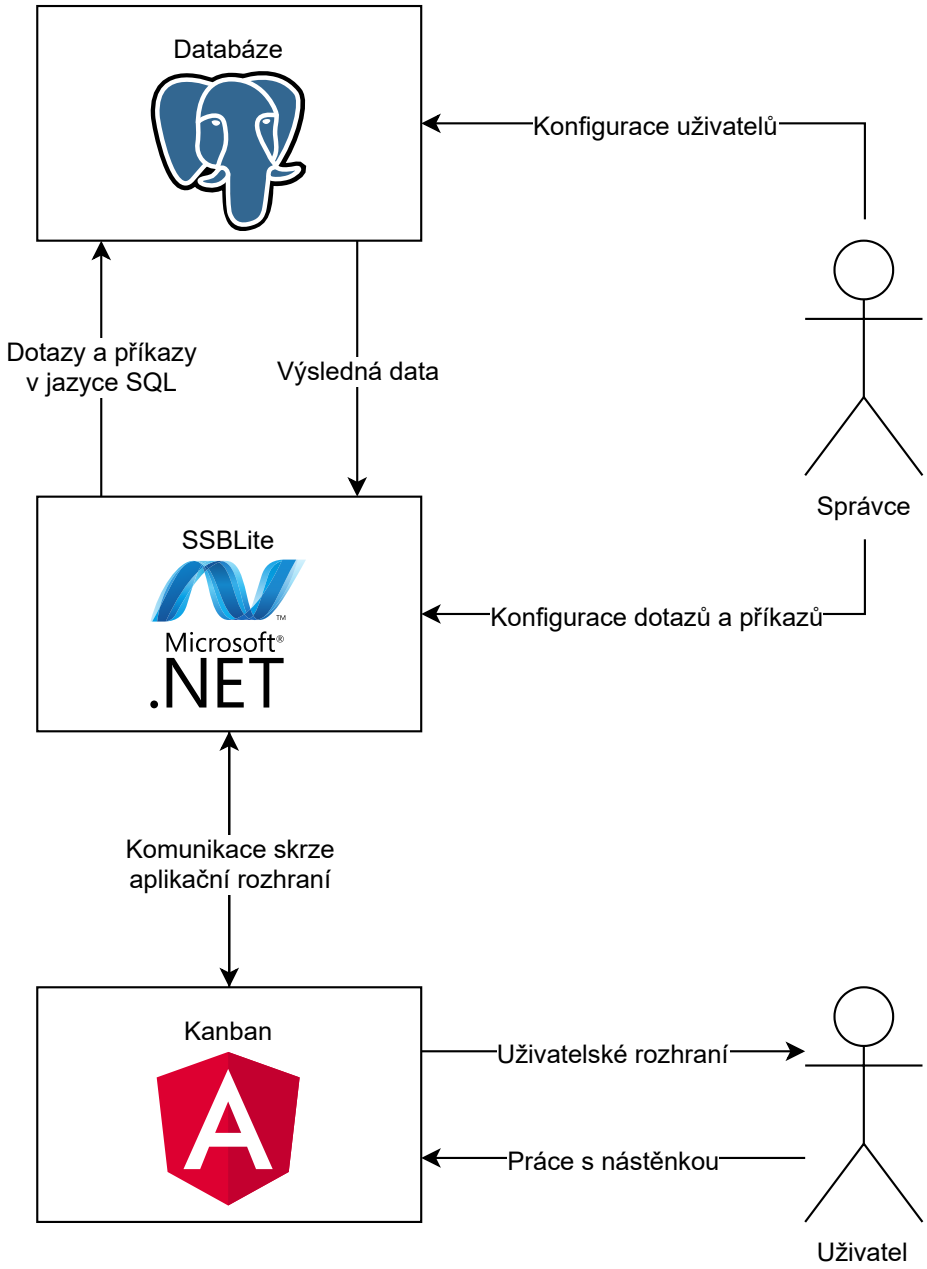
Návrh aplikace

Před samotnou implementací aplikace bylo nejprve nutné promyslet návrh tvořené aplikace. Zejména pak rozdělení aplikace do jednotlivých částí, určit jejich role v celkovém fungování systému a zmapovat interakci mezi těmito částmi. Některá rozhodnutí ohledně směru vývoje byly provedeny mnou samotným, některá ve spolupráci s konzultantem této bakalářské práce a jiná byla nařízena přímo vedením společnosti SEACOMP s.r.o.

Dále bylo třeba vytvořit návrh databáze, kde budou uložena data a také připravit grafický návrh aplikace. Mimo jiné bylo třeba zanalyzovat existující aplikační rozhraní a připravit jeho rozšíření. Právě těmto zmíněným oblastem se věnuje tato kapitola.

4.1 Schéma systému

Schéma systému je vidět na snímku 4.1. Jsou zde znázorněny tři její hlavní části – databáze, serverová část neboli SSBLite a v neposlední řadě webový klient Kanban. U každé z těchto částí je na snímku uvedena také její nejpodstatnější technologie. Dále zde lze pozorovat vzájemné interakce těchto části. Na pravé straně obrázku lze vidět dvě role uživatelů a také jejich interakci s aplikací.



Obrázek 4.1: Schéma vzájemných interakcí jednotlivých částí aplikace, jejich technologií a možnostech různých rolí uživatelů.

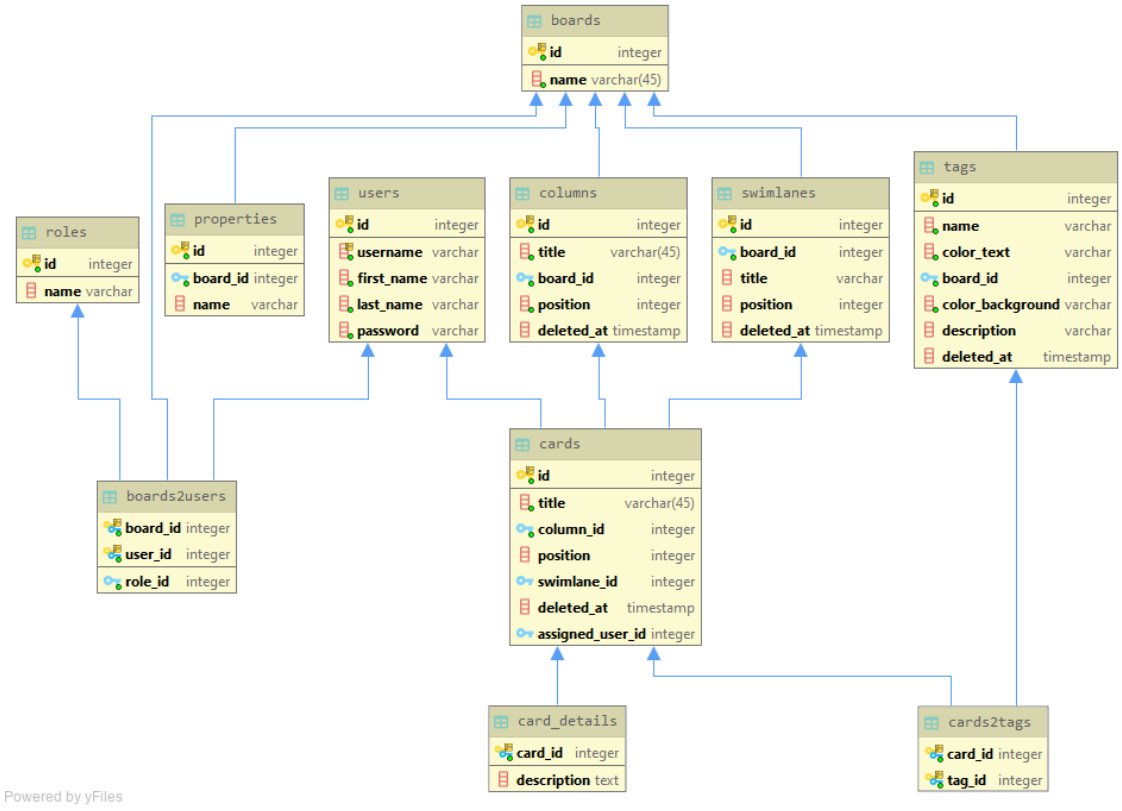
V tomto schématu se může jevit podivuhodná konfigurace uživatelů správcem přímo v databázi. V reálném prostředí, kde bude aplikace SSBLite nahrazena plnohodnotnou aplikací SSB, bude konfigurace probíhat již přímo v aplikaci SSB za pomocí uživatelského rozhraní. Nicméně pro účely této bakalářské práce postačí konfigurace tímto náhradním způsobem. Vzhledem k pozdějšímu nahrazení aplikace SSBLite by implementace této funkcionality byla bezpředmětná.

V práci taktéž není zakomponovaná správa samotných nástěnek. V současné aplikaci SSB totiž nástěnka odpovídá právě jednomu týmu uživatelů. Pro vytvoření nové nástěnky je tudíž nutné vytvořit celý nový tým. Z důvodů zachování tohoto chování a předcházení

zbytečné dvojí implementace je pro vytvoření nebo odstranění nástěnky či přiřazení přístupu uživatelům k nástěnce nutné upravit data přímo v databázi. Tato funkcionalita bude taktéž vyřešena pozdější integrací do stávající aplikace SSB4Web.

4.2 Entitně-vztahový diagram

Data aplikace jsou uložena v relační databázi. Systém SSB a tedy i SSBLite využívají databázový systém PostgreSQL. Data jsou uložena v jednotlivých tabulkách, které lze reprezentovat entitně-vztahovým diagramem znázorněným na obrázku 4.2. Názvy tabulek i sloupců jsou dle konvence společnosti v anglickém jazyce. Tabulky ve většině případu reprezentující entity (například nástěnky, karty nebo štítky), výjimku tvoří pouze propojovací tabulky, které reprezentují vztah M:N. Ty lze poznat podle číslovky 2 v názvu tabulky. Před a za touto číslovkou se pak nachází názvy entit, které ve vztahu figurují.



Obrázek 4.2: Entitně-vztahový diagram reprezentující relační databázi aplikace.

4.3 Grafický návrh

Prvotní verze aplikace, která vznikla řešením této bakalářské práce bude využita pouze pro interní účely společnosti. Z tohoto důvodu nebyl při zadání práce kladen příliš velký důraz na estetickou stránku aplikace. Jedním z požadavků v tomto směru však bylo, aby aplikace co nejvíce využívala grafický styl knihovny DevExtreme. Další požadavek zněl, aby byly v aplikaci použity barvy loga společnosti SEACOMP s.r.o.

Ve všech částech aplikace je přítomna hlavička. Na levé straně je vždy obsaženo logo společnosti a na pravé straně se nachází informace o přihlášeném uživateli a akce jako jeho odhlášení. Po vzoru oficiální webové stránky společnosti[1] hlavičku a obsah stránky dělí horizontální linka oranžové barvy, která je použita v samotném logu společnosti.

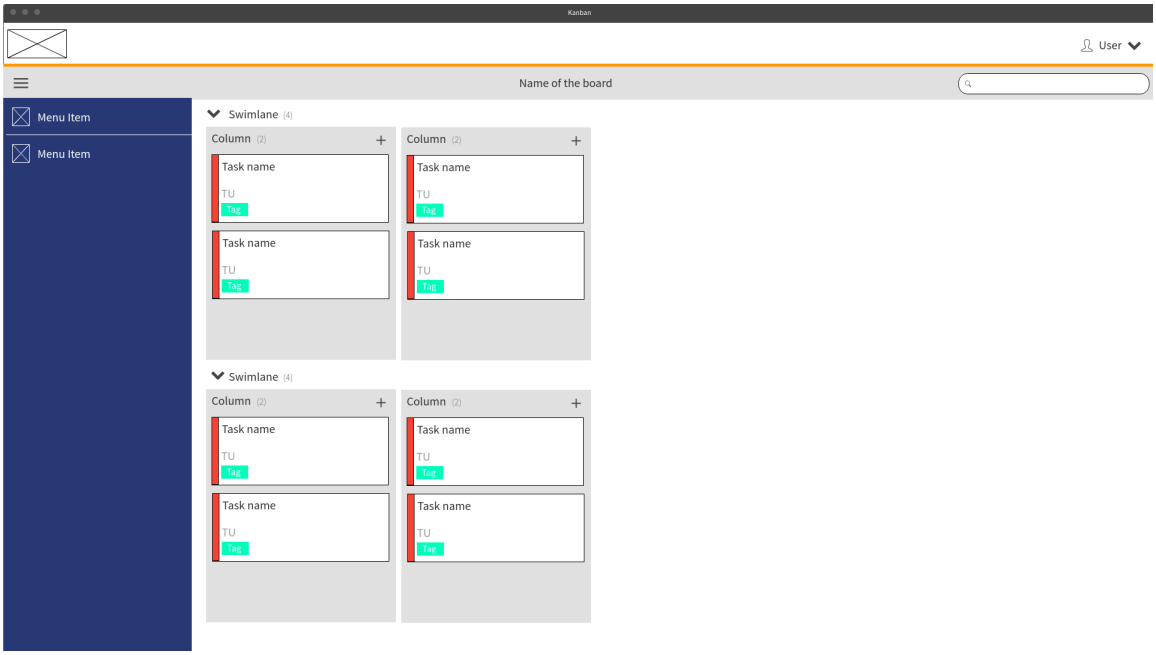
Co se grafického návrhu týče, nejrozsáhlejší část aplikace je jistě samotná nástěnka. Původní návrh je vyobrazen na snímku 4.3. Z důvodu co největší maximalizace počtu zobrazených karet se nástěnka rozpíná po celé šířce i výšce stránky s výjimkou hlavičky a nástrojové lišty.

Ze stejného důvodu je menu ve výchozím stavu skryto a lze ho otevřít pomocí tlačítka na levé straně nástrojové lišty. Lišta dále obsahuje název právě otevřené nástěnky.

Položky v menu i samotný detail karet se zobrazuje v modálním okně. Uživatel se tak například i při zobrazení detailu karty stále nachází na nástěnce a může se vrátit na její přehled téměř zcela bez prodlení na rozdíl od stávající aplikace využívané ve společnosti zadavatele, kde je pro zobrazení všech detailů nutné přejít na novou stránku a následný návrat na přehled je vzhledem k počtu zobrazených úkolu velmi pomalý.

Rozložení a vizuální podoba detailu karty je inspirována nástrojem Microsoft Team Foundation Server[2], který se ve společnosti SEACOMP s.r.o. v současnosti rovněž využívá.

Některé akce jako například odstranění karty lze provést podobně jako v desktopových aplikací pomocí kliknutí pravým tlačítkem myší na danou entitu. V případě dotykových obrazovek je kliknutí pravého tlačítka substituováno dlouhým stiskem.



Obrázek 4.3: Grafický návrh části aplikace, kde je zobrazena nástěnka kanban.

Další části aplikace jsou z pohledu uživatelského rozhraní triviální, a proto pro jejich tvorbu návrh nebyl vytvořen.

[1] Webové stránky společnosti SEACOMP s.r.o.: https://seacomp.cz

[2] Dnešní verze nástroje se již označuje jako Azure DevOps Server

4.4 Aplikační rozhraní

Komunikace mezi klientskou částí aplikace a částí serverovou probíhá za pomoci aplikačního rozhraní REST. Zprávy tohoto rozhraní využívají formát JSON a mají přesně definovaný obsah. Vzhledem k budoucí integraci mnou tvořené aplikace do již existující aplikace SSB4Web bylo nutné zachovat v obou těchto aplikacích jednotnou podobu rozhraní.

Vzhledem k zaměření mé aplikace nebylo třeba využít všechny možnosti existujícího rozhraní. Při práci byly z původního rozhraní použity pouze dva koncové body a to `SSBQuery/GetData`, umožňující dotazovat se na data uložena v databázi, a `SSBMethod/ExecuteMethod`, spouštějící uložené procedury či příkazy. Toto rozhraní však bylo nutné rozšířit o možnost autentizace uživatele. Původní koncové body v tomto případě nebylo možné využít, protože pouze propagují dotazy na databázi, kdežto pro autentizaci je třeba požadavek zpracovat přímo v aplikaci SSBLite. Z tohoto důvodu byl přidán nový koncový bod `SSBAuthentication/VerifyCredentials`.

Obě části aplikace – jak Kanban, tak SSBLite – využívají stejné datové objekty, díky tomu je možné mezi těmito dvěma částmi jednoduše přenášet data. Obsahem požadavku pro získání dat je například objekt `SSBQueryRequestDTO`, ten s sebou nese jednak název konkrétního před-připraveného dotazu pod klíčem `queryName` a dále pod klíčem `filters` může obsahovat i další objekt typu `SSBFilterDTO`. Ten se zase skládá z objektu `ssbIFilterOperator`, skrytým pod stejnojmenným klíčem, a dále z pole objektů `SSBFilterTermDTO` označeným klíčem `filterItems`. Objekty v tomto poli obsahují tři hodnoty. Pod klíčem `m_filterOperator` lze opět pozorovat objekt `ssbIFilterOperator`. Ve skutečnosti se jedná pouze o výčet logických operátorů, které jsou později použity pro bezpečné sestavení dotazu na databázi. Další dvě hodnoty nesou název sloupce v databázi a pole hodnot použitých v dotazu.

Konkrétní dotaz na získání karet s použitím filtru nástěnky tak může vypadat následovně:

```
{
  "queryName": "getCards",
  "limit": 0,
  "onlyVisibleColumns": true,
  "offset": 0,
  "filter":
    {
      "ssbIFilterOperator": 0,
      "filterItems":
        [
          {
            "m_filterOperator": 0,
            "m_column": "col.board_id",
            "m_values": [1]
          }
        ]
    }
}
```

Nevýhoda této podoby aplikačního rozhraní je však v nemožnosti provést více dotazů na databázi pomocí jednoho asynchronního volání. Například při prvotním načtení nástěnky je tak nutné odeslat vícero asynchronních požadavků najednou.

Tento nedostatek je způsoben tím, že je původní návrh rozhraní zaměřen hlavně na zobrazení tabulek ve vztahu hlavní–podrobné (angl. *master–detail*), kde k těmto situacím dochází minimálně. Zadavateli tématu diplomové práce bylo představeno několik alternativních řešení, nicméně bylo rozhodnuto pro zachování původního návrhu a setrvání v odesílání několika požadavků současně.

Kapitola 5

Implementace aplikace

Předmětem této kapitoly je popis vlastní implementace webové aplikace typu Kanban, jejíž vytvoření bylo cílem této bakalářské práce. V úvodu je popsaná klientská část aplikace, ke které uživatel přistupuje pomocí internetového prohlížeče. Další část kapitoly je věnována implementaci serverové části aplikace, s kterou uživatel přichází do styku pouze skrze aplikační rozhraní, které již bylo přiblíženo v kapitole 4.4. Závěr kapitoly je pak zaměřen na databázovou vrstvu aplikace, kde jsou uchovávány uživatelská a jiná data.

5.1 Klientská část

Klientskou částí aplikace je jednostránková webová aplikace vytvořená za pomoci rámce Angular. Základním stavebním kamenem toho rámce jsou bez pochyby komponenty. Právě ty jsou hlavním tématem této podkapitoly. Dále jsou zde zmíněny i jiné zajímavé části implementace klientské části. Obsah jednotlivých stránek je sestaven kombinací vlastních komponent, které jsou podrobněji popsané v této kapitole, a také před-připravených komponent knihovny DevExtreme. Rozeznat jejich původ lze pomocí předpony `app-` v případě vlastních komponent a pomocí předpony `dx-` v případě komponent z knihovny DevExtreme.

5.1.1 Struktura adresáře

Adresář, ve kterém se nachází zdrojové kódy aplikace je kvůli využití rámce Angular poměrně rozsáhlá. Některé jeho části, které stojí za zmínku jsou popsány v této podkapitole. Zajímavé části nejvyšší úroveň adresáře aplikace vypadají následovně:

- `node_modules` – Zdrojové kódy knihoven třetích stran
- `src` – Zdrojové kódy aplikace
  - `app` – Adresář obsahující zdrojové kódy aplikace v jazyce TypeScript
  - `assets` – Adresář obsahující statický obsah stránky jako například globální kaskádové styly a obrázky
  - `index.html` – Hlavní stránka aplikace
- `angular.json` – Konfigurace rámce Angular
- `package.json` – Konfigurace závislostí na knihovnách třetích stran

Nejdůležitějším z výše uvedených adresářů je jednoznačně adresář `app`. Pro lepší přehlednost je jeho nejpodstatnější obsah uveden zvlášť níže:

- `board/` – Adresář obsahující veškeré komponenty týkající se části aplikace zobrazující nástěnku
- `core/` – Adresář obsahující klíčové části aplikace nutné pro její fungování
- `dashboard/` – Adresář obsahující komponent týkající se části aplikace se seznamem nástěnek
- `login/` – Adresář obsahující komponenty týkající se přihlášení uživatele
- `shared/` – Adresář obsahující komponenty sdílené více částmi aplikace jako například správa stavu nebo datové objekty aplikačního rozhraní
- `app.component.ts` – Kořenová komponenta aplikace
- `app.module.ts` – Hlavní modul aplikace
- `app-routing.module.ts` – Modul sdružující cesty aplikace

5.1.2 Základní struktura stránky

Ať už se uživatel aktuálně nachází na libovolné části aplikace, základní struktura právě zobrazené stránky se bude vždy tvořit ze stejných prvků. Základ celé aplikace je komponenta `<app-root>`. Tento prvek je také kromě referencí na soubory obsahující zdrojové kódy jazyka JavaScript to jediné, co se v obsahu stránky zobrazí v případě, že internetový prohlížeč jazyk JavaScript nepodporuje či jej dokonce blokuje.

V případě, že je interpret jazyka JavaScript dostupný, vzniknou uvnitř tohoto elementu další komponenty. Vždy je přítomna komponenta `<app-header>` a `<router-outlet>`. První z komponent obsahuje hlavičku s logem společnosti SEACOMP s.r.o., která je vždy vidět ve vrchní části aplikace. Součástí hlavičky je také možnost odhlášení uživatele, za předpokladu, že je právě přihlášen. Tato komponenta je zachycena na snímku 5.1.

Obrázek 5.1: Ukázka komponenty hlavičky s právě přihlášeným uživatelem.

Druhá z uvedených komponent ve skutečnosti nemá žádný reálný obsah. Představuje však nedílnou součást aplikace, jelikož se stará právě o chování aplikace jako jednostránkové. Na základě aktuální cesty obměňuje komponentu, která v hierarchii uzlů HTML představuje jejího dalšího sourozence. Z počátku se tak lze na tomto místě setkat s komponentou `<app-login>`, následně `<app-dashboard>` a poté `<app-board>`. Současně s obměnou těchto komponent je také změněna adresa aktuální stránky v adresním řádku internetového prohlížeče. Navenek se tedy může zdát, že uživatel skutečně přechází z jedné stránky na druhou. Ve skutečnosti tomu tak ale není.

Cesty a jim odpovídající komponenty jsou definované v souboru `app-routing.module.ts`. U každé cesty je také možno nastavit ochranné služby.

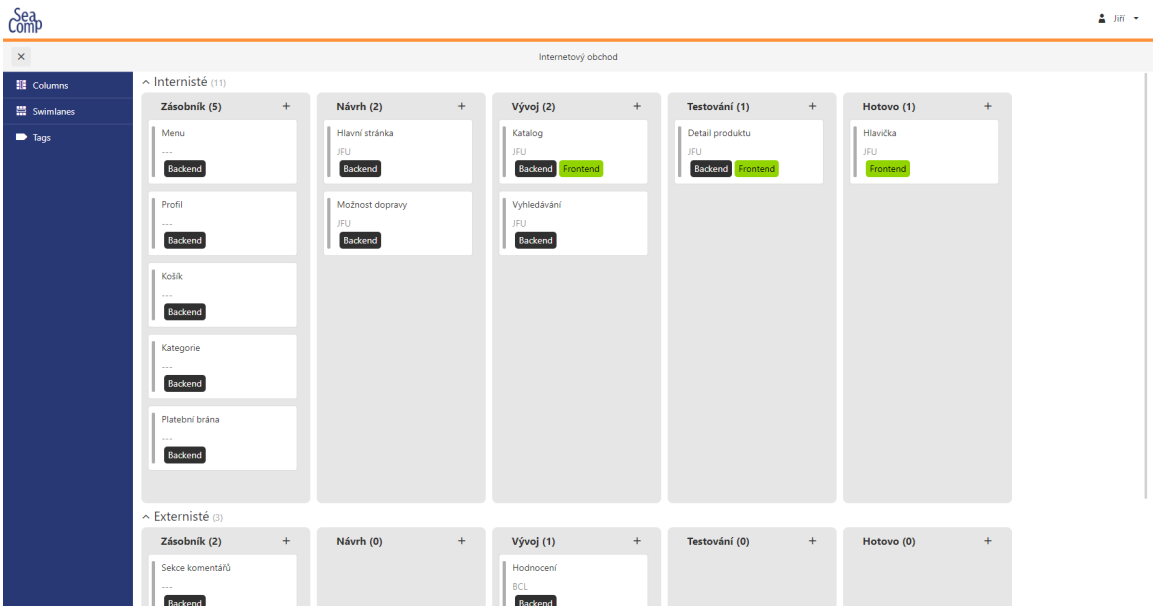
Jedná se o třídy implementující rozhraní `CanActivate`, které ve stejnojmenné metodě vrací logickou hodnotu. Pokud jedna z těchto služeb vrátí hodnotu nepravdy, je přístup uživateli k této cestě zamítnut. Tuto funkcionalitu v aplikaci využívám na ověření, zda je uživatel přihlášen. Výjimku tvoří obrazovka s přihlášením, ke které lze přistoupit v přesně opačné situaci a to právě když uživatel zatím ještě není přihlášen. Jedná se o třídy `AuthenticationGuardService` a `GuestGuardService`.

5.1.3 Nástěnka

Stránka s nástěnkou je bezesporu nejkomplexnější částí aplikace. Základní komponentou je v tomto případě `<app-board>`, jejím úkolem je mimo jiné také prvotní načtení všech dat z databáze, kterých se daná nástěnka týká. Jedná se například o sloupce, plavecké dráhy, karty, štítky a další entity. Tuto část aplikace je možné vidět na snímku 5.2.

Na nejvyšší úrovni této stránky se nachází několik komponent knihovny DevExtreme. Nejvýše položená komponenta je `<dx-toolbar>`, která obsahuje tlačítko pro zobrazení či skrytí menu a název nástěnky. Hned pod ní se nachází komponenta `<dx-drawer>`, ta slouží k zobrazení nabídky menu. Jistou zvláštností je, že celá zbylá část stránky se nachází uvnitř této komponenty, nicméně použitá knihovna takovýto přístup vyžaduje.

Obsahem zmíněné komponenty je další komponenta knihovny DevExtreme a to `<dx-scroll-view>`. V tomto případě nahrazuje nativní posuvné lišty internetového prohlížeče a to jak vertikální, tak i horizontální. Tím je zajištěno jednotné chování i vzhled těchto lišt napříč různými prohlížeči.



Obrázek 5.2: Ukázka části aplikace s nástěnkou obsahující úkoly týkající se vývoje internetového obchodu.

Plavecké dráhy

Uvnitř této komponenty se již nacházejí vlastní komponenty specifické pro nástěnku Kanban. V hierarchii se nejvýše nachází komponenty `<app-board-swimlane>`, kde jedna vždy odpovídá právě jedné plavecké dráze. Ty se skládají ze samotného záhlaví dráhy a dále

z obsahu. Ten lze pomocí kliknutí na záhlaví zobrazit či skrýt, což je užitečná funkcionalita v případě, že je těchto drah využito mnoho.

Sloupce

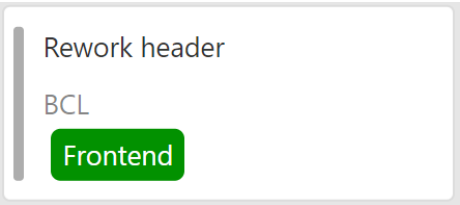
Uvnitř každé plavecké dráhy se pak pro každý sloupec nástěnky nachází právě jedna komponenta typu `<app-board-column>`. Ta sdružuje karty, náležící danému sloupci a plavecké dráze. Sloupec opět využívá komponentu `<dx-scroll-view>`, která v kombinaci s fixní výškou sloupce zajišťuje, že nadměrné hromadění karet v jednom sloupci neovlivní okolní sloupce, jako tomu je v současnosti u původní aplikace Kanboard.

Obsahem sloupců je také komponenta `<dx-sortable>`, která umožňuje přesun karet mezi jednotlivými sloupci a plaveckými drahami. Jedna z metod této komponenty však byla přepsána. Jedná se o metodu, který je spuštěna po přesunu karty. Ta kromě provedení samotné změny pozice také o této změně informuje serverovou část aplikace.

5.1.4 Karta

Přímo v nástěnce jsou úkoly reprezentovány jednotlivými kartami. Ty však zobrazují pouze klíčové údaje úkolu. Podrobnější informace jsou zobrazeny až po kliknutí na příslušnou kartu. Samotné karty se v aplikaci vyskytují jako komponenty `<app-card>`. Ty obstarávají jak samotné zobrazení karty, tak zároveň i obsahují další komponenty. Mezi ty patří seznam štítků náležící dané kartě aneb komponenta `<app-card-tags-list>` nebo kontextové menu, které se zobrazí po kliknutí na kartu pravým tlačítkem myši. Toto menu je realizováno s využitím knihovny DevExtreme za použití komponenty `<dx-context-menu>`. Součástí karty je i komponenta obsahující detail karty, která je blíže popsána v další podkapitole.

Karta je vizualizována jako obdélník obsahující název úkolu, informaci o přiřazeném uživateli a případně i přiřazenými štítky. Na levé straně karty se nachází pruh, jehož barva reprezentuje prioritu úkolu. Na základě priority jsou štítky taktéž sestupně seřazeny v rámci daného sloupce. Kartu lze libovolně přesouvat napříč sloupci a plaveckými drahami. Reálný příklad štítku je možné vidět na snímku 5.3.



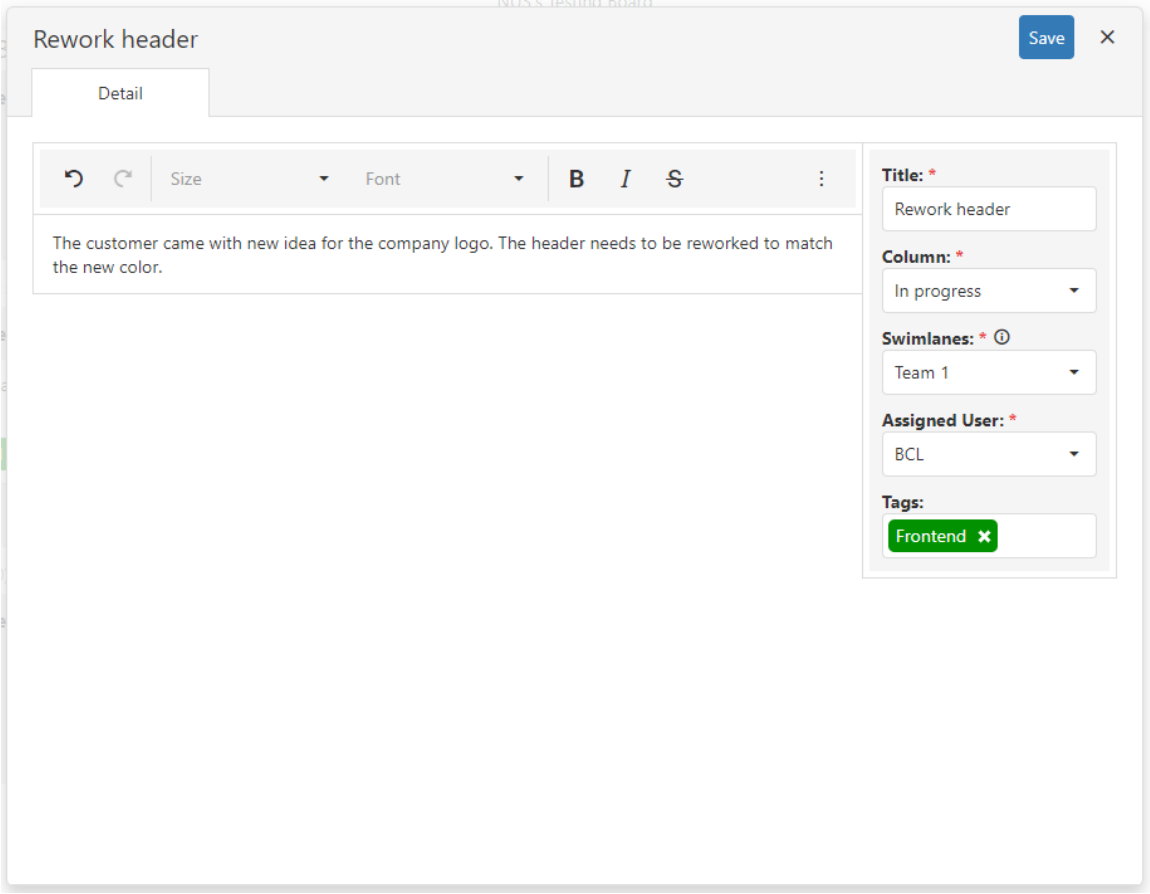
Obrázek 5.3: Ukázka části aplikace obsahující kartu úkolu.

Detail karty

Detail karty úkolu je realizován pomocí modálního okna. V aplikaci existují dva typy tohoto okna. Jeden z nich je použit pro vytváření nové karty a druhý při zobrazení již existující karty. Pro každý tento typ existuje vlastní komponenta, jedná se o `<app-card-edit-popup>` a `<app-card-create-popup>`.

Obě z těchto komponent však využívají znovupoužitelnou komponentu `<app-card-popup>`. Ta zajišťuje především integritu obou typů modálních oken detailu.

V případě zobrazení již existující karty je obsah okna rozdělen do dvou záložek. První obsahuje informace o úkolu a druhá zase obsahuje historii akcí, prováděných s touto kartou. Při vytváření nové karty je použita jen první z těchto záložek. Názorná ukázka zmiňovaného okna je prezentována snímkem 5.4.



Obrázek 5.4: Ukázka části aplikace s otevřeným detailem karty úkolu.

5.1.5 Nastavení

Ačkoliv některá nastavení se v současné aplikaci provádí editací dat v databázi, existuje i několik případů, kdy se nastavení nachází přímo v aplikaci a je realizováno pomocí uživatelského rozhraní. Implementace takových nastavení je popsána v této podkapitole.

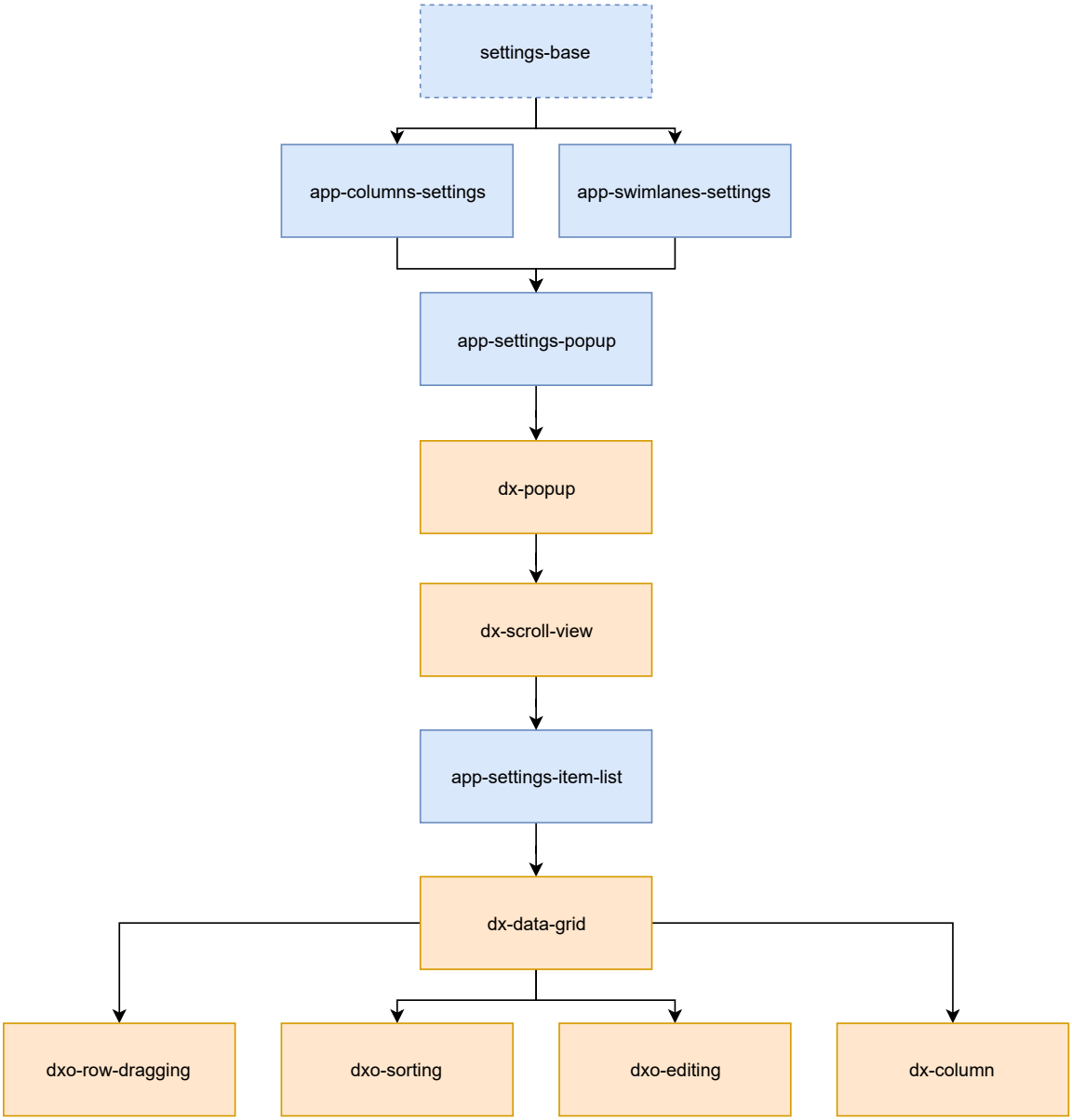
Nastavení štítků

Modální okno s nastavením štítků cíleně porušuje jedno z pravidel principu Flux. Uchovává totiž lokální kopii sdílených dat uložených ve skladu. Tento přístup jsem zvolil z jednoho prostého důvodu a to, aby se v případě zahození změn nemusel stav štítků napříč celou aplikací změnit. Místo toho jsem se přiklonil k opačnému řešení. Při úpravě štítků uživatel pracuje pouze s kopií sdílených dat. Jeho změny se tak neprojevují v jiných komponentách. V případě zahození změn při zavření dialogu tudíž není nutné provést vůbec žádnou akci. Obdobně se chovají i další nastavení v aplikaci.

Sdílené komponenty

Některá nastavení jsou si vzájemně velice podobná. Z toho důvodu bylo nastavení rozděleno do několika menších komponent a několik z nich tak lze použít například jak v nastavení sloupců, tak i v nastavení plaveckých drah.

Hierarchie těchto komponent je vyobrazena na snímku 5.5. Detailní popis jednotlivých komponent a jejich funkcí je popsán dále v této podkapitole.



Obrázek 5.5: Diagram znázorňující hierarchii jednotlivých komponent a tříd využitým pro realizování modálních oken pro nastavení nástěnky. Vlastní části řešení jsou označeny modrou barvou a komponenty knihovny DevExtreme jsou označeny oranžově. Komponenty jsou znázorněny plným ohraničením a abstraktní třída čárkovaným.

Třída SettingsBase

Jedná se o abstraktní třídu, z které dědí konkrétní komponenty pro jednotlivá nastavení jako například `<app-columns-settings>` pro nastavení sloupců nebo `<app-swimlanes-settings>` pro nastavení řádků. Hlavním účelem této abstraktní třídy je sjednotit způsob propagace otevření modálního okna.

Komponenta app-settings-popup

Veškerá nastavení nástěnky probíhají za pomoci modálních oken. Samotné grafické zobrazení oken je realizováno pomocí komponenty knihovny DevExtreme a to konkrétně `<dx-popup>`. Tato komponenta je však rozšířena o varování při pokusu o zavření okna s neuloženými daty. Zmíněné chování je součástí implementace komponenty `<app-settings-popup>`.

Komponenta app-settings-item-list

Další opakovaně se vyskytující částí nastavení je datová mřížka (angl. *data grid*). Tato mřížka by se dala označit jako rozšířená tabulka s akcemi umožňující manipulaci a editaci dat. Stejně jako u předchozí komponenty je pro grafické zobrazení této mřížky využita knihovna DevExtreme a její komponenta `<dx-data-grid>`. Komponenta mimo pouhé zobrazení umožňuje i samotnou práci s daty, nicméně během implementace byl zjištěn nedostatek při současném využití řazení dle sloupce a možnosti měnit pořadí řádků.

Při úpravě hodnot se mění pouze lokální kopie dat, kterou si komponenta uchovává. Tyto hodnoty se v původním zdroji dat projeví až po uložení. Nicméně řazení probíhá podle zdrojových dat a ne podle dat lokálních.

Především kvůli vlastní implementaci změny pořadí byla vytvořena komponenta `<app-settings-item-list>`. Mimo jiné slouží pro unifikaci nastavení samotné datové mřížky, které je realizováno pomocí vnořených elementů uvnitř komponenty `<dx-data-grid>`. Dále však obsahuje celou řadu metod, které téměř kompletně mění standardní chování komponenty mřížky. Realizuje zejména změnu pořadí řádků a vlastní chování tlačítek pro přidání a mazání řádků, obnovení změn a jejich ukládání.

Změna pořadí dat tedy v mé implementaci mění hodnoty jak v lokální kopii dat uvnitř komponenty `<dx-data-grid>`, tak i v jejím datovém zdroji. Jelikož je nutné neuložené změny projevovat i v datovém zdroji, tak existuje i třetí kopie dat, kterou si uchovává komponenta `<app-settings-item-list>` pro případnou obnovu dat při zahození změn.

Tlačítko pro smazání záznamu celý řádek odstraní z mřížky. Původní chování byla pouze změna barvy. Tlačítko pro přidání nového záznamu bylo pozměněno opět z důvodu zachování možnosti změny pořadí řádků. Tlačítko uložení pak bylo nutné rozšířit z důvodu absence události uložení dat v komponentě `<dx-data-grid>`. Pro případ použití v této aplikaci nestačí změnit hodnoty v datovém zdroji, ale je nutné změny projevit i v databázi na serveru a právě tento požadavek je nyní odeslán pomocí tlačítka uložení.

5.1.6 Asynchronní požadavky

Jak již bylo zmíněno v kapitole 3.1, jednostránkové aplikace fungují hlavně díky asynchronním požadavkům. Ty jsou tedy samozřejmě nedílnou součástí i této aplikace.

Po přihlášení uživatele každý jeho asynchronní požadavek v hlavičce protokolu HTTP[1] obsahuje žeton standardu JWT. O tuto funkci se stará třída `JwtInterceptor`, umístěná ve složce `interceptors`. Třída naslouchá všem odesílaným požadavkům protokolu HTTP a v případě, že je žeton již dostupný, je před samotným odesláním požadavku nejprve rozšířena hlavička tohoto požadavku. Do pole `Authorization` je vloženo slovo `Bearer` a za mezerou následuje samotný žeton.

Ve stejné složce lze také nalézt další třídu `ErrorInterceptor`, která obstarává oznámení uživateli v případě negativní odpovědi na odeslaný požadavek. Tato třída byla užitečná obzvláště během vývoje, jelikož chyby jazyka JavaScript se zobrazují v konzoli internetového prohlížeče, tudíž nejsou na první pohled zřetelné. Nicméně tato třída přináší užitek i pro koncové uživatele ve výsledné verzi aplikace.

V případě úspěšné odpovědi na asynchronní požadavek jsou data obsažená v této zprávě uložena do skladu dat. Tento proces je podrobněji vysvětlen v následující kapitole.

O samotné vytvoření zmíněných požadavků se starají služby `AuthenticationService`, `SSBMethodService` a `SSBQueryService`. Jejich chování lze částečně konfigurovat v souborech s prostředím umístěných ve složce `environments`.

5.1.7 Centrální úložiště

Data sdílena mezi více komponentami jsou uložena v centrálním úložišti stavu aplikace, který je realizován pomocí knihovny NgRx. Toto úložiště bývá také označováno jako sklad (angl. *store*). Při implementaci byl kladen důraz na co nejmenší zanořování dat, která jsou v tomto skladu přítomna. Každé sdílené entitě (např. nástěnkám, uživatelům, štítkům) tedy odpovídá právě jeden objekt v první úrovni zanoření.

Každý z těchto objektů je pak tvořen dalšími třemi hodnotami. Klíč `loading` obsahuje logickou hodnotu, která indikuje, zda se data právě načítají. Tato hodnota je kladná v období mezi odesláním asynchronního požadavku a přijmutím odpovědi od serveru. Uživateli tak může být dočasně skryt obsah části stránky, který je závislý na stále ještě nedostupných datech. V případě kladné odpovědi jsou přijatá data převedena do vhodného tvaru a poté uložena jako pole či objekt do hodnoty označené klíčem `data`. V případě neúspěchu dotazu je pak chyba uložena do hodnoty pod klíčem `error`.

O zmíněné převedení formátu se stará třída `SSBQueryDataResultParser`. Ta převádí nevhodný tvar dat přenášených pomocí aplikačního rozhraní, kde je informace o sloupcích a datech rozdělena do dvou polí. Metoda této třídy z těchto dvou polí vytvoří jedno pole obsahující objekty, ve kterých jsou již názvy sloupců a jejich hodnoty v podobě, jaká je u objektů typu JSON obvyklá.

Ani tato podoba dat však není finální. Výstupem zmíněné metody jsou totiž generické objekty. Pro veškeré přenášené entity však v aplikaci existuje odpovídající třída ve složce `models`. Generické objekty sice v aplikaci ve většině případů fungují, ale chybí jím specifické metody. V případě modelu uživatele lze například využít virtuální atribut `fullName`, který je ve skutečnosti pouze složenina dvou jiných atributů. Tento atribut by tedy v případě generického objektu dostupný nebyl. To je důvodem proč je využita funkce `plainToClass` knihovny `class-transformer`[2], která data dále převede do finální podoby.

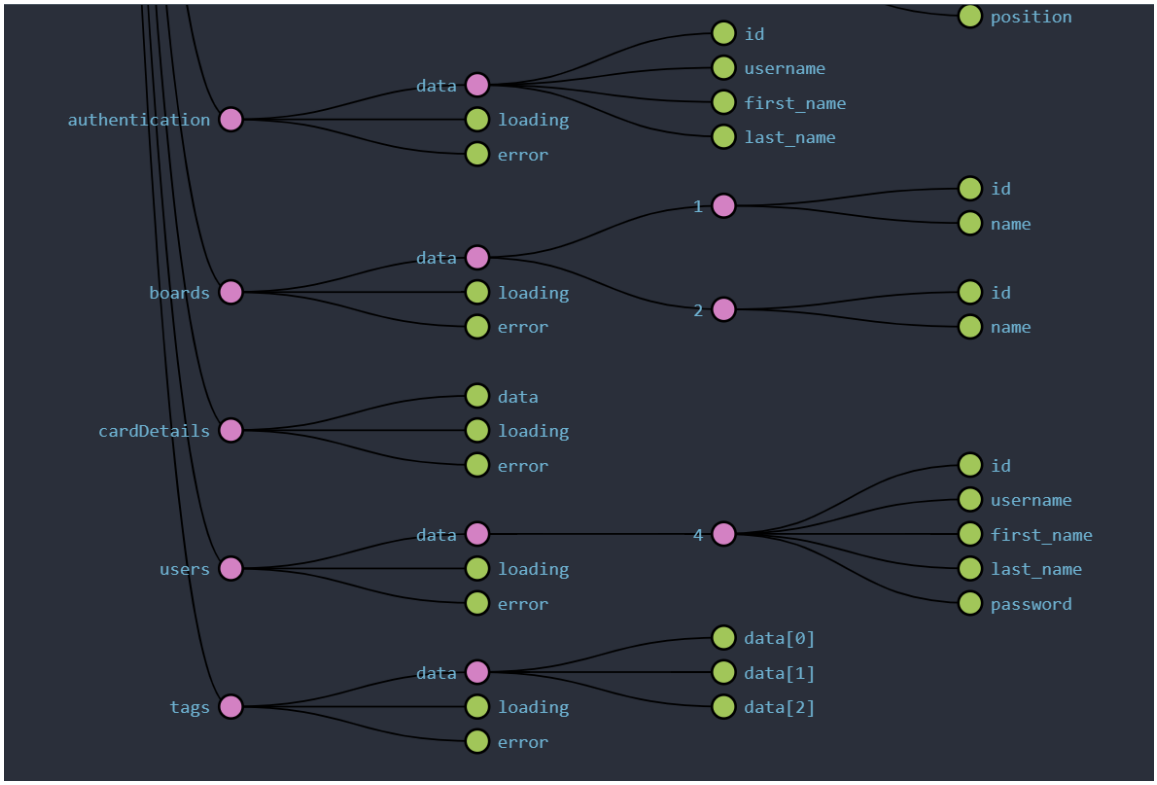
O samotné ukládání přijatých dat a změnu obsahu skladu se zase starají reduktory (angl. *reducers*), které využívají zmíněnou třídu pro převod dat.

[1] Hypertext Transfer Protocol neboli Hypertextový přenosový protokol

[2] Repozitář knihovny class-transformer: https://github.com/typestack/class-transformer

Okamžitá vizuální odezva je však pro uživatele velmi důležitá. V některých případech je tak tedy nutné data ve skladu změnit dříve, než je přijata odpověď od serveru. Je tomu tak například u přesunu karty. Jistě by bylo pro uživatele velice nepříjemné, kdyby přesunul kartu na nové místo a ta se vzápětí ihned vrátila na svou původní pozici a až po nějaké době by akci potvrdil server a karta by se přesunula na místo, kam uživatel původně zamýšlel. Proto je v takových případech předpokládán úspěch akce a data ve skladu jsou ihned změněna. I tuto práci obstarává reduktor.

Pro lepší ilustraci je níže uveden snímek 5.6, který ilustruje část dat aplikace uložených právě v popisovaném centrálním skladu.



Obrázek 5.6: Vizualizace dat obsažených v centrálním datovém skladu. Snímek je pořízen z nástroje Redux DevTools[4].

Další dvě nedílné součásti knihovny NgRx jsou efekty a akce. Jejich účel byl již popsán v kapitole 3.2.3. Každé ze sdílených entit naleží vlastní trojice těchto součástí knihovny. Obsah asynchronních požadavků je sestaven přímo v souboru s efekty, odkud jsou následně volány služby, které mají na starost tyto požadavky odeslat aplikaci SSBLite.

5.2 Serverová část

Z důvodu zachování firemního tajemství klientská část mé práce nekomunikuje přímo se systémem SSB, ale pouze se zjednodušenou verzí zvanou SSBLite, kterou jsem vytvořil v rámci řešení této bakalářské práce. Jedná se o webový server vytvořený s použitím systému

[4]Redux DevTools: https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd

ASP.NET. Server zpracovává požadavky aplikačního rozhraní REST na základě kterých sestrojuje a odesílá požadavky na databázi. Výsledky těchto požadavků jsou poté odeslány zpět klientovi ke zpracování a zobrazení uživateli.

Samotný webový server kromě autentizace uživatele neobsahuje žádnou logiku aplikace. Slouží pouze jako zprostředkovatel přístupu k databázi, ke které klient z důvodu bezpečnosti nesmí mít přímý přístup. Možné požadavky na databázi jsou předem definovány a uloženy v konfiguračních souborech. Díky tomu je možné v budoucnu upravit chování aplikace bez nutnosti přímého zásahu do zdrojového kódu serverové části aplikace.

5.2.1 Struktura adresáře

V této podkapitole je popsán adresář obsahující zdrojové aplikace SSBLite. Některé nepodstatné soubory a adresáře nejsou v seznamu uvedeny. Jedná se o podobu adresáře bez vygenerovaných souborů, tak jako na přiloženém médiu. Po úspěšném překladu se vytvoří nové soubory uvnitř adresáře `bin`.

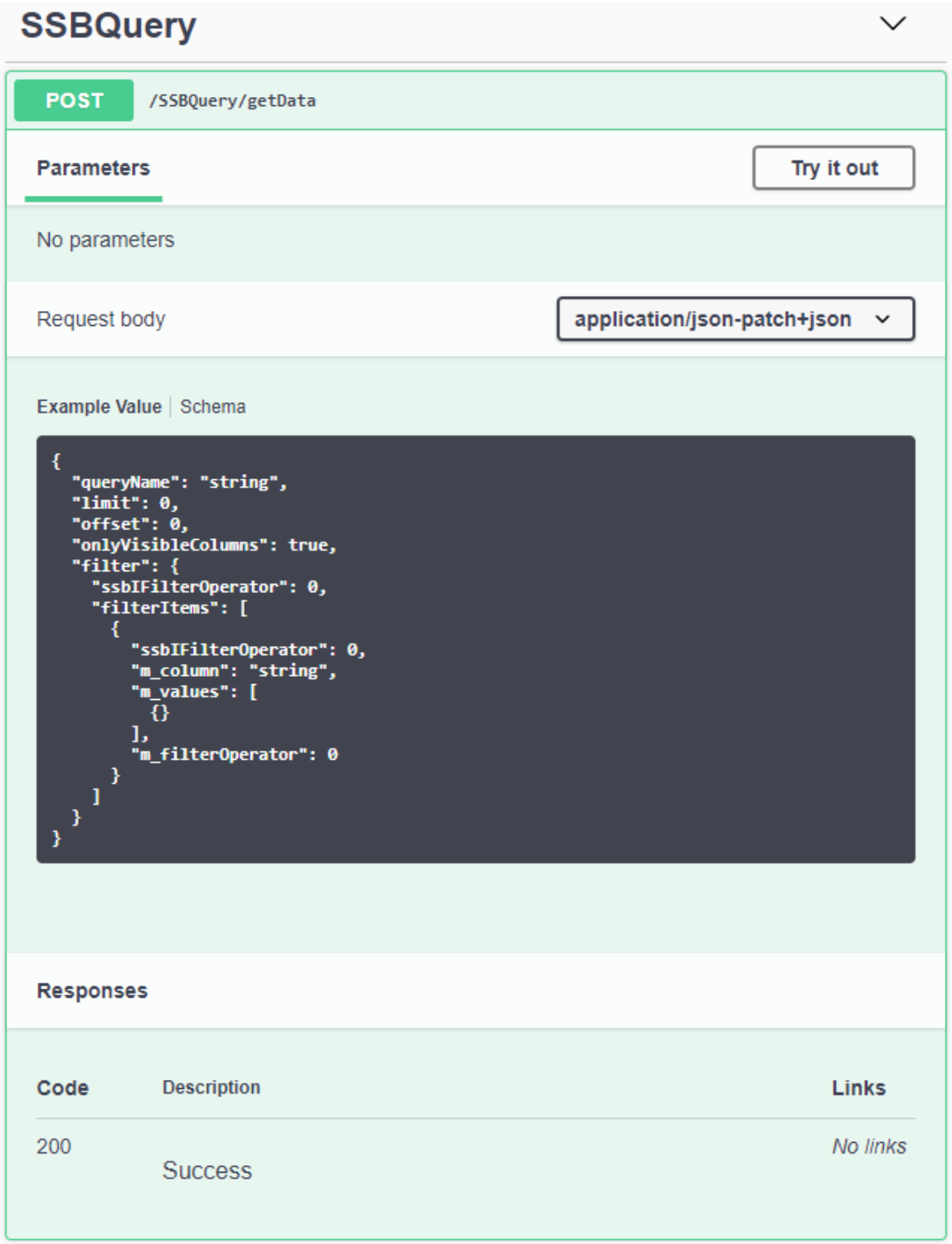
- `Configurations/` – Adresář obsahující konfigurační soubory metod a dotazů na databázi
- `Controllers/` – Adresář obsahující řídící třídy
- `Enums/` – Adresář obsahující objekty výčtů, používaných aplikačním rozhraním
- `Helpers/` – Adresář obsahující třídu, která umožňuje přístup k tajnému klíči aplikačního rozhraní
- `Interfaces/` – Adrsář s rozhraními
- `Models/` – Adresář s třídami objektů modelů
  - `Authentication/` – Modely týkající se autentizace uživatele
  - `Method/` – Modely týkající se metod a příkazů databázi
  - `Query/` – Modely týkající se dotazů na databázi
- `Services/` – Adresář se službami získávající data z databáze
- `appsettings.json` – Soubor s konfigurací aplikace
- `Startup.cs` – Soubor se spouštěcí třídou s další konfigurací

5.2.2 Rozhraní Swagger

V raných etapách vývoje, kdy ještě nebyla klientská a serverová část aplikace vzájemně propojena, byl pro testování vývoje používán rámec Swagger[5]. Díky jeho využití byly možnosti a podoba aplikačního rozhraní přehledně uvedena na jednom místě, odkud bylo také možné jednotlivé požadavky odesílat.

Tento nástroj je ve výsledné aplikaci stále dostupný při překladu aplikace pro vývojové prostředí. Vzhledem k absenci jeho použití v pozdějších fázích vývoje však neobsahuje nejaktuálnější změny v aplikaci, i tak jej však lze využít zejména pro orientaci v aplikačním rozhraní. Ukázka části tohoto rozhraní je vidět na snímku 5.7.

[5]Swagger: https://swagger.io



Obrázek 5.7: Popis části aplikačního rozhraní umožňujícího dotazovat se na data uložená v aplikaci znázorněný v uživatelském rozhraní rámce Swagger.

5.2.3 Konfigurační soubory

Konfigurační soubory využívají formát JSON a obsahují názvy jednotlivých požadavků a jejich znění. To však v mnoha případech přímo neobsahuje platný dotaz na databázi, ale obsažený dotaz je obohacen o klíčová slova, která umožňují přizpůsobení jeho znění pro aktuální potřebu klienta. Před samotným odesláním dotazu do databáze je nutné tyto klíčová slova zpracovat.

5.2.4 Zpracování požadavku GetData

Požadavek `GetData` je určen pro dotazování se nad daty uloženými v databázi. Požadavky tohoto typu jsou zpracovány pomocí třídy `SSBQueryController`, která na základě názvu obsaženého v těle požadavku dohledá v konfiguračních souborech příslušný dotaz. V dalším kroku jsou pak nahrazena klíčová slova ve znění dotazu a to konkrétně `{WHERE}` a `{AND}`, místo kterých se dotaz rozšíří o další podmínky na základě příslušného obsahu požadavku `GetData`. Příkladem, využívajícím klíčové slovo `{WHERE}`, je následující dotaz:

```
{
    "name": "getCards",
    "content": "SELECT car.* FROM cards car JOIN columns col ON col.id =
        car.column_id JOIN boards boa ON boa.id = col.board_id {WHERE}"
}
```

Ten má za následek získání informací o existujících kartách. Klíčové slovo v tomto případě bývá nahrazeno specifikací, o kterou nástěnku má klient zájem. Zpravidla se jedná o tu právě zobrazenou v jeho prohlížeči. Klíčové slovo `{AND}` se využívá v případě, že dotaz již nějaké podmínky obsahuje a ty požadované klientem jsou zřetězeny za již existující.

Dalším klíčovým slovem s kterým se lze setkat je `{USERID}`, které je nahrazeno identifikačním číslem klienta, který se na data dotazuje. V tomto případě klient již nemá možnost ovlivnit, jakou hodnotou bude klíčové slovo nahrazeno, a proto je vhodné tento prvek využít například pro zamezení přístupu k datům, které uživateli nenáleží. Nahrazení klíčových slov obstarává metoda `ReplaceKeywords`, jenž je součástí třídy `SSBQuery`.

Dalším krokem je pak samotné provedení dotazu a transformace získaných dat do formátu, jež odpovídá specifikaci aplikačního rozhraní. Tyto data jsou pak navrácena klientovi.

V aplikacích využívajících platformu ASP.NET bývá zvykem pro sestavení dotazů použít technologii LINQ, v tomto případě však použita není. Místo ní je využívána knihovna Npgsql a dotazy jsou sestrojovány přímo v cílovém jazyce PostgreSQL. Tento přístup bylo nutné aplikovat z důvodu zachování integrity se systémem SSB.

5.2.5 Zpracování požadavku ExecuteMethod

Funkcí požadavku `ExecuteMethod` je spuštění příkazu databáze. Může se jednat přímo o příkaz například `INSERT` či `UPDATE`, nicméně jeho nejčastější využití je spuštění uložené procedury pomocí příkazu `CALL`.

Samotné zpracování je téměř totožné jako u požadavku `GetData`. V tomto případě se však využívají třídy `SSBMethodController` a `SSBMethod`. V konfiguračních souborech příkazů se však vyskytuje z dříve uvedených klíčových slov pouze `{USERID}`. Jsou zde však přítomny i zástupné symboly s poznávacím znamením dvojtečky jako prvního znaku. Ihned za ní pak následuje název parametru. V konfiguračních souborech tak lze nalézt například následující záznam:

```
{
    "name": "createCard",
    "content": "CALL create_card(:columnId, :swimlaneId, :title,
                :description, :assignedUserId, :tagIds)"
}
```

Tyto zástupné symboly jsou nahrazeny hodnotami, které jsou obsaženy v požadavku zaslaným klientem. O tuto práci se stará metoda `FillParameters`, která nejprve zaslaná data převede do odpovídajícího typu a následně je za pomocí funkce knihovny Npgsql vloží do příkazu místo zástupných symbolů. Tato knihovna mimo jiné zabraňuje případným pokusům o napadení databáze za pomocí metody SQL injekce.

Po provedení příslušné akce je klientovi oznámeno zda jeho požadavek proběhl úspěšně či nikoliv.

5.2.6 Autentizace uživatele

Pro ověření přístupových údajů uživatele slouží požadavek `VerifyCredentials`. Ten je zpracován třídou SSBAuthenticationController, ale samotná autentizace probíhá až ve třídě UserService. Tato třída obstarává jak ověření zaslaných údajů s těmi v databázi, tak i vytvoření žetonu standardu JWT. Pro přístup k databázi je v tomto případě již využita technologie LINQ a samotný dotaz není načten z konfiguračních souborů. Vygenerovaný žeton je poté odeslán zpět k uživateli.

5.3 Databázová vrstva

Jak již bylo zmíněno, databáze využívá systém PostgreSQL. Ve většině podobných aplikací databáze slouží pouze pro úschovu dat. Nicméně kvůli univerzálnosti systému SSB je nutné v databázi uchovávat i logiku aplikace. Toho je docíleno pomocí uložených procedur. Koncepce uložení dat v databázi byla již zmíněna v podkapitole 4.2.

Jednodušší procedury jsou psány přímo v jazyce SQL a jedná se především o sekvenci vícero klasických příkazů, které lze spustit pomocí jednoho příkazu (spuštěním uložené procedury). Tento přístup je využit například při vytvoření nového štítku, kdy je část dat uložena v tabulce `cards` a část v tabulce `card_details`.

Pro komplexnější akce bylo však nutné využít jazyk PL/pgSQL[6]. Ten umožňuje používat například funkce nebo podmínky.

Dle požadavku zadavatele práce názvy uložených procedur dodržují konvenci, kdy v první části názvu je uvedena entita, které se daná operace týká a v druhé části je název akce. Lze se tedy setkat s názvy jako je `tas_sync`.

5.3.1 Synchronizace záznamů

V aplikaci je několik situací kdy je třeba synchronizovat záznamy. Jedná se o kombinaci tří akcí s daty – úprava, odstranění existujících záznamů a vytvoření nových. Tato situace nastává zejména při nastavování nástěnky. Konkrétně při nastavení sloupců, plaveckých drah a štítků. Tyto akce jsou provedeny pomoci uložených procedur `column_sync`, `swimlane_sync`, `tag_sync`.

[6] PL/pgSQL: https://www.postgresql.org/docs/9.6/plpgsql.html

Uvnitř této procedury se však nachází pouze volání další uložené procedury a to jedné a té stejné pro všechny zmíněné procedury. Jedná se o proceduru `sync_handler`, která umožňuje dynamicky provádět právě synchronizaci dat. Jejími parametry jsou jednak vstupní data původních procedur (identifikátor nástěnky a data k synchronizaci) a dále také specifikace nástěnky, kde jsou data uložena, a sloupce, kterých se synchronizace týká.

Díky tomu je značně zredukována redundance kódu a v kombinaci s generickým komponentami klientské aplikace lze celý systém jednoduše rozšířit o nastavení dalších prvků.

5.3.2 Autorizace požadavků

Při vytváření téměř jakékoliv aplikace je nutné počítat s možnými pokusy o její napadení. Požadavky uživatele je tedy nutné vždy autorizovat.

Dotazy na data lze omezit použitím klíčového slova `{USERID}`, představeného v kapitole 5.2.4. Do dotazu lze připojit i tabulku `users` například prostřednictvím propojovací tabulky `boards2users` a pomocí zmíněného klíčového slova vyfiltrovat jen ty tabulky, ke kterým má mít uživatel přístup.

Pro omezení spouštění metod je taktéž využito uvedené klíčové slovo. V tomto případě lze funkci předat údaje o uživateli jako parametr a následně v těle funkce ověřit jeho privilegia k provedení příslušné akce. Nicméně vzhledem k existující implementaci ukládání historie je jednodušší využít proměnnou aktuálního sezení přístupu k databázi.

5.3.3 Ukládání historie

Jeden z požadavků vývoje aplikace bylo zaznamenávání změn karty. Tato funkcionalita je implementována pomocí spouští, které jsou provedeny před samotným zápisem do klíčových tabulek jako `cards` či `card_details`. Spoušť v případě změny některých polí uloží jejich původní hodnoty do tabulky `changes`. Informace o uživateli, který je strůjcem dané akce, jsou spoušti předány pomocí proměnné aktuálního sezení přístupu k databázi a rovněž zaznamenány.

Kapitola 6

Testování a vyhodnocení

Tato kapitola popisuje provedené způsoby testování výsledné aplikace se zaměřením na uživatelské testování, které bylo prováděno ve spolupráci s několika zaměstnanci společnosti SEACOMP s.r.o. Následně je popsán výsledek testování a akce, které byly na základě zpětné vazby respondentů učiněny. V závěru kapitoly jsou uvedena další možná rozšíření aplikace.

Automatizované testování nebylo při práci využito. Bylo tak rozhodnuto po konzultaci jak s vedoucím tak i se zadavatelem práce. Pro ověření funkčnosti, bezpečnosti a dalších aspektů aplikačního rozhraní bylo využito manuální testování pomocí nástroje Postman[1], který slouží jako klient pro odesílání požadavků aplikačního rozhraní REST a pro zobrazení odpovědí na tyto požadavky.

6.1 Uživatelské testování

Testování na uživatelích se zúčastnilo 10 zaměstnanců společnosti SEACOMP s.r.o. Jednalo se jak o zástupce z řad vedoucích týmů, kteří v rámci své pozice pracují s nástěnkou Kanban dennodenně a mají na starost i její konfiguraci, tak i o řadové programátory, kteří tento nástroj využívají zřídka a čistě jen z uživatelského pohledu.

Vzhledem k současné pandemii koronaviru bylo nutné využít pouze prostředky pro vzdálenou komunikaci. Testování tak bylo rozděleno do dvou částí. První část obnášela samotné vyzkoušení aplikace uživatelem. Druhá část byla realizována pomocí dotazníku.

6.1.1 Testovací data

Pro každého účastníka uživatelského testování byl vytvořen vlastní účet a k němu také separátní nástěnka se základními sloupci a plaveckými drahami. K nástěnce bylo přiřazeno dalších deset uživatelů. Obsahem nástěnky však byla pouze jediná karta. Uživatelé tak tedy začali téměř s netknutou nástěnkou. Tyto testovací data byla vytvořena pomocí databázové procedury `test_table_create`.

6.1.2 Sledování chování uživatelů

Uživatel nebyl dopředu s aplikací nijak seznámen, protože vzhledem k zaměření jeho povolání byl předpoklad, že se s podobným nástrojem v minulosti již setkal.

Aplikace třetích stran umožňující zaznamenat pohyb myši a průchod aplikací bohužel nebylo možné využít, jelikož nedokázaly správně zpracovat kaskádové styly aplikace a vý-

[1] Postman: https://www.postman.com

sledný záznam tak byl takřka nepoužitelný. Náhradní řešení tedy bylo sledování vybraných uživatelů pomocí sdílení obrazovky prostřednictvím aplikace Skype[2].

6.1.3 Dotazník

Po vyzkoušení aplikace každý z testovacích uživatelů obdržel odkaz na elektronický dotazník realizovaný pomocí online nástroje Formuláře Google[3]. Otázky dotazníku byly rozděleny do dvou skupin a jejich znění je uvedeno v příloze B.

První skupina obsahovala deset výroků na téma použitelnosti aplikace a respondent byl požádán o vybrání hodnoty na škále od jedné do pěti, zda s daným výrokem souhlasí či nesouhlasí. Znění těchto výroků vychází z překladu standardizovaného dotazníku zvaného Měřítko použitelnosti systému[4] (angl. *System Usability Scale* neboli SUS).

Druhá skupina obsahovala pět otevřených otázek zaměřených na silné a slabé stránky aplikace a její srovnání s původní aplikací, která se ve společnosti SEACOMP s.r.o. právě využívá.

6.2 Vyhodnocení

Vyhodnocení bylo rozděleno do dvou částí. První část se věnuje čistě jen samotnému dotazníku Měřítka použitelnosti systému. Další část se zabývá poznatky zjištěnými druhou částí dotazníku a sledováním uživatelů a případně i jejich nápravou.

6.2.1 Měřítko použitelnosti systému

Jednotlivé výsledky dotazníku Měřítka použitelnosti systému lze pomocí vzorce převést na hodnotu v intervalu 0 až 100, která se označuje jako skóre. Následně je nutné zprůměrovat všechna vypočítaná skóre, čímž je získáno celkové skóre aplikace. Aby tuto hodnotu bylo možné vhodně interpretovat jako výsledek testování, je možné testovanému produktu přidělit známku na základě percentilu výsledku dalších 241 obdobných studií. Zmíněné známky jsou uvedeny v tabulce 6.1.

[2]Skype: https://www.skype.com/cs/

[3]Formuláře Google: https://www.google.com/intl/cs/forms/about/

[4]Měřítko použitelnosti systém: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Tabulka 6.1: Známkovací stupnice určená pro vyhodnocení výsledku dotazníku Měřítka použitelnosti systému. Přeloženo z [42].

Známka	Skóre SUS	Rozpětí percentilu
A+	84,1 – 100	96 – 100
A	80,8 – 84,0	90 – 95
A-	78,9 – 80,7	85 – 89
B+	77,2 – 78,8	80 – 84
B	74,1 – 77,1	70 – 79
B-	72,6 – 74,0	65 – 69
C+	71,1 – 72,5	60 – 64
C	65,0 – 71,0	41 – 59
C-	62,7 – 64,9	35 – 40
D	51,7 – 62,6	15 – 34
F	0 – 51,6	0 – 14

Z 8 výsledků dotazníku bylo vypočítáno průměrné skóre 78, což dle stupnice uvedené v tabulce 6.1 odpovídá známce B+.

Z odpovědí dále vyplynulo, že se jedná o jednoduchou, konzistentní a snadno použitelnou aplikaci, s kterou se uživatel naučí rychle pracovat. Převážně pozitivně bylo také vnímáno začlenění jednotlivých funkcí. Takřka nerozhodný výsledek se objevil u pocitu jistoty při práci s aplikací, stále však mírně převažovaly pozitivní výsledky. Podobně se respondenti vyjádřili i u otázky, zda by aplikaci rádi používali opakovaně. Tento výsledek je zapříčiněn tím, že většina respondentů při náplni své práce nástěnku Kanban téměř či vůbec nepožívají. Vedoucí týmů, čili klíčoví respondenti, tvořili ve vzorku respondentů pouze menšinu.

6.2.2 Sledování uživatelů a otevřené otázky

Uživatelské testování vedlo k objevení jak několika technických chyb, které byly po sléze opraveny, tak i k objevení několika menších nedostatků návrhu aplikace. Část těchto poznatků byla získána sledováním chování uživatelů při používání aplikace a část je založena na odpovědích na otevřené otázky dotazníku.

Většina připomínek se týkala drobných nedostatků jako je třeba nemožnost odeslat přihlašovací údaje stisknutím tlačítka *Enter* nebo problémy s kaskádovými styly, které byly po dokončení testování opraveny.

Z testování také vyplynulo, že ačkoliv se s metodikou Kanban všichni účastníci do jisté míry již dříve setkali, část z nich neznala koncept plaveckých drah. Proto byla do aplikace na vhodné místo přidána informační ikona se stručným popisem jejich principu.

Dále se ukázalo, že uživatelé ve webových aplikacích nejsou zvyklí na využití pravého tlačítka myši jako kontextového menu. Pro akce z toho menu tedy byla přidána i klasická tlačítka v detailu karty.

Pro vyhodnocení úspěchu aplikace je stěžejní srovnání s aktuálně používanou aplikaci ve společnosti. Obecně byla aplikace vytvořená v rámci této bakalářské práce vnímána jako mnohem přehlednější a vyzdvižena byla také rychlost její odezvy. Často zmiňovaným nedostatkem však byla absence některých funkcí současné aplikace Kanboard.

6.3 Možnost úprav

Mimo již několikrát zmíněnou integraci do stávajícího systému společnosti lze aplikaci vzhledem k její rozsáhlé povaze rozšířit o celou řadu jednoduchých i komplexních funkcí.

Při větším množství úkolů přítomných na jedné nástěnce může docházet k částečné nepřehlednosti informací. Zcela jistě by tedy bylo užitečné přidat možnost filtrovat zobrazené úkoly. Pro sestavení takového filtru by bylo možné využít například komponentu `<dx-filter-builder>` knihovny DevExtreme, případně je možnost implementovat přímo vlastní dotazovací jazyk.

Další možností rozšíření by mohla být možnost částečné automatizace nástěnky, například automatické archivování hotových úkolů. Díky této funkci by správci nástěnek nebyli nuceni provádět tolik rutinních úkonů.

Jelikož v budoucnu může být nástroj využíván v nejrůznějších odvětvích průmyslu, nabízí se umožnit uživatelům plně si přizpůsobit obsah úkolů pomocí konfigurovatelných polí jako je tomu například v nástroji Jira. Tím by bylo docíleno možnosti jednoduše upravit nástěnku pro potřeby různorodých zákazníků.

Konkurence v oboru těchto aplikací je veliká. Pro případnou migraci z nebo do jiného nástroje s podobných zaměřením by také mohlo být užitečné umožnit importovat či exportovat úkoly a případně i nastavení nástěnky.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvořit webovou aplikaci typu Kanban dle požadavků společnosti SEACOMP s.r.o., která v budoucnu nahradí v současnosti využívaný nástroj třetí strany. Základní verze této aplikace byla úspěšně vytvořena a v nynější podobě ji lze použít pro řízení projektů. Toto tvrzení bylo ověřeno uživatelským testováním. Před jejím reálným nasazením ve společnosti je však nutné nejprve ji zaintegrovat do aplikace SSB4Web a rozšířit o několik pokročilých funkcí.

Výsledná aplikace obsahuje klíčové funkce pro správu nástěnky a v ní obsažených úkolů. Během vývoje byla dodržena všechna omezení určené zadavatelem práce jako jsou architektura a technologie aplikace.

Práce pro mne měla osobní přínos především novou zkušeností s tvorbou jednostránkových aplikací, osvojením si nových rámců pro vývoj webových aplikací a realizací řešení se striktními omezeními způsobu implementace.

Výsledná aplikace bude v budoucnu sloužit společnosti SEACOMP s.r.o. pro interní řízení vývoje softwarových produktů.

Literatura

[1] Allen, I. The Brutal Lifecycle of JavaScript Frameworks. *StackOverflow Blog* [online]. Leden 2020 [cit. 2020-04-14]. Dostupné z: https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/.

[2] Atlassian. The Agile Coach. *Atlassian | Software Development and Collaboration Tools* [online]. 2020 [cit. 2020-03-22]. Dostupné z: https://www.atlassian.com/agile/kanban/boards.

[3] Atlassian. Jira Software vs Trello Comparison. *Atlassian | Software Development and Collaboration Tools* [online]. 2020 [cit. 2020-01-26]. Dostupné z: https://www.atlassian.com/software/jira/comparison/jira-vs-trello.

[4] Atlassian. Jira | Issue & Project Tracking Software. *Atlassian | Software Development and Collaboration Tools* [online]. 2020 [cit. 2020-02-05]. Dostupné z: https://www.atlassian.com/software/jira.

[5] Atlassian. Ceník Jira – cena měsíčního a ročního předplatného na uživatele. *Atlassian | Software Development and Collaboration Tools* [online]. 2020 [cit. 2020-04-04]. Dostupné z: https://www.atlassian.com/cs/software/jira/pricing.

[6] Atlassian. Customers. *Atlassian | Software Development and Collaboration Tools* [online]. 2020 [cit. 2020-04-06]. Dostupné z: https://www.atlassian.com/customers.

[7] Atlassian. About. *Trello* [online]. 2020 [cit. 2020-01-26]. Dostupné z: https://trello.com/about.

[8] Atlassian. Ceník Trella. *Trello* [online]. 2020 [cit. 2020-04-04]. Dostupné z: https://trello.com/cs/pricing.

[9] Auth0. Introduction to JSON Web Tokens. *JSON Web Tokens – jwt.io* [online]. 2020 [cit. 2020-04-08]. Dostupné z: https://jwt.io/introduction/.

[10] Azanha, A., Argoud, A. R., Camargo Junior, J. de a Antoniolli, P. Agile project management with Scrum: A case study of a Brazilian pharmaceutical company IT project. *International Journal of Managing Projects in Business*. Leden 2017, roč. 10.

[11] Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W. et al. *Manifest Agilního vývoje software* [online]. 2020 [cit. 2020-02-23]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html.

[12] Bierman, G., Abadi, M. a Torgersen, M. Understanding TypeScript. In: Jones, R., ed. *ECOOP 2014 – Object-Oriented Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, s. 257–281. ISBN 978-3-662-44202-9.

[13] Cheng, T. a Podolsky, S. *Just-in-Time Manufacturing: An introduction*. Springer Netherlands, 1996. ISBN 9780412735400.

[14] Dave Gavigan. The History of Angular. *Medium – Get smarter about what matters to you.* [online]. 2020-04-03 [cit. 2020-02-10]. Dostupné z: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7.

[15] Developer Express Inc.. DevExtreme Features. *DevExtreme* [online]. 2020 [cit. 2020-05-17]. Dostupné z: https://js.devexpress.com/Overview/.

[16] Fendrych, A. System Usability Scale česky. *Použitelnost.info – Boty a botky v použitelnosti* [online]. Červen 2015 [cit. 2020-04-21]. Dostupné z: https://www.pouzitelnost.info/2015/06/03/system-usability-scale-cesky/.

[17] Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, 2006. ISBN 9780596554477.

[18] Flora, H., Wang, X. a Chande, S. Adopting an Agile Approach for the Development of Mobile Applications. *International Journal of Computer Applications*. Květen 2014, roč. 94, s. 43–50.

[19] Fukizi, K., De Oliveira, J. a Bruchet, M. *Learn ASP.NET Core 3: Develop modern web applications with ASP.NET Core 3, Visual Studio 2019, and Azure, 2nd Edition*. Packt Publishing, 2019. ISBN 9781789618877.

[20] Google. Introduction to components and templates. *Angular* [online]. 2020 [cit. 2020-05-17]. Dostupné z: https://angular.io/guide/architecture.

[21] Google. Using OAuth 2.0 for Server to Server Applications. *Google Developers* [online]. 2020 [cit. 2020-04-08]. Dostupné z: https://developers.google.com/identity/protocols/oauth2/service-account.

[22] Guillot, F. kanboard/kanboard: Kanban project managment software. *GitHub* [online]. 2020 [cit. 2020-03-19]. Dostupné z: https://github.com/kanboard/kanboard.

[23] Guillot, F. *Kanban Project Managment Software – Kanboard* [online]. 2020 [cit. 2020-03-19]. Dostupné z: https://kanboard.org/.

[24] Jadhav, M. A., Sawant, B. R. a Deshmukh, A. Single Page Application using AngularJS. *International Journal of Computer Science and Information Technologies*. 2015, roč. 6, č. 3, s. 2876–2879. ISSN 0975-9646.

[25] Jiang, L. a Eberlein, A. An analysis of the history of classical software development and agile development. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. Oct 2009, s. 3733–3738. ISSN 1062-922X.

[26] Khanna, M. Why Web Applications Are Becoming Popular Than Standalone Applications? *Atlassian | Software Development and Collaboration Tools* [online]. Březen 2019 [cit. 2020-04-07]. Dostupné z: https://www.atlassian.com/customers.

[27] Kniberg, H. a Skarin, M. *Kanban and Scrum – Making the Most of Both.* C4Media Inc., 2010. ISBN 978-0-557-13832-6.

[28] Konik, J. Trello vs Jira: Settling a Sibling Rivalry in 2020. *Cloudwards* [online]. Leden 2020 [cit. 2020-04-06]. Dostupné z: https://www.cloudwards.net/trello-vs-jira/.

[29] Kumar, G. a Bhatia, P. Impact of Agile Methodology on Software Development Process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE).* Srpen 2012, roč. 2, s. 2249–6343.

[30] Lander, R. Announcing .NET Core 1.0. *.NET Blog* [online]. Červen 2016-06-27 [cit. 2020-02-12]. Dostupné z: https://devblogs.microsoft.com/dotnet/announcing-net-core-1-0/.

[31] Lynn, R. Kanban 101: Learn the Basics. *Planview Portfolio Managment and Work Managment Software* [online]. 2020 [cit. 2020-03-26]. Dostupné z: https://www.planview.com/resources/articles/kanban-101/.

[32] Microsoft. What is ASP.NET? *.NET* [online]. 2020 [cit. 2020-02-12]. Dostupné z: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet.

[33] NgRx Community. What is NgRx? *NgRx* [online]. 2020 [cit. 2020-02-11]. Dostupné z: https://ngrx.io/docs.

[34] NgRx Community. @ngrx/store. *NgRx* [online]. 2020 [cit. 2020-01-26]. Dostupné z: https://ngrx.io/guide/store#diagram.

[35] Noring, C. *Architecting Angular Applications with Redux, RxJS, and NgRx: Learn to build Redux style high-performing applications with Angular 6.* Packt Publishing, 2018. ISBN 9781787121751.

[36] Pryor, M. Trello Common Questions. *Trello Blog – Organize anything, together* [online]. Září 2011 [cit. 2020-04-14]. Dostupné z: https://blog.trello.com/trello-common-questions.

[37] RANDOM HOUSE, INC. Definition of Kanban. *Dictionary.com* [online]. 2020 [cit. 2020-01-12]. Dostupné z: https://www.dictionary.com/browse/kanban.

[38] Roberts, B. a Ryan, M. Reducing the Boilerplate with NgRx. *YouTube* [online]. Duben 2018 [cit. 2020-04-07]. Dostupné z: https://www.youtube.com/watch?v=t3jx0EC-Y3c.

[39] Romanato, G. Vanilla JavaScript: pros and cons. *Gabriele Romanato / Web development and stuff* [online]. Leden 2020 [cit. 2020-04-14]. Dostupné z: https://gabrieleromanato.name/vanilla-javascript-pros-and-cons.

[40] Saks, E. *JavaScript frameworks: Angular vs React vs Vue.* 2019. Diplomová práce. Haaga-Helia University of Applied Sciences. Dostupné z: http://urn.fi/URN:NBN:fi:amk-2019110720797.

[41] Salameh, H. What, When, Why, and How? A Comparison between Agile Project Management and Traditional Project Management Methods. *International Journal of Management Reviews.* Říjen 2014, Vol.2,.

[42] Sauro, J. a Lewis, J. *Quantifying the User Experience: Practical Statistics for User Research*. Elsevier Science, 2016. ISBN 9780128025482.

[43] SEACOMP s.r.o.. *Portfolio* [online]. 2019 [cit. 2019-12-05]. Dostupné z: https://seacomp.cz.

[44] Serrador, P. a Pinto, J. Does Agile work? — A quantitative analysis of agile project success. *International Journal of Project Management*. Březen 2015, roč. 33.

[45] Seymour, D. T. a Hussein, S. The History Of Project Management. *International Journal of Management & Information Systems (IJMIS)*. Září 2014, roč. 18, s. 233.

[46] Siriwardena, P. Base64 URL Encoding. In: *Advanced API Security: OAuth 2.0 and Beyond*. Berkeley, CA: Apress, 2020, s. 397–399. Dostupné z: https://doi.org/10.1007/978-1-4842-2050-4_20. ISBN 978-1-4842-2050-4.

[47] Stecky Efantis, M. 5 Easy Steps to Understanding JSON Web Tokens (JWT). *Medium – Get smarter about what matters to you.* [online]. 2020 [cit. 2020-04-08]. Dostupné z: https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec.

[48] Stones, R. a Matthew, N. *Beginning Databases with PostgreSQL: From Novice to Professional*. Apress, 2006. Books for professionals by professionals. ISBN 9781430200185.

[49] Teršl, A. *Webový klient pro nemocniční systém*. 2019. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Dostupné z: https://is.muni.cz/th/ggxzm/.

[50] The jQuery Foundation. *JQuery* [online]. 2020 [cit. 2020-04-14]. Dostupné z: https://jquery.com.

[51] The PostgreSQL Global Development Group. About. *PostgreSQL: The world's most advanced open source database* [online]. 2020 [cit. 2020-05-13]. Dostupné z: https://www.postgresql.org/about/.

[52] The PostgreSQL Global Development Group. Procedural Languages. *PostgreSQL: The world's most advanced open source database* [online]. 2020 [cit. 2020-05-18]. Dostupné z: https://www.postgresql.org/docs/9.1/external-pl.html.

[53] TOYOTA MOTOR CORPORATION. Development and Deployment of the Toyota Production System. *Toyota Motor Corporation Official Global Website* [online]. 2012 [cit. 2020-01-11]. Dostupné z: https://www.toyota-global.com/company/history_of_toyota/75years/text/entering_the_automotive_business/chapter1/section4/item4.html.

Příloha A

Obsah přiloženého média

Stručný přehled základního obsahu přiloženého média.

- **README.txt** - Textový soubor popisující obsah tohoto média a instrukce k instalaci
- **src/** - Adresář obsahující zdrojové kódy aplikace
- **thesis/** - Adresář obsahující písemnou zprávu a její zdrojové kódy
- **misc/** - Adresář obsahující výsledky uživatelského testování

Příloha B

Uživatelský dotazník

Znění otázek dotazníku použitého při uživatelském testování.

B.1 Měřítko použitelnosti systému

Český překlad standardizovaného dotazníku pro měření použitelnosti systému [16].

- Rád bych systém používal opakovaně
- Systém je zbytečně složitý
- Systém se snadno používá
- Potřeboval bych pomoc člověka z technické podpory, abych mohl systém používat
- Různé funkce systém jsou do něj dobře začleněny
- Systém je příliš nekonzistentní
- Řekl bych, že většina lidí se se systémem naučí pracovat rychle
- Systém je příliš neohrabaný
- Při práci se systémem se cítím jistě
- Musel jsem se hodně naučit, než jsem se systémem dokázal pracovat

B.2 Názor uživatele

Otevřené otázky zaměřené na subjektivní názor uživatele.

- Co se Vám na aplikaci nejvíce líbí?
- Co se Vám na aplikaci líbí nejméně?
- Jak aplikaci hodnotíte ve srovnání s aplikací Kanboard používanou v současnosti?
- Co Vám v aplikaci chybí?
- Co by se v aplikaci mohlo zlepšit?