VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA SYSTÉMOVÉHO INŽENÝRSTVÍ

Návrh a vývoj webové aplikace pro správu konferencí
Design and development of web application for conference management

Student: Bc. Filip Rutar
Vedoucí diplomové práce: Ing. Radek Němec, Ph.D.

Ostrava 2023

PODĚKOVÁNÍ

Rád bych touto cestou poděkoval svým blízkým za poskytnutou podporu při vypracovávání této práce a také vedoucímu této diplomové práce, panu Ing. Radku Němcovi, Ph.D., za jeho odborné vedení, přínosné rady a čas, který mi při vypracovávání této práce věnoval. Dále bych chtěl poděkovat paní Mgr. Ing. Lucii Chytilové, Ph.D., za skvělou spolupráci a poskytnuté informace, bez kterých by tato diplomová práce nemohla vzniknout.

Bc. Filip Rutar

Obsah

1 Úvod

V rámci mé diplomové práce jsem se rozhodl zaobírat tématem webových aplikací, jelikož se o toto téma velmi zajímám a tato tématika je součástí mého profesního života již několik let.

V poslední době lze pozorovat obrovský skok kupředu v oblasti vývoje a užívání webových aplikací. Tyto aplikace přinášejí svým uživatelům obrovské výhody hlavně ve své přístupnosti a využitelnosti. Vzhledem k tomu, že přístup k internetu již patří mezi neodmyslitelné prvky každodenního života a spousta lidí si život bez přístupu na internet již ani nedokáže představit, dá se očekávat, že se tímto směrem bude zaobírat i vývoj různých aplikací a systémů. Jako příklad mohou posloužit např. firemní systémy CRM, ERP, fakturační systémy nebo systémy na správu schůzek atd.

Hlavním cílem této diplomové práce je vyvinout novou webovou aplikaci, určenou k vytváření a řízení konferencí pro Ekonomickou fakultu Vysoké školy báňské – Technické univerzity v Ostravě. Na trhu existuje mnoho komerčních aplikací, které by tento účel dokázaly pokrýt, ale nakonec bylo rozhodnuto vytvořit aplikaci vlastní a přizpůsobit ji na míru budoucím uživatelům. Vývoj bude probíhat na bázi agilního přístupu, pomocí krátkých schůzek.

V první kapitole této diplomové práce budou objasněna teoretická a praktická východiska vývoje moderních webových aplikací. Budou zde popsány a vysvětleny veškeré technologie, které byly využité k vývoji aplikace. V další kapitole proběhne zhodnocení současného stavu a budou uvedeny požadavky na funkčnost vytvářené aplikace. Dále následuje kapitola, která se bude zabývat samotným návrhem a vývojem webové aplikace. Bude zde popsán vývoj veškerých částí aplikace. V předposlední kapitole proběhne uživatelské testování nově vyvinuté aplikace a proběhne zhodnocení výsledků a budou uvedeny případné návrhy pro zlepšení. Na závěr této diplomové práce bude zhodnoceno, jestliže bylo dosaženo všech jednotlivých cílů a proběhne shrnutí veškerých poznatků.

2 Teoretická a praktická východiska vývoje moderních webových aplikací

V této kapitole diplomové práce bude blíže objasněno, co pojem webová aplikace znamená a postupně popsán vývoj jednotlivých částí architektury webových aplikací. Budou zde uvedeny techniky vývoje Front-end a Back-end částí webových aplikací a také nebudou opomenuty principy vývoje databázové vrstvy aplikace.

2.1 Úvod do webových aplikací

Webovou aplikaci je možné označit jako aplikaci, ke které lze přistupovat prostřednictvím webových prohlížečů na každém zařízení s přístupem k internetu. Je umístěna na serveru, který poskytuje uživatelům přístup k aplikaci přes internet.

Existuje několik druhů webových aplikací, které mohou mít různé funkce, od jednoduchých webových stránek, které poskytují uživateli informace o různých produktech nebo službách, až po složité interaktivní aplikace, jako jsou například různé e-shopy, bankovní aplikace nebo sociální sítě.

Velkou výhodou webových aplikací je způsob aktualizace. U desktopových aplikací je nutné stahovat různé balíčky s aktualizacemi a samostatně tyto aktualizace na svém zařízení instalovat. U webové aplikace se pouze přehraje verze umístěná na externím serveru a konečný uživatel již nemusí nic řešit.

Důvodů, proč se upouští od využívání desktopových aplikací může být několik. Jedním z hlavních důvodů je jednoduchost užívání aplikace. Desktopové aplikace musejí být stále optimalizovány z důvodu existence velkého počtu operačních systémů, tudíž je nutné pro každý tento systém vytvářet samostatné verze, ať už se jedná o počítač či mobilní telefon. U aplikace webové se tento problém řešit nemusí, protože je umístěna na externím serveru a instalace na lokální zařízení již není tedy potřeba. Vše, co je vlastně potřeba k užívaní webové aplikace je zařízení, které má přístup k internetu. V tomto ohledu je nutné pouze aplikaci optimalizovat pro různé typy velikostí displejů těchto zařízení.

V minulosti byla většina webových aplikací vyvíjena frameworky, které běžely na webovém serveru a vykreslovaly webové stránky přímo do webového prohlížeče uživateli. Při každém provedení akce na stránce, webový prohlížeč neboli klient přenesl

požadavek pomocí HTTP protokolu na webový server a ten vykreslil novou stránku s jinými daty, která byla následně opět přes HTTP protokol odeslána zpět klientovi.

Protokol HTTP se postupem času začal využívat mnohem více s příchodem technologie AJAX, která umožňovala vytvářet požadavky asynchroně pomocí JavaScriptu.

Dnešní moderní webové aplikace jsou složeny z několika oddělených aplikací, které spolu komunikují pomocí REST API. Tato aplikační rozhraní existují pro předávání požadavků a odpovědí mezi jednotlivými aplikacemi. Jako největší rozdíl mezi moderními webovými aplikacemi a aplikacemi vyvíjenými v minulosti lze označit fakt, že dnešní aplikace jsou rozděleny na minimálně 3 nezávislé celky, kterými jsou Front-endová část, Back-endová část a databázová část aplikace (Hoffman, 2020).

Webová aplikace, které bude v rámci této diplomové práce vyvíjena lze označit jako single-page webovou aplikaci. Jedná se o typ webové aplikace, která vypadá a funguje jako jedna webová stránka. V tomto typu aplikace je prezentační vrstva řízena pouze webovým prohlížečem a ne serverem. Tento přístup má výhodu v tom, že aplikace, která je umístěna na serveru se nemusí starat o to, jak jsou data uživateli prezentovány.

V single-page webové aplikaci nejsou jednotlivé komponenty kompletní HTML stránky, ale pouze části DOM, které vytvářejí jednotlivé prvky. Po načtení úvodní stránky jsou všechny potřebné nástroje pro tvorbu a zobrazování komponent staženy a připraveny k použití. Když je zapotřebí zobrazit novou komponentu, je vygenerována lokálně ve webovém prohlížeči a dynamicky vykreslena do DOM JavaScriptem. To znamená, že není zapotřebí znovu načítat stránku (Scott, 2016).

2.2 Management vývoje webových aplikací

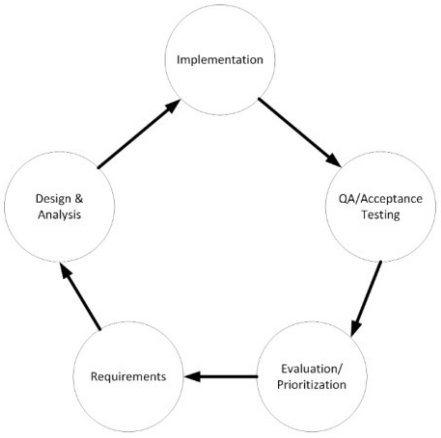
Aby byl celý proces vývoje vznikající webové aplikace úspěšný, je zapotřebí ho dobře řídit zodpovědnou osobou, která má v dané oblasti zkušenosti. V praxi se většinou jedná o projektového manažera, který zvolí určitou metodiku, která je pro daný projekt vhodná. Projektový manažer je osoba, která vytváří tzv. komunikační spojku mezi vývojovým týmem, zákazníkem a vedením organizace. Měl by být zapojen do celého životního cyklu vyvíjeného produktu od konceptuálního návrhu, přes vývoj, implementaci, až do výsledného uvedení aplikace na trh a následné propagování (Sommerville, 2020).

2.2.1 Tradiční a agilní metodiky

Pojem metodika chápeme jako souhrn určitých doporučených praktik a postupů, které pokrývají celý životní cyklus vyvíjené aplikace. V současné době lze pozorovat dvě hlavní skupiny metodických přístupů, které jsou označovány jako tradiční (rigorózní) metodiky a agilní metodiky.

Tradiční metodiky vycházejí z principu, že procesy, které jsou využity při vývoji, lze popsat, řídit a jsou měřitelné. Tyto metodiky jsou založeny na sériovém vývoji, což chápeme tak, že jednotlivé fáze probíhajícího vývoje probíhají postupně za sebou. Jako příklad tradičních metodik můžeme uvést například vodopádový model nebo model spirála (Buchalcevová, 2005).

Agilní metodiky vznikly v devadesátých letech dvacátého století jako reakce na nespokojenost s pomalou odezvou na funkcionální změny vyvíjeného softwaru podle tradičních metodik. Všechny agilní metodiky jsou založeny na inkrementálním vývoji a doručení zákazníkovi. Inkrementální vývoj lze chápat tak, že si software představíme jako soubor funkcionalit a každá funkcionalita má pro budoucího uživatele softwaru přidanou hodnotu. Po vytvoření a implementaci se dané funkcionality zákazníkovi předloží ke kontrole a následně podle zpětné vazby se dále upravují (Sommerville, 2020). Tento přístup je také nazýván iterativní metodikou. Každá iterace obsahuje část vyhotovených požadavků na celkový systém a na konci každé iterace je k dispozici fungující a otestovaná verze softwaru. V rámci vývoje této webové aplikace bylo rozhodnuto, se inspirovat tímto přístupem. Nejznámějšími agilními metodikami jsou SCRUM a Extrémní programování. Graficky si lze prohlédnout příklad iterativní metodiky na obrázku 2.1 níže.



Obr. 2.1 Iterativní metodika, zdroj: Ingeno, 2018

Tradiční metodiky mají pevný plán s definovanými kroky a požadavky, které jsou dopředu přesně specifikované a dokumentované. Testování probíhá až na konci celého projektu. Jedná se o obecně pomalejší a méně flexibilní metodiky, pokud jde o změny v průběhu projektu. Agilní metodiky se zaměřují na flexibilitu a inkrementální vývoj, který umožňuje změny v požadavcích na základě feedbacku. Jsou méně rizikové, zaměřují se na průběžné testování a vývoj softwaru ve fázích, což umožňuje rychle odhalit a řešit problémy. Jsou více zaměřeny na týmovou spolupráci a komunikaci (Ingeno, 2018).

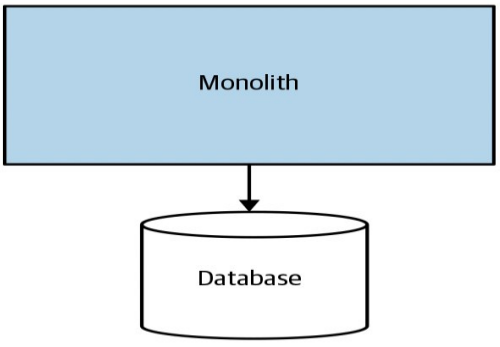
2.3 Druhy architektur webových aplikací

Existuje několik základních architektur, podle kterých je možnost danou webovou aplikaci vyvinout. Pro daný typ aplikace je zapotřebí zvolit tu nejvhodnější. V této podkapitole diplomové práce bude představeno několik architektur, které jsou vhodné pro implementaci webových aplikací.

2.3.1 Monolit

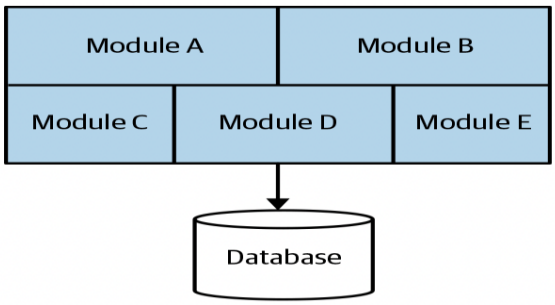
Jedná se o druh aplikační architektury, kdy je nutné všechny implementované funkce v aplikaci nasadit společně, aby byla aplikace funkční. Celá aplikace je soběstačná a je nasazena jako celek. Pokud by bylo nutné aplikaci replikovat, je to nutné provést i s funkcemi, které nejsou zapotřebí.

Nejvíce běžným příkladem monolitické aplikace, je systém, ve kterém je celý zdrojový kód nasazen jako jediný proces. Je možné mít několik instancí tohoto procesu kvůli robustnosti, ale v zásadě je celý kód zabalen do jednoho procesu. Tyto jedno-procesní systémy mohou být samy o sobě jednoduchými distribuovanými systémy, protože pořád čtou data z databáze a také je do ní ukládají. Graficky si lze jedno-procesní monolitickou aplikaci prohlédnout na obrázku 2.3 níže.



Obr. 2.3 Jedno-procesní monolit, zdroj: Newman, 2019

Druhým příkladem monolitické aplikace je modulární monolit. Tento typ je podmnožinou jedno-procesního monolitu. U tohoto typu se proces skládá z rozdělených modulů, na kterých je možné pracovat samostatně, ale stále musí být nasazeny společně. Graficky je možné si modulární monolit prohlédnout na obrázku 2.4 níže.



Obr. 2.4 Modulární monolit, zdroj: Newman, 2019

Monolitická aplikace má své výhody a také nevýhody. Mezi výhody lze zařadit např. jednoduchost v nasazování aplikace. To může mít za následek jednoduchý monitoring aplikace, detekci chyb a také může být zjednodušen end-to-end testing. Nevýhodou může být v tomto případě architektury to, že po každé implementované funkcionalitě a každé úpravě systému je potřeba celý systém znovu kompilovat a provést testování. Další nevýhodou je, že v rámci dlouhodobého vývoje by mohl být problém si udržet programátory, kteří celou aplikaci dobře znají (Newman, 2019).

2.3.2 Mikroslužby

Mikroslužby jsou malé samostatně nezávislé běžící služby, které zapouzdřují svoji funkcionalitu, kterou poskytují dalším službám přes aplikační rozhraní. Takovýmto rozhraním může být například metoda REST nebo GraphQL, přes které si mikroslužby vyměňují data nejčastěji ve formátu JSON (Mitra a Nadareishvili, 2020).

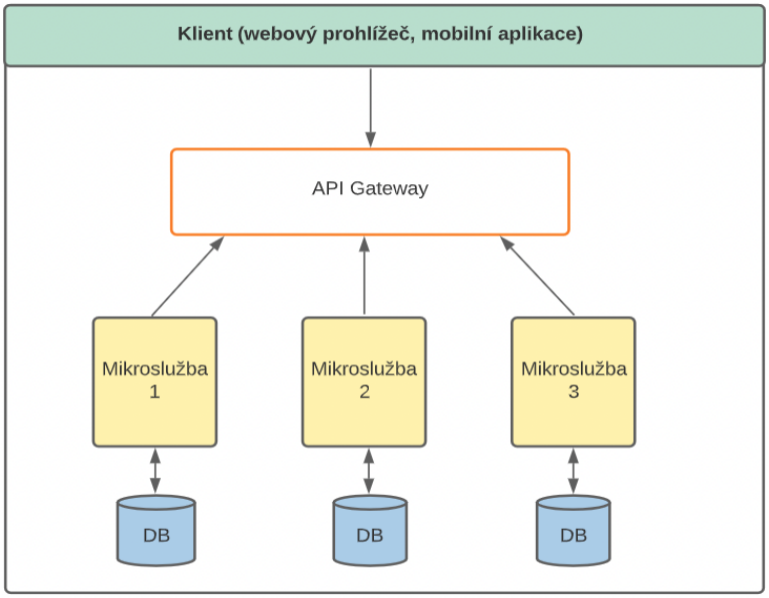
Mikroslužby zapouzdřují jeden nebo více síťových koncových bodů, přes které spolu komunikují, a to z nich vytváří určitou formu distribuovaného systému. Mikroslužby se většinou vyhýbají sdíleným databázím a místo toho má každá mikroslužba svou vlastní. Také zapouzdřují ukládání a vystavování dat, tudíž jsou jejich databáze skryty mimo danou mikroslužbu. Potřebuje-li jedna mikroslužba od druhé specifická data, musí o to požádat přes výše zmíněné aplikační rozhraní.

Mezi výhody architektury mikroslužeb patří určitě jejich nezávislost na ostatních službách. To nám otevírá nové možnosti rozšiřitelnosti systému o další funkcionality systému a také míchat technologie, které jsou pro danou funkcionalitu nejvhodnější. Další

výhodou je to, že na mikroslužbách jde pracovat paralelně, tudíž na systému může pracovat více programátorských týmů najednou aniž by se navzájem jakkoliv omezovali. Další výhodou je jejich malá velikost a jednoduchost, která zajistí jednoduchou správu a případnou úpravu funkcionality bez nutnosti znát okolní kód.

Mezi nevýhody lze zařadit fakt, že mikroslužby spolu komunikují většinou pomocí internetové či intranetové sítě, kde můžou nastávat výpadky. To znamená, že nikdy nemůže být jistota, že mikroslužby mezi sebou ztratí možnost komunikace (Newman, 2019).

Graficky je architektura pomocí mikroslužeb zobrazena na obrázku 2.5 níže.



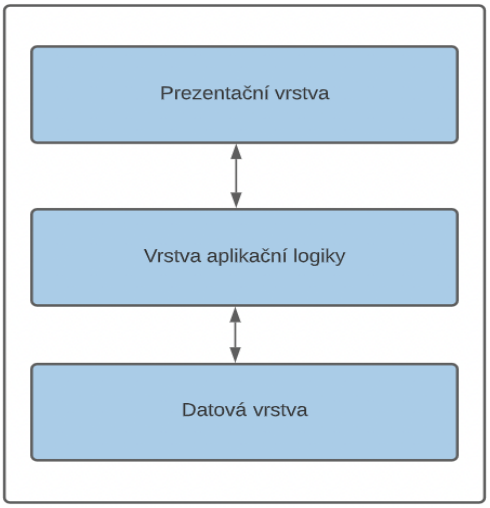
Obr. 2.5 Architektura mikroslužeb, zdroj: vlastní

2.3.3 Vrstvená architektura

Vrstvená architektura, také známá jako n-vrstvá architektura, je jednou z nejpoužívanějších architektur při vývoji aplikací. Tento styl architektury je de facto standardem pro velkou většinu aplikací především kvůli její jednoduchosti, oblíbenosti a nízkým nákladům. Komponenty ve vrstvené architektuře jsou organizovány do logických horizontálních vrstev, kde každá vrstva vykonává specifickou roli v celé aplikaci. Ačkoliv neexistují žádná psaná pravidla a omezení pro počet vrstev, nejčastěji jsou využity tři vrstvy. Těmi vrstvami jsou prezentační vrstva, aplikační vrstva a vrstva datová. Prezentační vrstva řeší, jak prezentovat data uživateli aplikace, v aplikační vrstvě jsou popsány logické operace, založeny na zdrojových datech aplikace a ve vrstva datová se stará o ukládání a čtení dat z databáze.

Vrstvená architektura je vhodným stylem pro menší, jednoduché aplikace nebo dynamické webové stránky. Kvůli jednoduchosti a velké oblíbenosti mezi programátory a softwarovými architekty, je tato architektura vhodná pro projekty s malým rozpočtem a časovým omezením. Tato architektura je také dobrou volbou v případě, že softwarový architekt stále analyzuje potřeby podniku a je si nejistý, která z architektur bude prodanou aplikaci, či informační sytém nejvhodnější.

Jestliže aplikace využívající vrstvenou architekturu stále rostou (funkcionálně), je tím ohrožena udržitelnost, hbitost, testovatelnost a nasaditelnost aplikace. Z tohoto důvodu by měly větší aplikace a systémy, využívající vrstvenou architekturu, být transformovány do vhodnějších, více modulárních architektonických stylů (Richards a Ford, 2020). Jelikož se nejedná o rozsáhlý systém, byla pro vyvíjenou aplikaci v rámci této diplomové práce byla vybrána právě tato architektura. Na obrázku 2.6 níže si lze prohlédnout vrstvenou architekturu.



Obr. 2.6 Vrstvená architektura, zdroj: vlastní

2.4 Programovací paradigmata

Druh programovacího paradigma označuje styl programování. Jak programování dospívalo, programátoři a počítačoví vědci nacházeli výhody a nevýhody v programovacích stylech a ty se následně vyvinuly do nových a modernějších. Dnešní moderní programovací jazyky zastávají dva nebo i více programovacích paradigmat, které je možné kombinovat dle preferencí, nebo potřeb vyvíjené aplikace či organizace (Bansal, 2013).

Programovacích paradigmat je celá řada, ale v této kapitole budou objasněny čtyři nejčastější. Těmi jsou funkcionální paradigma, objektově orientované paradigma, imperativní (procedurální) paradigma a aspektově orientované paradigma. V rámci programovacího jazyka JavaScript, který bude primárně využit v této práci, lze kombinovat všechny níže uvedené paradigmata.

2.4.1 Imperativní programování

Základem imperativního nebo také procedurálního programování je přiřazovací příkaz, který mění stav nízko úrovňového von Neumenova stroje. Programátor manuálně překládá logiku explicitně, aby řekl počítači, co má dělat. Proměnné jsou v imperativním programování namapovány do určitého místa v paměti počítače, které lze opakovaně upravovat za pomoci přiřazovacích příkazů. Výsledným efektem tohoto přiřazení je fakt, že nová hodnota se zapíše do paměti a stará je ztracena, což má za následek, že není možné starou hodnotu využít v budoucnu.

Imperativní programování je prvním paradigmatem, vyvinutým v padesátých letech dvacátého století. Nejznámější programovací jazyky, které toto paradigma využívaly jsou FORTRAN a ALGOL, C a Pascal (Bansal, 2013).

2.4.2 Funkcionální programování

Funkcionální programování je založeno na principu postavit neměnný program, založený na čistých funkcích. Čistá funkce má následující vlastnosti:

- Spoléhá se jen na poskytnutý vstup, nikoliv na externí stav, který se může změnit během volání ostatních funkcí
- Nezpůsobuje změny, přesahující jejich rozsah, jako například změnu globálního objektu, nebo poskytnutého parametru.

S funkcemi se ve funkcionálním programování zachází velmi podobně jako s objekty v OOP. Funkcionální programy jsou velmi vhodné na práci s poli, či seznamy, kvůli využití funkcí, které originální pole nebo seznam nepřepíšou.

Jelikož kód, který využívá čisté funkce má nulovou šanci změnit nebo rozbít globální stav programu, je tento kód více udržovatelný a mnohem lépe testovatelný. Funkcionální programování využívá deklarativní styl psaní kódu. To vylepšuje a zjednodušuje celkovou čitelnost aplikace a výsledný kód je díky využívání kombinace funkcí a lambda výrazů kratší a čitelnější. Program napsaný funkcionálním stylem by ve

výsledku měl popisovat problém, který se snažíme řešit, nikoliv mechanismus řešení (Atencio, 2016).

2.4.3 Objektově orientované programování

Při využití objektově orientovaného přístupu klademe důraz na znovupoužitelnost, jelikož je sestavení programu z již existujících komponent výhodnější, přehlednější a levnější. Pokud se v programu vyskytne chyba, stačí ji v tomto případě opravit pouze na jednom místě. Tento přístup psaní kódu je také mnohem přehlednější pro další lidi, kteří na vývoji aplikace spolupracují.

V tomto přístupu se snažíme odpoutat od toho, jak vytvářený program vidí počítač, ale snažíme se soustředit na to, jak program vidí samotný programátor.

Základní jednotkou v objektově orientovaném přístupu je objekt, který má své atributy a metody. Za atribut objektu můžeme označit vlastnost nebo data, která uchovává a za metodu můžeme označit schopnost, kterou vykonává. Dalším důležitým pojmem v objektově orientovaném programování je třída, která představuje vzor, jak se daný objekt má chovat. Jinak řečeno, definuje jeho vlastnosti a schopnosti (atributy a metody).

Všechny objektově orientované programovací jazyky poskytují mechanismy, které pomáhají programátorům implementovat tzv. objektově orientovaný model. Těmito mechanismy jsou zapouzdření, dědičnost a polymorfismus (mnohotvárnost), které představují tři základní pilíře objektově orientovaného programování.

Zapouzdření je mechanismus, který svazuje kód a data se kterými manipuluje a ochraňuje je před vnějším zneužitím. Jinými slovy umožnuje skrýt vybrané atributy a metody tak, aby zůstali viditelné jen uvnitř dané třídy, ve které jsou deklarovány. Existuje zde možnost označit atributy a metody jako veřejné nebo soukromé. K soukromým datům existuje přístup pouze uvnitř daného objektu třídy anebo s pomocí veřejných metod, se kterými lze s těmito daty pracovat.

Dědičnost se dá označit za proces, ve kterém jeden objekt třídy přebírá neboli dědí vlastnosti jiné nadřazené třídy. Díky tomuto procesu, získáváme podporu hierarchické klasifikace. Bez využití hierarchií by každá třída musela mít v sobě explicitně definovány všechny charakteristiky, které již má rodičovská třída definovány. Díky dědičnosti, může mít třída nadefinovány jen ty vlastnosti, které ji činí jedinečnou.

Polymorfismus je vlastnost, která umožňuje využívat jedno rozhraní pro práci s různými typy objektů. Specifická akce je určena přesnou povahou dané situace. Podstatou polymorfismu jsou tedy metody, které jsou definovány v rodičovské třídě, nebo v rozhraní, které třída implementuje. Tyto metody jsou definovány se stejnou hlavičkou, ale jiným tělem (Schildt, 2021).

2.4.4 Aspektově orientované programování

Vznik tohoto typu programování zapříčinila potřeba modularizace často se opakující části kódu, nacházejících se na různých místech dané aplikace. Jako klasický případ se dá uvést například logování operací, které se provádí nad daty. Takovéto části jsou zpravidla označeny jako *„záležitosti“*. Oddělení těchto jednotlivých záležitostí do programových bloků, které lze nezávisle využít v různých částech aplikace. Tyto bloky jsou nazývány *„aspekty“* a jsou základem tohoto typu programování.

Při využití aspektového programování jsou nejprve nalezeny nezávislé *„záležitosti“*, což mohou být například transakce nad databázemi, které se opakují a jejich bezpečnostní procedury atd. Poté se naprogramuje kód jednotlivých *„aspektů“*, které se připojí k jednotlivým odpovídajícím modulům příslušnou technikou a následně se provádí případná kompilace. Výsledný modul obsahuje jak obchodní logiku dané aplikace, tak i zdrojové kódy jednotlivých aspektů (Vymětal, 2009).

2.5 UML (Unified Modeling Language)

UML je standartní modelovací jazyk pro vývoj softwarových systémů. Návrh systémů je v každém měřítku složitou činností. Cokoliv od jednoduché webové aplikace po vícevrstvý podnikový systém, může být tvořeno stovkami softwarovými a hardwarovými komponentami. V systémovém návrhu se vytvářejí modely hlavně kvůli zvládnutí složitosti. Modelování dokáže pomoci při zachycení a zdokumentování důležitých aspektů návrhu daného systému (Miles a Hamilton, 2006).

Modelování přináší pochopení systému, ale žádný model není nikdy dostačující samotný. Většinou je zapotřebí více modelů, které jsou vzájemně propojeny, aby bylo systému řádně porozuměno. S pomocí UML lze modelovat různé pohledy na architekturu systému a jak se vyvíjí v průběhu celého životního cyklu.

Pravidla jazyka UML říkají, jak lze vytvářet modely a jak je číst, ale neříkají, kdy a jaké modely vytvářet. To je role procesu vývoje softwaru. Dobře definovaný proces dokáže být nápomocen při rozhodování, jaké artefakty vytvořit, jaké činnosti a

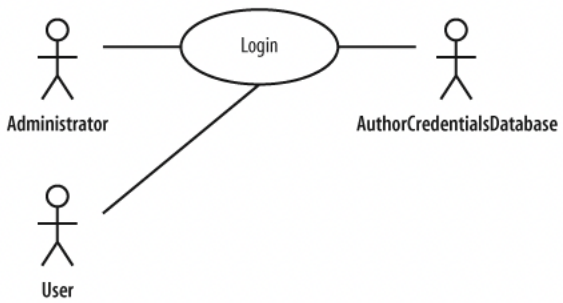
pracovníky využít k jejich vytvoření a právě, a také jak tyto artefakty správně využít k měření a řízení projektu jako celku.

UML lze využít při psaní programů, ale není považován za programovací jazyk, jelikož mu chybí syntaktické vlastnosti a také sémantika. Přesto existují užitečné nástroje, které poskytují generátory kódů z UML do různých druhů programovacích jazyků (Booch, Rambaugh a Jacobson, 2005).

UML popisuje třináct druhů diagramů, z kterých jsou níže popsány ty, které budou v dalších kapitolách této diplomové práce využity k návrhu architektury webové aplikace.

2.5.1 Modelování požadavků

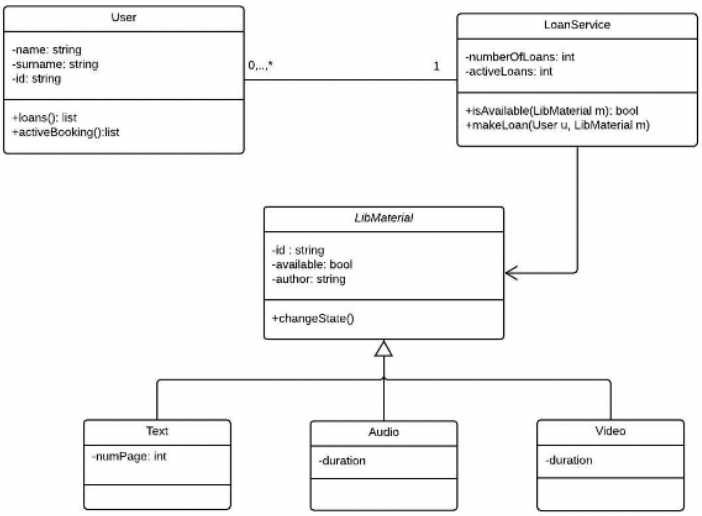
Pro modelování požadavků lze využít diagram případu užití (use case diagram), který popisuje systémové požadavky z pohledu vnějších uživatelů. Specifikuje, jakou přidanou hodnotu systém doručuje uživatelům. Jelikož tyto diagramy popisují funkcionality vyvíjené aplikace či systému, mely by být prvním výstupem modelování po startu projektu (Miles a Hamilton, 2006). Graficky si lze diagram případu užití prohlédnout na obrázku 2.7 níže.



Obr. 2.7 Use Case diagram, zdroj: Miles a Hamilton, 2006

2.5.2 Modelování architektury aplikace

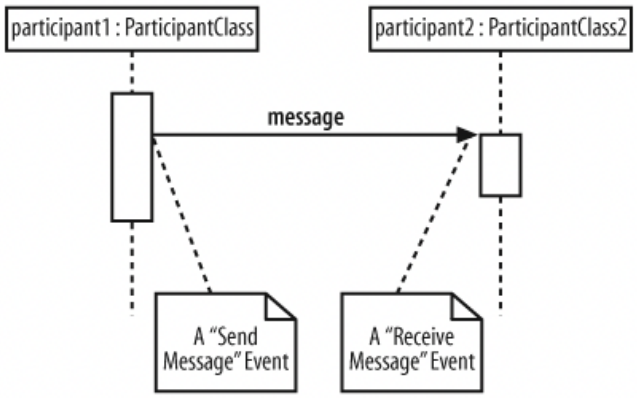
K modelování aplikační architektury lze využít diagram tříd. Tento diagram popisuje strukturu vyvíjeného systému pomocí tříd, které reprezentují odlišné druhy objektů, které vyvíjená aplikace či systém obsahuje. V tomto diagramu jsou popsány také jaké jsou vztahy mezi jednotlivými třídami (Miles a Hamilton, 2006). V grafické podobě si lze tento diagram prohlédnout na obrázku 2.8.



Obr. 2.7 Diagram tříd, zdroj: vlastní

2.5.3 Modelování komunikace mezi jednotlivými částmi aplikace

K modelování komunikace lze využít sekvenční diagramy, které patří do skupiny interakčních diagramů a slouží k zachycení pořadí interakcí mezi jednotlivými částmi aplikace nebo systému. Pomocí těchto diagramů lze popsat, jaké interakce budou spuštěny při provedení konkrétního případu užití a v jakém pořadí k těmto interakcím dojde (Miles a Hamilton, 2006). V grafické podobě je sekvenční diagram znázorněn na obrázku 2.8.

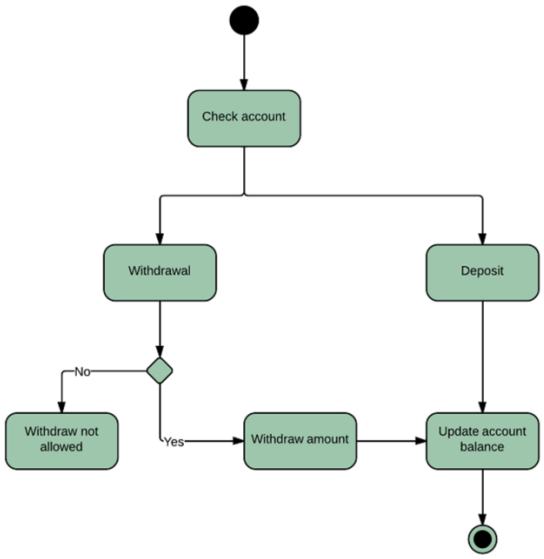


Obr. 2.8 Sekvenční diagram, zdroj: Miles a Hamilton, 2006

2.5.4 Modelování funkcí systému

Pro modelování jednotlivých funkcí systému lze využít aktivity diagram. Tento diagram je nástroj, který se v softwarovém inženýrství využívá k popisu toků činností neboli aktivit v rámci vyvíjené aplikace nebo systému. Umožňuje lépe porozumět

procesům a činnostem v rámci systému a identifikovat oblasti, které je potřeba vylepšit. V grafické podobě si lze aktivity diagram prohlédnout na obrázku 2.9 níže.



Obr. 2.9 Aktivity diagram, zdroj: https://www.lucidchart.com/pages/uml-activity-diagram

2.6 Front-end aplikace

V této podkapitole diplomové práce budou popsány technologie, využité pro klientskou část vyvíjené webové aplikace. Jako klientskou část označujeme část aplikace, která slouží k interakci s uživatelem. Za hlavní technologie této části označujeme HTML pro tvorbu struktury, CSS pro implementaci stylů a JavaScript pro tvorbu logiky. Všechny tyto technologie budou detailněji popsány níže.

2.6.1 Tvorba struktury

Při tvorbě webové stránky či aplikace je potřeba vytvořit obsahovou strukturu jednotlivých prvků, které má daná aplikace obsahovat. Pro tuto činnost se využívá HTML (Hypertext Markup Language), což je značkovací jazyk pro tvorbu struktury obsahu webu. Tento jazyk není nikým vlastněn a jeho prvky jsou výsledkem práce spousty organizací. HTML dokument je textový dokument, který se dá vytvořit v libovolné textovém editoru. Tento dokument obsahuje tagy neboli značky, které tvoří jeho strukturu. Tyto tagy mohou také obsahovat určité direktivy, přes které se lze odkazovat na určité styly napsané v jazyce CSS anebo logické prvky, napsané v jazyce JavaScript. Příkladem těchto direktiv může být například direktiva *„id“*, která slouží k identifikaci elementu a v rámci jednoho dokumentu jsou její hodnoty jedinečné, nebo direktiva

„class“, která obsahuje názvy jednotlivých elementů. HTML dokument je rozdělen na dvě části které jsou vymezeny tagy *„head“*, který slouží pro různé informace o dokumentu, jako například jméno autora, nebo způsob kódování znaků a *„body“* pro viditelnou obsahovou část dokumentu.

Poslední verzí tohoto jazyka je HTML5. Tato verze při vzniku vzbudila značnou oblíbenost, kvůli novým přidaným prvkům jako je například *„canvas“* pro zobrazování obrázků a animací. Dalším důležitým rozdílem oproti předchozím verzím je podpora videí a zvukových prvků. Asi největším přínosem, který tato verze přinesla jsou nové druhy tagů pro definování částí dokumentu a lepší orientaci. Těmi hlavními jsou tagy *„header“* pro hlavičku viditelné části dokumentu, *„footer“* pro patičku dokumentu, *„main“* pro hlavní obsahovou část a *„section“* pro definování jednotlivých sekcí na webu (Meyer, 2022).

2.6.2 Vizuální úpravy obsahu

Poté co je vytvořena obsahová struktura jednotlivých obsahových sekcí webové stránky či aplikace, je potřeba je vizuálně upravit, aby vypadaly, jak je požadováno. Pro tuto činnost se využívá CSS (Cascading Style Sheets), což je jazyk pro specifikaci vzhledu zobrazovaných HTML dokumentů. CSS umožňuje volit barvy textů a tlačítek, ale také umožňuje tvorbu vzhledu a pozic elementů na různých velikostech zobrazovacích zařízení a také animace.

CSS byl poprvé navržen v roce 1994 na konferenci v Chicagu a v roce 1996 byla představena jeho první verze jakožto nástroje pro stylování HTML dokumentů. CSS se skládá z pravidel, které se využívají k cílení na elementy HTML pomocí selektorů, které popisují ty elementy, na které mají být styly aplikovány.

Existují tři možnosti, jak využívat CSS v HTML dokumentu. První možností je využití tzv. vnořených stylů přímo v daném elementu přes atribut *„style“*, kde se zapisují požadované stylistické vlastnosti, které má daný prvek splňovat. Další možností, jak stylovat HTML prvky je přes tzv. stylové bloky. Toho lze docílit přidáním elementu *„style“* do hlavičky nebo těla dokumentu a psaním CSS stylů do tohoto bloku. Třetí, v dnešní době nejpoužívanější možností je vytvoření externího CSS souboru, který je následně propojen s daným HTML dokumentem pomocí tagu *„link“*.

V dnešní době existují tzv. CSS preprocessory jako například Sass, LESS a Stylus, které poskytují přidané funkcionality, které v čistém CSS nejsou implementovány.

Těmito funkcionalitami je vnořování, tvorba proměnných a mixin atd. Při buildu aplikace preprocessor převede napsané styly do čistého CSS podporovaného webovým prohlížečem (Attardi, 2020).

2.6.3 Tvorba logické části

V dnešní době je prakticky nutné prvky umístěné na webové stránce nějak dynamicky oživovat. Ať už se jedná o vytváření animací, nebo složitější asynchronní operace s daty, využívá se k těmto věcem JavaScript. JavaScript je skriptovací programovací jazyk, který se využívá nejvíce na straně klienta webových aplikací. Mimo klientské části se dá využít i na části serverové. Díky tomu, že JavaScript využívá velká většina moderních webových stránek a aplikací a také všechny moderní webové prohlížeče v sobě mají zabudovaný JavaScriptový interpreter, je JavaScript nejrozšířenějším programovacím jazykem v historii. Díky NodeJS, což je systém, navržený pro využití JavaScriptu na serverové části webových aplikací, se tento jazyk stal v posledním desetiletí také nejvíce využívaným mezi všemi softwarovými vývojáři (Flanagan, 2020).

Díky názvu, jaký tento jazyk má, hodně vývojářů očekává klienstkou verzi programovacího jazyka JAVA, ovšem podobnost těchto jazyků je pouze v jejich názvu. Syntakticky tento jazyk vychází z jazyka C, jeho rozhraní, kde můžeme zahrnout vestavěné funkce, objekty a proměnné se podobají Javě a jeho objektový a funkcionální model je inspirován programovacími jazyky Self a Scheme. Jedná se o dynamicky typovaný jazyk, což znamená, že o datových typech definovaných proměnných je rozhodnuto až za běhu programu. Také jeho funkce jsou v něm k dispozici jako běžný datový typ.

Během prvních let své existence, JavaScript díky své oblíbenosti odsunul na vedlejší kolej svůj jediný konkurující jazyk VBScript od společnosti Microsoft a stal se de facto jediným programovacím jazykem použitelným v rámci HTML dokumentů (Žára, 2022).

JavaScript byl vytvořen společností Netscape (Mozilla) a jeho název je ochrannou známkou společnosti Sun Microsystems (Oracle), používanou k popsání implementace jazyka touto společností. Netscape odeslal jazyk ke standardizaci do ECMA (European Computer Manufacturer’s Association) a kvůli problémům se značkami byla standardizovaná verze jazyka pojmenována ECMAScript. Standard ES5, který vznikl

v roce 2009 byl plně podporován všemi webovými prohlížeči. Průlomovou verzí byla verze ES6, také známá jako ECMAScript 2015. Ta přinesla mnoho nových funkcí a vylepšení například:

- Lepší syntaxe – přidání tzv. *„arrow“* funkcí (zkrácený zápis funkcí), deklarace proměnných let (lze přepsat) a const (nelze přepsat), rozšířené výrazy pro přiřazení atd,
- Moduly – lepší organizace a správa kódu a možný import a export jednotlivých funkcí a proměnných mezi moduly,
- Nové typy dat – např. široké řetězce (template literals) a nové způsoby práce s poli,
- Možnost klasického OOP – umožňuje vývojářům využívat koncepty jako třídy, dědičnost a objektové instance,
- Asynchronní programování – podpora pro asynchronní programování pomocí nových funkcí např. async/await, což umožňuje vytvořit kód mnohem jednodušší a čitelnější (Flanagan, 2020).

K vytvoření plně dynamické webové aplikace, je zapotřebí asynchronní komunikace mezi webovým prohlížečem a serverem. Pro tyto účely byla vytvořena technologie AJAX, která je v dnešní době již zastaralá a ve velké většině aplikací je nahrazována efektivnější a novější technologií REST služeb, ale stále je v některých aplikacích pro tyto účely využívána. AJAX neboli Asynchronous JavaScript and XML je technologie webového vývoje, která se využívá pro získávání dat z webového serveru bez nutnosti znovunačtení stránky. To umožňuje vývojářům vytvořit interaktivní a rychle se načítající aplikace. AJAX není programovací jazyk nebo knihovna, ale kombinace zabudovaného XMLHttpRequest objektu ve webovém prohlížeči a programovacího jazyka JavaScript (Svekis, Putten a Percival, 2021).

DOM (Document Object Model) je rozhraní, které v sobě obsahuje HTML elementy určitého dokumentu a pomocí JavaScriptu s nimi můžeme manipulovat. Těmito elementy mohou být tagy, texty, atributy atd. Pomocí tohoto rozhraní je možné vytvářet interaktivní webové stránky namísto stránek statických (Svekis, Putten a Percival, 2021).

2.6.4 Frameworky pro vývoj klientské části webové aplikace

V rámci vývoje webových aplikací existuje pro účely vývoje klientské části spousta moderních frameworků. Všechny tyto frameworky jsou založeny na programovacím jazyce JavaScript. JavaScriptové frameworky jsou knihovny softwarových nástrojů, které slouží k usnadnění a také urychlení vývoje webových aplikací. Tyto frameworky nabízejí jednotný a strukturovaný přístup k vývoji aplikací, a tím umožňují vývojářům vyhnout se opakovanému psaní kódu a zaměřit se na implementaci funkcí aplikace.

Existuje mnoho těchto frameworků a jejich počet se neustále zvyšuje. Mezi nejznámější patří ReactJS, vyvinutý společností Facebook (Meta), AngularJS, vyvinutý společností Google, VueJS, který vyvinul Evan You a EmberJS, vyvinutý společností Tidle. Tyto frameworky se od sebe liší výkonem, složitostí a funkcemi a volba mezi nimi závisí hlavně na preferencích vývojáře a specifikací vyvíjené aplikace.

V rámci této diplomové práce bude využit framework VueJS. Tento framework byl zvolen hlavně z důvodu, že jde o mladý progresivní framework, jeho rostoucí popularity mezi vývojáři a jeho jednoduchosti a intuitivnosti oproti ostatním zmiňovaným. Tento framework byl zvolen také s ohledem na požadavky na funkčnost aplikace, které jsou popsány v následující kapitole této práce a zkušenostmi, kterými s tímto frameworkem disponuji.

2.6.5 VueJS

VueJS je jednou z nejpopulárnějších JavaScriptových knihoven využívaných k vývoji front-endové části webových aplikací. Byla vytvořena v roce Evanem You v roce 2014. Vyznačuje se jednoduchou syntaxí a zároveň umožňuje vysoce sofistikovaný vývoj webových aplikací. Je označována za progressivní knihovnu, což znamená, že je možné ji využít k vývoji jednoduchých webových prvků např. widgetů a rovněž k vývoji složitých robustních webových aplikací s mnoha komponentami.

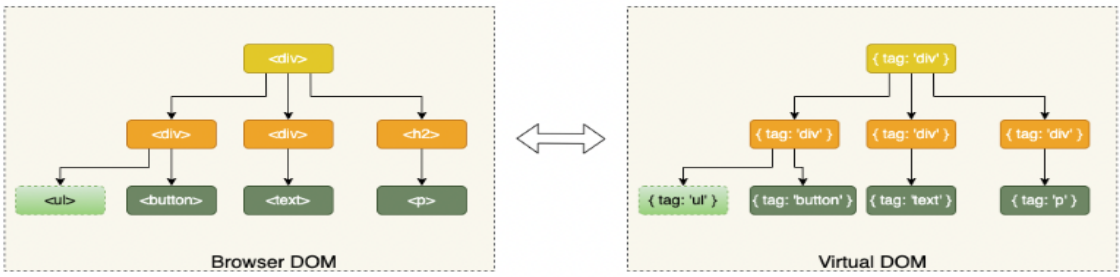
Díky své komponentové architektuře lze snadno rozdělit vyvíjenou webovou aplikaci na malé, samostatně psané komponenty, což usnadňuje vývoj a udržování webové aplikace. Kromě toho také poskytuje nástroje pro správu stavu a renderování, což umožňuje snadno vytvářet aplikace s dynamickým uživatelským rozhraním.

VueJS také poskytuje širokou škálu nástrojů pro optimalizaci a testování webové aplikace. Tyto nástroje umožňují snadno testovat jednotlivé komponenty a zajistit, aby

webová aplikace byla rychlá a spolehlivá. Navíc je tento framework dobře dokumentován a má rozsáhlou komunitu uživatelů, což znamená, že je k dispozici mnoho návodů, tutoriálů a knihoven třetích stran, které dokážou pomoci s vývojem dané webové aplikace pomocí tohoto frameworku (Shavin, 2023).

DOM může obsahovat mnoho uzlů, dotazování a aktualizace jednoho nebo více uzlů může být extrémně náročné a může aplikaci zpomalovat. Proto je zde implementován tzv. virtuální DOM, který tento problém eliminuje. Virtuální DOM je odlehčená virtuální kopie skutečného DOM prohlížeče s jinou datovou strukturou (většinou objektovou). Virtuální DOM je umístěn mezi Vue aplikací a skutečnou DOM strukturou prohlížeče. Prostřednictvím interakcí uživatelského rozhraní uživatel říká Vue, v jakém stavu si přeje, aby prvek byl. Poté je požadavek předán virtuálnímu DOM a ten aktualizuje reprezentovaný objekt daného prvku na požadovaný tvar. Nakonec se spustí komunikace se skutečným DOM webového prohlížeče a ten provede přesné aktualizace na změněných prvcích struktury.

Jelikož je virtuální DOM stromem vlastních JavaScriptových objektů, aktualizace komponenty se rovná aktualizaci vlastního JavaScriptového objektu. Tato operace není náročná, protože se při ní nevolá žádná DOM API. Jakmile virtuální DOM aktualizuje sám sebe, synchronizuje se se skutečnou strukturou originálního DOM, a to provede požadované změny v prohlížeči. Tento proces synchronizace je znázorněn na obrázku 2.9 níže (Shavin, 2023).



Obr. 2.9 Komunikace mezi virtuálním DOM a skutečným DOM prohlížeče, zdroj: Shavin,2023

Aktuální verzí VueJS frameworku je Vue 3, která byla spuštěna v druhé polovině roku 2020. Tato verze přinesla sadu změn, jako lepší možnost integrace s TypeScriptem, cože je nadstavba nad jazykem JavaScript, která mu umožňuje statické typování proměnných a funkcí. Další změna je nové rozhraní API pro řešení velkých,

škálovatelných případů využití a pevný základ pro dlouhodobé budoucí vylepšování tohoto frameworku (Djirdeh, Murray a Lerner, 2021).

Velkou změnou verze Vue 3 je podpora Composition API. Jedná se o nový způsob vytváření Vue komponent s více optimalizovaným způsobem psaní kódu a s podporou pro TypeScript. V tomto nové způsobu deklarace Vue komponenty, se všechny data, objekty a funkce deklarují v rámci *„setup“* vlastnosti objektu a ta vrátí vše co komponenta potřebuje v požadovaném pořadí. V nejnovější aktualizaci tento objekt již není zapotřebí zakládat, protože existuje možnost přidání atributu *„setup“* přímo do *„<script>“* tagu, kde je potřebná logika zapsána. (Ribeiro, 2020).

Mezi hlavní výhody Composition API patří:

- Modularita a čistší kód: Composition API umožňuje oddělit kód do menších, samostatných funkcí, což zlepšuje čitelnost a udržitelnost kódu,
- Opětovné využití kódu: Funkce vytvořené pomocí Composition API mohou být použity v různých komponentách, což umožňuje využívat kód vícekrát a zlepšuje efektivitu práce,
- Lépe organizovaný kód: Composition API umožňuje skupovat podobné funkce do logických celků, což usnadňuje orientaci v kódu,
- Flexibilita: Composition API umožňuje tvorbu složitých funkcionalit bez nutnosti vytváření složitých komponent, což zvyšuje flexibilitu vývoje aplikací,
- Lépe se řeší složité problémy: Composition API umožňuje řešení složitých problémů pomocí funkcí, což zvyšuje čitelnost kódu a snižuje počet chyb.

Ve Vue 3 byly také představeny nové integrované komponenty, které lze využít pro řešení problémů, kvůli kterým se ve starších verzích frameworku musely instalovat pluginy třetích stran. Prvním z těchto komponent je komponenta *„Fragments“*, která zabezpečuje možnost vkládání zdrojových HTML elementů přímo do struktury Vue komponenty. V minulých verzích musely být tyto elementy obaleny do dalšího elementu např. elementu *„div“*. Další přidanou komponentou je komponenta *„Teleport“*, která může být využita k přesunu elementů z jedné komponenty do druhé. Poslední komponentou je komponenta *„Suspence“*, která lze využít např. při velké prodlevě při stahování potřebných dat. Tato komponenta zabezpečuje, že se má uživateli zobrazit

nějaký prvek např. načítací animace než se stáhnout potřebná data k zobrazení požadované stránky. K tomuto bylo zapotřebí v předchozích verzích psát vlastní logiku, ale ve Vue 3, díky této komponentě to již není zapotřebí (Ribeiro, 2020).

2.7 Back-end aplikace

V této podkapitole diplomové práce budou popsány technologie, které byly využity k vývoji serverové části webové aplikace. Za serverovou část považujeme část aplikace, která slouží ke zpracovávání dat a komunikaci s databází. K vývoji této části existuje spousta nástrojů a také cloudových řešení třetích stran. Lze volit například mezi programovacími jazyky Java, Python, C# nebo JavaScriptovým frameworkem Node.js.

Za účelem vývoje webové aplikace, která je vyvíjena v rámci této diplomové práce, nebude vytvořen vlastní backend, protože ke všem potřebným funkcionalitám a komunikaci s databází bude využita cloudová služba Firebase od společnosti Google. Tato služba bude v této diplomové práci využita z důvodu její jednoduchosti implementace a rychlosti. Dále bude využita knihovna email.js, která bude využita pro odesílání informativních e-mailů.

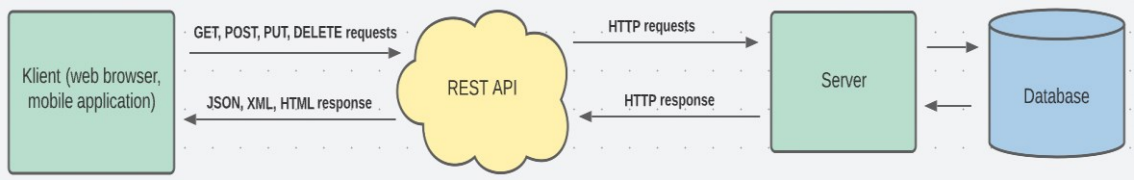
2.7.1 REST API

Komunikace s výše uvedenými produkty bude realizována pomocí API, což je jednoduše řečeno spojení mezi oddělenými částmi softwarového systému. Definuje nám množinu operací, které jedna systémová komponenta poskytuje k využití ostatním částem systému, které mohou tvořit části klientské (webový aplikace, mobilní aplikace) nebo ostatní API bez uživatelského rozhraní. API samo o sobě může navazovat spojení s ostatními API a kontrolovat jejich funkčnost.

API bylo tradičně poskytováno za pomoci softwarových knihoven, které mohly být propojeny s aplikací staticky nebo za běhu aplikace ke znovuvyužití procedur a funkcí pro specifické problémy, jako OpenGL, což je průmyslový standard specifikující multiplatformní rozhraní pro tvorbu počítačové grafiky, nebo knihovny pro síť na bázi protokolu TCP/IP. Takovéto API jsou stále běžné, ale s rostoucím počtem API jsou nyní k dispozici na internetu jako RESTful webové služby.

REST neboli Representional State Transfer je styl, který byl vytvořen Royem Fiedingem k popsání principů, které vedly k úspěchu HTTP protokolu a webu. Později byl adaptován jako množina principů pro návrh API. REST API zdůrazňují standardní formáty zpráv a malý počet generických operací, aby se redukovalo propojení mezi

klientem a konkrétním API. Využití hypertextových odkazů v navigaci rozhraní API snižuje riziko selhání klientů, protože se rozhraní API v průběhu času vyvíjí. Naprostá většina REST API využívá pro přenos dat formát JSON, což je datový model založený na principu klíč-hodnota (Madden, 2021).



Obr. 2.10 Princip fungování REST API, zdroj: vlastní

JSON (JavaScript Object Notation) je formát pro přenos dat mezi částmi webové aplikace nebo také softwarovými systémy. Alternativou pro tento formát je XML. Oproti XML je tento formát jednodušší, má menší velikost a mnohem rychlejší zpracování. Data jsou v něm zobrazena jako textové řetězce, které jsou velmi podobné objektům v programovacím jazyce JavaScript a můžou být na ně velmi lehce převedeny. Tyto data mohou obsahovat hodnoty jako jsou čísla, logické hodnoty ve formátu true/false, nebo další vnořená pole a objekty. Tento formát je velmi výhodný v případech, že je nutné předávání velkých objemů dat mezi aplikacemi.

Spousta dnešních webových prohlížečů podporují tento formát zápisu dat a jeho analýzu. Proto je JSON v dnešní době jedním z nejpoužívanějších a nejvhodnějších formátů pro přenos dat např. mezi klientskou aplikací a aplikací serverovou (Bassett, 2015). Ukázku JSON objektu si lze prohlédnout na obrázku 2.11 níže.

```
{
    "person": {
        "name": "Lindsay Bassett",
        "heightInInches": 66,
        "head": {
            "hair": {
                "color": "light blond",
                "length": "short",
                "style": "A-line"
            },
            "eyes": "green"
        }
    }
}
```

Obr. 2.11 JSON objekt, zdroj: Bassett, 2015

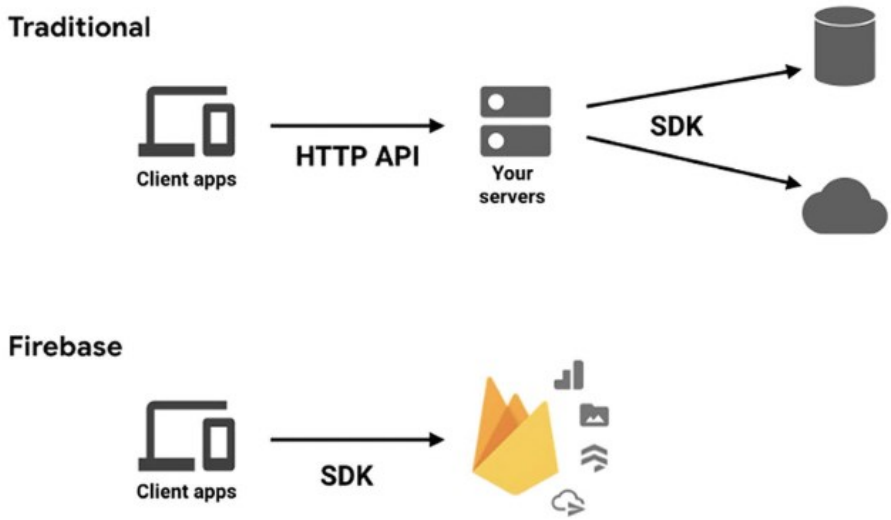
2.7.2 Firebase

Firebase je platforma od společnosti Google pro vývoj aplikací. Tato služba je naprogramována v jazyce Java a je často nazývána *„back-end as a service“* (Baas) a obsahuje široké spektrum vytvořených funkcionalit a nástrojů, které snadno umožňují vytvářet webové a mobilní aplikace. Firebase poskytuje mnoho funkcí pro vývoj moderních webových aplikací, do nichž patří tyto služby:

- Autentifikace: Přihlášení uživatele a přiřazování identity,
- Real-time databáze: Real-time No-SQL databáze, umístěná na cloudu,
- Cloud Firestore: Novější a rychlejší verze výše zmíněné databáze,
- Cloud úložiště: Velké úložiště souborů např. obrázků,
- Cloud funkce: back-endové funkce řízené událostmi,
- Firebase hosting: Globální webový hosting,
- ML kit: SDK pro běžné úkoly pro machine learning.

Všechny tyto služby jsou poskytovány jako služby v cloudu. Velká výhoda v tomto přístupu je, že se vývojář nemusí starat o instalaci, konfiguraci a správu serverové infrastruktury a tím pádem se lze více soustředit na samotný vývoj aplikace.

S pomocí Firebase lze ulehčit práci hlavně front-end vývojářům s integrací back-endu do jejich vyvíjených aplikací, bez zbytečných vytváření API a dalšího back-end kódu. Firebase využívá svou vlastní API, se kterou lze pracovat pomocí volání specifických Firebase funkcí. Lze tedy říci, že Firebase dokáže do určité části eliminovat zbytečné vytváření vlastní serverové části a zasílat požadavky přímo z klientské aplikace do Firebase, která je umístěna na Google platformě. Tyto přístupy vývoje lze ovšem i kombinovat a využívat Firebase API i za pomoci serverových aplikací, napsaných například v Node.js. Na obrázku 2.12 níže si lze prohlédnout rozdíly ve vývoji aplikací s pomocí Firebase a bez ní (Biswas, 2022).



Obr. 2.12 Rozdíly architektur aplikací s Firebase a bez ní, zdroj: Biswas, 2022

2.8 Datové úložiště aplikace

Při vývoji webové aplikace lze volit z několika druhů databází na základě požadavků na funkčnost vyvíjené aplikace, kapacit a podmínek daného projektu. Pro tento projekt bude datové úložiště vytvořeno ve službě Firestore database, kterou poskytuje výše zmíněná služba Firebase.

Databáze je pojem, který označuje soustavu dat, která je organizována tak, aby byla dostupná pro vyhledávání, úpravu a použití jednotlivých dat. Za databázi lze považovat také např. telefonní seznam, jelikož obsahuje údaje jako telefonní čísla, jména a adresy. Kvůli těžkopádné povaze papírových databází byly některé z prvních počítačových aplikací vyvinuty databázové systémy, což jsou počítačové mechanismy pro ukládání a vyhledávání dat. Vzhledem k tomu, že databázový systém ukládá data elektronicky, je tento systém schopen získávat přístup k těmto datům mnohem rychleji, indexovat je více způsoby a poskytovat aktuální informace komunitě uživatelů (Beaulieu, 2020).

Donedávna byly nejpoužívanějším způsobem pro ukládání a efektivní správu dat tradiční databázové systémy, které mohly být založené na relačním paradigmatu, objektovém paradigmatu, objektově-relačním paradigmatu, XML atd. Nejpopulárnějším z těchto paradigma je relační přístup k ukládání informací, které vycházejí z matematického pojmu relace, který si lze představit jako tabulku obsahující řádky a sloupce. Takovéto systémy mají mnoho společných vlastností, jako je například persistentní způsob ukládání dat a dotazovací jazyky sloužící pro přístup k datům. Tyto

systémy jsou typicky založeny na architektuře klient/server, což znamená, že jsou všechna data uložena na jednom serveru, k němuž je umožněn přístup jednotlivým klientským programům. Logicky z toho vyplývá, že čím více dat je zapotřebí na serveru ukládat, o to více je poté potřeba zvyšovat diskovou kapacitu serveru, což může být technicky a finančně náročný proces. Poté je potřeba data někde zálohovat v případě ztráty nebo poškození paměťových médií serveru, proto je potřeba mít ještě servery záložní na nichž jsou data duplikována. Z těchto důvodu vznikli tzv. NoSQL databáze, které začaly vznikat začátkem tohoto tisíciletí pro podporu efektivního zpracování velkého objemu dat (Holubová, Kosek, Minařík a Novák, 2015).

2.8.1 Datové modelování

Pro určení správného formátu dat je potřeba provést analýzu datových požadavků. Takovýto proces se nazývá datové modelování. Výsledkem tohoto procesu je datový model, což je reprezentace dat a vztahů mezi nimi. V podstatě se jedná o abstraktní popis dat, který popisuje, jak jsou data organizována a jak jsou navzájem propojena. Datový model se používá jako základ pro návrh a implementaci databázového systému, který slouží k uchování a zpracování dat. Datový model je obvykle dokumentován pomocí diagramů, tabulek a jiných vizuálních nástrojů.

Existují různé typy datových modelů, které se používají v závislosti na potřebách organizace a konkrétních požadavcích. Mezi nejčastěji používané datové modely patří hierarchický datový model, síťový datový model, relační datový model, objektově orientovaný datový model a objektově relační datový model.

Proces datového modelování se skládá z několika fází, kterými jsou konceptuální datové modelování, které se zaměřuje na definování vztahů mezi různými entitami, logické datové modelování, které se zaměřuje na specifikaci atributů a vztahů mezi entitami a fyzické datové modelování, které se zaměřuje na návrh databázového schématu a specifikaci technických detailů, jako jsou datové typy, indexy, omezení, klíče a další. Fyzický datový model popisuje, jak jsou data ukládána v databázi a jakým způsobem jsou zpracovávána aplikací (Umanath a Scamell, 2014).

2.8.2 NoSQL databáze

Tyto databázové systémy jsou převážně nerelační, distribuované, škálovatelné a podporují replikaci. Velmi často to jsou databáze bez datového schématu, open-source přístupem a jednoduchým rozhraním pro práci s daty. Tento typ databází se nejlépe hodí

pro aplikace s rychle se měnícími požadavky na data, pro zpracování velkých objemů dat, jako jsou například texty, obrázky a videa.

Existuje několik typů těchto databází. Příkladem může být například dokumentová databáze, key-value databáze a grafové databáze. Každý tento typ se liší svým modelem pro ukládání a práci s daty.

Mezi hlavní výhody těchto databázových systémů patří:

- Škálovatelnost: NoSQL databáze jsou obecně škálovatelnější než SQL databáze. SQL databáze jsou většinou škálovatelné vertikálně, kdežto noSQL databáze jsou škálovatelné horizontálně.
- Flexibilita: NoSQL databáze jsou obecně flexibilnější než SQL databáze, což umožňuje snadnější změny v datovém modelu.
- Kompatibilita s moderními aplikacemi: Mnoho moderních webových a mobilních aplikací vyžaduje rychlé čtení a zápis dat v reálném čase.
- Složitost dotazování: SQL databáze jsou silné v práci s daty, které jsou ukládány v tabulkách, ale když se potřebují provádět složitější dotazy na nestrukturovaná data, jako jsou JSON dokumenty, mohou být omezené.

Tyto systémy ovšem mají i také své nevýhody oproti klasickým databázovým systémům. Mezi tyto nevýhody například patří omezení v oblasti transakcí a konzistence dat a také mohou vyžadovat větší úsilí při implementaci zabezpečení. Také jelikož neexistuje žádný standardizovaný přístup k datům jako u relačních databází v podobě SQL, tak prakticky každá NoSQL databáze má vlastní dotazovací jazyk a vlastní API. Z tohoto důvodu můžou být zapotřebí netriviální programátorské znalosti v případě zadávání složitějších dotazů. Nejpoužívanějšími z NoSQL databází jsou například MongDB, Cassandra a DynamoDB (Holubová, Kosek, Minařík a Novák, 2015).

2.8.3 Firestore databáze

Firestore je cloudová NoSQL databáze, vyvinutá a spravována společností Google. Je součástí služby Firebase. Jelikož se jedná o NoSQL databázi, tak jsou zde data ukládány jako dokumenty, a ne jako tabulky, obsahující řádky a sloupce jako v případě databází relačních.

Nabízí několik funkcionalit pro zabezpečení dat, jako jsou pravidla, která lze použít k omezení přístupu k datům a také funkce pro šifrování dat. Od svých konkurentů

se tato databáze odlišuje velmi rychlým a snadným přístupem k datům, což může být pro mnoho aplikací kritické. Firestore také umožňuje aplikacím získávat data v reálném čase, což pomáhá vytvářet interaktivní aplikace s rychlým přenosem dat. Firestore má také jednoduchou syntaxi, která umožňuje lehce pracovat s daty prostřednictvím API (Biswas, 2022). Přestože Firebase podporuje práci s jinými databázemi, bylo rozhodnuto o využití této databáze z důvodu její rychlosti, flexibilitě a škálovatelnosti.

2.9 Shrnutí kapitoly

V této kapitole byly popsány veškeré principy a technologie, které jsou při návrhu a následném vývoji této webové aplikace využity. Také byly zmíněny důvody výběru daného využitého přístupu a technologií, spolu s konkurenčními či alternativními technologiemi. Další kapitola se bude zabývat analýzou současného stavu a požadavky na aplikaci, kde budou některé z výše zmíněných technologií využity.

3 Analýza současného stavu a sběr požadavků

Tato kapitola se bude zabývat analýzou současného stavu, ve kterém se nyní proces vytváření konferencí a jejich následná správa nachází. Také zde budou popsány jednotlivé požadavky na nově vznikající webovou aplikaci a jejich zpracování.

3.1 Současný stav

Jak již bylo řečeno v úvodu této diplomové práce, tato webová aplikace je vytvářená pro Ekonomickou fakultu Vysoké školy báňské – Technické univerzity v Ostravě a bude sloužit k vytváření a správě konferencí a jejich následných recenzí od účastníků. Momentálně se k tomuto úkolu nevyužívá žádná aplikace, či informační systém a vše je řešeno administrativně formou e-mailů a excelovských tabulek. Informace o plánované konferenci jsou umístěny na internetových stránkách fakulty a dále jsou rozesílány potenciálním účastníkům ve formě PDF dokumentů.

Přihlášení uživatelů probíhá primárně přes e-mail nebo přes formulář, umístěný na webových stránkách fakulty. Údaje o přihlášených jsou následně vedeny v Excelovských dokumentech. Přes e-mail jsou rovněž vedeny i recenze k dané konferenci pomocí vytvořených formulářů, které recenzent obratem zasílá zpět vyplněné. Tyto formuláře obsahují údaje veřejné údaje, které se dále přeposílají autorům a soukromé údaje, které jsou určeny pouze redakční komisi. Tento způsob správy je časově náročný, a proto vznikl požadavek na vytvoření aplikace, která by tento proces zjednodušila.

3.2 Vymezení požadavků na webovou aplikaci

Požadavky na webovou aplikaci byly stanoveny v průběhu několika schůzek se zadavatelem. Tyto schůzky probíhali formou rozhovorů osobně nebo online formou, kde byla probrána požadovaná funkčnost a vzhled aplikace. Výsledkem měla být webová aplikace s jednoduchým a intuitivním uživatelským rozhraním, která bude usnadňovat plnění výše zmíněných aktivit. Tato aplikace by měla být také rozšiřitelná pro budoucí požadavky na funkčnost. V softwarovém inženýrství se rozlišují dva typy požadavků, a to požadavky nefunkční a požadavky funkční.

V případě nefunkčních požadavků, popisují, jaké vlastnosti by měla vyvíjená aplikace mít. Tyto požadavky nejsou přímo spojeny s funkcemi aplikace, ale spíše s její výkonností, bezpečností, uživatelským rozhraním atd.

V případě funkčních požadavků se jedná o požadavky, které popisují, co musí aplikace dělat, tedy jaké funkcionality musí být v aplikaci implementovány. Jsou to požadavky, které musí být splněny, aby mohla být aplikace považována za plně funkční.

3.2.1 Nefunkční požadavky na webovou aplikaci

V rámci schůzek byly stanoveny tyto nefunkční požadavky:

- Uživatelské rozhraní aplikace by mělo být jednoduché a intuitivní
- Aplikace by měla být dostupná přes webový prohlížeč na desktopových i mobilních zařízeních
- Aplikace by měla rychle reagovat na uživatelské akce, bez nutnosti načítání velkého množství dat
- Měly by existovat dva typy uživatelských účtů s různými právy
- Do aplikace nesmí být možné se samostatně registrovat
- V aplikaci by měli být přístupné všechny konference, které byly vytvořeny a ne jen aktuální

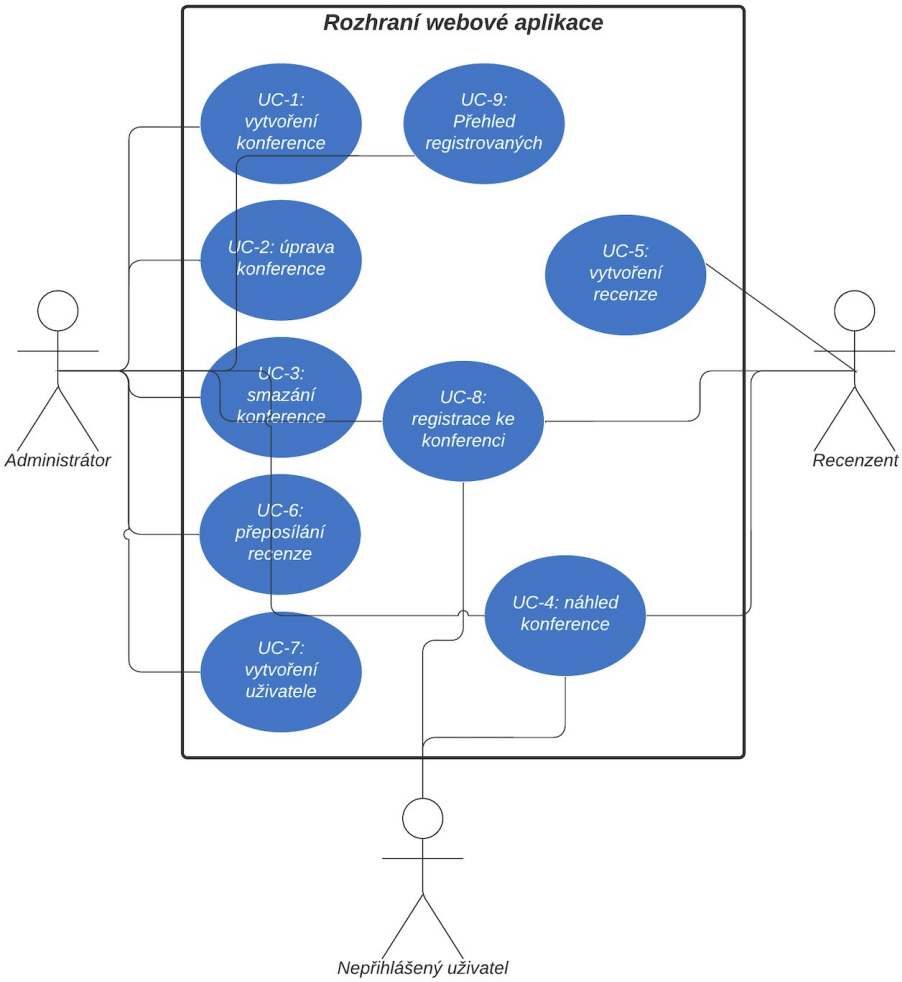
3.2.2 Funkční požadavky na webovou aplikaci

V rámci schůzek byly stanoveny tyto funkční požadavky:

- P-1: Správa konferencí
- P-2: Registrace na konferenci bez nutnosti přihlášení
- P-3: Náhled konference bez nutnosti přihlášení
- P-4: Vytváření recenzí
- P-5: Přeposílání recenzí
- P-6: Vytváření uživatelů

3.2.3 Use Case diagram

V této podkapitole jsou popsáni všichni aktéři a jejich případy užití. Jak je možné vidět na diagramu 3.1 níže, v aplikaci existují tři typy uživatelů se specifickými možnostmi užívání aplikace. Tyto případy užití byly vytvořeny na základě uvedených požadavků výše a detailněji budou popsány v následujícím textu.



Obr. 3.1 Use Case diagram, zdroj: vlastní

3.2.4 Aktéři

Z diagramu znázorněného výše je jasné, že v aplikaci existují tři typy uživatelů, kteří mají různé oprávnění. Prvním typem je administrátor, který může dělat téměř vše. Může vytvářet jednotlivé konference, následně upravovat jejich obsah nebo je mazat. U vytvořených konferencí má právo vidět seznam registrovaných účastníků a také jednotlivé vytvořené recenze včetně soukromých údajů, které může dále přeposílat na určenou e-mailovou adresu. Také má právo vytvářet jiné uživatelské účty ať už s administrátorskými oprávněními, nebo s oprávněními recenzenta.

Dalším typem je recenzent. Uživatel s právy recenzenta má právo pouze vidět vytvořené konference s možností registrace a dále je oprávněn vytvářet recenze k jednotlivým vytvořeným konferencím.

Posledním typem je uživatel, který je nepřihlášen. Tento typ uživatele si může jen prohlížet vytvořené konference a registrovat se na ně. Všechno ostatní má v aplikaci zakázáno.

3.2.5 Matice sledovatelnosti požadavků

Matice sledovatelnosti požadavků neboli RTM matice je dokument, pomocí kterého sledujeme požadavky uživatelů pomocí jednotlivých případů užití. Hlavním účelem vytváření RTM matice je ověřit, zda všechny jsou všechny funkční požadavky řádně kontrolovány a testovány pomocí případů užití. Vytvořenou matici si lze prohlédnout v tabulce 3.1 níže.

Tab. 3.1 Matice sledovatelnosti požadavků, zdroj: vlastní

RTM	Případy užití								
Požadavky	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
P-1	x	x	x	x					x
P-2								x	
P-3				x					
P-4					x				
P-5						x			
P-6							x		

3.3 Shrnutí kapitoly

V rámci této kapitoly byl objasněn současný stav správy konferencí na ekonomické fakultě Vysoké školy báňské – Technické univerzity v Ostravě. Dále byly popsány nefunkční a funkční požadavky na nově vyvíjenou webovou aplikaci. Pomocí těchto funkčních požadavků byl sestaven diagram případů užití a dále pak sestavena matice sledovatelnosti požadavků, kde lze vidět, které případy užití splňují konkrétní požadavky. Další kapitola se bude věnovat samotnému návrhu, vývoji a implementací výše uvedených požadovaných funkcionalit.

4 Návrh řešení, vývoj a implementace webové aplikace

Tato kapitola poskytne podrobný pohled na celý proces návrhu, vývoje a implementace webové aplikace. Bude zde podrobně popsán návrh databáze a jednotlivých komponent webové aplikace, včetně ukázek a popisu uživatelského rozhraní.

4.1 Využité technologie k vývoji webové aplikace

Celá aplikace včetně všech jejích komponent je vyvíjena v rámci výše specifikovaných požadavků. V rámci těchto požadavků byly vybrány v současnosti nejnovější verze technologií uvedeny v tabulce 4.1 níže.

Tab. 4.1Využité technologie k vývoji, zdroj: vlastní

Technologie	Verze	Účel využití
VueJS	3.2.45	Tvorba uživatelského rozhraní
Vite	4.1.0	Kompilace webové aplikace
Pinia	2.0.30	State management
UUID	9.0.0	Tvorba unikátních identifikátorů
Vee-validate	4.8.1	Ošetřování vstupů dat
Vue-router	4.1.6	Routování v aplikaci
Firebase	9.17.1	Backendová služba
Bootstrap	5.3.0	Tvorba layoutu jednotlivých náhledů
Email.js	3.10.0	Odesílání e-mailů

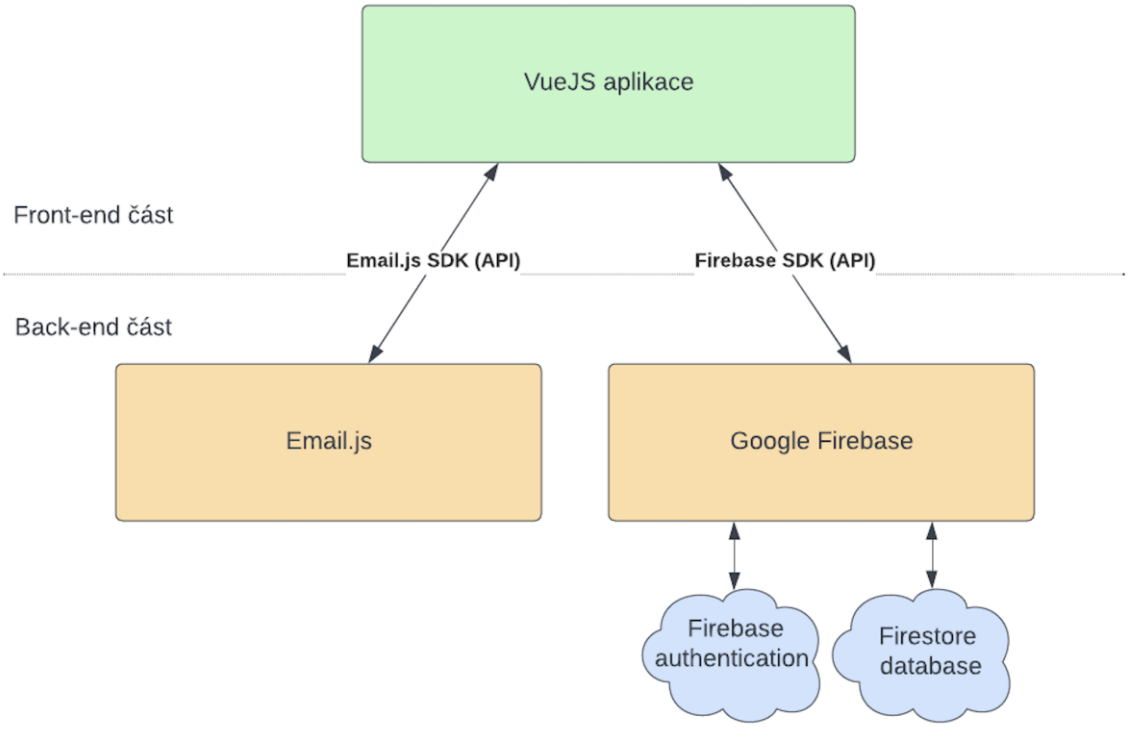
4.2 Architektura webové aplikace

Celá architektura vyvíjené webové aplikace je implementována v souladu s principy vrstvené architektury, které jsou vysvětleny v druhé kapitole této diplomové práce.

Front-endová aplikace je naprogramována v programovacím jazyce JavaScript za pomoci frameworku VueJS jako single-page webová aplikace. Veškeré načítání dat a komunikace s Back-endem probíhá v podobě asynchronních požadavků, a tudíž není zapotřebí refreshovat okno prohlížeče, kde je aplikace spuštěna. Celá aplikace se dle jejích komponent dá rozdělit do dvou vrstev. Těmito vrstvami jsou vrstva prezentační a aplikační.

V prezentační vrstvě jsou umístěny komponenty, které se starají o správné zobrazování dat do uživatelského rozhraní a interakci s uživatelem. V aplikační vrstvě jsou umístěny komponenty, které se starají o manipulaci a ukládání dat do správných formátů, aby je mohli komponenty v prezentační vrstvě správně zobrazovat. V těchto komponentách vrstvy aplikační logiky jsou také umístěny funkce Firebase SDK, které se starají o komunikaci s back-endovou službou Firebase pomocí API. Příkladem těchto komponent může být například State management Pinia využívaný v této webové aplikaci.

Velká část back-endu aplikace je, jak bylo uvedeno výše zajišťován službou Firebase od společnosti Google, která v aplikaci zajišťuje komunikaci s cloudovou databází Firestore a veškeré potřebné operace nad uloženými daty. Rovněž tato služba zajišťuje autentifikaci uživatelů a s fornt-endem komunikuje pomocí již zmíněné API. Tato služba nám tedy logicky zajišťuje z části funkci vrstvy aplikační logiky a vrstvy datové. Druhou využívanou službou je služba poskytována knihovnou Email.js, pomocí které lze z aplikace posílat e-maily. Komunikace s touto knihovnou opět probíhá za pomoci SDK funkcí přes API rozhraní. Výsledná architektura webové aplikace je k nahlédnutí na obrázku 4.1 níže.



Obr. 4.1 Architektura webové aplikace, zdroj: vlastní

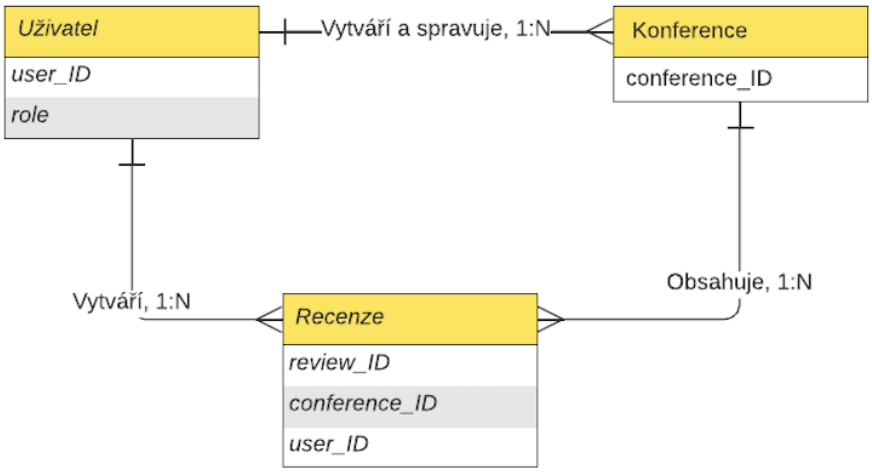
4.3 Návrh struktury databáze

Tato webová aplikace je primárně určena ke správě konferencí a jejich následných recenzí. Je tedy potřeba data spojená s konferencemi a uživateli aplikace někde efektivně ukládat a za pomoci webové aplikace k nim mít přístup a následně efektivně informace zobrazovat uživateli v uživatelském rozhraní. Za tímto účelem bylo nutné navrhnout a následně implementovat databázovou strukturu.

Jako první krok byl vytvořen konceptuální datový model, podle kterého byly určeny potřebné entity. Na základě tohoto konceptuálního datového modelu byl následně vytvořen finální model databázové struktury.

4.3.1 Konceptuální datový model

Za pomoci komunikace se zadavatelem bylo zapotřebí určit, jak se má výsledná webová aplikace chovat a co má zobrazovat. Zpracováním těchto získaných informací byl vypracován konceptuální datový model, který je k nahlédnutí na obrázku 4.2 níže.



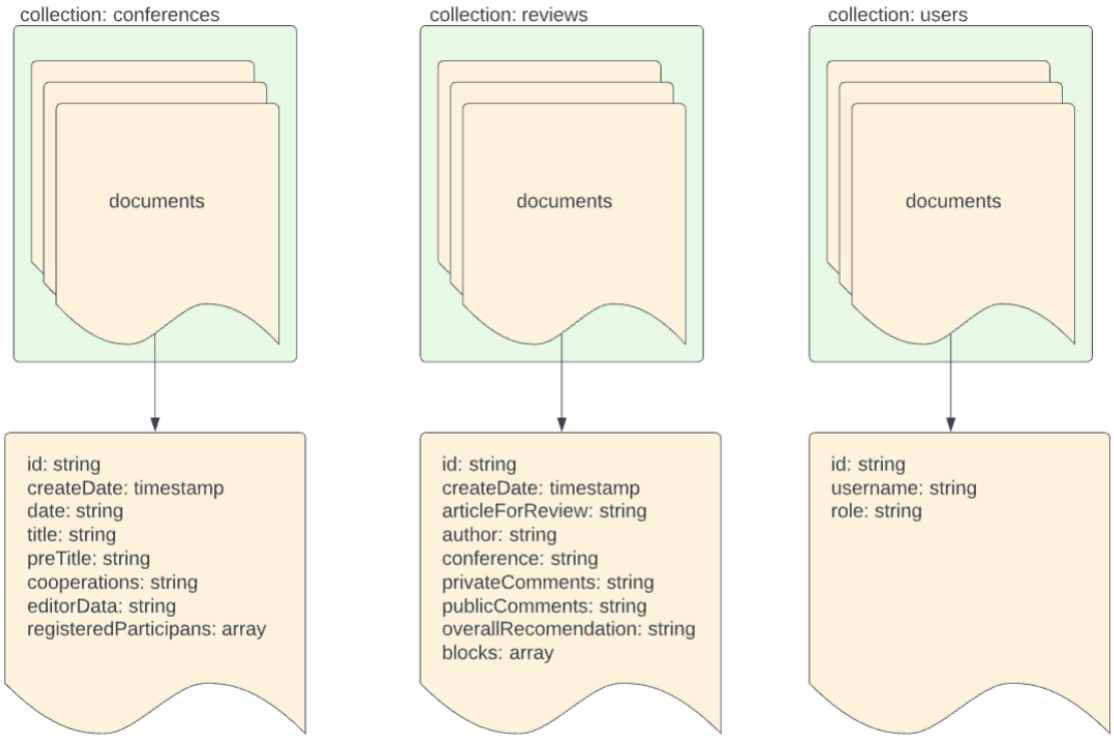
Obr. 4.2 Konceptuální datový model, zdroj: vlastní

V navrženém konceptuálním datovém modelu jsou zobrazeny tři entity – *Uživatel, Konference a Recenze*. Entita *Uživatel*, která představuje existující uživatelské účty ve webové aplikaci může na základě svých oprávněních (administrátor, recenzent) vytvářet konference, spravovat je a dále je recenzovat. Entita *Konference* představuje vytvořenou konferenci, která může obsahovat více recenzí. Poslední entitou je *Recenze*, která představuje vytvořenou recenzi, kterou vytváří uživatel aplikace s oprávněním

recenzenta. Každá jednotlivá recenze musí být přiřazena pouze jedné existující konferenci.

4.3.2 Logický datový model

V rámci návrhu logického datového modelu je již počítáno s tím, že bude využita databáze Firestore, kterou poskytuje služba Firebase od Google. Jak již bylo zmíněno Firestore je dokumentová databáze, to znamená, že její obsah tvoří jednotlivé kolekce s dokumenty. Vytvořený logický datový model si lze prohlédnout na obrázku 4.3 níže.



Obr. 4.3 Logický datový model, zdroj: vlastní

První věcí, které si lze všimnout je, že entity z konceptuálního modelu byly převedeny do jednotlivých dokumentů v kolekcích *conferences, reviews a users.* Tyto kolekce obsahují dokumenty ve formátu JSON s unikátními identifikátory, které představují jednotlivé entity. Ačkoliv tyto typy databáze podporují ukládání dokumentů s různou strukturou v jedné kolekci, byla kvůli přehlednosti databáze navržena struktura jednotlivých dokumentů, které se do kolekcí budou ukládat. Další věcí, kterou se tato databáze liší od klasické relační je, že zde neexistují žádné relace. V tomto typu databáze jsou cizí klíče reprezentovány jako pole, které obsahuje identifikátor dokumentu z jiné kolekce. V případě dokumentů v kolekci *reviews* jsou to pole *author,* kde je uložen identifikátor uživatele, který recenzi vytvořil a *conference*, kde je uložen identifikátor

konference, ke které tato recenze patří. Oba tyto údaje nejsou do dokumentů vkládány uživatelsky, ale generovány automaticky webovou aplikací.

Obrovskou výhodou tohoto přístupu je velmi jednoduchá rozšiřitelnost aplikace. Kdyby bylo například potřeba rozšířit obsah recenze, stačí rozšířit pouze strukturu objektu generovaného webovou aplikací, jehož struktura dat následně tvoří strukturu dat dokumentů v databázi. V samotné databázi nejsou zapotřebí žádné úpravy. V relační databázi by tato úprava vyžadovala i úpravu samotných tabulek v databázi.

Kolekce conferences

Tato kolekce obsahuje jednotlivé dokumenty ve formátu JSON, který obsahuje jednotlivé vlastnosti. První touto vlastností je *id* ve které je uložen jedinečný identifikátor dokumentu, vygenerovaný webovou aplikací jako řetězec náhodných znaků při vzniku dokumentu. Další vlastnost je *createDate*, která je ve formátu celočíselné hodnoty časové značky (timestamp), která značí, kdy byla konference vytvořena a generuje se automaticky při vzniku dokumentu.

Další vlastnosti tohoto dokumentu jsou editovatelné uživatelem, který má oprávnění administrátora a jejich hodnoty jsou zadávány v rámci formuláře určeného k vytvoření konference, umístěného ve webové aplikaci. Jedná se o vlastnosti *preTitle, title, cooperations* a *editorData*.

Poslední vlastností je *registeredParticipans*, jehož hodnota je ve formátu pole (array), jehož položky jsou jednotlivé objekty obsahující informace o registrovaných účastnících. Vlastnosti těchto objektů jsou jméno, příjmení, e-mailová adresa a telefon. Tyto objekty se vytváří automaticky webovou aplikací při úspěšném odeslání registračního formuláře, pomocí kterého se dá na konferenci zaregistrovat.

Kolekce reviews

V této kolekci jsou umístěny dokumenty, které obsahují informace o jednotlivých vytvořených recenzích. Jednotlivé dokumenty opět využívají formát JSON. Vlastnosti *id* a *createDate* mají stejnou funkci jako u dokumentů předchozí zmiňované kolekce a jsou generovány stejně. Další vlastnosti *konference* a *author* jsou generovány automaticky při úspěšném odeslání formuláře určeného k vytváření recenzí a jsou v nich umístěny identifikátory konference ke které byla daná recenze vytvořena a uživatele, který danou recenzi vytvořil.

Vlastnost *articleForReview* obsahuje informaci, ke kterému článku na konferenci je tato recenze určena. Hodnota této vlastnosti je vyplňována v rámci příslušného formuláře.

Dalšími vlastnostmi jsou *publicComments*, kde je uložen veřejný komentář recenzenta vyplněný pomocí příslušného formuláře. Následuje vlastnost *privateComments*, která se vyplňuje stejně jako vlastnost předchozí s tím rozdílem, že její obsah je viditelný pouze uživatelům s oprávněním administrátora. Vlastnost *overallRecomendation* je uživatelsky vyplňována v rámci formuláře.

Poslední vlastností je *blocks* která je opět typu pole (array), uvnitř tohoto pole jsou dynamicky generované objekty s hodnotícími kritérii, vyplňovanými v rámci formuláře při tvorbě recenze. Tyto objekty obsahují vlastnosti název hodnotícího kritéria a jeho hodnota.

Kolekce users

Tato kolekce je tvořena dokumenty, které poskytují informace o zaregistrovaných uživatelích aplikace. Na obrázku 4.3 výše si lze všimnout, že dokumenty umístěné v této kolekci obsahují pouze tři vlastnosti a to *id, username* a *role*. Je tomu tak z toho důvodu že ona kolekce je vytvořena pouze jako rozšíření informací o uživatelích, které jsou umístěné v interní databázi služby Firebase authentication. V této databázi jsou umístěny údaje o zaregistrovaném uživateli jako id, e-mail, šifrovaná hesla, ověřovací tokeny atd, ke kterým pomocí webové aplikace máme přístup a jejichž hodnoty lze upravovat. Problémem je ale vytváření vlastních dodatečných údajů o uživateli a přidávání je do databáze, což není povoleno. Jelikož tato aplikace vyžaduje rozdělovat zaregistrované uživatele podle oprávnění, bylo nutné v databázi Firestore vytvořit pomocnou kolekci, která obsahuje dokumenty s těmito dodatečnými vlastnostmi.

V rámci webové aplikace se při vytvoření uživatele přes registrační formulář automaticky vytvoří i dokument v této kolekci, který obsahuje jedinečný identifikátor uživatele, jeho oprávnění a uživatelské jméno, se kterými se pak ve webové aplikaci dále pracuje.

4.3.3 Zabezpečení databáze

Databáze je klíčovou součástí webové aplikace a obsahuje informace, jako jsou uživatelská data a data s informacemi o jednotlivých konferencích a recenzích. Zabezpečení databáze je proto klíčové, aby se zabránilo neoprávněnému přístupu, nebo

nechtěným úpravám a odstraňování dat. Co se týče ochrany proti hackerským útokům, databáze Firestore poskytuje zabezpečené HTTPS spojení mezi webovou aplikací a databází. Tento druh spojení zajišťuje, že data jsou šifrovaná během přenosu a jsou chráněna před útoky. Dalším důležitým prvkem je, že Firestore umožňuje správu přístupů k datům pomocí pravidel, kterými lze určit kdo může číst a zapisovat data do databáze. Tyto pravidla mohou být konfigurována tak, aby povolila přístup pouze určitým uživatelům a také mohou zahrnovat další pravidla jako například časová omezení.

Pomocí těchto pravidel byly nastaveny oprávnění pro jednotlivé kolekce dokumentů v databázi. Pro kolekci *conferences* kde jsou uloženy data vytvořených konferencí bylo nastaveno, že číst data mohou i neautorizovaní uživatelé, všechny ostatní operace nad daty může provádět pouze uživatel, který má oprávnění administrátora. Výjimku v zápisu dat tvoří pouze pole *registeredParticipans*, do kterého mohou zapisovat i neautorizovaní uživatelé, protože se zde vkládají data při registraci na konferenci, což může provádět přes příslušný formulář i neautorizovaný uživatel webové aplikace.

Pro kolekci *reviews* je povoleno čtení dat všem uživatelům, zápis je povolen pouze uživatelům s oprávněním recenzenta a všechny ostatní operace jsou povoleny pouze administrátorům.

Pro kolekci *users* je povoleno čtení a zápis pouze autorizovaným uživatelům a další akce pouze administrátorům.

Jako příklad nastavených pravidel pro přístup k databázi je níže na obrázku 4.4 k nahlédnutí nastavení pravidel pro dokumenty v kolekci *conferences*.

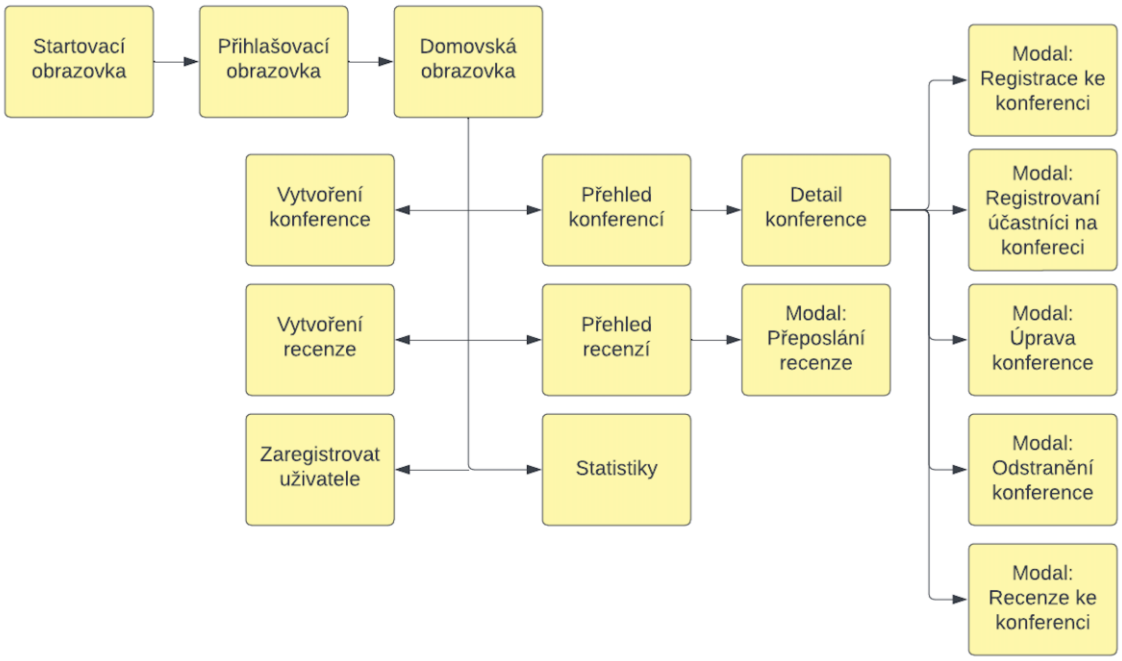
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents/conferences {
    match /{document=**} {
      allow read;
      allow write, update, delete: if request.auth != null && get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
      allow write: if request.auth == null && request.resource.data.diff(resource.data).affectedKeys().hasOnly(['registeredParticipans']);
    }
  }
}
```

Obr. 4.4 Zabezpečení dokumentů kolekce conferences v databázi, zdroj: vlastní

4.4 Návrh struktury uživatelského rozhraní webové aplikace

Struktura uživatelského rozhraní byla navržena tak, aby byla pro uživatele co nejvíce intuitivní. Na jednotlivé obrazovky se lze dostat po přihlášení z domovské obrazovky webové aplikace, nebo přes hlavní navigační menu, umístěné v hlavičce

aplikace. Celá webová aplikace je responzivní, což znamená, že s ní uživatel dokáže pracovat z jakéhokoliv zařízení s přístupem na internet přes webový prohlížeč. Pro lepší přehled struktury jednotlivých částí uživatelského rozhraní byl vytvořen screen-flow diagram, který je k nahlédnutí na obrázku 4.5 níže.



Obr. 4.5 Struktura uživatelského rozhraní aplikace, zdroj: vlastní

Přístup k jednotlivým částem uživatelského rozhraní je samozřejmě zabezpečen a každému uživateli je umožněn přístup podle jeho oprávnění.

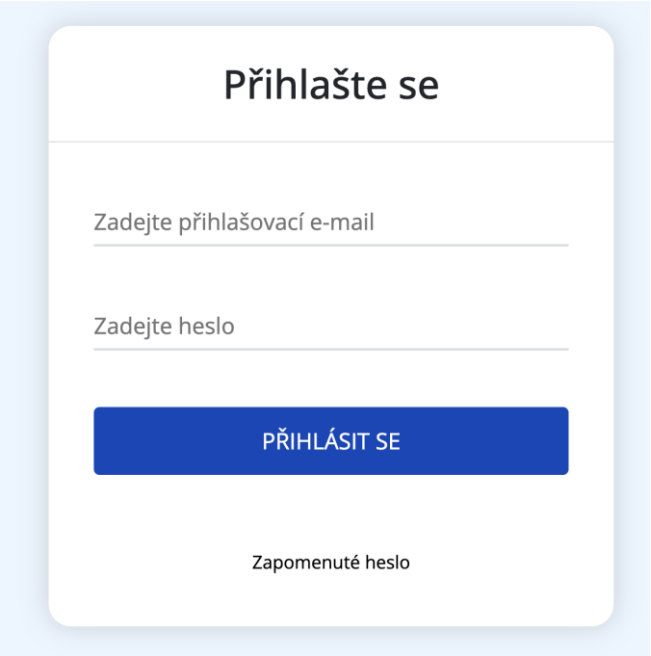
4.5 Návrh a implementace komponent webové aplikace

V této podkapitole této diplomové práce bude popsán návrh a implementace jednotlivých komponent webové aplikace, které byly navrženy tak, aby webová aplikace jako celek dokázala plnit požadovanou funkčnost.

Komponenty jsou základním stavebním kamenem VueJS aplikace. Lze je chápat jako znovupoužitelné bloky kódu, které mohou obsahovat HTML, CSS a JavaScript, a které lze použít v různých částech aplikace. Pomocí těchto komponent lze snadno a efektivně organizovat kód, čímž se zlepší přehlednost, udržovatelnost a opakovatelnost kódu. Důležitým prvkem je, že jsou komponenty na sobě nezávislé, dají se vnořovat nebo vkládat pod sebe a dokážou si předávat navzájem data, které zobrazují.

4.5.1 Komponenta pro přihlášení

Na přihlašovací obrazovku se dá dostat ze startovací stránky aplikace nebo z hlavního navigačního menu, umístěného v hlavičce aplikace. Je přístupná pouze pro nepřihlášené uživatele. Kdokoliv přihlášený, kdo se pokusí na tuto obrazovku dostat je automaticky přesměrován na domovskou obrazovku. Kdokoliv nepřihlášený se pokusí dostat na jakoukoliv jinou obrazovku kromě výpisu a detailu konferencí, například zadáním URL adresy bude přesměrován na přihlašovací obrazovku. Přihlašovací obrazovka, která zobrazuje komponentu pro přihlášení je k nahlédnutí na obrázku 4.6 níže.



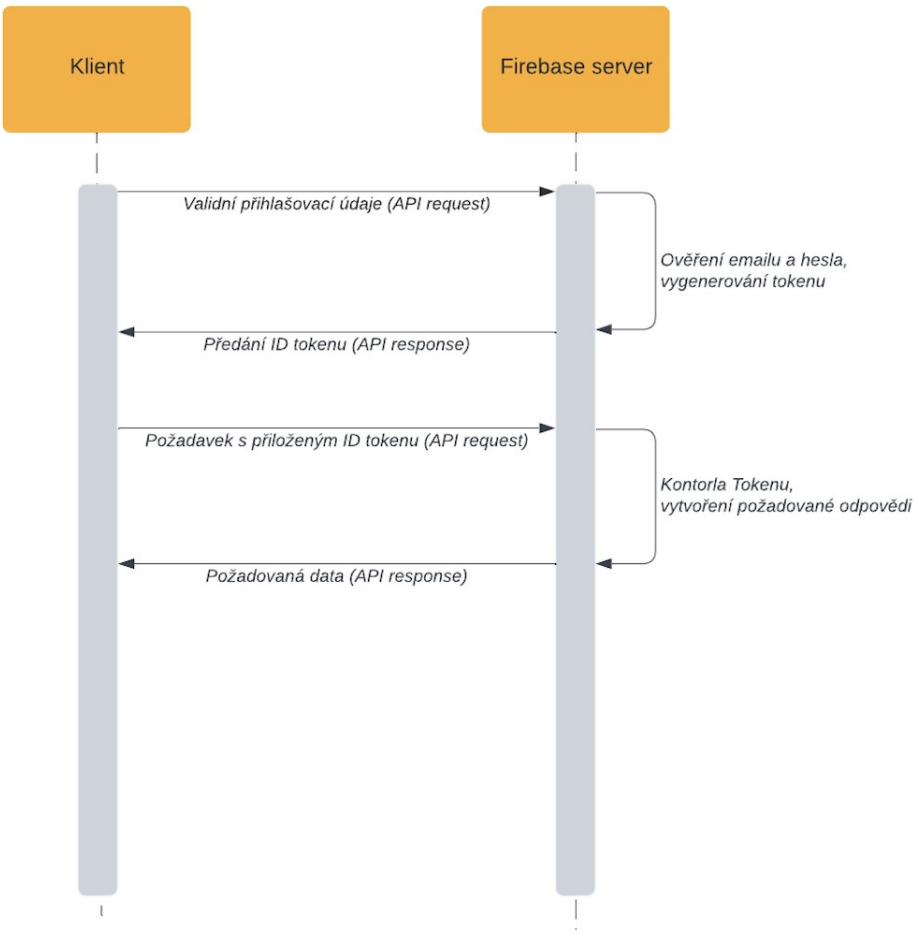
Obr. 4.6 Komponenta pro přihlášení do aplikace, zdroj: vlastní

Přihlašovací komponenta v sobě obsahuje formulář s dvěma poli. Do prvního pole uživatel zadává e-mail, pod kterým je registrován a do druhého heslo. Také je zde umístěn odkaz *Zapomenuté heslo*, na který po kliku otevře modální okno s formulářem, pomocí kterého lze zažádat o resetování hesla. Pokud se nachází uživatelem zadaný e-mail v databázi registrovaných uživatelů, přijde na tento e-mail automaticky generovaný odkaz, pomocí kterého lze heslo aktualizovat.

Přihlašovací komponenta, stejně jako všechny komponenty, které obsahují formulář, obsahuje validační proces. Tento proces se provádí na klientské části aplikace. Při zadání neplatného formátu e-mailové adresy se pod formulářovým polem zobrazí chybová hláška, která uživatele upozorní na to, že zadaná e-mailová adresa není validní.

Pokud se uživatel pokusí odeslat formulář s jedním polem nevyplněným nebude mu to umožněno a bude upozorněn zase formou chybových hlášek. Formulář se podaří odeslat jen v případě, že jsou obě pole správně vyplněny.

Po úspěšném odeslání formuláře následuje autentifikační proces na backendu aplikace, který je zajišťován službou Firebase authentication. Odesláním formuláře se zavolá funkce *signInWithEmailAndPassword*, která je obsažena ve Firebase SDK. Ta pomocí HTTPS protokolu odešle API se zadanými údaji na Firebase server. Tento Server ověří správnost údajů a pokud jsou správné, pošle zpět odpověď s tokenem pro Firebase Authentication. Klient následně přihlašovací token zpracuje a uloží si ho do paměti pro pozdější použití. Uživatel je následně přihlášen a může začít používat webovou aplikaci. Celý tento proces probíhá asynchronně, a tedy není nutné aktualizovat okno prohlížeče. Tento proces autentizace je znázorněn v sekvenčním diagramu na obrázku 4.7 níže.



Obr. 4.7 Autentifikační proces, zdroj: vlastní

Po úspěšném přihlášení je uživatel přesměrován na domovskou obrazovku aplikace a zobrazí se mu notifikační oznámení, že byl úspěšně přihlášen. Při pokusu o

přihlášení, které je neúspěšné, server vrátí odpověď s kódem 400 a s názvem chyby. V případě, že nastane tato možnost, je uživatel upozorněn notifikační hláškou, že se přihlášení nezdařilo. Na obrázku 4.8 níže je příklad serverové odpovědi při neúspěšném přihlášení z důvodu zadání špatného hesla.

Obr. 4.8 Serverová odpověď při neúspěšném přihlášení, zdroj: vlastní

4.5.2 Hlavička webové aplikace

Pro hlavičky všech stránek webové aplikace je využita pouze jedna sdílená komponenta, která má svůj specifický styl a je zobrazována v celé webové aplikaci. V hlavičce je umístěno hlavní navigační menu, jehož struktura se mění podle oprávnění uživatele. Hlavička také obsahuje informaci o přihlášeném uživateli s možností odhlášení se z aplikace. Na obrázcích 4.9 a 4.10 lze vidět různé podoby této komponenty v závislosti na oprávnění uživatele.

Pro nepřihlášené uživatele hlavička obsahuje pouze navigační menu s položkami, ke kterým má nepřihlášený uživatel přístup. Může se pomocí nich dostat na přihlašovací stránku, na stránku s výpisem jednotlivých konferencí a na domovskou stránku, která má v tomto případě podobu startovací stránky aplikace s informacemi, že pro další akce je nutné se přihlásit.

Obr. 4.9 Hlavička webové aplikace pro nepřihlášené uživatele, zdroj: vlastní

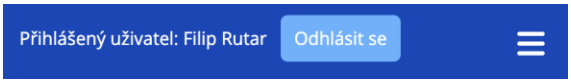
Na obrázku 4.10 níže je zobrazena podoba hlavičky pro uživatele s administrátorskými právy. Lze si všimnout, že se v hlavičce změnilo navigační menu a přibyla informace, který uživatel je k aplikaci přihlášen s možností odhlášení se. Po kliku na toto tlačítko dojde k vymazání všech údajů o stavu přihlášeného uživatele uložených

v lokálním úložišti aplikace. Následně je uživatel informován notifikační hláškou, že byl úspěšně odhlášen.

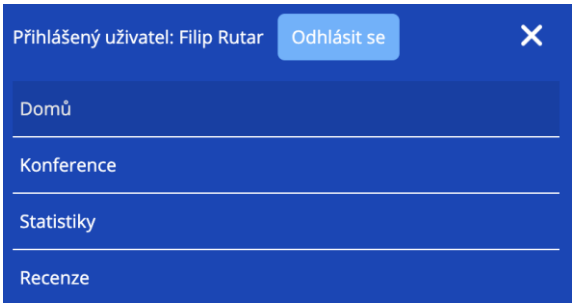
Obr. 4.10 Hlavička webové aplikace pro administrátory, zdroj: vlastní

Tato komponenta je dokonalým příkladem toho, jak lze ve VueJS pracovat s komponentami a podle potřeb dynamicky měnit jejich obsah bez zbytečného duplikování kódu.

Kvůli responzivitě aplikace je komponenta hlavičky webové aplikace optimalizována tak, že se na mobilních zařízeních prvky navigačního menu skryjí a zobrazí se místo nich tzv. *"burger button"*, který následně po kliku jednotlivé položky v menu odkrývá. Tyto stavy lze vidět na obrázcích 4.11 a 4.12.



Obr. 4.11 Responzivní hlavička webové aplikace se skrytým menu, zdroj: vlastní



Obr. 4.12 Responzivní hlavička webové aplikace s odkrytým menu, zdroj: vlastní

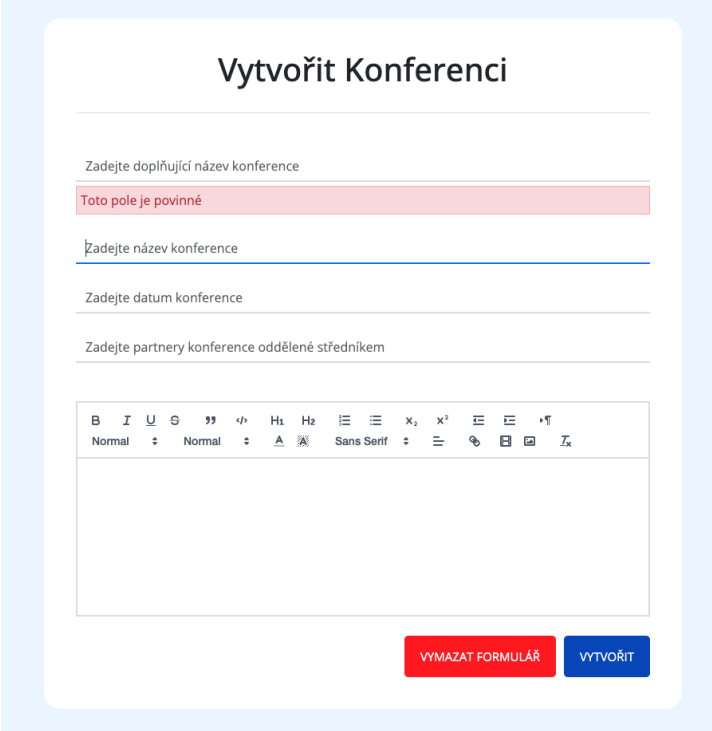
4.5.3 Vytváření konference

Vytváření konferencí je realizováno pomocí komponenty, která v sobě obsahuje formulář určený k tomuto účelu. Lze se do ní dostat z domovské obrazovky za podmínky, že je uživatel přihlášen a jeho účet disponuje administrátorskými právy.

Formulář se skládá z pěti polí, z nichž čtyři jsou povinné a bez jejich vyplnění formulář nelze odeslat. Stejně jako u přihlašovacího formuláře jsou i zde validační podmínky, které musí být úspěšně splněny, aby bylo možné formulář úspěšně odeslat. Pokud se uživatel pokusí odeslat formulář bez vyplnění povinných polí, odeslání mu nebude umožněno a pod každým nevyplněným polem se zobrazí chybová hláška, která uživatele informuje, že je pole potřeba vyplnit. Tyto povinné pole obsahují název

konference, doplňující název konference, datum konání konference a partnery konference.

Vyplňovat páté pole formuláře povinné není, jelikož je toto pole tvořeno integrovaným wysiwyg (what you se eis what you get) editorem, pomocí kterého uživatel může vytvořit vlastní doplňující obsahovou část konference. Tento editor obsahuje integrované funkcionality pro úpravu vkládaného textu a jeho stylování. Při odeslání formuláře s validními daty bude tento vstup převeden do HTML formátu a do databáze uložen jako řetězec. Při zobrazování obsahu konference v komponentě, určené k tomuto účelu, bude tento HTML kód vložen do určeného místa v komponentě následně zobrazen webovým prohlížečem. Komponenta určená pro vytváření konferencí je k nahlédnutí na obrázku 4.13 níže.

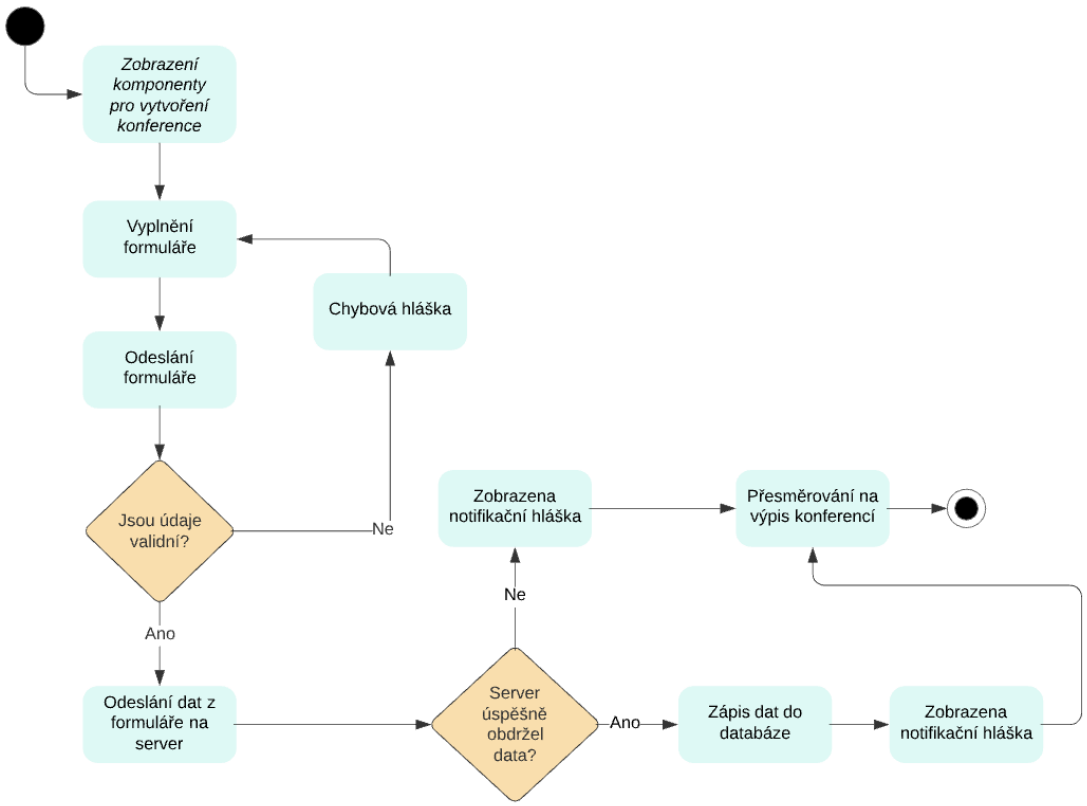


Obr. 4.13 Komponenta k vytváření konferencí, zdroj: vlastní

Všechny formuláře, které webová aplikace jsou kvůli přehlednosti nastylovány do jednotného formátu. Všechny pole jsou podtrženy šedou linkou, která se při focusu daného pole obarví do modra. Na obrázku 4.13 výše si lze všimnout, že pod prvním polem je vygenerována chybová hláška, která říká, že pole je povinné k vyplnění. Je to z důvodu, že validační funkce v aplikaci je reaktivní a reaguje na opuštění focusovaného pole, před jeho vyplněním. Jestliže tedy uživatel klikne do prvního pole a následně klikne do pole následujícího, před jeho řádným vyplněním, zobrazí se chybová hláška.

Dále si lze všimnout, že formulář obsahuje dvě tlačítka, jedno slouží k odesílání formuláře a druhé po kliku vymaže vyplněný formulář. Toto tlačítko tam je z důvodu komfortu při opravování nechtěných vstupů. Pokud by uživatel formulář vyplnil a poté si řekl, že ho chce vyplnit jinak, stačí kliknout na jedno tlačítko místo toho, aby všechna pole mazal postupně po jednom.

Při úspěšném odeslání formuláře dojde k zápisu dat do databáze a je mu zobrazena notifikační hláška, že konference byla úspěšně vytvořena. Při neúspěšném zápisu dat do databáze, který může nastat například z důvodu nedostupnosti připojení k serveru, je uživateli zobrazena notifikační hláška, že konference nebyla vytvořena. Uživatel je následně automaticky přesměrován na komponentu s výpisem konferencí, kde je při úspěšném zápisu do databáze nově vytvořená konference k nahlédnutí. Celý proces vytváření konferencí je znázorněn pomocí aktivity diagramu níže na obrázku 4.14.



Obr. 4.14 Komponenta k vytváření konferencí, zdroj: vlastní

4.5.4 Zobrazení přehledu konferencí

Komponenta pro zobrazování přehledu konferencí je navržena, tak aby jednotlivé vytvořené konference zobrazovala ve formě dlaždic, které se řadí vedle sebe, nebo pod sebou podle velikosti zobrazovací plochy. Jednotlivé vytvořené konference jsou zde

zobrazovány od nejnovější po nejstarší. Každá dlaždice zobrazuje název konference a její datum vytvoření.

Po kliku na dlaždici dojde k přesměrování na detail konference, jehož obsah se do komponenty detailu konference načte z databáze pomocí speciálního identifikátoru.

Tato komponenta je dostupná z domovské obrazovky aplikace anebo skrze hlavní navigační menu webové aplikace. Komponenta pro zobrazování přehledu konferencí jek nahlédnutí na obrázku 4.15 níže.

Obr. 4.15 Přehled vytvořených konferencí, zdroj: vlastní

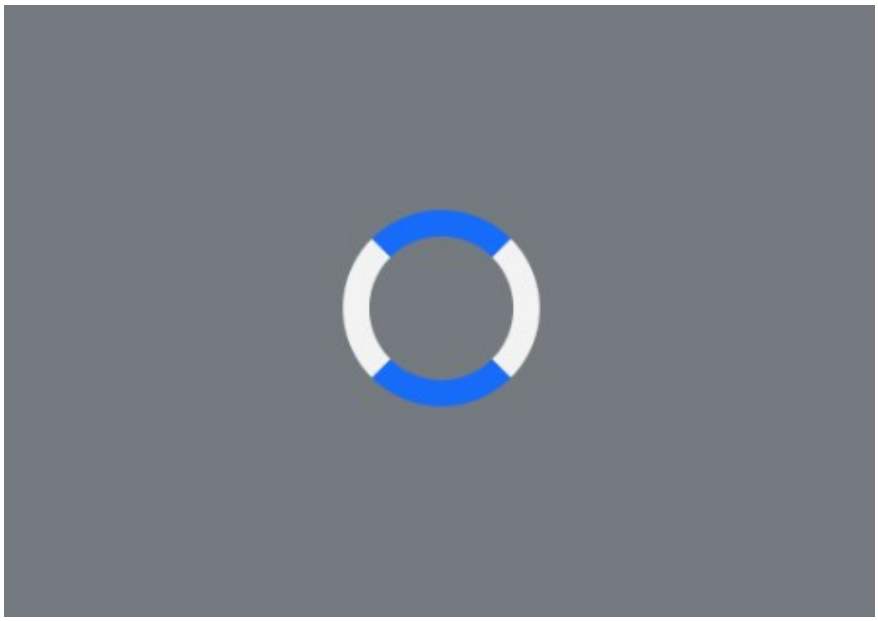
4.5.5 Detail konference

V první části této podkapitoly bude popsán vzhled a funkčnost komponenty na zobrazování detailu jednotlivých konferencí. Ve druhé části budou následně popsány všechny funkcionality sloužící ke správě konference a zobrazování jednotlivých přehledů informací, které se dané konference týkají.

Komponenta pro zobrazování detailu konference zobrazuje informace, spojené s jednotlivými konferencemi pomocí jejich speciálního identifikátoru, uloženém v databázi. Detail konference je přístupný po kliku na určitou konferenci v přehledu konferencí. Data, která jednotlivé konference obsahují jsou většinou obsáhlá, a proto nejsou do aplikace načítána hned při startu aplikace, ale až při zavolání komponenty detailu konference.

Načítání dat probíhá asynchronně hned po tom, co se struktura komponenty vykreslí do DOM. K tomuto účelu je využit jeden z hooků životního cyklu VueJS komponenty, který tento framework poskytuje. Tento hook je volán pomocí VueJS integrované funkce *onMounted*. Data jsou následně načtena do reaktivního objektu, pomocí kterého lze v komponentě dále s daty pracovat a přizpůsobovat je podobě, kterou potřebujeme.

Načítání takového většího objemu dat do aplikace a jejich následné převádění na požadované výstupy, zobrazené na obrazovce většinou zabere nějaký čas, a proto je potřeba uživatele nějakým způsobem upozornit, že se komponenta načítá. K tomuto účelu je zde využita načítací komponenta, která se zobrazí přes celou stránku, než se tato data načtou. Tato komponenta má tmavé průhledné pozadí a uprostřed načítací kolečko, které je plynule animováno tak, aby bylo uživateli jasné, že se aplikace nezasekla, ale načítá data. Příklad načítací komponenty si lze prohlédnout na obrázku 4.16 níže.



Obr. 4.16 Načítací komponenta, zdroj: vlastní

Podoba struktury a stylu komponenty je inspirována předlohou, která byla poskytnuta při pravidelných schůzkách se zadavatelem. Obsahová struktura detailu konference se dá rozdělit na dvě části.

První část je tvořena fixním layoutem, inspirovaným předlohou. Do určitých míst této části komponenty se dynamicky propisují data z databáze, které obsahují informace o termínu konání konference, názvu a partnerech konference.

Do druhé části layoutu komponenty se propisují data, které jsou v databázi uloženy v podobě řetězce, obsahujícího HTML kód, generovaného pomocí wysiwyg editoru, popsaného v podkapitole 4.5.3 výše. Tato část je kompletně uživatelsky editovatelná a může obsahovat jakékoliv dodatečné informace o konferenci včetně obrázků. Příklad zobrazení detailu konference je k nahlédnutí na obrázku 4.17 níže.

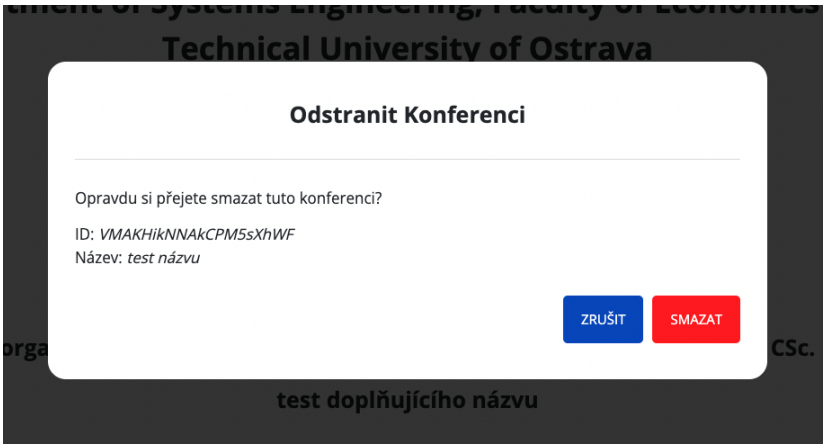
Obr. 4.17 Detail konference, zdroj: vlastní

Dále si je možné všimnout pěti tlačítek, umístěných v pravé horní části komponenty. Jedná se o tlačítka, pomocí kterých lze konferenci spravovat a zobrazovat přehledy informací, které jsou s konferencí spojeny. Všechna tato tlačítka, kromě tlačítka sloužícího pro registraci na konferenci, jsou přístupná pouze uživatelům s oprávněním administrátora. Jednotlivé akce, související se správou konferencí jsou realizovány pomocí modálních oken.

Využití modálních oken je v moderních webových aplikacích velmi oblíbené hlavně pro zobrazování důležitých informací, které je potřeba uživateli předat a je zbytečné pro ně zakládat vlastní nové stránky aplikace. Velkou výhodou těchto oken je, že jsou zobrazovány jen v případě potřeby po kliku na některé z tlačítek uvedených výše v textu.

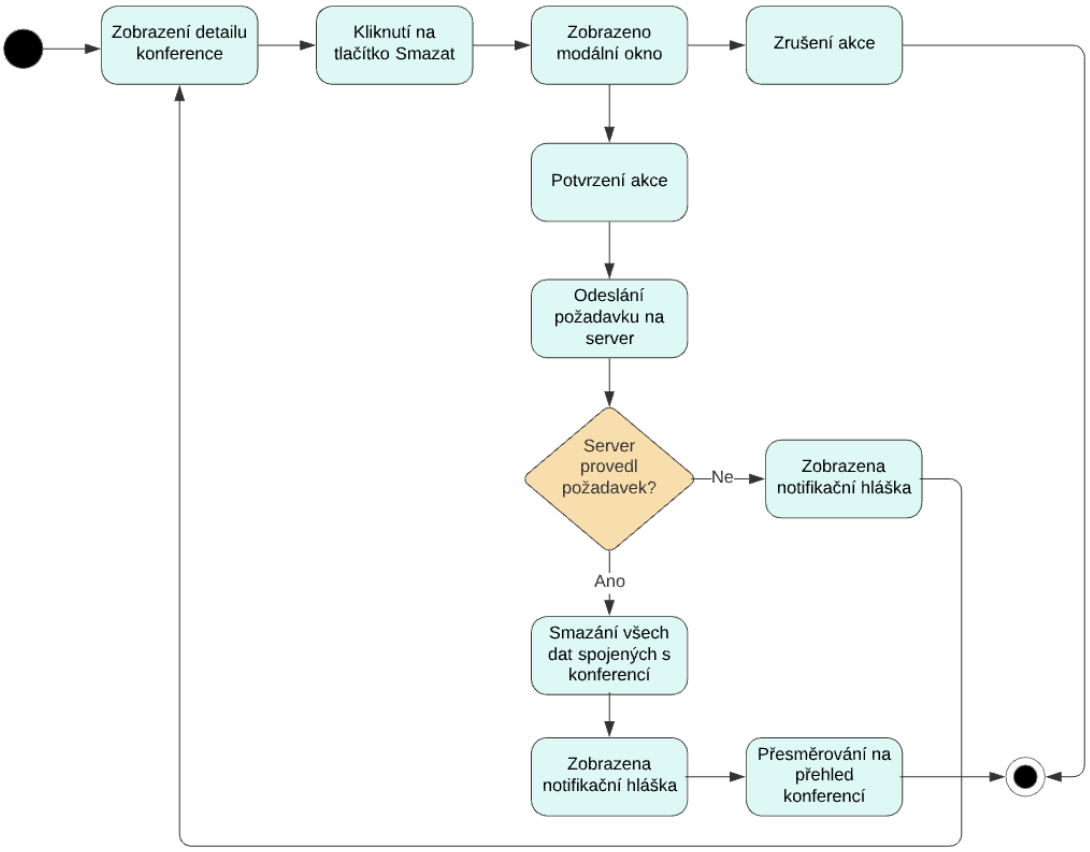
Mazání konference

První akcí, kterou může uživatel s oprávněním administrátora provádět je mazání konference. Tato akce se provede po potvrzení modálního okna, které se na obrazovce objeví po kliku na tlačítko *Smazat* umístěného v pravé horní části komponenty. Toto modální okno obsahuje informace, které se týkají spravované konference a má uživatele upozornit, zda chce opravdu danou konferenci vymazat. Modální okno obsahuje dvě tlačítka, z nichž jedno slouží pro potvrzení akce a druhé pro zrušení, které modální okno opět zavře. Příklad tohoto modálního okna je k nahlédnutí na obrázku 4.18 níže.



Obr. 4.17 Modální okno pro smazání konference, zdroj: vlastní

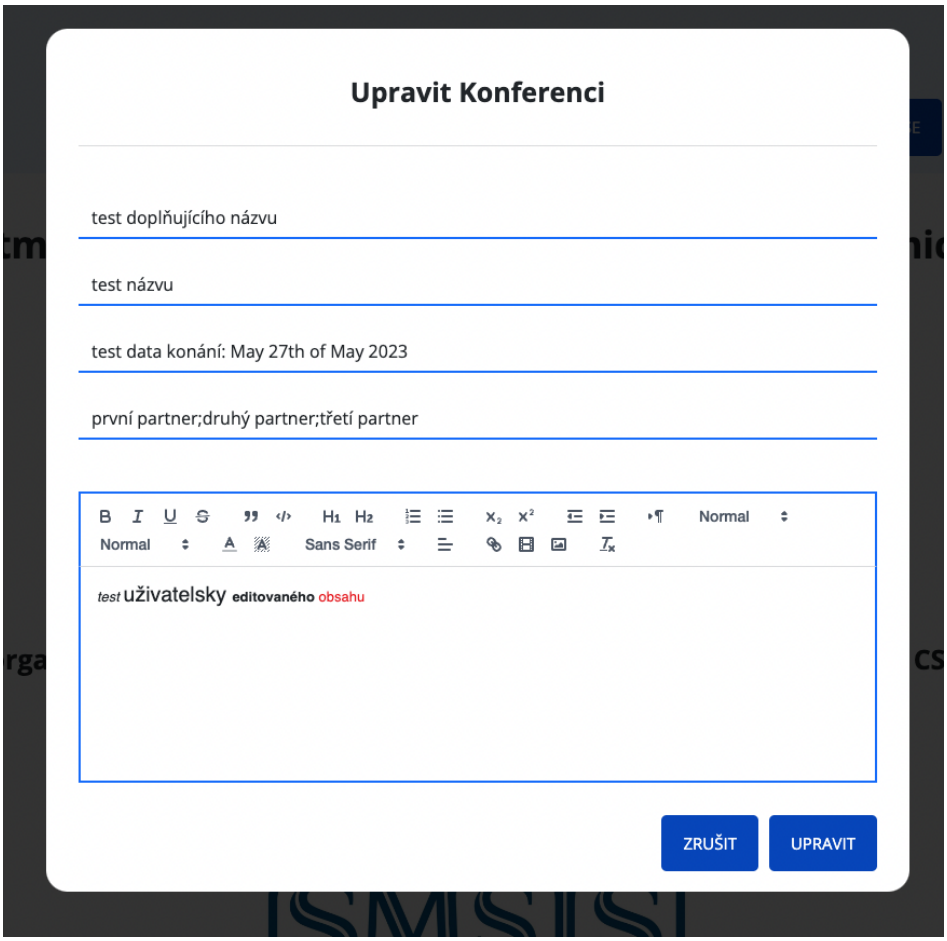
Při potvrzení této akce v modálním okně se zavolá funkce, která odešle na Firebase server požadavek na vymazání dat spojených s danou konferencí, včetně recenzí, které jsou s danou konferencí spojeny. Pokud je tento požadavek serverem úspěšně realizován, je uživateli zobrazena notifikace, že byla konference úspěšně smazána a následně je přesměrován na přehled konferencí. Pokud nastane nějaký problém a server nedokáže požadavek zpracovat, je uživatel upozorněn notifikační hláškou, že konference nemohla být smazána. Celý proces mazání konferencí je znázorněn pomocí aktivity diagramu na obrázku 4.18 níže.



Obr. 4.18 Proces mazání konference, zdroj: vlastní

Úprava konference

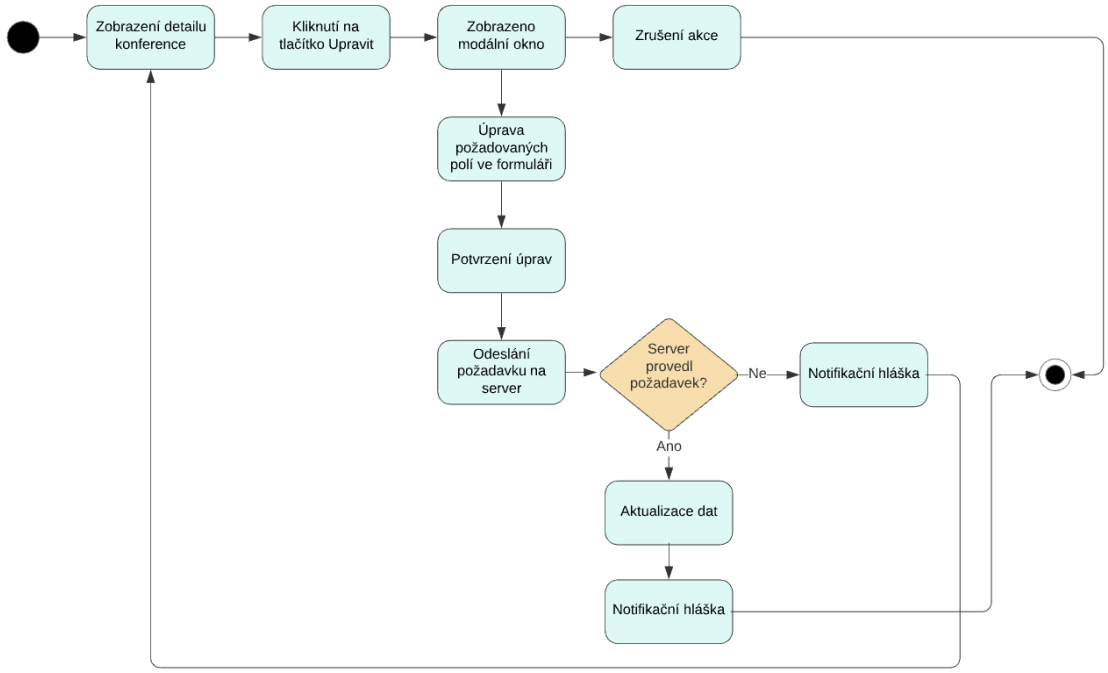
Další akcí, kterou může administrátor webové aplikace provádět na této komponentě je úprava obsahu dané konference. Po kliku na tlačítko *Upravit*, které se nachází hned vedle tlačítka *Smazat* se otevře modální okno, které v sobě obsahuje formulář stejné podoby jako na stránce pro vytváření konferencí. Při zavolání funkce, která se aktivuje po kliku na tlačítko se spolu s otevřením modálního okna do příslušných polí formuláře nahrají data, která jsou určená k úpravě. Tato data se nestahují z databáze, ale jsou do formuláře předávány rodičovskou komponentou, což je komponenta, zobrazující detail konference. Příklad tohoto modálního okna je k dispozici na obrázku 4.19 níže.



Obr. 4.19 Modální okno pro úpravu konference, zdroj: vlastní

Jelikož jsou data uloženy v reaktivním objektu komponenty, už jen pouhá změna, provedená v těchto formulářových polích data v objektu upraví a tyto úpravy se okamžitě propíší do komponenty, kde je zobrazen detail konference. Takto provedená změna by se ale při aktualizaci prohlížeče zase vymazala a konference by obsahovala neaktuální data. Proto se zde nachází tlačítko pro potvrzení změn, které při kliku zavolá funkci, která odešle aktualizovaná data na Firebase server, který provede aktualizaci dat v databázi. Dále se zde nachází opět tlačítko na zrušení změn, které modální okno zase zavře.

Po úspěšném odeslání požadavku na server se data v databázi aktualizují a uživatel je opět upozorněn notifikační hláškou, že jsou změny úspěšně uloženy. V opačném případě je uživatel také upozorněn notifikační hláškou, že změny nebyly provedeny. Celý proces úpravy konference je znázorněn pomocí aktivity diagramu na obrázku 4.20 níže.

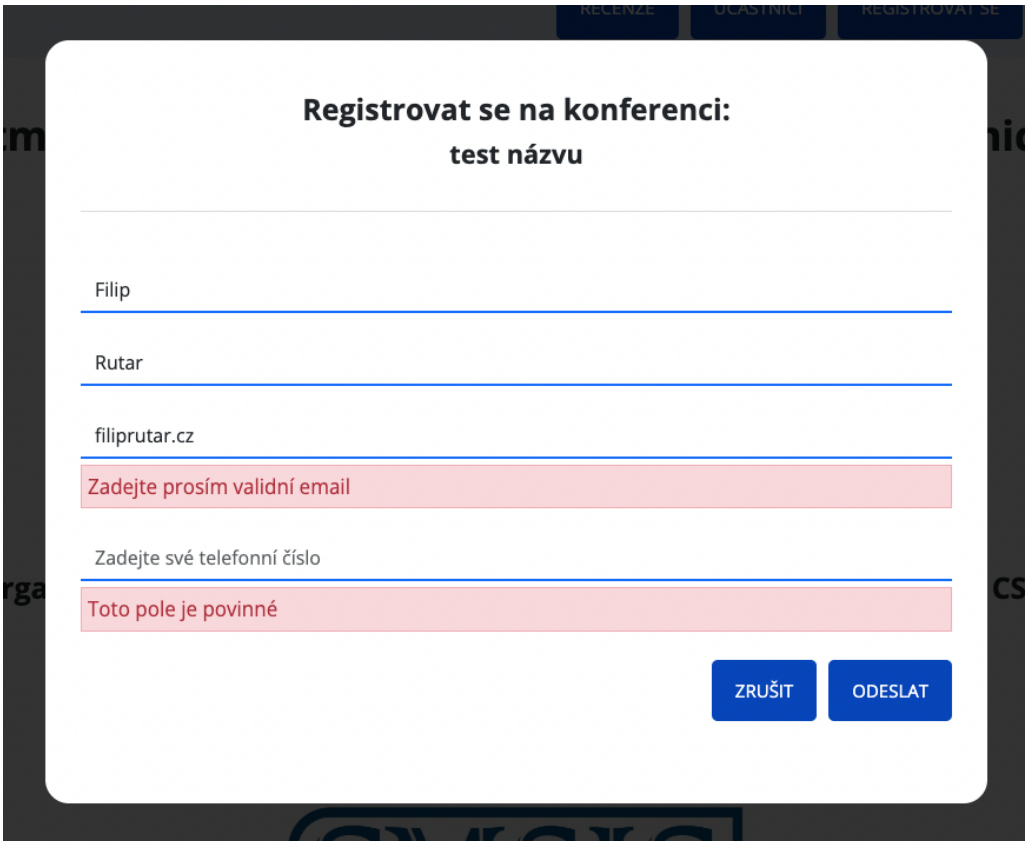


Obr. 4.20 Proces úpravy konference, zdroj: vlastní

Registrace na konferenci

Tato akce je jako jediná přístupná všem druhům uživatelů webové aplikace. Opět je jako všechny předchozí popsané funkcionality přístupná přes modální okno. Toto modální okno se otevře po kliku na tlačítko *Registrovat se* a obsahuje zase formulář, který má ovšem jinou strukturu. Jeho strukturu tvoří čtyři pole pro zadání křestního jména, příjmení, e-mailové adresy a telefonního čísla.

Pro všechna tato pole jsou nastaveny validační podmínky, bez jejichž splnění nelze formulář odeslat. Každé pole formuláře je povinné a jestliže se uživatel pokusí formulář odeslat bez jejich vyplnění, zobrazí se pod jednotlivými poli chybové hlášky, které uživatele upozorní, že je potřeba tato pole vyplnit. Pole pro zadání e-mailové adresy a telefonního čísla také požadují, aby byla vyplněna ve správném formátu. Na obrázku 4.21 níže se nachází příklad chybně vyplněného formuláře, který neobsahuje validní formát e-mailové adresy a nemá vyplněné telefonní číslo.

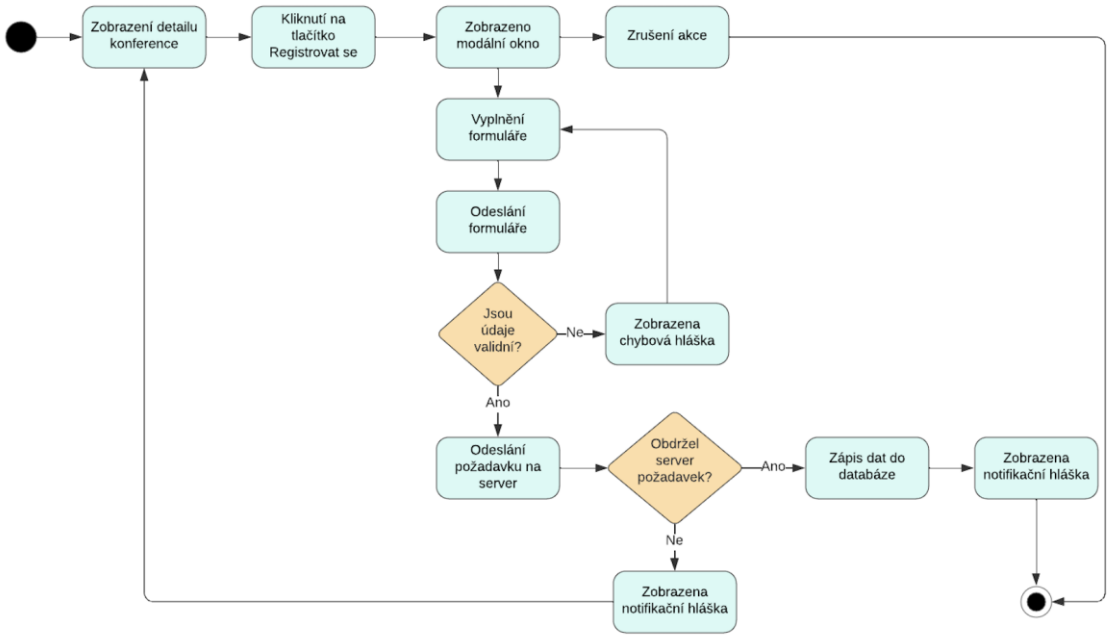


Obr. 4.21 Modální okno pro registraci na konferenci, zdroj: vlastní

Po vyplnění formuláře validními údaji lze tento formulář odeslat pomocí příslušného tlačítka a pokud se uživatel rozhodne akci neprovádět, tak je zde druhé tlačítko, sloužící k zavření modálního okna.

Po kliknutí na tlačítko *Odeslat* dojde ke kontrole, zda zadaný uživatel již na konferenci není zaregistrován. Pokud uživatel do registračního formuláře zadá e-mailovou adresu, která je v seznamu registrovaných již vedena, nebude mu registrace umožněna a bude upozorněn notifikační hláškou, že je tento uživatel na tuto konferenci již registrován.

Po úspěšném odeslání požadavku na registraci ke konferenci na Firebase server, proběhne zápis těchto dat do databáze a uživatel bude informován notifikační hláškou, že byl na konferenci úspěšně registrován. Pokud Firebase server úspěšně požadavek nezpracuje, bude uživatel upozorněn notifikační hláškou, že registrace na konferenci nebyla úspěšně dokončena. Průběh procesu registrace na konferenci je znázorněn pomocí aktivity diagramu na obrázku 4.22 níže.



Obr. 4.22 Proces registrace na konferenci, zdroj: vlastní

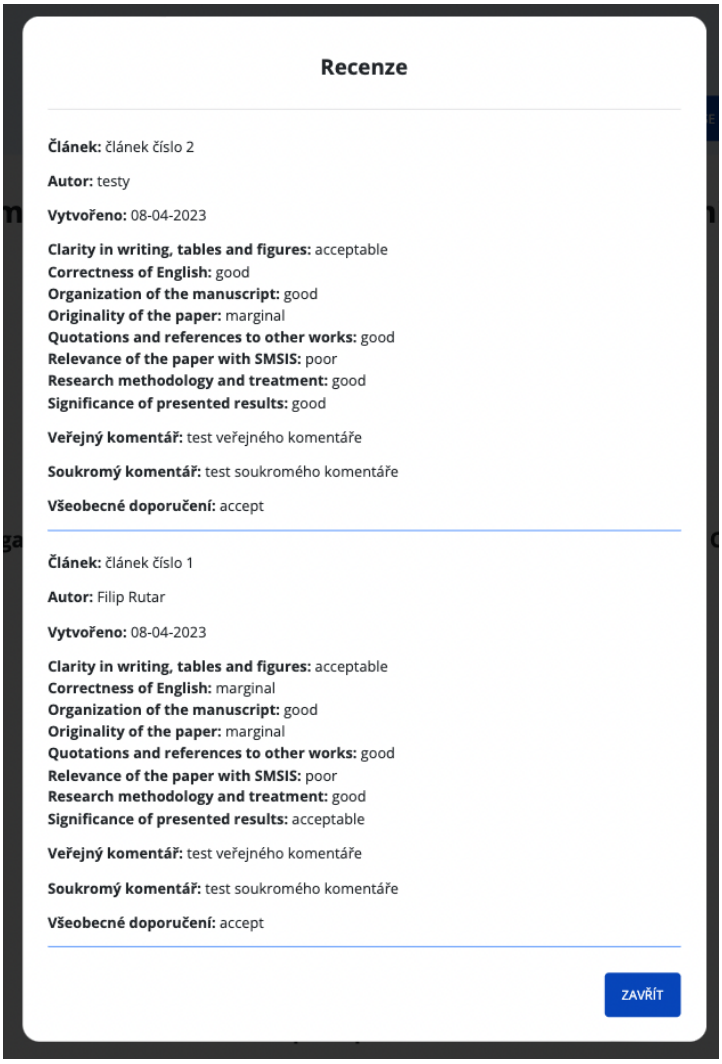
Přehled registrovaných účastníků

Zobrazení přehledu registrovaných účastníků je realizováno opět pomocí modálního okna. Pro zobrazení tohoto přehlede je zapotřebí být do webové aplikace přihlášen jako administrátor. Modální okno v tomto případě obsahuje tabulku s jednotlivými údaji o registrovaných účastnících konference. Tyto údaje se skládají z jména a přímení účastníka, e-mailové adresy a telefonního čísla. Data, která obsahují tyto informace jsou modálnímu oknu předány rodičovskou komponentou ve formě reaktivního objektu, tudíž neprobíhá načítání těchto dat z databáze až při zobrazení modálního okna, což by celý průběh zobrazení zpomalovalo. V tomto případě modální okno obsahuje pouze jedno tlačítko, které slouží k jeho skrytí. Příklad přehledu registrovaných účastníků je k nahlédnutí na obrázku 4.23 níže.

Obr. 4.23 Modální okno pro zobrazení přehledu registrovaných, zdroj: vlastní

Přehled recenzí

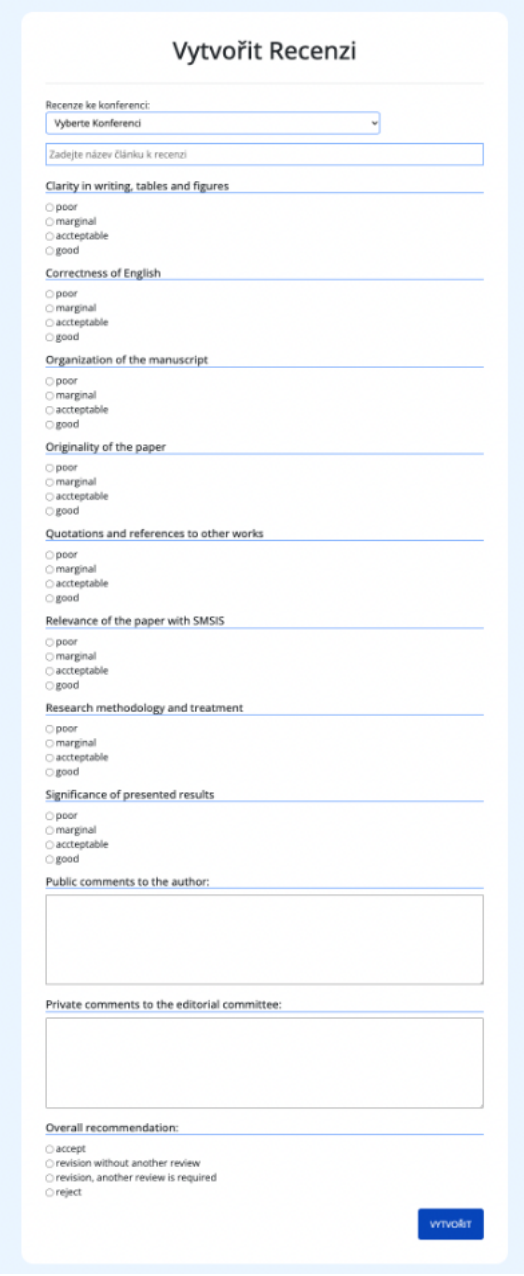
Zobrazení přehledu recenzí pro určitou konferenci je zde realizováno opět za pomoci modálního okna. Pro zobrazení tohoto přehledu je opět potřeba být přihlášen jako administrátor. Data, která obsahují informace o jednotlivých recenzích jsou modálnímu oknu předávány, opět stejně jako u přehledu registrovaných účastníků, formou reaktivního objektu. Jednotlivé recenze zde zobrazeny formou bloků, obsahujících jednotlivé informace, které jsou od sebe odděleny modrou linkou a jsou seřazeny podle sestupně podle data vytvoření. Jednotlivé recenze obsahují informace jako název článku, pro který je recenze vytvořena, autora recenze, kdy byla recenze vytvořena, jednotlivá hodnotící kritéria a dále soukromý a veřejný komentář. Tvorba recenzí bude popsána v následující podkapitole. Příklad modálního okna, obsahujícího přehled jednotlivých recenzí pro danou konferenci je k nahlédnutí na obrázku 4.24 níže.



Obr. 4.24 Modální okno pro zobrazení přehledu recenzí, zdroj: vlastní

4.5.6 Vytváření recenzí

Komponenta, která slouží pro vytváření recenzí obsahuje formulář s poli pro výběr konference, ke které má recenze být přiřazena, název článku dané konference, kterého se recenze týká, jednotlivá hodnotící kritéria, veřejný a soukromý komentář. Je přístupná pouze pro uživatele s právy recenzenta a lze se na ní dostat přes domovskou obrazovku webové aplikace nebo pomocí hlavního navigačního menu, umístěného v hlavičce webové aplikace. Komponenta, která slouží k vytváření jednotlivých recenzí je k nahlédnutí na obrázku 4.25 níže.



Obr. 4.25 Komponenta k vytváření recenzí, zdroj: vlastní

Výběr konference je v tomto formuláři řešen pomocí select boxu, kde jsou uloženy názvy všech konferencí, které jsou ve webové aplikaci vytvořeny. Uživatel si následně pomocí tohoto boxu vybere konferenci, ke které má být recenze přiřazena. Další pole je určeno pro zadávání názvu článku, kterého se daná recenze týká. Poté následují hodnotící bloky, z nichž každé obsahuje název a čtyři hodnotící kritéria, které jsou řešeny pomocí výběrových polí. Tyto čtyři možnosti obsahují pro každý blok hodnocení *poor*, *marginal*, *acceptable* a *good*. V každém bloku je možnost vybrat pouze jedno kritérium.

Poté následují pole pro zadání veřejných komentářů, které jsou viditelné všem uživatelům a soukromých komentářů, které jsou viditelné pouze administrátorovi aplikace.

Poslední částí formuláře k vytváření recenzí je všeobecné doporučení, které je realizováno opět pomocí výběrových polí a obsahuje možnosti *accept*, *revision without another review*, *revision, another review is required* a *reject*.

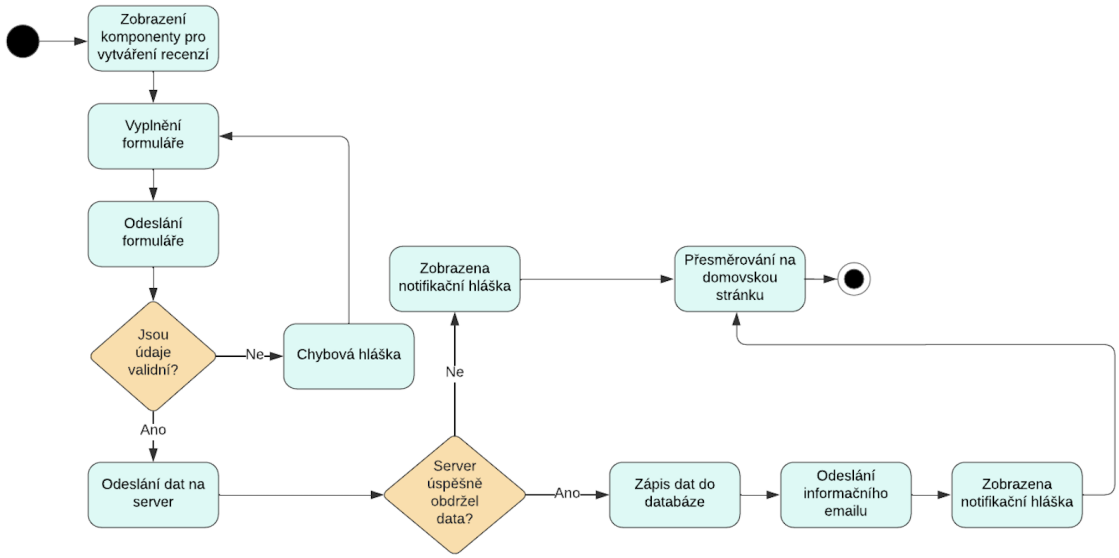
Každé pole je povinné a při pokusu o odeslání formuláře bez vyplnění některého pole je uživatel upozorněn chybovou hláškou stejně jako ve všech ostatních formulářích, popsaných v předchozích kapitolách.

Po úspěšném splnění validačních podmínek lze formulář odeslat pomocí příslušného tlačítka *Vytvořit*. Po kliku na toto tlačítko se zavolá funkce, která je k dispozici přes Firebase SDK a odešle se požadavek na server, který po jeho úspěšném obdržení vytvoří dokument s vyplněnými informacemi se specifickým identifikátorem v kolekci *reviews*, dostupné v databázi této webové aplikace. Jakmile je obdržena odpověď, zaslaná serverem zpět klientovi, je uživatel upozorněn notifikační hláškou, jestli byla recenze úspěšně vytvořena nebo v opačném případě nebyla.

Průběh procesu vytváření recenzí je velmi podobný procesu vytváření konferencí, popsaného v podkapitole 4.5.3. Tento proces pro vytváření recenzí se liší jen v pár detailech a je k nahlédnutí na obrázku 4.26 níže.

Po úspěšném zápisu recenze do databáze je vygenerován automatický informační e-mail, který je odeslán na adresu uživatele, který danou recenzi vytvořil a jeho kopie je odeslána na adresu administrátora webové aplikace. E-mail je odesílán pomocí služby Email.js, které se za pomoci poskytnutého SDK odešle požadavek s daty, které se nahrají připravené šablony. Vygenerovaný e-mail obsahuje URL adresu vytvořené recenze a

konference, ke které je recenze přiřazena. Dále obsahuje informace o uživateli, který recenzi vytvořil.



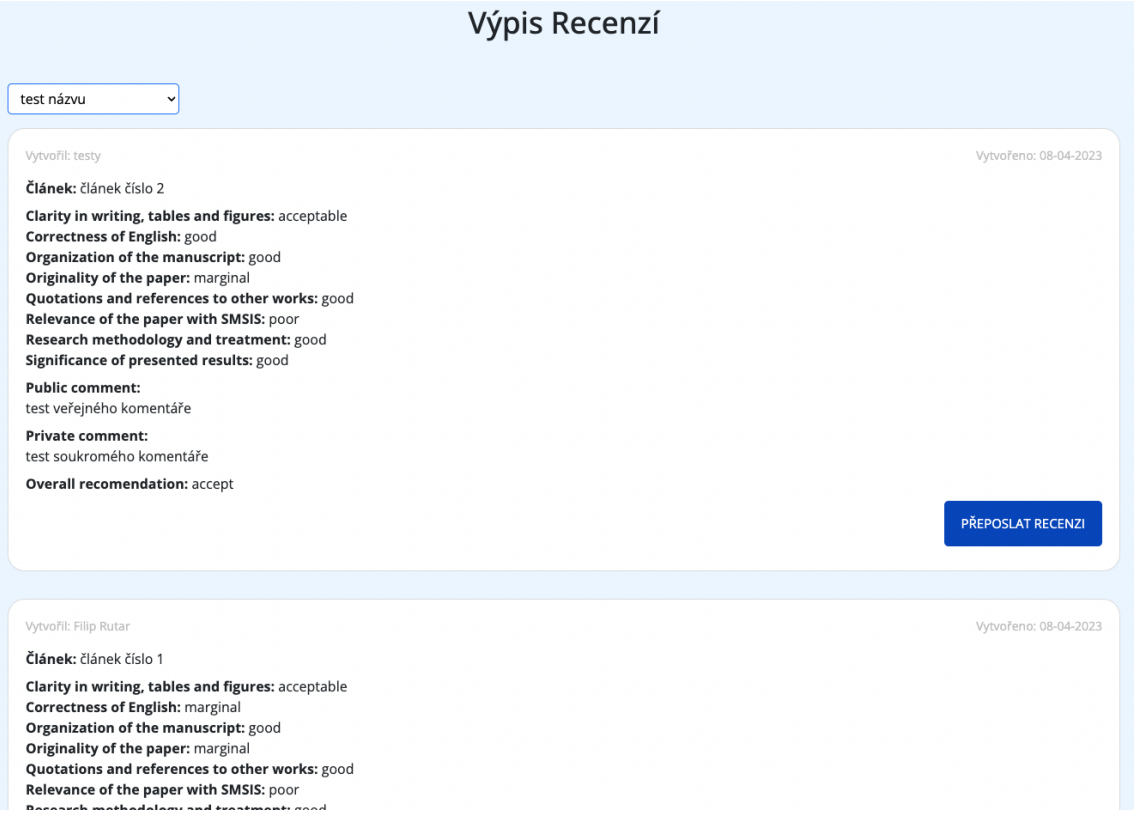
Obr. 4.26 Proces vyváření recenze, zdroj: vlastní

4.5.7 Přehled vytvořených recenzí

V této komponentě jsou k nahlédnutí všechny recenze, které byly pomocí této webové aplikace vytvořeny. Komponenta je přístupná pouze administrátorovi webové aplikace a lze se na ní dostat pomocí domovské obrazovky webové aplikace nebo přes navigační menu.

V komponentě je umístěn select box, pomocí kterého lze vybrat konferenci, pro kterou se mají zobrazit vytvořené recenze. Jednotlivé recenze jsou zde zobrazeny v podobě bloků pod sebou, podobně jako v modálním okně, přístupnému v komponentě detailu konference.

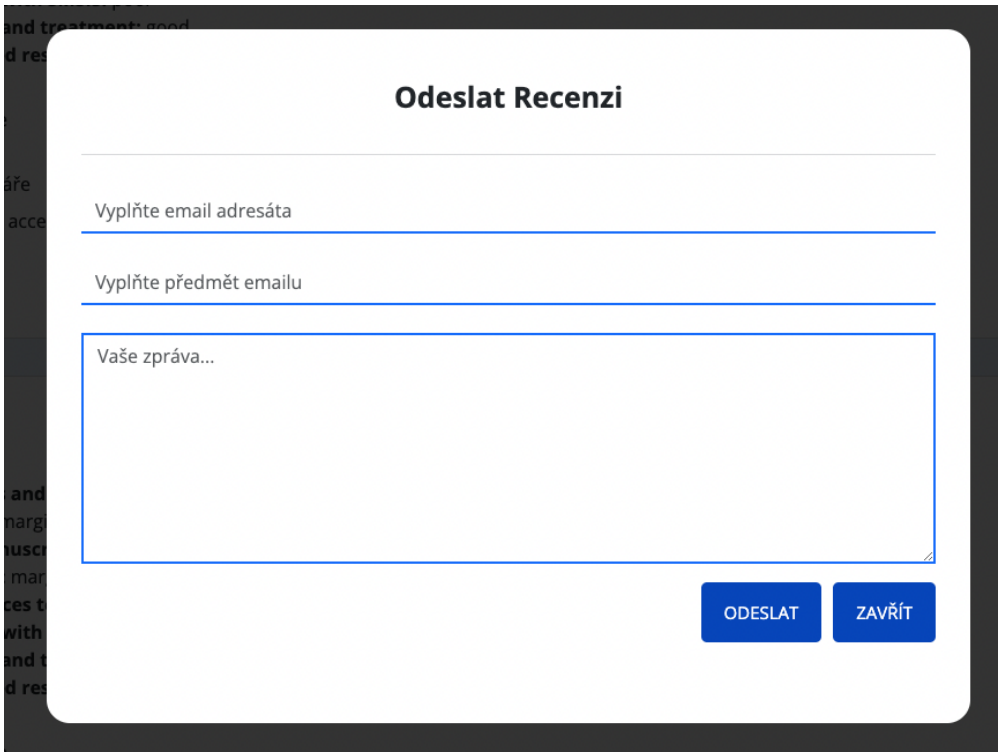
Po vyfiltrování recenzí pomocí select boxu se zobrazí recenze, které souvisí s vybranou konferencí. U každé z vypsaných recenzí je možnost ji přeposlat na e-mail autorovi recenzovaného článku. Tato akce lze provádět pomocí příslušného tlačítka, umístěného v pravém dolním rohu každé konference. Náhled komponenty, která slouží k výpisu recenzí si lze prohlédnout na obrázku 4.27 níže. Náhled obsahuje výpis recenzí, určených k vytvořené testovací konferenci.



Obr. 4.27 Komponenta pro zobrazování přehledu recenzí, zdroj: vlastní

Po kliknutí na tlačítko *Přeposlat recenzi* vyskočí modální okno s formulářem, který je tvořen třemi poli, které slouží k vytvoření obsahu požadovaného e-mailu. Tato pole jsou určeny k zadání e-mailové adresy adresáta, předmětu e-mailu a zprávy určené adresátovi. Modální okno v tomto případě obsahuje také dvě tlačítka, z nichž jedno slouží k odeslání e-mailu s požadovaným obsahem a druhé ke zrušení akce a zavření modálního okna. Formulář je, jako všechny ostatní formuláře v této webové aplikace, ošetřen validačními podmínkami, bez jejichž splnění nelze formulář odeslat.

Při splnění validačních podmínek se požadavek s daty správného formátu odešle na server již uvedené služby Email.js, která následně odešle e-mail požadovanému adresátovi. Obsahem e-mailu jsou také automaticky odkazy v podobě URL adres na detail recenze a také na konferenci spojenou s touto recenzí. Po kliku na odkaz spojený s detailem přeposlané recenze, je uživateli zobrazena požadovaná recenze s veřejně přístupnými informacemi. To znamená, že pokud není uživatel do webové aplikace přihlášen jako administrátor, je mu skryt obsah soukromého komentáře. Příklad modálního okna s výše popsaným formulářem je k nahlédnutí na obrázku 4.28 níže.



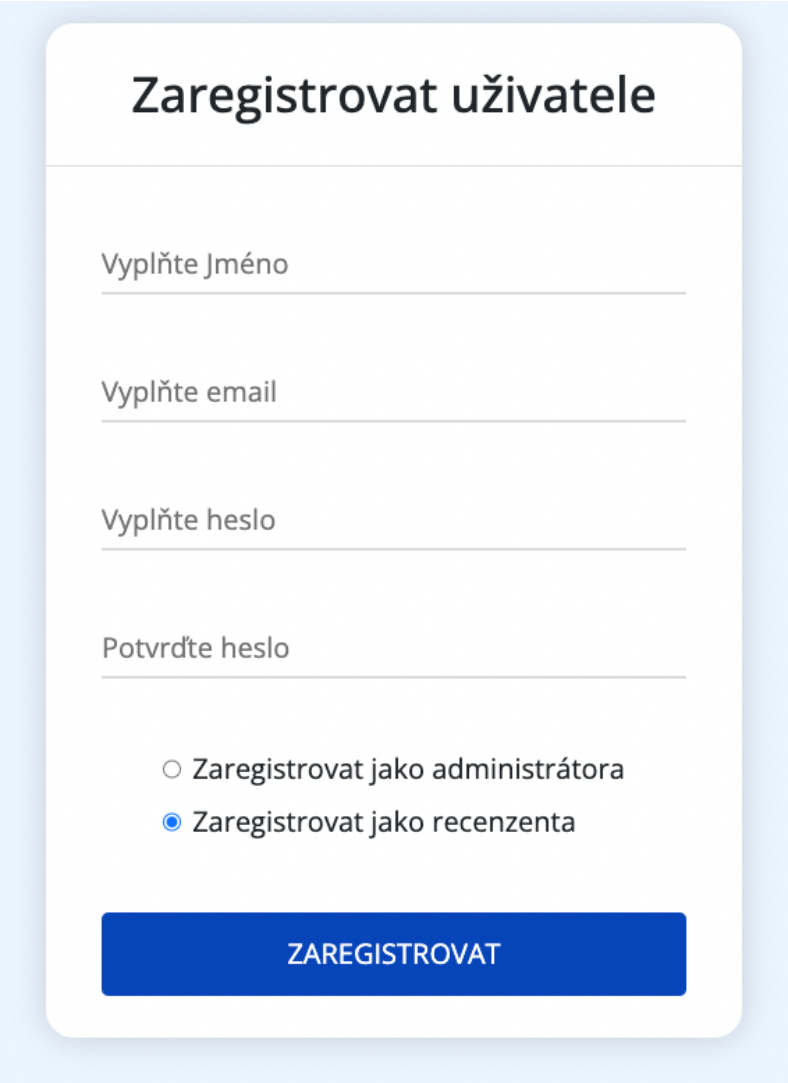
Obr. 4.28 Modální okno pro přeposílání recenzí, zdroj: vlastní

4.5.8 Registrování uživatelů aplikace

Předposlední komponentou webové aplikace je komponenta sloužící pro registraci uživatelů aplikace. Zaregistrovat nové uživatele má být dle požadavků schopen pouze administrátor webové aplikace, proto je tato komponenta přístupná pouze z administrátorského účtu. Tato komponenta obsahuje formulář, který obsahuje čtyři pole pro zadání jména, e-mailové adresy, přes kterou se uživatel bude do aplikace přihlašovat a dále pole pro zadání hesla. Čtvrté pole slouží pro opakované zadání hesla, které se musí shodovat. Pokud se hesla neshodují, je na to uživatel upozorněn v podobě chybové hlášky. Dále formulář obsahuje výběrové boxy pro výběr, zda má mít uživatel administrátorské oprávnění, nebo mu mají být přiděleny práva recenzenta. Všechna tato pole jsou povinná a jsou opět ošetřeny příslušnými validačními podmínkami, bez jejichž splnění nelze požadavek na registraci uživatele odeslat.

Při úspěšném potvrzení formuláře se pošle požadavek s daty na server Firebase, který pomocí služby Firebase authentication provede kontrolu, jestli uživatel již neexistuje. Pokud ne, tak je uživatelský účet vytvořen v interní databázi služby Firebase authentication a následně jsou uloženy do databáze Firestore také doplňující údaje o něm. Těmito doplňujícími údaji jsou jeho jméno a oprávnění.

Po úspěšném vytvoření uživatelského účtu server pošle odpověď na požadavek klientovi se stavovým kódem 200, který značí že byl požadavek úspěšně splněn a uživatel je upozorněn notifikační hláškou, která ho informuje, že byl požadovaný uživatelský účet vytvořen. V opačném případě pošle server odpověď se stavovým kódem 400 a uživatel je upozorněn notifikační hláškou, že vytvoření uživatele nebylo úspěšné. Komponenta, která slouží k registraci uživatelů do webové aplikace je k nahlédnutí na obrázku 4.29 níže.



Obr. 4.29 Komponenta pro registraci uživatelů, zdroj: vlastní

Poslední komponentou této webové aplikace je komponenta pro zobrazování statistik. Tato komponenta slouží pouze pro zobrazení počtu zaregistrovaných uživatelů, vytvořených konferencí a vytvořených recenzí. Do webové aplikace byla tato komponenta přidána pouze z důvodu přehledu, a proto je zde zmíněna pouze okrajově.

4.6 Zabezpečení webové aplikace

Protože budou tuto aplikaci využívat uživatelé s různým oprávněním a některé komponenty mají být určeny pouze pro oprávněné uživatele, je potřeba tyto komponenty webové aplikace nějakým způsobem zabezpečit. Akce, na které uživatelé nemají oprávnění jsou již zabezpečeny na backendové úrovni za pomoci pravidel, které jsou popsány v podkapitole 4.3.3. Pomocí těchto pravidel je nastaveno, že uživatelé, kteří nemají příslušné oprávnění, nemohou manipulovat s určitými daty. Toto zabezpečení není dostačující z důvodu, že se na příslušné komponenty mohou stále dostat například pomocí zadání odkazu na příslušnou stránku do webového prohlížeče.

Například kdyby se přihlášený uživatel chtěl dostat na přihlašovací komponentu, tak by mu stačilo do URL adresy dosadit */login*, což by nemělo být možné. Proto je nutné zabezpečení provést i v klientské části webové aplikace.

Jelikož se jedná o single page webovou aplikace, je veškeré směrování na příslušné stránky řešeno dynamicky programovacím jazykem JavaScript. Ve frameworku VueJS je k tomuto účelu využíván interní plugin vue-router, pomocí kterého lze také požadované cesty zabezpečit. V následujícím obrázku 4.30 lze vidět příklad zabezpečení výše zmíněné cesty na přihlašovací komponentu aplikace.

```
const routes = [
    {
        path: '/',
        name: 'home',
        component: Home
    },
    {
        path: '/login',
        name: 'login',
        component: Login,
        beforeEnter: (to, from) => {
            const storeAuth = useAuthStore()
            if (storeAuth.user.id) {
                return to.name = '/'
            }
        }
    },
```

Obr. 4.30 Zabezpečení cesty na přihlašovací komponentu, zdroj: vlastní

Na obrázku 4.30 si lze všimnout úryvku JavaScriptového kódu, který obsahuje objekt s vlastnostmi, popisujícími přístup ke komponentě *Login*, což je komponenta, která obsahuje přihlašovací formulář. Lze si zde všimnout vlastnosti *beforeEnter*, která obsahuje vnořenou funkci, která zajistí, že před vstupem na tuto cestu bude

zkontrolováno, zda je uživatel přihlášen. Pokud ano, tak mu vstup umožněn nebude a bude přesměrován na domovskou stránku webové aplikace. Podobně jsou ošetřeny všechny cesty webové aplikace, které zobrazují komponenty, co nejsou přístupné všem druhům uživatelů.

Na závěr byla klientská část webové aplikace nahrána na platformu Firebase hosting, což je platforma pro hostování webových stránek a aplikací na serverech Firebase. Tato platforma byla zvolena hlavně kvůli jednoduchosti nasazení aplikace a velmi dobré komunikace s ostatními službami, které Firebase poskytuje. Zde je aplikace připravena na závěrečné uživatelské testování, které bude popsáno v následující kapitole této diplomové práce.

4.7 Shrnutí kapitoly

V této kapitole diplomové práce byla popsána navržená architektura webové aplikace, která je určená ke správě konferencí. Byl zde popsán návrh struktury databáze a jednotlivých dokumentů, umístěných do příslušných kolekcí. Také zde byly detailně popsány jednotlivé komponenty webové aplikace a jejich činnost, která slouží k naplnění požadavků, uvedených ve třetí kapitole této diplomové práce. Nebyl opomenut také popis zabezpečení webové aplikace v její backendové i frontendové části.

5 Uživatelské testování webové aplikace a zhodnocení výsledků

V této poslední kapitole této diplomové práce budou shrnuty výsledky uživatelského testování webové aplikace za pomoci reálných dat a dále bude následovat celkové zhodnocení výsledků.

5.1 Uživatelské testování

Jelikož byl vývoj webové aplikace inspirován agilním přístupem, bylo uživatelské testování aplikace bylo prováděno na pravidelných schůzkám se zadavatelem. Při každé schůzce byly zadavateli představeny nově přidané funkcionality a provedlo se testování na testovacích datech. Výsledkem byla zpětná vazba, která sloužila k případným úpravám funkcionalit, dokud nebyl zadavatel spokojen. Zadavateli byl také v aplikaci zřízen uživatelský účet, pomocí kterého si používání aplikace mohl sám vyzkoušet.

Hlavním účelem tohoto průběžného testování bylo vyladit jednotlivé funkcionality a prvky uživatelského rozhraní tak, aby bylo rozhraní webové aplikace co nejvíce intuitivní a zadavatel byl spokojen s prováděním funkcionalit, které vedou k úspěšnému splnění všech požadavků na webovou aplikaci.

Po skončení vývoje webové aplikace, bylo vyzkoušeno vytváření konference, která obsahuje reálná data, dle dodané předlohy. Podobu vytvořené konference, která tato reálná data obsahuje si lze prohlédnout v příloze 1.

Pro další případné testování je webová aplikace nasazena na testovací server, kde si můžou případní budoucí uživatelé testovat již hotovou webovou aplikaci a navrhovat případné nové funkcionality. Novou funkcionalitou by mohla být například implementace reCAPTCHA do přihlašovací komponenty aplikace, či implementace přihlášení pomocí třetích stran jako je Facebook nebo Google. Také bylo navrženo rozšíření funkcionality tvorby recenzí, kdy by si administrátor mohl sám vytvořit několik podob formuláře pro tvorbu recenzí.

5.2 Zhodnocení výsledků

Jak již bylo v této práci uvedeno, vývoj a implementace této webové aplikace byl založen na požadavcích zadavatele, které byly určeny za pomoci pravidelných schůzek. V průběhu celého procesu vývoje byly jednotlivé kroky tohoto vývoje neustále konzultovány a jednotlivé funkcionality spolu s uživatelským rozhraním se za pomoci

těchto konzultací upravili tak, aby se co nejvíce přiblížili představám zadavatele. Uživatelské rozhraní je vytvořeno tak, aby bylo co nejvíce intuitivní a přístup k jednotlivým částem webové aplikace co nejjednodušší. Komponenty webové aplikace jsou specificky navrženy tak, aby dokázaly plnit uvedené požadavky.

Celá webová aplikace je vyvinuta pomocí nejmodernějších technologií určených pro tvorbu webových aplikací a byla vyvinuta způsobem, aby byla co nejjednodušeji rozšiřitelná a spravovatelná. Celý zdrojový kód webové aplikace je přístupný přes github a může být spravován pomocí verzovacího nástroje GIT.

6 Závěr

Hlavní cíl této práce bylo vytvoření webové aplikace pro Ekonomickou fakultu Vysoké školy báňské – Technické univerzity v Ostravě, která bude sloužit ke správě konferencí. Pomocí této webové aplikace lze tedy vytvářet a spravovat jednotlivé konference a jejich recenze snadnějším způsobem, než tomu bylo doposud.

Tato webová aplikace je vyvinuta na principech vrstvené architektury a lze ji rozdělit na frontendovou (klientskou) a backendovou (serverovou) část. Obě tyto části jsou vyvinuty za pomoci nejmodernějších přístupů a technologií. Návrh a vývoj této webové aplikace byl realizován tak, aby splnil veškeré stanovené požadavky a zároveň aby byla webová aplikace stále jednoduše rozšiřitelná a připravená k implementaci dalších případných funkcionalit.

Vytvořená webová aplikace byla prozatím nasazena na testovací doménu, kde proběhlo veškeré potřebné testování. Výsledkem tohoto testování bylo ověřeno, že je aplikace schopná pracovat s reálnými daty a splňuje všechny očekávané požadavky. Z tohoto důvodu lze považovat cíl této diplomové práce za splněný. Aplikaci je možné nasadit na jakýkoliv hosting s doménou, který má nainstalovaný NodeJS. Nyní je pro přístup k aplikaci využit Firebase hosting, který tyto vlastnosti splňuje, ale v době psaní této diplomové práce nebyla prozatím založena doména, přes kterou se bude k aplikaci přistupovat. Z tohoto účelu byl přístup k aplikaci prozatím ponechán na testovací doméně.

Při vypracovávání této diplomové práce jsem se nesetkal s vývojem aplikace na míru dle požadavků poprvé, jelikož se již nějakou dobu tímto oborem zabývám. Ovšem technologie využité k vývoji této webové aplikace pro mě byly premiérou, a proto tuto diplomovou práci beru jako obrovskou zkušenost do budoucna hlavně z důvodu získaných znalostí spojených s těmito technologiemi.

Seznam použité literatury

Odborná kniha

ATENCIO, Luis. *Functional Programming in JavaScript*. Shelter Island: Manning Publications Co, 2016. ISBN 9781617292828.

ATTARDI, Joe. *Modern CSS: Master the Key Concepts of CSS for Modern Web Development*. 1. Billerica: Apress, 2020. ISBN 978-1-4842-6293-1.

BANSAL, Arvind Kumar. *Introduction to Programming Languages*. Boca Raton: Crc Press, 2013. ISBN 978-1-4665-6514-2.

BASSETT, Lindsay. *Introduction to JavaScript Object Notation*. 1. Sebastopol: O'Reilly Media, 2015. ISBN 978-1-491-92948-3.

BEAULIEU, Alan. Learning SQL, 3rd Edition. California: O'Reilly Media, 2020. 380 p. ISBN 978-1-492-05761-1.

BISWAS, Nabendu. *Beginning React and Firebase: Create Four Beginner-Friendly Projects Using React and Firebase*. 1. Bhopal: APress Media, 2022. ISBN 978-1-4842-7811-6.

BOOCH, Grady, James RUMBAUGH and Ivar JACOBSON. *The unified modeling language user guide*. 2nd ed. Upper Saddle River: Addison-Wesley, 2005. ISBN 03-212-6797-4.

BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.

DJIRDEH, Hassan, Nate MURRAY and Ari LERNER. *Fullstack Vue 3: The Complete Guide to Vue.js.* San Francisco, California: Fullstack.io, 2021. ISBN 978-1987595291.

FLANAGAN, David. *JavaScript: The Definitive Guide*. 7. Sebastopol: O'Reilly Media, 2020. Všechny cesty k informacím. ISBN 978-1-491-95202-3.

HOFFMAN, Andrew. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Sebastopol: O`Reilly, 2020. 291 p. ISBN 978-1-492-05311–8.

HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. *Big Data a NoSQL databáze*. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

INGENO, Joseph. *Software Architect's Handbook*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-406-0.

MADDEN, Neil. API Security in Action. New York: Manning publications, 2021. 576 p. ISBN 978-16-1729-602-4.

MEYER, Jeanine. *The Essential Guide to HTML5: Using Games to Learn HTML5 and JavaScript*. 3. New York: Apress, 2022. ISBN 978-1-4842-8721-7.

MILES, Russ a Kim HAMILTON. *Learning UML 2.0*. Sebastopol: O´Reilly, 2006. ISBN 978-059-6009-823.

MITRA, Ronnie and Irakli NADAREISHVILI, 2020. *Microservices: Up and Running*. Sebastopol: O'Reilly Media. ISBN 978-1-492-07545-5

NEWMAN, Sam. *Monolith to Microservices*. Sebastopol: O'Reilly Media, 2019. ISBN 978-1-492-04784-1.

RIBEIRO, Heitor Ramon. *Vue.js 3 Cookbook*. 1. Birmingham: Packt Publishing, 2020. ISBN 978-1-83882-622-2.

RICHARDS, Mark and Neil FORD. *Fundamentals of Software Architecture*. Sebastopol: O'Reilly Media, 2020. ISBN 978-1-492-04345-4.

SCHILDT, Herbert. *Java: The Complete Reference, Twelfth Edition*. 12. New York: McGraw-Hill, 2021. ISBN 978-1-26-046342-2.

SCOTT, Emmit. *SPA design and architecture: understanding single-page web applications*. Shelter Island: Manning, 2016. ISBN 978-1-61729-243-9.

SHAVIN, Maya. *Learning Vue*. 1. Sebastopol: O'Reilly Media, 2023. ISBN 978-1-492-09876-8.

SOMMERVILLE, Ian. *Engineering Software Products: An Introduction to Modern Software Engineering*. New Jersey: Pearson, 2020. ISBN 978-1292376349.

SVEKIS, Laurence Lars, Maaike van PUTTEN a Rob PERCIVAL. *JavaScript from Beginner to Professional: Learn JavaScript quickly by building fun, interactive, and dynamic web apps, games, and pages*. Birmingham: Packt Publishing, 2021. ISBN 978-1-80056-252-3.

UMANATH, Narayan S. and Richard W. SCAMELL. *Data Modeling and Database Design*. 2nd ed. Boston: Cengage Learning, 2014. ISBN 9781305177543.

VYMĚTAL, Dominik. *Informační systémy v podnicích: teorie a praxe projektování*. Praha: Grada, 2009. Průvodce (Grada). ISBN 978-802-4730-462.

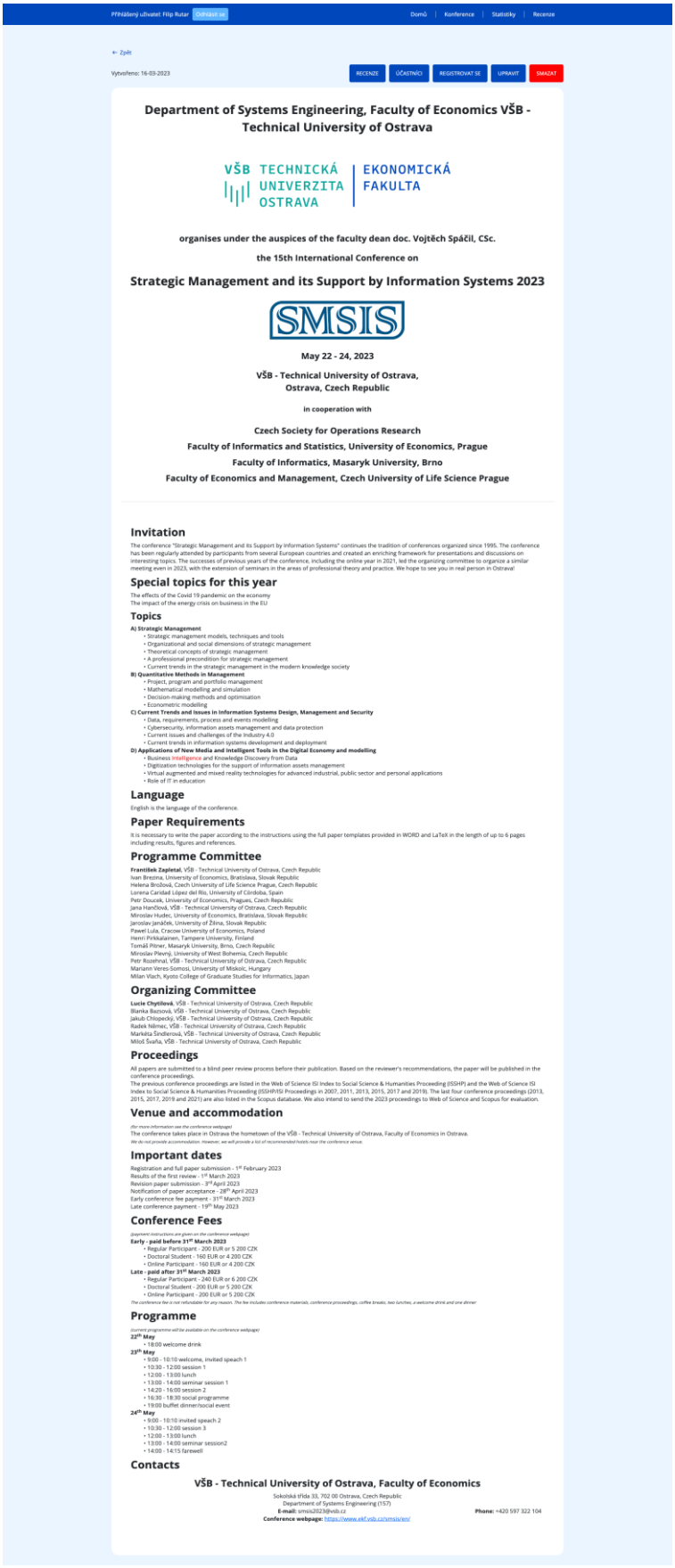
ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie.* 2. vydání. Brno: Computer Press, 2021. ISBN 978-80-251-5026-9.

Seznam zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
NoSQL	Not Only SQL
OOP	Object Oriented Programming
PDF	Portable Document Format
REST	Representational State Transfer
RTM	Requirements Tracebility Matrix
SDK	Software Development Kit
SQL	Structured Query Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Seznam příloh

Příloha 1



Přihlášený uživatel: Filip Rutar Odhlásit se

Domů | Konference | Statistiky | Recenze

← Zpět

Vytvořeno: 16-03-2023

RECENZE | ÚČASTNÍCI | REGISTROVAT SE | UPRAVIT | SMAZAT

Department of Systems Engineering, Faculty of Economics VŠB - Technical University of Ostrava

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | EKONOMICKÁ FAKULTA

organises under the auspices of the faculty dean doc. Vojtěch Spáčil, CSc.

the 15th International Conference on

Strategic Management and its Support by Information Systems 2023

SMSIS

May 22 - 24, 2023

VŠB - Technical University of Ostrava,
Ostrava, Czech Republic

in cooperation with

Czech Society for Operations Research
Faculty of Informatics and Statistics, University of Economics, Prague
Faculty of Informatics, Masaryk University, Brno
Faculty of Economics and Management, Czech University of Life Science Prague

Invitation

The conference "Strategic Management and its Support by Information Systems" continues the tradition of conferences organized since 1995. The conference has been regularly attended by participants from several European countries and created an enriching framework for presentations and discussions on interesting topics. The successes of previous years of the conference, including the online year in 2021, led the organizing committee to organize a similar meeting even in 2023, with the extension of seminars in the areas of professional theory and practice. We hope to see you in real person in Ostrava!

Special topics for this year

The effects of the Covid 19 pandemic on the economy
The impact of the energy crisis on business in the EU

Topics

A) Strategic Management
- Strategic management models, techniques and tools
- Organizational and social dimensions of strategic management
- Theoretical concepts of strategic management
- A professional precondition for strategic management
- Current trends in the strategic management in the modern knowledge society

B) Quantitative Methods in Management
- Project, program and portfolio management
- Mathematical modelling and simulation
- Decision-making methods and optimisation
- Econometric modelling

C) Current Trends and Issues in Information Systems Design, Management and Security
- Data, requirements, process and events modelling
- Cybersecurity, information assets management and data protection
- Current issues and challenges of the Industry 4.0
- Current trends in information systems development and deployment

D) Applications of New Media and Intelligent Tools in the Digital Economy and modelling
- Business Intelligence and Knowledge Discovery from Data
- Digitization technologies for the support of information assets management
- Virtual augmented and mixed reality technologies for advanced industrial, public sector and personal applications
- Role of IT in education

Language

English is the language of the conference.

Paper Requirements

It is necessary to write the paper according to the instructions using the full paper templates provided in WORD and LaTeX in the length of up to 6 pages including results, figures and references.

Programme Committee

František Zapletal, VŠB - Technical University of Ostrava, Czech Republic
Ivan Brezina, University of Economics, Bratislava, Slovak Republic
Helena Brožová, Czech University of Life Science Prague, Czech Republic
Lorena Caridad López del Río, University of Córdoba, Spain
Petr Doucek, University of Economics, Prague, Czech Republic
Jana Hančlová, VŠB - Technical University of Ostrava, Czech Republic
Miroslav Hudec, University of Economics, Bratislava, Slovak Republic
Jaroslav Janáček, University of Žilina, Slovak Republic
Paweł Lula, Cracow University of Economics, Poland
Henri Pirkkalainen, Tampere University, Finland
Tomáš Pitner, Masaryk University, Brno, Czech Republic
Miroslav Plevný, University of West Bohemia, Czech Republic
Petr Rozehnal, VŠB - Technical University of Ostrava, Czech Republic
Mariann Veres-Somosi, University of Miskolc, Hungary
Milan Vlach, Kyoto College of Graduate Studies for Informatics, Japan

Organizing Committee

Lucie Chytilová, VŠB - Technical University of Ostrava, Czech Republic
Blanka Bazsová, VŠB - Technical University of Ostrava, Czech Republic
Jakub Chlapecký, VŠB - Technical University of Ostrava, Czech Republic
Radek Němec, VŠB - Technical University of Ostrava, Czech Republic
Markéta Šindlerová, VŠB - Technical University of Ostrava, Czech Republic
Miloš Šuařa, VŠB - Technical University of Ostrava, Czech Republic

Proceedings

All papers are submitted to a blind peer review process before their publication. Based on the reviewer's recommendations, the paper will be published in the conference proceedings.
The previous conference proceedings are listed in the Web of Science ISI Index to Social Science & Humanities Proceeding (ISSHP) and the Web of Science ISI Index to Social Science & Humanities Proceeding (ISSHPYR) Proceedings in 2007, 2011, 2013, 2015, 2017 and 2019). The last four conference proceedings (2013, 2015, 2017, 2019 and 2021) are also listed in the Scopus database. We also intend to send the 2023 proceedings to Web of Science and Scopus for evaluation.

Venue and accommodation

(for more information see the conference webpage)
The conference takes place in Ostrava the hometown of the VŠB - Technical University of Ostrava, Faculty of Economics in Ostrava.
We do not provide accommodation. However, we will provide a list of recommended hotels near the conference venue.

Important dates

Registration and full paper submission - 1st February 2023
Results of the first review - 1st March 2023
Revision paper submission - 3rd April 2023
Notification of paper acceptance - 28th April 2023
Early conference fee payment - 31st March 2023
Late conference payment - 19th May 2023

Conference Fees

(payment instructions are given on the conference webpage)

Early - paid before 31st March 2023
- Regular Participant - 200 EUR or 5 200 CZK
- Doctoral Student - 160 EUR or 4 200 CZK
- Online Participant - 160 EUR or 4 200 CZK

Late - paid after 31st March 2023
- Regular Participant - 240 EUR or 6 200 CZK
- Doctoral Student - 200 EUR or 5 200 CZK
- Online Participant - 200 EUR or 5 200 CZK

The conference fee is not refundable for any reason. The fee includes conference materials, conference proceedings, coffee breaks, two lunches, a welcome drink and one dinner

Programme

(current programme will be available on the conference webpage)

23th May
- 18:00 welcome drink

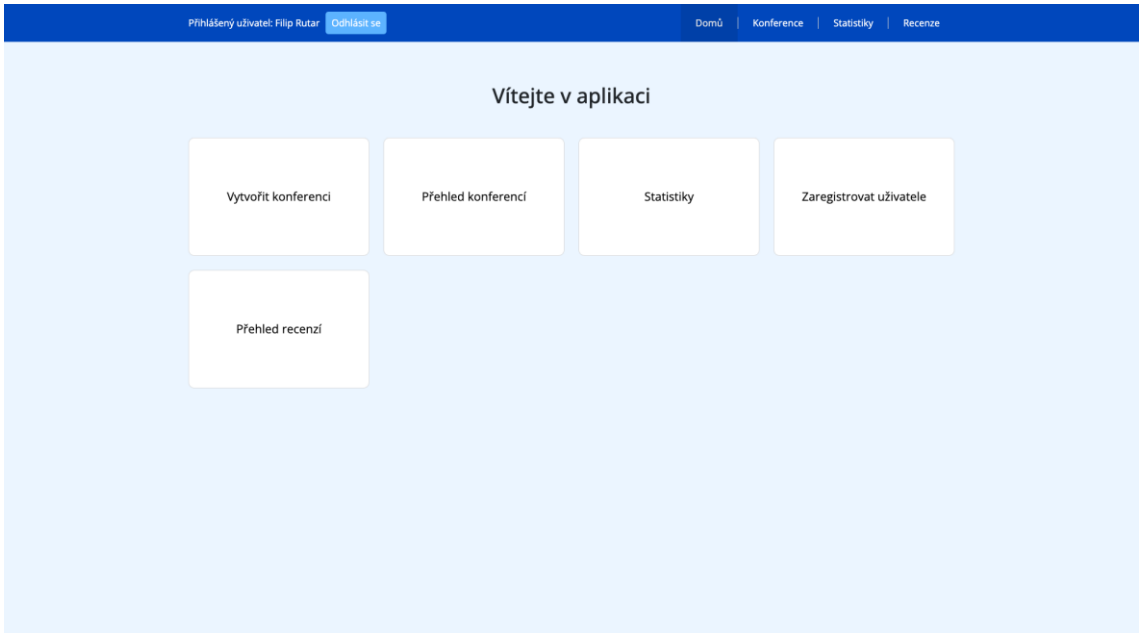
23th May
- 9:00 - 10:10 welcome, invited speach 1
- 10:30 - 12:00 session 1
- 12:00 - 13:00 lunch
- 13:00 - 14:00 seminar session 1
- 14:20 - 16:00 session 2
- 16:30 - 18:30 social programme
- 19:00 buffet dinner/social event

24th May
- 9:00 - 10:10 invited speach 2
- 10:30 - 12:00 session 3
- 12:00 - 13:00 lunch
- 13:00 - 14:00 seminar session2
- 14:00 - 14:15 farewell

Contacts

VŠB - Technical University of Ostrava, Faculty of Economics
Sokolská třída 33, 702 00 Ostrava, Czech Republic
Department of Systems Engineering (157)
E-mail: smsis2023@vsb.cz
Phone: +420 597 322 104
Conference webpage: https://www.ekf.vsb.cz/smsis/en/

Příloha 2


Přihlášený uživatel: Filip Rutar
Odhlásit se
Domů
Konference
Statistiky
Recenze
Vítejte v aplikaci
Vytvořit konferenci
Přehled konferencí
Statistiky
Zaregistrovat uživatele
Přehled recenzí

Příloha 3

```vue
<script setup>
import { reactive } from 'vue'
import { useAuthStore } from '@/stores/storeAuth'
import { Field, Form, ErrorMessage } from 'vee-validate';
import * as yup from 'yup';

const storeAuth = useAuthStore()

const credentials = reactive({
  email: '',
  password: '',
  name: '',
  rights: 'reviewer'
})
const schema = yup.object().shape({
  username: yup.string().required('Toto pole je povinné'),
  email: yup.string().required('Toto pole je povinné').email('Zadejte prosím validní email'),
  password: yup.string().required('Toto pole je povinné').min(6,'Heslo musí mít minimálně 6 znaků'),
  passwordConfirmation: yup
    .string()
    .required('Toto pole je povinné')
    .oneOf([yup.ref('password')], 'Hesla se neshodují'),
})
const onSubmit = () => {
  if (!credentials.email || !credentials.password || !credentials.name) {
    alert('Vyplňte prosím všechny pole.')
  }
  else {
    storeAuth.registerUser(credentials)
  }
}
</script>
<template>
  <section class="s-register-form">
    <div class="container">
      <div class="row">
        <div class="s-register-form__wrapper">
          <h1 class="s-register-form__title">Zaregistrovat uživatele</h1>
          <Form @submit="onSubmit" :validation-schema="schema">
            <div class="s-register-form__row">
              <Field type="text" class="text-input" v-model="credentials.name" name="username" id="username" placeholder="Vyplňte Jméno"/>
              <ErrorMessage name="username" class="alert alert-danger"/>
            </div>
            <div class="s-register-form__row">
              <Field type="email" class="text-input" v-model="credentials.email" name="email" id="email" placeholder="Vyplňte email"/>
              <ErrorMessage name="email" class="alert alert-danger"/>
            </div>
            <div class="s-register-form__row">
              <Field type="password" class="text-input" v-model="credentials.password" name="password" id="password" placeholder="Vyplňte heslo"/>
              <ErrorMessage name="password" class="alert alert-danger"/>
            </div>
            <div class="s-register-form__row">
              <Field type="password" class="text-input" name="passwordConfirmation" id="passwordConfirmation" placeholder="Potvrďte heslo"/>
              <ErrorMessage name="passwordConfirmation" class="alert alert-danger"/>
            </div>
            <div class="s-register-form__row justify-content-start">
              <fieldset>
                <div class="d-flex mb-2">
                  <input type="radio" id="admin" name="rights" value="admin" v-model="credentials.rights">
                  <label for="admin">Zaregistrovat jako administrátora</label>
                </div>
                <div class="d-flex">
                  <input type="radio" id="reviewer" name="rights" value="reviewer" v-model="credentials.rights">
                  <label for="reviewer">Zaregistrovat jako recenzenta</label>
                </div>
              </fieldset>
            </div>
            <div class="s-register-form__row">
              <button>Zaregistrovat</button>
            </div>
          </Form>
        </div>
      </div>
    </div>
  </section>
</template>
```