VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ PLATFORMA PRO ONLINE MARKETING

WEB PLATFORM FOR ONLINE MARKETING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ KŘIVÁNEK**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

152997

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Křivánek Tomáš, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Webová platforma pro online marketing**
Kategorie:	Elektronický obchod
Akademický rok:	2023/24

Zadání:

1. Seznamte se se současnými technologiemi pro tvorbu webových aplikací s podporou multimediálního obsahu.
2. Analyzujte požadavky na aplikaci propojující tvůrce uživatelsky generovaného obsahu a manažery online marketingu. Prozkoumejte rovněž existující řešení v této oblasti.
3. Na základě provedené analýzy navrhněte architekturu aplikace a odpovídající uživatelské rozhraní.
4. Po konzultaci s vedoucím implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte systematické testování vytvořené aplikace.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Krug, S.: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Kleppmann, M.: Designing Data-Intensive Applications, ISBN: 9781449373320
- dále dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

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

Vedoucí práce:	**Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Práce se zaměřuje na celý proces vývoje webové aplikace pro podporu propojení tvůrců uživatelsky generovaného obsahu s manažery online marketingu firem. Toto marketingové odvětví je v České republice velice nové a tak ještě neexistují platformy, které by navázání kontaktu mezi zmíněnými účastníky zjednodušovaly. Proces vývoje začíná pečlivým sběrem a analýzou požadavků potenciálních uživatelů tohoto nového systému. Na základě sesbíraných požadavků je následně navržena architektura systému s využitím moderních technologií. Pro vývoj systému byl zvolen backend ve formě REST API, které je naprogramováno pomocí Python frameworku FastAPI. Dále je využit framework Vue na klientské straně. Pro uchování dat jsou použity databáze MySQL, MongoDB a cloudové úložiště. Po výběru konkrétních technologií je dále popsána implementace celého systému a také jak bylo prováděno testování celé aplikace.

Abstract

The work focuses on the entire process of developing web application to facilitate the connection between creators of user-generated content and online marketing managers of companies. This marketing industry is relatively new in the Czech Republic, and there are currently no platforms that simplify the interaction between these mentioned participants. The development process begins with a careful collection and analysis of the requirements of potential users of this new system. Based on the gathered requirements, the system architecture is then designed using modern technologies. For the system development, a backend in the form of a REST API is chosen, programmed using the Python FastAPI framework. Additionally, the Vue framework will be utilized on the client side. Data storage will involve MySQL and MongoDB databases, along with a cloud storage solution. After the selection of specific technologies, the implementation of the system is described, as well as how the testing of the whole application was carried out.

Klíčová slova

UGC, online, marketing, web, vývoj, aplikace, mysql, mongodb, fastapi, python, vue, javascript, fastapi, cloudové úložiště

Keywords

UGC, online, marketing, web, development, application, mysql, mongodb, fastapi, python, vue, javascript, cloud storage

Citace

KŘIVÁNEK, Tomáš. *Webová platforma pro online marketing.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Webová platforma pro online marketing

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Křivánek
17. května 2024

Poděkování

Chtěl bych poděkovat svému vedoucímu doc. Ing. Radku Burgetovi Ph.D. za veškerou pomoc při psaní diplomové práce, za veškeré připomínky a nápady pro zlepšení jak textu diplomové práce, tak i samotné aplikace. Dále bych chtěl poděkovat své snoubence, která mi poskytla prvotní nápad pro vytvoření této aplikace a jelikož se v tomto oboru pohybuje, poskytla mi kontakt i na další potenciální uživatele pro testování. Dále také děkují svým rodičům, kteří mi během celého studia poskytovali velkou podporu.

Obsah

Kapitola 1

Úvod

Tato práce si klade za cíl vytvořit kompletní webovou platformu, která bude sloužit pro propojení tvůrců uživatelsky generovaného obsahu (UGC) s firmami, které tento obsah potřebují. Marketingoví manažeři na této platformě budou zveřejňovat nabídky spoluprací, kde popíší jaký obsah by si s jejich produktem představovali. Tvůrci jim prostřednictvím systému budou zasílat žádosti. Z těchto žádostí si manažeři vyberou tvůrce, se kterým pak spolupráci v systému povedou.

V druhé kapitole se věnuji objasnění celého tématu. Vysvětluji zde, co vlastně UGC obsah je a jak vzniká. Jak se historicky z tohoto obsahu stala zajímavá možnost propagace firem na sociálních sítí a kolik si jeho tvůrci mohou touto tvorbou vydělat v zahraničí, protože u nás tento druh obsahu je teprve na vzestupu.

V další kapitole se věnuji sběru požadavků. Uvádím zde jak jsem požadavky od potenciálních uživatelů sbíral a také je zde detailně popsána celá platforma. Detailněji zde popisuji průběh celé spolupráce a ukazuji na něm, jak jsem z požadavků získal přehled o potřebných technologiích pro vývoj systému. Také je zde uvedena analýza podobných zahraničních řešení, které jsem mezi sebou porovnal a nakonec i uvedl rozdíl oproti nové aplikaci.

Ve čtvrté kapitole je navržen celý systém. Jsou zde uvedeny konkrétní technologie, které jsou při implementaci využity. Dále jsou zde popsány diagramy případů užití, které ukazují kteří uživatelé budou moci provádět v systému jaké akce. Je zde také datový model v podobě ER diagramu a jelikož aplikace nebude využívat pouze relační databázi, jsou zde také uvedeny formáty objektů, které budou uloženy v NoSQL databázi. V závěru této kapitoly je demonstrován postup při návrhu uživatelského rozhraní aplikace na konkrétní stránce. Zbytek snímků uživatelského rozhraní je k nahlédnutí v přílohách práce.

V páté kapitole detailně popisuji technologie, které byly vybrány pro implementaci a zdůvodňuji, proč byly zvoleny. Jsou zde porovnány dva nejvyužívanější frontendové frameworky Angular a Vue a knihovna React. Po frontendových technologiích jsou zde také popsány backendové technologie. Jelikož je zde větší rozmanitost, jsou zde prvně porovnány jazyky, ve kterých se nejčastěji API implementují. Následně je z těchto jazyků vybrán jeden a v něm jsou porovnány nejpoužívanější frameworky. Z nich je nakonec jeden vybrán pro implementaci.

V šesté kapitole je popsán samotný vývoj aplikace. Zvláštní pozornost je zde věnována organizaci souborů na cloudovém uložišti a využité principy při práci s ním. Následně specifikuji různé formy registrace a přihlášení do systému, které platforma nabízí. Detailnější pozornost věnuji i částem aplikace, které využívají komunikaci přes protokol WebSocket.

Kromě předchozích je zde stručně popsáno i fungování různých částí systému a využití frontendového frameworku Vue.

V sedmé kapitole se věnuji popisu testování implementované aplikace, které bylo rozděleno do dvou fází. Testování API probíhalo pomocí jednotkových testů a testování klientské strany řešení bylo založeno na testování použitelnosti webového aplikace, kde byli využity různé testovací scénáře.

V předposlední kapitole jsou navrženy možná rozšíření webové aplikace a obohacení o aplikaci mobilní. V závěru shrnuji dosažené výsledky celé práce.

Kapitola 2

Definování problému

Jedním ze současných problémů online marketingu na českém trhu je, že neexistuje platforma pro navázání kontaktu mezi firmou a potenciálním tvůrcem UGC. Pojem UGC je detailně popsán v následující kapitole.

2.1 User-generated content a producer-generated-content

Přesnou a jednoznačnou definici UGC (user–generated content) není tak jednoduché formulovat, protože nikde nebyla ustanovena. Podle [63] je user–generated content obsah publikovaný uživateli na různých online platformách. To je velmi nekonkrétní definice. Dále např. [40] uvádí, že UGC je obsah dobrovolně vytvořen jednotlivcem nebo konsorciem a distribuován prostřednictvím online platformy. Většina definic UGC zdůrazňuje fakt, že se jedná o obsah distribuovaný na online platformách [44]. Podle [48] by každé UGC mělo splňovat následující vlastnosti:

1. Musí se jednat o obsah publikovaný na internetu.
2. Obsah musí odrážet nějakou kreativní snahu tvůrce.
3. Obsah je vytvořen bez použití profesionálních praktik a rutin.

Pod samotným pojmem UGC se tedy skrývá jakýkoliv text, audio, obrázek, video a jiná multimédia, která jsou distribuována např. na blozích, podcastových repositářích, Twitteru, Youtube, Instagramu nebo také na stránkách online novin [40].

UGC [6] bývá také někdy nazýváno jako eWOM[1]. Electronic-word-of-mouth narozdíl od WOM se odlišuje tím, že místo ústní komunikace mezi spotřebiteli je využíváno nástrojů na online sociálních sítí. Mezi tyto nástroje můžeme zařadit online chat mezi spotřebiteli, Facebook zeď s příspěvky, instagramový feed nebo také zeď příspěvků na Twitteru.

Dalším způsobem propagace firem je PGC (professionally-generated-content) [6]. Rozdíl mezi PGC a UGC spočívá ve způsobu, jakým je obsah vytvořen a kdo za ním stojí. PGC se týká obsahu, který je vytvářen profesionály nebo zkušenými tvůrci, kteří mají odborné znalosti a dovednosti v dané oblasti. Tento obsah je často produkován za účelem komerčního využití a může zahrnovat například kvalitní videa, články, fotografie nebo design. Na druhou stranu UGC se týká obsahu, který je vytvářen samotnými uživateli, kteří nejsou nutně

[1] Samotné Word-of-mouth je každá ústní komunikace v rámci spotřebitelského chování. Takový marketing se odkazuje na ústní sdělení informací o produktu mezi přáteli, rodinou a kolegy [79].

profesionály. Uživatelé generují tento obsah dobrovolně a často ho sdílejí na online platformách. UGC může zahrnovat různé formy obsahu, jako jsou recenze, komentáře, amatérská videa, fotky a další projevy kreativity.

Významným rozdílem je tedy způsob tvorby a zkušenosti tvůrců stojících za obsahem. PGC je tvořen odborníky a má často profesionální úroveň, zatímco UGC zahrnuje rozmanité projevy a přístup široké veřejnosti, což může vést k pestřejší a autentičtější škále obsahu.

Výsledkem tedy je, že zákazníci si mnohem více věří navzájem než profesionálně vytvořeným reklamám. V roce 2020 Retail Reputation Report [45] zjistil, že až 92 % zákazníků více věří svým vrstevníkům (v tomto smyslu tedy vrstevník znamená podobný zákazník) než tradičnímu pojetí propagování firmy. Zákazníci chtějí vidět ostatní používat či ukazovat produkt nebo službu a podle toho se často rozhodují, jestli si ji sami obstarají.

2.2 Využití uživatelsky vytvořeného obsahu pro účely propagace firmy

Firmy začali tento obsah využívat pro svou propagaci z již dříve zmíněných důvodů. Uživatelé mnohem více věří autentické recenzi od jiných uživatelů než od profesionálních herců [44]. Za tímto účelem v zahraničí vzniklo několik webových nástrojů, které propojují tvůrce s firmami. Na sociálních sítích zahraničních firem se mnohem častěji objevují různé formy tohoto obsahu – recenze, rozbalování zaslaných produktů, různé záběry přímo od uživatelů a firmy tímto obsahem vyplňují své účty na sociálních sítích. User-generated-content totiž potenciálním zákazníkům mnohem více přiblíží jak daný produkt či služba zapadne do jejich každodenního života a dokáže tak více propojit potenciální spotřebitele s poskytovatelem služby či produktu [28].

User-generated-content se stal nedílnou součástí každodenní zkušenosti s online prostorem. Od sociálních sítí po stránky specializované na recenze, se změnil způsob, jakým zákazníci vyhledávají, nachází nové produkty, služby či nápady [43].

Pro marketingové zástupce firem by bylo velmi složité všechen tento obsah shromažďovat samostatně, proto vznikly specializované webové nástroje, které jim v tomto pomáhají. Mezi nejvyužívanější zahraniční nástroje patří stránky Grin, Creator.co, Upfluence, Intellifluence atd [28].

Jak již bylo zmíněno, tak historicky se UGC bere jako jakýkoliv obsah sdílený na internetu. V této době již ale firmy začali využívat právě již zmíněné autentičnosti obsahu a tvůrcům, kteří tento obsah vytváří, dobrovolně zasílají produkty, se kterými tito tvůrci (anglicky UGC creator) vytvářejí nový obsah. Tito tvůrci nejen, že dostávají produkty zdarma, ale také velmi často dostávají i za jejich poskytnutý materiál zaplaceno [11]. Tyto částky se od každého tvůrce liší v závislosti na kvalitě poskytovaného obsahu. Například zde [68] autorka blogu uvádí, že během prvního měsíce vytváření obsahu si přivydělala $540. Dle [11] se ale částky za tvoření tohoto druhu obsahu mohou vyšplhat až k měsíčním výplatám v řádech tisíců dolarů. Díky tomu, že jsou v zahraničí tito tvůrci velmi dobře placeni, mohou se tomuto odvětví zabývat na plný úvazek.

I s tímto zahraniční webové aplikace počítají a zprostředkovávají možnost, jak si i začínající UGC tvůrce může přivydělat. Firmy na těchto stránkách se poptávají po tvůrcích, kteří by pro ně mohli obsah generovat a jakmile s nimi naváží kontakt, zašlou jim jejich produkty nebo se domluví na způsobu jejich spolupráce. Cílem je tedy od UGC tvůrce získat materiál, který působí velmi autenticky a věrohodně. UGC tvůrce je zde tedy staven do role, jakoby šlo o opravdového zákazníka [11].

Kapitola 3

Analýza požadavků a dosavadních řešení

Podle [51] je několik principů, které jsou využity v každém obecném frameworku, který se zabývá analýzou, návrhem a vývojem softwarových produktů.

Prvním z nich jsou **principy komunikace**. Předtím než vůbec nějaké požadavky od zákazníka mohou být analyzovány, modelovány a specifikovány musí být sesbírány pomocí nějaké komunikační aktivity. Efektivní komunikace podle [51] patří k nejnáročnějším činnostem, se kterými se softwarový inženýr musí potýkat. V této souvislosti uvádí 10 nejlepších principů, které vedou ke zlepšení komunikace mezi zákazníkem a softwarovým inženýrem.

1. Naslouchat – softwarový inženýr se musí skutečně soustředit na to co mu zákazník říká a doptávat se pokaždé, kdy něčemu jen trochu nerozumí, ale měl by se vyvarovat častému přerušování. Nikdy by neměl působit ať už ve slovech nebo v činnech nesouhlasně či nechápavě: např. vrtění hlavou.

2. Příprava před komunikací – softwarový inženýr by měl věnovat čas porozumění problému před setkáním s ostatními. Pokud je to možné měl by si i udělat jistou rešerši, aby rozuměl žargonu v obchodní oblasti. Také by si měl připravit program nebo postup při jednání.

3. Vůdce komunikace – každá komunikace by měla mít nějakého vůdce, facilitátora, aby se konverzace posouvala produktivním směrem.

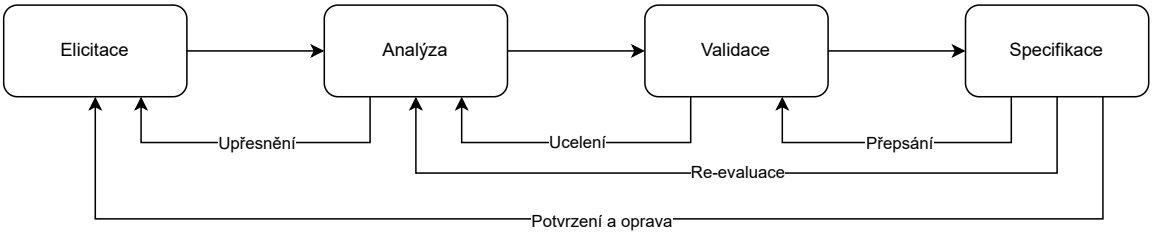
4. Komunikace z očí do očí je nejlepší, ale musí být doplněna i dalšími formami – kreslením, psaním atd.

5. Dělat si poznámky a dokumentovat rozhodnutí – během konverzace by měl někdo zaznamenávat důležitá rozhodnutí a poznatky ze schůzky.

6. Usilovat o spolupráci – každá malá spolupráce vede k budování důvěry a vytváří tak společný cíl.

7. Zůstat soustředěný, rozkouskovaní komunikace – čím více lidí se konverzace účastní, tím je větší tendence měnit téma a neposouvat se dopředu. Vedoucí konverzace by měl téma opustit až v momentě, kdy je toto téma dořešeno, ale zároveň musí být dodržen devátý bod.

8. Jestli je něco nejasné, nakreslit to – obrázek dokáže někdy nahradit tisíce slov.



Obrázek 3.1: Fáze analýzy a specifikace požadavků podle [78].

9. Pokud je na něčem dohodnuto posuňte se dál. Pokud se na něčem nemůžete shodnout, posuňte se dál. Pokud je nějaká funkcionalita nejasná a nemůže být teď vyřešena, posuňte se dál.

10. Vyjednávání není soutěž ani hra. Funguje nejlépe, když obě strany vyhrají.

Analýzu požadavků lze formulovat ve 4 vzájemně propojených fázích [78]. Tyto fáze se nazývají elicitace, analýza, specifikace a validace. Navazují na sebe, ale nelze je provézt v jednom lineárním sledu (Obr. 3.1). Elicitace zahrnuje všechny aktivity, které vedou ke shromáždění požadavků jako jsou – rozhovory, workshopy, analýza dokumentů, prototypování atd [78]. Analýza zahrnuje detailnější pochopení každého požadavku, které byly sesbírány ve fázi elicitace. Zde by se také mělo rozhodnout o jednotlivých prioritách, nebo-li jak jednotlivé funkcionality budou po sobě implementovány. Specifikace se zabývá sepsáním a uložením získaných požadavků v perzistentním a dobře organizovaném stylu. V této fázi jsou sesbírané potřeby uživatelů sepsány do dokumentů a diagramů, které se následně budou porovnávat a validovat. Cílem posledním fáze, validace, je potvrzení sepsaných požadavků, které poté pomohou vývojářům vyvinout produkt, po kterém zákazník touží.

Právě v předem zmíněných fázích probíhalo sbírání požadavků na systém. Požadavky byly sesbírány od reprezentativních uživatelů formou rozhovorů – jak online, tak i při osobním setkání. Během setkání byl i kladen důraz na všech 10 principů, které jsou zmíněny dříve. Setkání probíhala formou dialogu, kdy jsem se dotazoval na předem připravené otázky a nechal jsem dotazované co nejvíce odpověď rozvézt. Kromě zapisování odpovědi byl pořizován z rozhovorů i záznam, abych se k požadavkům a k jejich formulaci mohl kdykoliv vrátit. Během prvních setkání byl kladen důraz na sesbírání, co nejvíce požadavků na systém a během těch dalších byly sesbírané požadavky kontrolovány a dospecifikovány. Z těchto rozhovorů jsem následně sepsal detailnější popis systému.

3.1 Popis systému

Platforma by měla usnadnit domluvu a spolupráci mezi tvůrci uživatelsky generovaného obsahu a firmami, které tyto tvůrce hledají. Tito tvůrci a firmy se teď musejí kontaktovat na webových službách, které na tento typ spoluprací nejsou stavěné – např. přes sociální sítě. Navíc na sociálních sítích je omezení, co se týče vyhledávání nových kvalitních tvůrců. Pokud je nějaký tvůrce již známější je sociálními sítěmi při vyhledávání ukazován na popředí, a tak je pro firmy obtížné najít tvůrce nové.

Tvůrci budou po úspěšné registraci přesměrování na jejich profil, který si musí nějakým způsobem upravit. Profil tvůrce by mělo působit, jako profil na sociálních sítích s tím, že zde bude mít prostor pro ukázání své práce. Měl by mít možnost na svůj profil přidat 4 fotografie a 4 videa. Toto omezení plyne z toho, že chceme co nejvíce usnadnit výběr

manažerům. Aby se nemuseli rozhodovat na základě desítek videí, tak každý tvůrce bude mít velice omezené množství materiálu, který bude ostatním manažerům zpřístupňovat. Zároveň by na portfoliu měl mít tvůrce možnost sám sebe nějak textově popsat. Krom portfolia budou mít manažeři také možnost na základě žádosti o spolupráci přímo tvůrci psát zprávy prostřednictvím chatu a zde se budou moci doptat na cokoliv.

Firmy si budou moci založit svůj účet pomocí e-mailu a jejich identifikačního čísla IČO. Při založení účtu firmy by měla osoba zakládající tento účet zároveň poskytnout údaje o sobě a tím se stane hlavním manažerem, který bude moci spravovat údaje o firmě. V budoucích verzích systému bude hlavní manažer schopen přidávat nové manažery k firmě, kteří budou mít podobná práva jako hlavní manažer, ale s tím omezením, že nebudou moci upravovat informace o firmě.

Hlavní částí budou nabízené jednorázové spolupráce, které budou dvojího typu: barterové a placené. Barterové spolupráce jsou takové, kde tvůrce za natočené materiály (videa/fotografie/audio atd.) dostane pouze produkt. Za placené spolupráce dostávají tvůrci i peněžní ohodnocení. Manažer bude schopný vytvořit nabídku spolupráce. Tato nabídka bude obsahovat strukturovaný text s možností různých úrovní nadpisu, možnost tvorby odrážek a také podtržení či zvýraznění textu. Měla by zde být i možnost vložit fotografie či video propagovaného produktu. Dále budou muset zvolit kategorii do, které tato nabídka spadá (jednu či více) a zvolit typ spolupráce (barter/placená). V této chvíli jsou známé tyto kategorie: beauty, jídlo, lifestyle, technologie, unboxing, vzdělávací, zábava a ostatní. Kategorií je tedy prozatím 8, ale je možné, že se tyto kategorie budou přidávat či odebírat.

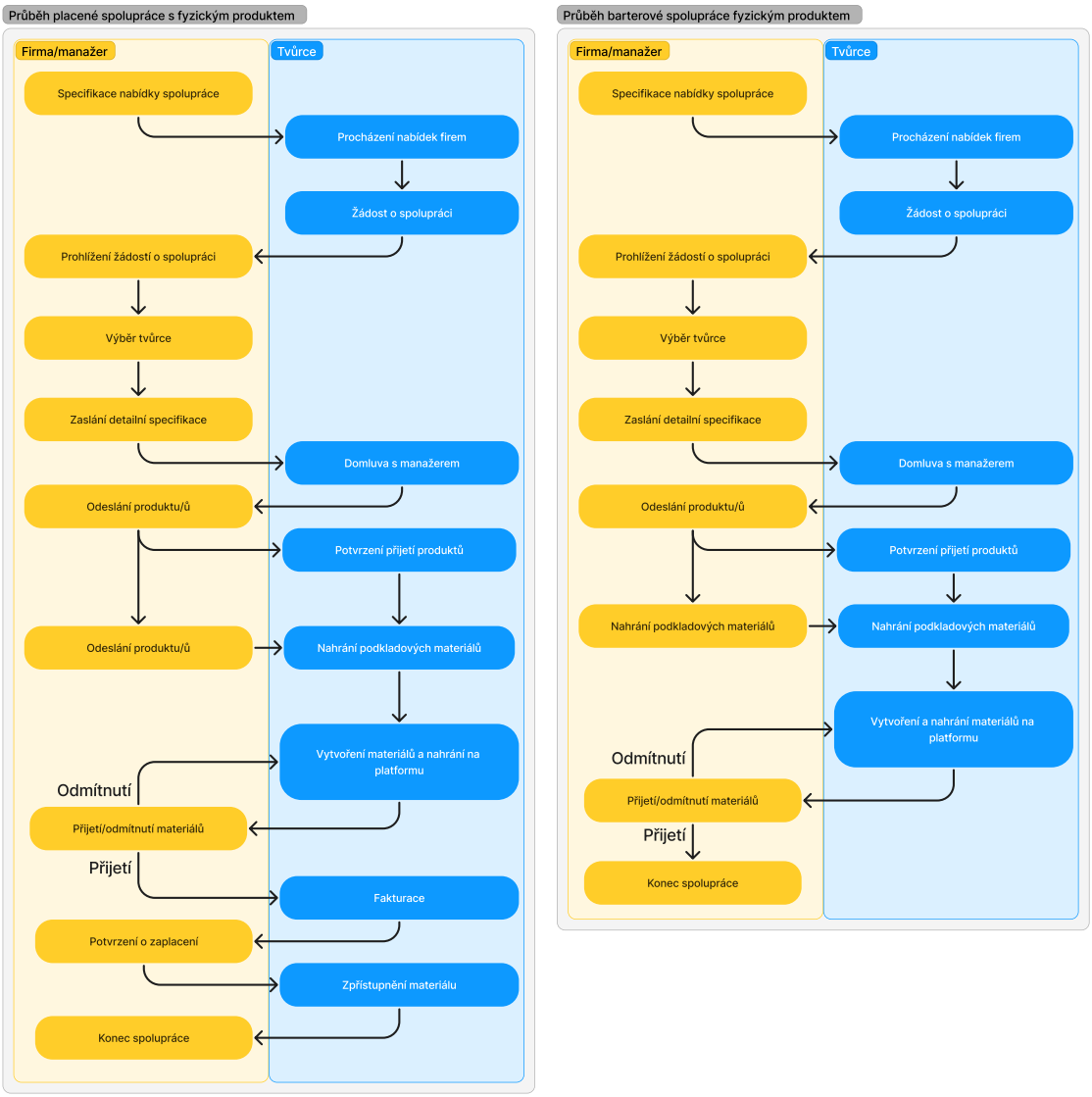
Tvůrci si nabídky budou moci procházet a reagovat na ně zprávou. Manažer, který nabídku vytvořil, bude mít tyto zprávy k dispozici v přehledném seznamu, který bude pro každou spolupráci zvlášť a každou zprávu bude moci zamítnout nebo přijmout – přijmout budou moci i více žádostí, protože firma nemusí hledat jen jednoho tvůrce. V případě zamítnutí je pro manažera volitelné zda-li udá důvod odmítnutí či nikoliv. Po přijetí jsou o této skutečnosti tvůrci informování e-mailem, aby mohli ve spolupráci co nejdříve začít. Pro konkrétní domluvu na produktech a způsobu spolupráce bude k dispozici chat. Tento chat by měl být zobrazen v průběhu celé spolupráce, aby se tvůrce i manažer mohli pohodlně a rychle doptávat na co je napadne.

Aplikace by tedy měla poskytovat jednoduché uživatelské rozhraní, ve kterém bude přehledně zobrazena probíhající spolupráce. Při spolupráci je důležité poznamenat, že se nemusí týkat fyzického produktu (může se jednat např. o natočení UGC videa s nějakou aplikací). Pokud se ale fyzického produktu týká, je třeba, aby uživatel byl schopen v chatu (nebo klidně i do odděleného prostoru) zapsat adresu a způsob zaslání: např. Doručení na adresu Ulice 123, Město, 111 11 pomocí České Pošty. Toto by mělo být viditelně odděleno od zbytku chatu, aby manažer věděl, kam má případný balík poslat. Tvůrce po přijetí balíku musí vytvořit smluvený materiál. K tomuto jsou často potřeba i různé podkladové materiály – inspirační fotografie, scénáře, smlouvy, takže by aplikace měla nabízet i prostor, kam uložit tyto dokumenty. Po vytvoření požadovaného materiálu a jeho nahrání na platformu je třeba, aby se přes videa a fotografie přikreslil vodoznak aplikace nebo jméno autora. Vodoznakem je potřeba označit videa/fotografie, které jsou součástí placené spolupráce, pokud je spolupráce barterová, není nutné nijak materiály upravovat. Zároveň je nutné, aby se manažer k neoznačenému materiálu nemohl dostat do té doby, dokud mu tvůrce vysloveně nedá přístup. Manažer může pořízený materiál tvůrcem odmítnout a poslat tak tvůrci zprávu, aby obsah nějak upravil. Toto se může opakovat několikrát. Jakmile je manažer s poskytnutým materiálem spokojen, tak tento obsah schválí. Pokud se jednalo o barterovou spolupráci, je tímto spolupráce uzavřená a je možné využívat ještě služby chatu. Při placené

spolupráci je postup komplikovanější, jelikož nejprve je potřeba od tvůrce faktura, kterou zašle manažerovi. Ten následně potvrdí, že platbu odeslal. Pak je již na tvůrci, kdy svůj poskytnutý materiál zpřístupní. Po zpřístupnění videa je firmě poskytnuto originální video bez vodoznaku a v plné kvalitě. Toto video si může firma stáhnout alespoň po následující měsíc.

3.2 Popis průběhu spolupráce

Z popisu systému v minulé kapitole je zřejmé, že hlavní součástí systému bude možnost vést spolupráci s protistranou. Na obrázku 3.2 je postup znázorněn na diagramu. Spolupráce bude mít celkově 4 typy. Tyto typy se odvíjí od typu produktu (fyzický či digitální) a formy odměny (placená či barterová).



Obrázek 3.2: Diagram zobrazující posloupnost akcí ze stran firmy/manažera a UGC tvůrce pro typy: placená spolupráce s fyzickým produktem a barterová spolupráce s fyzickým produktem.

Pro lepší srozumitelnost byl postup rozepsán v následujících bodech. Původně bylo navrženo, že pro vedení spolupráce bude navržen čítač, který bude určovat v jaké etapě daná spolupráce je. Při testování ovšem vyšlo najevo, že je tento způsob příliš svazující - uživatelé potřebují mít při vedení spolupráce jistou volnost a nechtějí být omezení např. tím, že na počátku spolupráce musejí nejprve specifikovat požadovaný materiál a až poté přijde na řadu fáze, kdy uživatel může nahrát smlouvy či svoji adresu. Proto jsem od tohoto návrhu upustil a systém vedení spolupráci je více volnější a záleží pouze na tvůrci a manažerovi, jak svoji spolupráci povedou. Následující výčet ukazuje, jak jsem z uživatelských požadavků získal detailní specifikaci systému. To je uvedeno na typickém popisu spolupráce:

1. **Specifikace nabídky spolupráce** – zde manažer vytváří popis nabízené spolupráce. Může různě členit text pomocí různých úrovní nadpisu, odrážek, podtržení atd.

    - **Požadavky:** Uživatelé chtějí flexibilní strukturu textu, která se dá jednoduše zachytit pomocí formátu JSON, proto bude potřeba použít NoSQL dokumentovou databázi MongoDB.

2. **Procházení nabídek firem** – tvůrce je schopen procházet všechny nabídky na spolupráci v systému.

3. **Žádost o spolupráci** – tvůrce zasílá zprávu firmě s textem žádosti.

    - **Požadavky:** Je třeba entita v databázi, která bude uchovávat text zprávy, autora, adresáta a také identifikátor spolupráce, na kterou je žádost poslána.

4. **Prohlížení žádostí o spolupráci** – Manažer je schopen zobrazit všechny žádosti, které se týkají spoluprací, které vypsal. Zároveň jediný manažer (zakladatel firmy) je schopen zobrazit veškeré nabídky všech manažerů, kteří pracují pro firmu, kterou založil.

    - **Požadavky:** Manažeři musí mít atribut, který bude určovat zda firmu vytvořili či ne – jak už ale bylo zmíněno dříve, rozdělení na hlavní a vedlejší manažery je pouze skutečnost se kterou se musí počítat do budoucna. V první verzi aplikace je nutný pouze jediný manažer pro každou firmu.

5. **Výběr tvůrce** – Manažer bude mít možnost vždy přijmout nebo odmítnout žádost tvůrce.

    - **Požadavky:** Uživatelé musí být o odmítnutí informováni, tedy je zapotřebí uchovávat notifikace v databázi. Je velmi pravděpodobné, že tyto notifikace se budou používat i v jiných částech systémů a bude jich několik typů tzn. vzniklá entita v databázi by měla mít: název, typ, text notifikace, čas vytvoření a údaj zda byla uživatelem již zobrazena či nikoliv. Tato entita bude uchována v NoSQL databázi, jelikož může mít proměnlivou strukturu – při kliknutí na notifikaci v uživatelské rozhraní, musí být uživatel přesměrován na stránku odkud notifikace vzešla. Tímto se každá notifikace bude lišit (každá notifikace potřebuje jiné údaje pro přesměrování). Pokud jsou uživatelé přijati je třeba vytvořit novou entitu v databázi, která bude uchovávat informace o probíhající spolupráci. Zároveň z popisu systému vyplývá, že je zapotřebí i zasílat e-maily o skutečnosti, že byl tvůrce přijat. Je tedy potřeba knihovna pro zajištění spojení s SMTP serverem a odeslání e-mailu.

6. **Zaslání detailní specifikace** – Manažer si s uživatelem budou moci vyměňovat zprávy.

    - **Požadavky:** Backend musí být schopen navázat spojení přes WebSocket, jelikož ten dovoluje plně oboustrannou komunikaci (full-duplex) [16]. Toto je potřebné z důvodu aktualizace obsahu chatu v reálném čase bez potřeby obnovovat stránku na straně klienta. Zároveň tento způsob komunikace bude použit pro upozornění uživatele na nové notifikace.

7. **Domluva s manažerem** – v rámci domluvy si budou schopni manažeři i tvůrci mezi sebou zasílat soubory. Speciálně důležitým dokumentem jsou smlouvy, které budou často párovány s dalším dokumentem, který bude obsahovat podepsanou kopii od druhé strany. Zároveň v rámci domluvy předává tvůrce manažerovi doručovací údaje.

    - **Požadavky:** V databázi musí být entita uchovávající informace o každém souboru, který je přiřazen k probíhající spolupráci. Také zde musí být poznačen autor, čas nahrání na platformu a typ. Dále zde bude možnost této entitě připsat vztah k dalšímu souboru, který bude využit v případě, že se bude jednat o smlouvu. Autoři mohou být dvojího typu, manažer či tvůrce, tzn. je zapotřebí mít atributy dva, kde vždy pouze jeden bude nastaven se smysluplnou hodnotou a druhý bude mít hodnotu null. Dalším požadavkem je zaznamenání doručovacích údajů ke spolupráci. Tyto doručovací údaje bude moct uživatel vyplnit a nebo si vybrat z předvyplněných adres, které si předtím uložil. To vede k vytvoření nové entity v databázi – uložená adresa. Zároveň je potřeba tyto informace uvézt ke každé spolupráci (nestačí pouze cizí klíč na uloženou adresu), jelikož při smazání uložené adresy by neexistoval záznam o tom, jakou adresu uživatel vyplnil.

8. **Odeslání produktu:** – toto není nutné v systému nijak modelovat. Při testování se ukázalo, že tento krok je spíše na obtíž – blokuje rychlé provedení spolupráce.

9. **Potvrzení přijetí produktu:** – stejně jako předchozí bod, není nutné toto nijak zpracovávat v systému

10. **Nahrávání podkladových materiálů** – tvůrce bude mít možnost na platformu nahrát různé dokumenty/fotografie/krátká videa, která budou sloužit jako podkladový materiál, aby manažer měl představu, jak bude výsledné video vypadat.

    - **Požadavky:** již zmíněny v kroku Domluva s manažerem .

11. **Vytvoření a nahrání požadovaného materiálu na platformu** – tvůrce by měl mít každý požadovaný materiál odděleně. Každé video/fotografie atd. by měla být na speciální podstránce, jelikož každý tento obsah bude mít své zadání a svá specifika. Do těchto podstránek se bude vytvořený obsah přidávat.

    - **Požadavky:** Pro zajištění škálovatelnosti úložiště a také částečné přesměrování zátěže bude využito cloudového úložiště. To zajistí, že zasílání videí a jejich nahrání budou zajišťovat jiné servery a server s API nebude tak zatížený. Dalším požadavkem na databázi je zde nová entita tvořeného materiálu, který bude připsán ke spolupráci. Tento materiál bude uložen na cloudovém uložišti a tedy v relační databázi bude uložena pouze cesta (klíč), pod kterým jej systém bude

schopný vyhledat. Zároveň u videí a fotografií je v případě placené spolupráce třeba vytvořit kopii videa/fotky s vodoznakem. Tedy je důležité aby v databázi byla případně uložena i cesta k materiálu s vodoznakem. Vodoznak má zamezit odcizení fotky/videa před zaplacením.

12. **Přijetí/odmítnutí videa** – manažer má možnost video odmítnout z důvodu nespokojenosti. A tím pádem musí tvůrce vytvořit nový požadovaný materiál. Při odmítnutí je důležité, aby manažer specifikoval důvod zamítnutí. V opačném případě je pouze tvůrce informován, že daný materiál byl schválen. V okamžiku, kdy je schválen veškerý obsah, který byl ve spolupráci zadán, je tvůrci zpřístupněno tlačítko pro odemčení materiálů, které neobsahují vodoznak.

    - **Požadavky:** Video musí mít atribut, který bude uchovávat informaci zda je video odmítnuto. Pokud je video odmítnuto měla by u entity videa v databázi být možnost přidat komentář, proč bylo video odmítnuto.

13. **Fakturace** – uživatel zasílá fakturu manažerovi.

    - **Požadavky:** Nová entita v databázi, která bude ukládat informace o zaslané faktuře.

14. **Zpřístupnění videa** – tvůrce může zpřístupnit materiál bez vodoznaku, je pouze na jeho uvážení zda-li bude čekat na proplacení faktury či nikoliv.

    - **Požadavky:** Každý zadaný materiál musí mít i atribut zda-li je odemčen či nikoliv.

Pomocí tohoto přístupu, kdy jsou jednotlivé procesy v systému detailněji rozepsány, jsem navrhl schéma databáze a potřebných technologií pro vývoj celého systému. To je zpracováno v kapitole 4.

3.3 Analýza dosavadních řešení

Během analýzy požadavků byla zároveň provedena analýza dosavadních řešení. V čase psaní není známa žádná aplikace na českém trhu, která by měla podobný účel jako tato, proto jsou zde popsány pouze zahraniční webové platformy. Jsou zde zmíněny funkcionality a hlavní zaměření webových aplikací: Grin[1],Upfluence[2] a Creator.co[3].

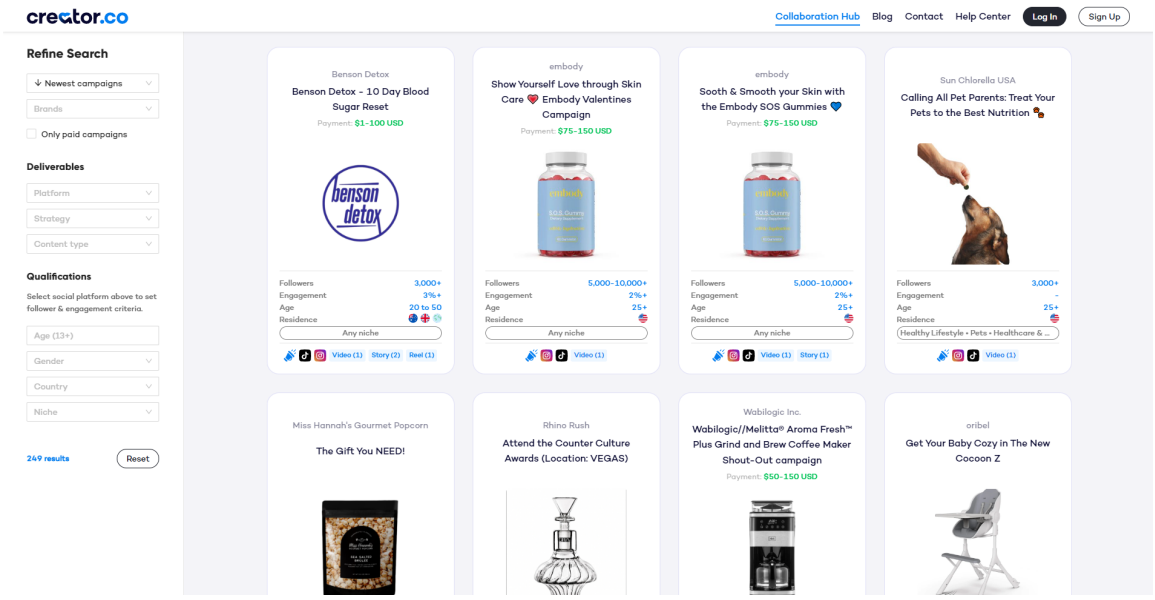
Aplikace Grin je zaměřena hlavně na firmy. Poskytuje jim možnost najít ty správné tvůrce pro jejich sociální sítě. Klade důraz na budování vztahu mezi firmou a tvůrcem pomocí funkcionality, kterou nazývají Creator Relationships Unwrapped. Následně nabízí pokročilou integraci právě se sociálními sítěmi a i jinými nástroji. Je zde také pokročilá analýza úspěšnosti příspěvků. Všeobecně je tento nástroj mířen spíše na tvůrce s vyšším dosahem než na tvorbu organického obsahu.

Stejně jako předchozí platforma je Upfluence orientovaná na hledání správných tvůrců pro registrované firmy. Snaží se pro firmu vyhledávat takové tvůrce, kterým skutečně bude produkt dané firmy vyhovovat. To dělají na základě databáze firmy (příspěvků, produktů) a porovnávají je s tvorbou různých tvůrců na sociálních sítí. Pokud firma nemá tuto databázi

[1] `https://grin.co/`
[2] `https://www.upfluence.com/`
[3] `https://creator.co/`

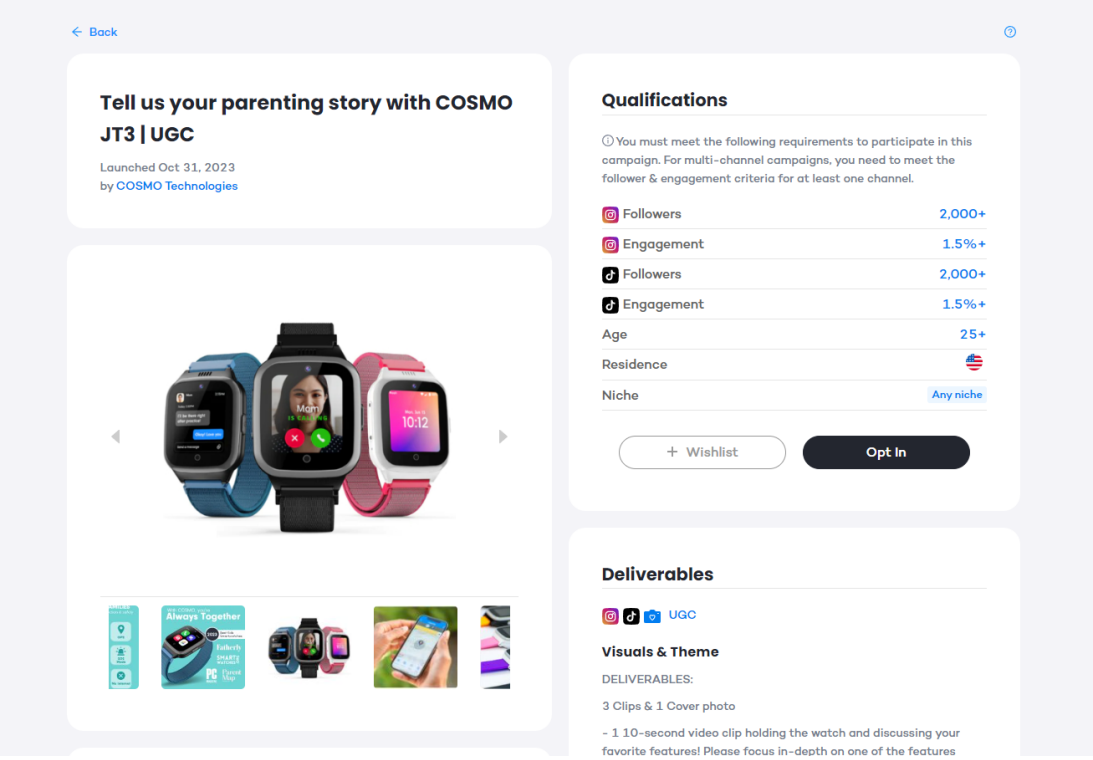
k dispozici např. z důvodu, že jde o začínající firmu, mohou procházet databázi tvůrců, kteří jsou na této platformě k dispozici. Snaží se identifikovat i zákazníky firmy, kteří by se mohli stát tvůrci a tvořit tak věrohodný obsah. Aplikace také dovoluje organizovat celé reklamní kampaně.

Creator.co je platforma, která se zaměřuje jak na firmy, tak i na tvůrce. Hned na úvodní stránce si uživatel vybírá z jaké skupiny uživatelů je a tím jim stránku přizpůsobí. Pro firmy/manažery je zde ukázáno, jak jim platforma dokáže pomoci s hledáním nových tvůrců. Pro nalezení je zde použito vyhledávání na základě atributů tvůrců. Firma/manažer může specifikovat přesně jakého tvůrce hledá – může specifikovat např. počet sledujících, zaměření tvůrce, míru dosahu, věk tvůrce, národnost nebo jazyk, kterým tvůrce mluví a i další. Pro firmy je zde popsán i postup, jak aplikace funguje. Prvně musí stručně specifikovat jakou kampaň si představují, následně se tvůrci na tuto kampaň hlásí a firmy si z těchto tvůrců vybírají. Tento postup by měl podobně fungovat i na aplikaci, která je vyvíjena pod touto diplomovou prací. Jako jedna z mála stránek dovoluje i neregistrovaným uživatelům nahlédnout do nabízených spoluprací. Podle nabízených spoluprací, ale převážně firmy hledají tvůrce z USA, tedy pro český trh je tato služba nepoužitelná.



Obrázek 3.3: Snímek z podstránky Collaboration Hub z webové platformy Creators.co. V levé části je implementováno pokročilé vyhledávání, kde tvůrce může specifikovat své vlastnosti a na základě nich služba zobrazí adekvátní nabízené spolupráce. O každé nabídce jsou zde vždy základní údaje, které by měl tvůrce splňovat - počet sledujících, věk, národnost a dosah. Zároveň je zde specifikována peněžní odměna pokud se jedná o placenou spolupráci.

Jak již bylo zmíněno, prakticky všechny tyto aplikace jsou vytvořeny pro zahraniční, převážně americký trh. Některé aplikace sice nabízejí spolupráce i celosvětově, ale stejně tvůrci musí umět ovládat angličtinu, jelikož vytvářená videa musí být namluvena anglicky. Nová aplikace tedy bude přizpůsobena pro český trh a české značky, které potřebují tvůrce. Dalším rozdílem oproti předešlým aplikacím je zaměření se na tvůrce. Předešlé aplikace jsou většinou koncentrovány na nalezení těch pravých tvůrců pro firmy, ale nová aplikace výběr tvůrců nechá doopravdy na firmách a tvůrci si budou schopni vybrat nabídky spoluprací, které vyhovují jim.



Obrázek 3.4: Detail spolupráce na platformě Creators.co, která ukazuje všechny detaily, které tvůrce musí splňovat pro navázání spolupráce. Důležitými prvky jsou zde detailní zobrazení produktu, název spolupráce a také možnost kontaktovat danou firmu.

Kapitola 4

Návrh aplikace

Na základě popisu systému v kapitole 3.1 a sesbíraných požadavků od uživatelů byla provedena podrobná specifikace systému pomocí diagramů v následujících kapitolách.

4.1 Diagramy případů užití

Na základě popisu systému v kapitole 3.1 bylo identifikováno celkem 4 typy uživatelů v tomto systému: neregistrovaný uživatel, tvůrce, manažer a hlavní manažer. V budoucích verzích aplikace určitě přibudou ještě uživatelé s rolí administrátor a superadministrátor, proto se s nimi počítá i v návrhu relační databáze.

První diagram 4.1 zobrazuje případy užití pro dva typy uživatelů – neregistrovaní uživatelé a registrovaní tvůrci. Neregistrovaní uživatelé mohou v systému provádět následující případy užití:

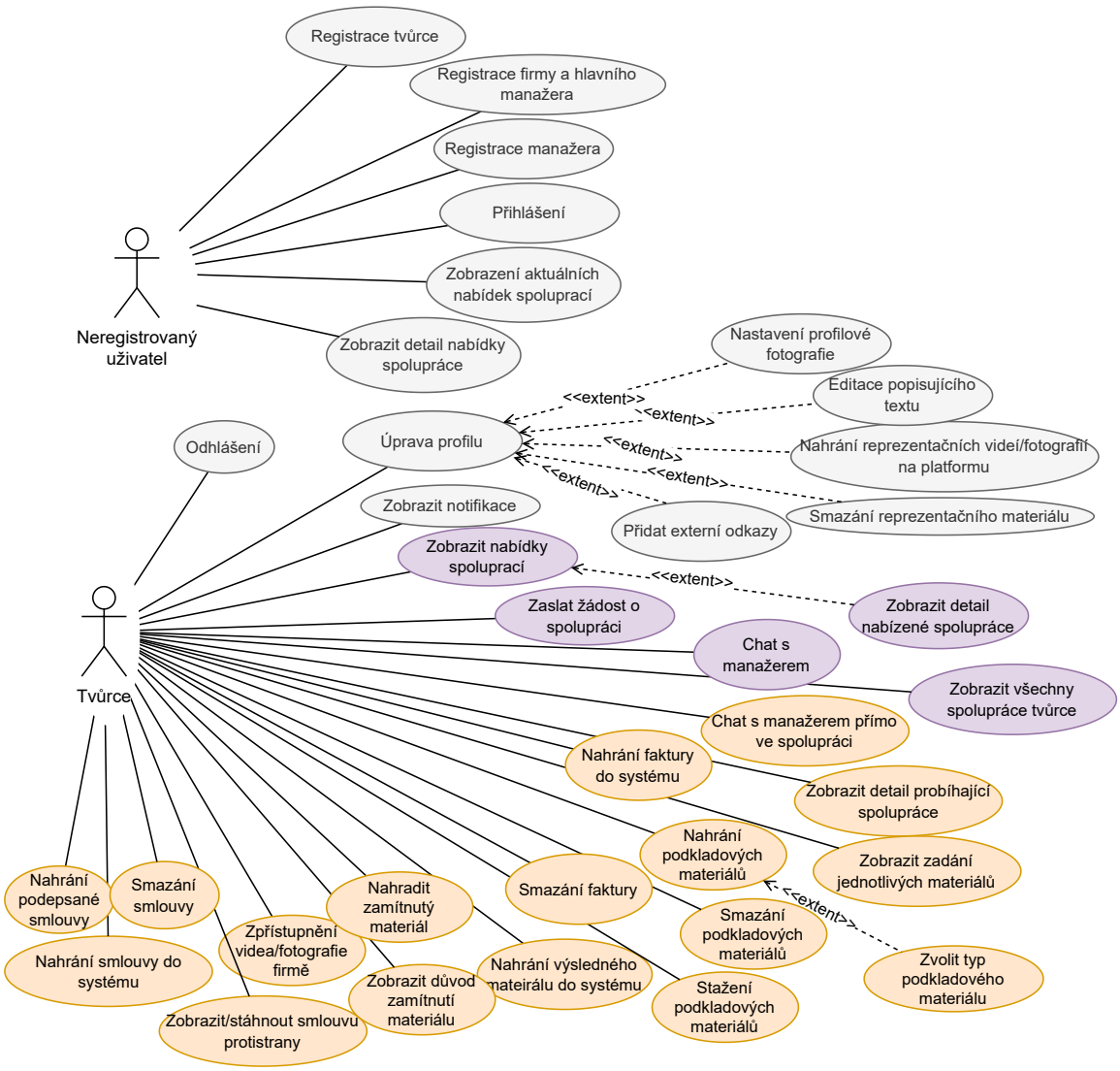
- **Registrace tvůrce** – po registraci je pro uživatele vytvořen nový účet tvůrce.
- **Registrace firmy** – po registraci je pro uživatele vytvořen nový účet hlavního manažera a zároveň je vytvořena v systému i firma, kterou tento manažer spravuje.
- **Zobrazení aktuálních nabídek spoluprací** – i neregistrovaný uživatel má možnost zobrazit nabídku všech spoluprací. Může tak vidět jak je aplikace aktivní, kolik je zde možností spolupráce atd.
- **Registrace manažera** – po registraci je pro uživatele vytvořen nový účet manažera. Samotný manažer v systému prakticky nemůže provádět žádné akce, jelikož potřebuje být zapsán u firmy, pro kterou pracuje. Z tohoto důvodu bude probíhat registrace jinak než registrace tvůrce či firmy. V první verzi systému bude možné tvořit pouze hlavní manažery s firmami. Tvorba ostatních manažerů bude doimplementována později.
- **Přihlášení** – uživatelé se budou moci přihlásit buď pomocí e-mailu a hesla, nebo také pomocí externích služeb, které poskytují např. Google či Facebook.
- **Zobrazení aktuálních nabídek spoluprací** – neregistrovaní uživatelé budou moci procházet nabídky spoluprací, které budou rozřazeny do kategorií.

Pro tvůrce je diagram užití rozdělen do tří barevně odlišných částí. První část jsou obecné případy užití, kde jsou specifikovány věci jako odhlášení, úprava profilu a zobrazení notifikací. Další barevně odlišná část zahrnuje případy užití, které přímo předchází nebo se netýkají systému vedení spolupráce, zde jsou například následující:

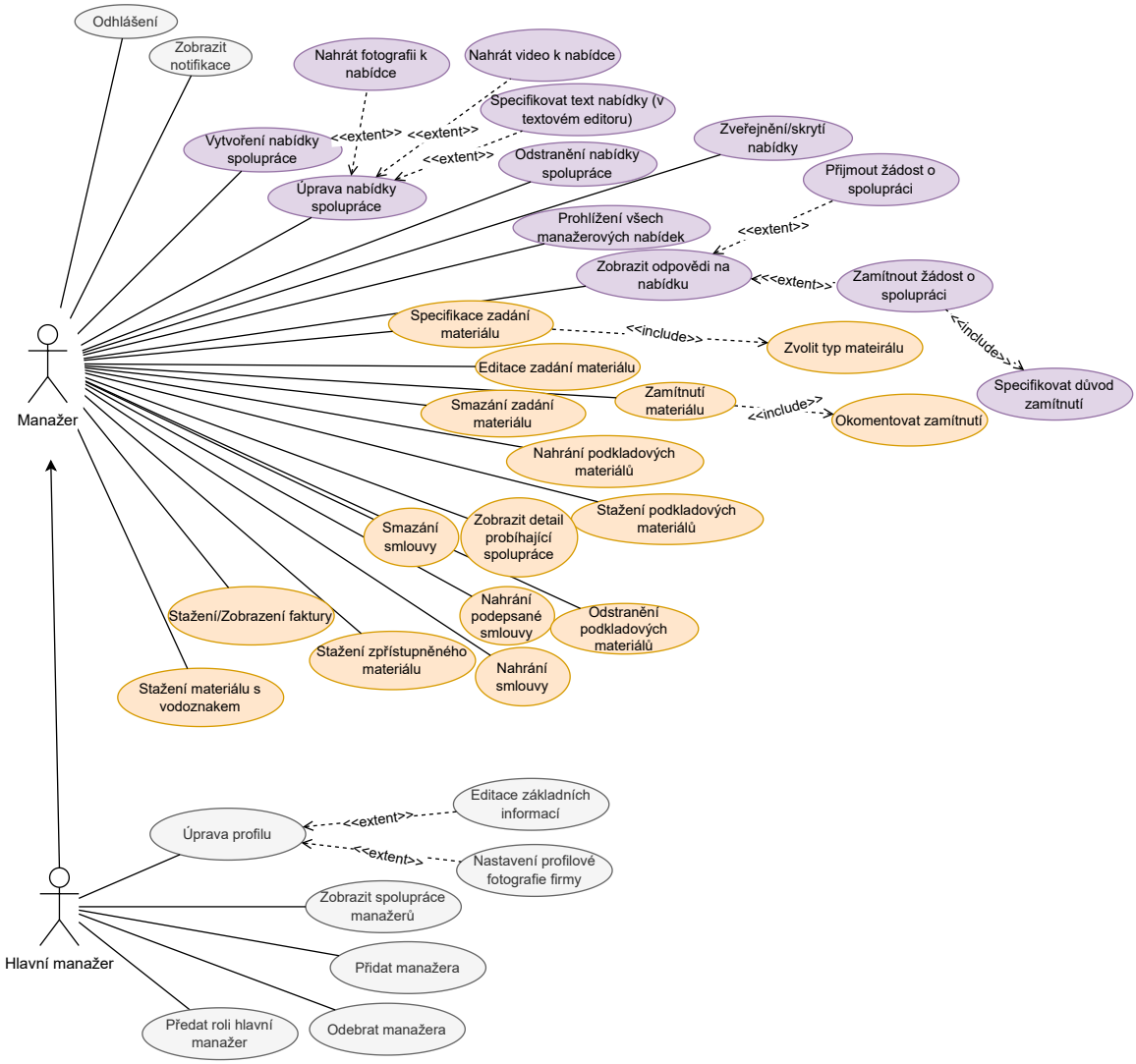
- **Zaslat žádost o spolupráci** – při prohledávání nabízených spolupráci zde budou uživatelé u každé mít možnost zaslat žádost. Tím zašlou manažerovi, který nabídku vytvořil zprávu a ten si bude moci zprávu přečíst, popřípadě přejít i na profil tvůrce a následně pak vybrat konkrétního tvůrce pro spolupráci.
- **Chat s manažerem** – pokud manažer kontaktuje tvůrce přímo s nějakými otázkami musí mu být tvůrce schopen odpovědět.
- **Zobrazit všechny spolupráce tvůrce** – tvůrce by měl mít přehledný seznam všech aktuálních spoluprací.

Třetím druhem případu užití pro tvůrce jsou ty, které se týkají již probíhající spolupráce. Detailní postup spolupráce je sepsán v kapitole 3.2. Z diagramu v této kapitole byly vytvořeny případy užití a dále byly doplněny o další, které se k nim vážou, jako např. nahrání podepsané smlouvy, zobrazit zadání atd. Uvádím detailnější popis jen některých z nich:

- **Nahrání materiálu do systému** – tvůrce musí být schopen nahrát video či fotografii (nebo jiné materiály, na kterých jsou s manažerem domluveni) do spolupráce. Video či fotografie je prakticky hlavní výsledek spolupráce.
- **Nahrání faktury do systému** – za poskytnutý materiál tvůrci nejčastěji vystavují faktury, které prostřednictvím systému budou moci manažerům zasílat.
- **Zpřístupnění videa/fotografie firmy** – pokud jde o placenou spolupráci, musí tvůrce po zaplacení faktury videa zpřístupnit. Videa po dobu spolupráce jsou totiž opatřeny vodoznakem a až po zpřístupnění tvůrcem jsou dostupná bez něj.



Obrázek 4.1: Diagram případů užití pro neregistrované uživatele a tvůrce. V diagramu šedě podbarveny obecné případy užití, fialově případy užití, které probíhají před započetím spolupráce a oranžově jsou ty, které se již týkají systému vedení spolupráce.



Obrázek 4.2: Diagram případů užití pro manažery sociálních sítí a hlavního manažera. V diagramu jsou zeleně zvýrazněny případy užití, které se týkají přímé žádosti o spolupráci s firmou. Oranžově jsou znázorněny ty, které se týkají spolupráce, která byla vytvořena na základě nabídky od manažera. Šedé případy užití jsou obecné.

Dalším typem uživatelů systému jsou manažeři a hlavní manažer (to je zpracováno na obr. 4.2). Jak už bylo zmíněno, tak v první verzi systému bude figurovat pouze hlavní manažer, ale jelikož ze specifikace požadavků vyplynulo, že toto rozdělení uživatelů bude někdy v budoucnu potřeba, je dobré si již teď uvědomit, co který manažer bude moci provádět. Hlavní manažer je od začátku ten, který založil profil firmě. Zároveň tento hlavní manažer má možnost tvořit ostatní manažery firmy nebo je odstraňovat, prohlížet jejich probíhající spolupráce. Zároveň bude mít hlavní manažer svoji roli administrátora firmy předat jinému manažerovi. Zajímavým případem užití je zde již zmíněné přidání manažera do firmy, jelikož chceme zamezit aby se manažeři registrovali sami do systému bez specifikované firmy. To tedy bude fungovat následovně:

1. Hlavní manažer klikne na tlačítko Přidat manažera

2. Poté specifikuje e-mail, kam bude zaslán jednorázový odkaz platný pouze nějakou dobu.

3. Přidávaný manažer na odkaz zaslaný v e-mailu bude moci kliknout a následovně si sám specifikuje své heslo.

Hlavní manažer, oproti normálním manažerům, má možnost spravovat profil firmy a upravovat jej. Jinak má hlavní manažer možnost dělat všechny ostatní případy užití stejně jako normální manažer.

Normální manažer bude v systému prakticky dělat správu podmnožiny spoluprací dané firmy. Bude schopný vytvářet nabídky spoluprací, vybírat ze žádosti na spolupráci tvůrce a dále s ním tuto spolupráci vézt. Opět uvádím jen několik příkladů případů užití:

- **Vytvoření nabídky spolupráce** – Manažer bude mít k dispozici textový editor, kde bude specifikovat nabídku spolupráce. Pro vytvoření nabídky je třeba krom popisu zadat i typ produktu, kterého se týká tato spolupráce (fyzický, digitální, služba). Manažer také musí specifikovat zda se jedná o barterovou spolupráci či placenou a případně v jaké výši odměnu nabízí. Také specifikuje počet tvůrců, které hledají. Jako poslední musí specifikovat i kategorii do které nabídka spadá.

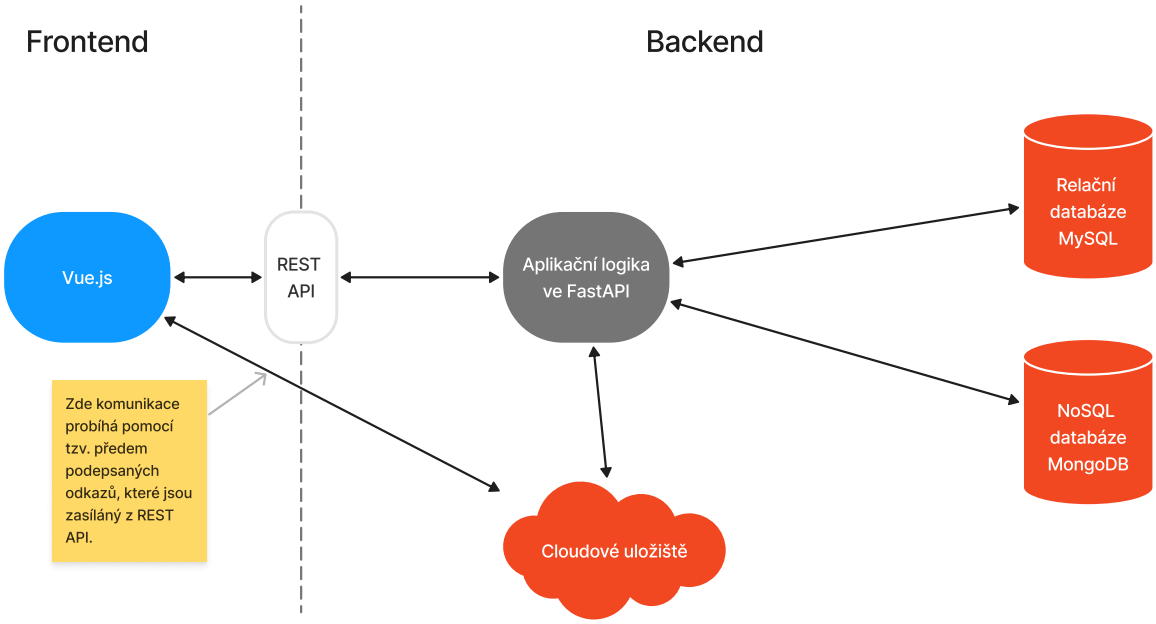
- **Specifikace zadání materiálu** – Manažer po spuštění spolupráce s tvůrcem musí specifikovat, jak mají požadované materiály vypadat. Zde specifikuje typ materiálu (video/fotografie/audio/jiné) a následně může zadat textový popis. Manažer, ale není povinen tento textový popis zadávat, jelikož v praxi se vyskytuje spousta spoluprací, kdy manažer výsledný materiál nechává jen na tvůrci.

- **Zamítnutí materiálu** – Manažer musí být schopen zamítnout poskytnutý materiál tvůrcem. V tomto případě musí specifikovat i důvod proč je materiál zamítnut, aby tvůrce věděl na co se má při změně soustředit.

- **Zobrazit detail probíhající spolupráce** – Manažer bude přesměrován do systému vedení spolupráce.

4.2 Architektura systému

Tato kapitola popisuje architekturu navrženého systému. Detailnější popis technologií a jejich porovnání je v následující kapitole 5.



Obrázek 4.3: Navržená architektura systému

- Frontend - klientská strana aplikace bude naprogramována pomocí javascriptového frameworku Vue, který je blíže popsán v kapitole 5.1.2. Dále klientská strana bude přímo komunikovat s cloudovým úložištěm pomocí předem podepsaných odkazů. Tímto způsobem bude docíleno menšího zatížení serveru s API.
- Backend - serverová strana aplikace bude rozdělena do několika částí:
  - REST API – rozhraní, které bude přijímat HTTP požadavky z klientské části systému, zpracovávat je a odesílat odpovědi. Zároveň bude sloužit jako prostředník pro komunikaci s dalšími prvky systému jako jsou databázová či cloudová úložiště.
  - Relační databáze MySQL – tato databáze bude sloužit pro uložení všech relačních dat jako jsou např. uživatelé, nabídky spolupráce, probíhající spolupráce, kategorie atd. Detailní popis ve formě ER diagramu je v následující kapitole 4.3.
  - NoSQL databáze MongoDB – tato databáze je použita pro uchování textů jednotlivých nabídek spoluprací, jelikož ty budou ukládány ve formátu JSON. Dále jsou zde dokumenty, které reprezentují chat mezi uživateli nebo také notifikace. Detailní popis položek v této databázi je také v následující kapitole.
  - Cloudové úložiště – zde se budou uchovávat prakticky veškerá multimediální data. Budou zde uložena jak videa/fotografie, která mají tvůrci v rámci svých portfolií, tak i videa/fotografie a další, která jsou odeslána v rámci spoluprací.

4.3 Struktury relační a nerelační databáze

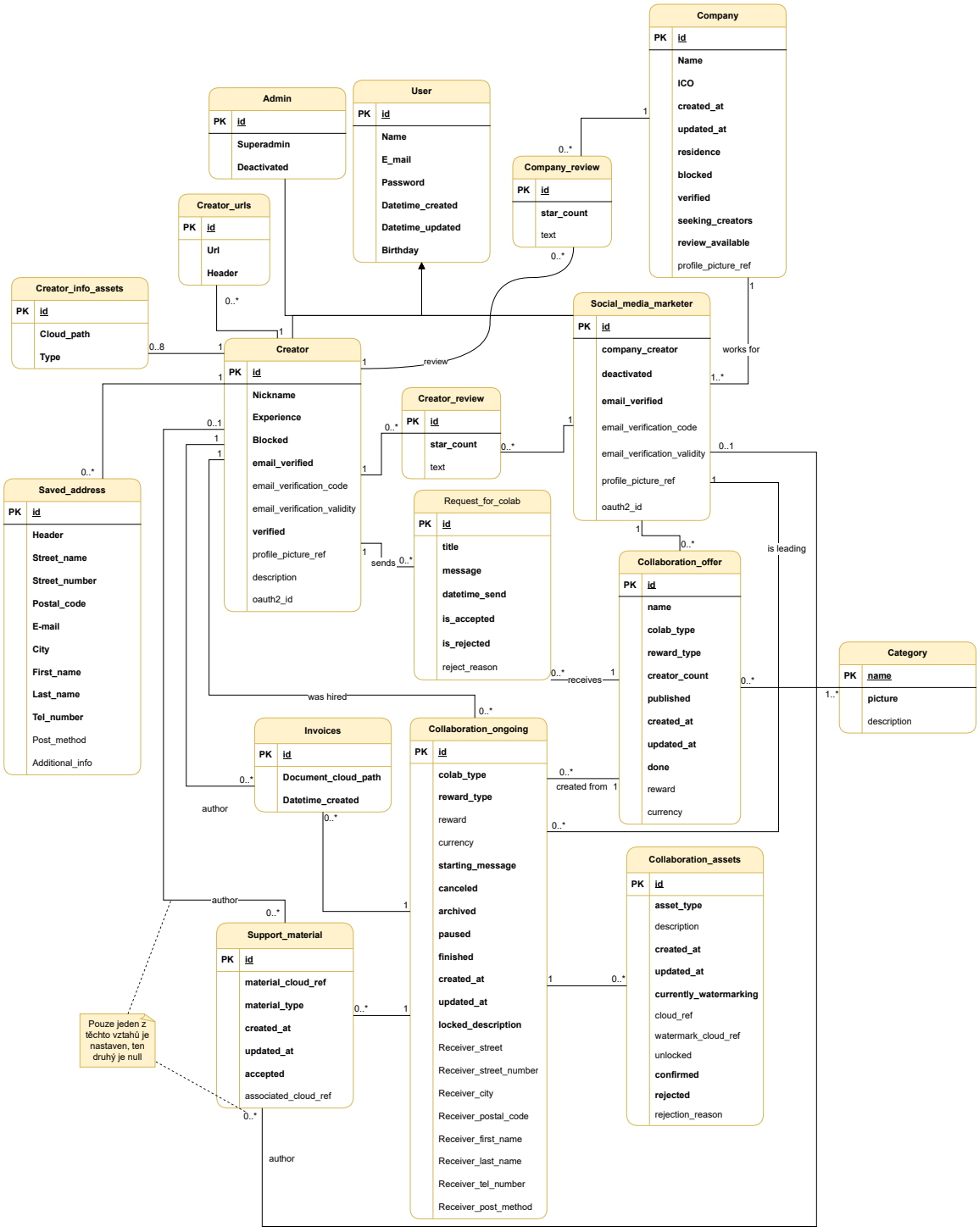
Jak již bylo zmíněno v předchozí kapitole v aplikaci budou jak relační tak nerelační data. Ta nerelační budou uchována v databázi MongoDB a budou uchovávat informace o hierarchii textu nabídek spolupráce. V této databázi budou také uloženy chaty mezi uživatelem a manažerem při spolupráci, stejně tak zde budou uchovány zprávy mezi manažery a tvůrci ještě před začátkem jakékoliv spolupráce. Chaty zde budou uloženy ze dvou důvodů. První je ten, že je jednodušší uchovávat atributy od koho a komu je zpráva směrována, jelikož zde nejsou integritní omezení na cizí klíče a zároveň zde budou i další atributy u každé zprávy. Odstranění integritních omezení byl mimo jiné jeden z důvodů, proč NoSQL databáze vznikly [32]. Mezi další důvody můžeme zařadit např. potřebu lepší škálovatelnosti, potřeba specializovaných vyhledávacích dotazů, které relační databáze nezvládaly atd. Dalším důvodem, proč byla zvolena NoSQL databáze, je potenciální různá struktura zpráv v chatu. Je velmi pravděpodobné, že v budoucích verzích budou zprávy různě strukturované s možností např. zvýrazňovat v textu, přikládat ke zprávám obrázky, označovat na kterou zprávu je odeslána zpráva odpovědí atd.

Relační data budou uchovány v databázi MySQL. Na obrázku 4.4 je viditelná struktura jednotlivých entit. Jelikož se jedná pouze o ER diagram bude výsledných tabulek v databázi více, jelikož jsou zde i vztahy N ku M. V návrhu, ale dominují vztahy 1 ku N, což znamená, že v databázi o moc víc tabulek nevznikne. Na diagramu lze vidět 17 entit, které budou zajišťovat persistenci systému.

Jelikož všechny typy uživatelů mají několik společných atributů je v diagramu využita generalizace, tzn. že entity `Creator`, `Admin` a `Social_media_markter` sdílí atributy jméno, e-mail, heslo, čas vytvoření, čas poslední aktualizace a datum narození. Tyto atributy jsou zobrazeny v entitě `User`.

Entita Creator bude obsahovat informace o tvůrci, zároveň je to entita, na kterou bude odkazovat nejvíce cizích klíčů. Přehled všech entit, které odkazují na tabulku tvůrce je uveden v následujícím výčtu:

- `Creator_urls` – entita, která uchovává odkazy, které uživatel specifikuje – např. na sociální sítě, blog či vlastní portfolio tvůrce.
- `Creator_info_assets` – tato entita uchovává informace o propagačních materiálech tvůrce. Je zde zajímavý vztah, kdy každý tvůrce může mít maximálně 6 takových připsaných entit, jelikož tvůrce se může prezentovat pomocí 3 fotografií a 3 videí. To bude kontrolováno na úrovni API aby odpovídal počet jednotlivých typů materiálu. To z důvodu šetření úložiště, ale hlavně také kvůli šetření času manažerů. Tvůrci si vyberou doopravdy ten nejlepší materiál, který je reprezentuje a pro manažery je tak jednodušší vybrat správného tvůrce.
- `Saved_adress` – uživatel má možnost si uložit několik adres, které pak může vkládat přímo do spolupráce. Ty obsahují číslo domu a název ulice, poštovní směrovací číslo, město, e-mail, křestní jméno i příjmení, způsob zaslání, telefonní číslo a další informace. Také je zde titulek, kterým budou moci uloženou adresu popsat. Křestní jméno i příjmení je u adresy z toho důvodu, že tvůrce si může uložit i dodání na adresu a jméno někoho jiného (např. kvůli nepřítomnosti). Do atributu s dalšími informacemi může uživatel vyplnit např. patro či číslo bytu.
- `Request_for_collab` – tato entita bude uchovávat zprávy, které uživatelé budou zasílat jako žádost o nabídku spolupráce. Bude obsahovat nadpis zprávy, text zprávy



Obrázek 4.4: Entity-Relationship diagram databáze pro platformu Content Cupid.

a čas zaslání zprávy a pak také informace jestli byla žádost přijata či zamítnuta a případný důvod zamítnutí.

- `Company_review` – pokud bude mít firma zapnutou možnost hodnocení firmy budou uživatelé schopni pomocí počtu hvězdiček a textu firmu hodnotit na základě předchozí spolupráce.
- `Collaboration_ongoing` – entita s nejvíce atributy. Detailněji je popsána dále v této kapitole.
- `Invoices` – záznamy této tabulky budou informace o fakturách, které tvůrce vytvořil za spolupráci. Záznamy v této tabulce budou tvořeny pouze klíčem, pod kterým jsou uloženy v cloudu, a datem vytvoření.
- `Support_material` – dodatečný materiál ke spolupráci – scénáře, pokyny k videu atp.
- `Creator_review` – manažeři, kteří vedli spolupráci s tvůrci budou moci psát hodnocení stejně jako tvůrci na firmy.

Informace o tvůrci, které budou uloženy v systému jsou následující: přezdívka, popis (nepovinný), zkušenost (počítadlo úspěšných spoluprací), booleovské hodnoty zda-li je tvůrce zablokován či nikoliv a zda je jeho účet verifikován pomocí e-mailu, který mu byl zaslán a také je zde odkaz na profilovou fotografii (nepovinný).

Dalším uživatelem systému budou manažeři. O manažerovi jsou kromě společných atributů z entity `User` uloženy informace: jméno, odkaz na profilovou fotografii (nepovinné), booleovská hodnota zda se jedná o hlavního manažera a booleovská hodnota zda není manažer zablokován. O tomto typu uživatele je také uchována informace pro kterou firmu pracuje, jaké nabídky spolupráce vytvořil a jaký podpůrný materiál vložil do spoluprací. Entita `Company` je zde využita pouze jako prezentační – za firmu se nikdo nemůže přihlásit. Jediný hlavní manažer, který pro firmu pracuje, ji může měnit.

Entita `Collaboration_ongoing` reprezentuje probíhající spolupráci. Jsou zde uloženy atributy: typ spolupráce, fáze spolupráce, booleovské atributy jestli je spolupráce uzavřená, archivovaná či pozastavená. Dále je zde uchována i první zpráva, kterou specifikuje manažer při přijetí tvůrce na spolupráci. Jsou zde také informace, kdy byla spolupráce naposledy aktualizována a kdy byla vytvořena. Entita je přítomna ve vztazích s následujícími entitami:

- `Collaboration_assets` – odkaz na materiál, který je brán jako hlavní cíl spolupráce. Tato entita uchovává informace jako – typ materiálu, zadání, datum vytvoření/poslední aktualizace. Dále jsou zde klíče k materiálu v plné kvalitě a jeho verzi s vodoznakem a je zde také uložena informace jestli materiál je v přítomnosti v procesu, kdy je vodoznakem opatřován.
- `Social_media_marketer` – odkaz na manažera, který vede spolupráci.
- `Invoices` – tabulka faktur, která se odkazuje na běžící spolupráci.
- `Support_material` – podpůrné materiály, které jsou připsány této spolupráci.
- `Creator` – tvůrce, který byl vybrán pro spolupráci.
- `Colaboration_offer` – odkaz na nabídku spolupráce, ze které byla vytvořena entita `Collaboration_ongoing`.

Poslední entitou v databázi budou administrátoři, kteří budou mít opět společné atributy z entity `User`. Kromě toho, ale budou mít atributy, které specifikují zda se jedná o superadministára a zda administrátor není deaktivovaný – nebo-li nemůže se k účtu přihlásit. Některé z entit v databázi nebyli při vývoji použity, jelikož nebyly součástí první verze systému. Jsou zde ale uvedeny kvůli úplnosti a také kvůli tomu, aby pro budoucí změny v implementaci nebylo potřeba provádět větší zásahy do databáze.

Nerelační databáze MongoDB bude obsahovat tři kolekce JSON souborů:

- `chats` – kolekce, kde každý JSON záznam bude zachycovat jeden chat mezi manažerem a tvůrcem. V této kolekci budou uloženy jak chaty v rámci jedné spolupráce, tak i ty, které se žádné spolupráce přímo netýkají. Každý objekt bude tvořen atributy:identifikátor, první uživatel, druhý uživatel, a pole zpráv. Toto pole bude obsahovat jednotlivé zprávy v podobě objektů – každá zpráva bude obsahovat informace o odesílateli i adresátovi, čas odeslání, booleovskou hodnotu zda byla zpráva zobrazena a případně čas zobrazení a nakonec text zprávy. Zde je nutné podotknout, že při vývoji se zprávy nijak nešifrovali, ale jakmile bude aplikace dostupná pro širokou veřejnost musejí být zprávy uloženy v šifrované podobě. Chaty, které budou součástí probíhající spolupráce budou opatřeny i identifikátorem, ke které spolupráci patří.

- `notifications` – záznamy této kolekce budou jednotlivé notifikace. Ty budou uchovávat informace o adresátovi a také o uživateli, který notifikaci vyvolal. Dále je zde nadpis a zpráva notifikace a také její typ.

- `offers` – záznamy v této kolekci specifikují podobu nabídky spolupráce. Tyto záznamy budou mít stejný identifikátor jako nabídka v relační databázi a dále budou obsahovat atribut, který reprezentuje textový obsah nabídky spolupráce a pak pole fotografií a videí, které byly ke spolupráci přiloženy

Dokumenty, které budou uchovány v těchto kolekcích jsou zobrazeny na obrázku 4.5.

a)

```
{
    "_id": 1,
    "created_by_id": 15,
    "content": {
        "type": "doc",
        "content": [ ... ]
    },
    "videos": ["key/to/cloud/video1.mp4"],
    "images": ["key/to/cloud/image1.mp4"]
}
```

b)

```
{
    "id": "ObjectId( ... )",
    "chat_user_1": "user@email.com",
    "chat_user_1_type": "SMM",
    "chat_user_2": "user2@email.com",
    "chat_user_2_type": "CREATOR",
    "messages": [
        {
            "id": 3,
            "from_user": "user@email.com",
            "to_user": "user2@email.com",
            "message": "Hello, how are you?",
            "datetime_send": "2024-04-12 09:50:58.150891",
            "viewed": false,
            "viewed_at": null
        }
    ],
    "collaborationId": 1
}
```

c)

```
{
    "_id": "ObjectId( ... )",
    "type": "NEW_REQUEST",
    "header": "Nová žádost o spolupráci",
    "message": "{Uživatel} vám zaslal/a žádost na vaši nabídku o spolupráci",
    "viewed": false,
    "viewed_at": null,
    "created_at": "2024-04-10 09:50:58.150891",
    "notified_user_type": "SMM",
    "notified_user_id": 1,
    "associated_user_type": "CREATOR",
    "associated_user_id": 5,
    "additional_info": {
        "collab_request_id": 1,
        "collaboration_offer_id": 3
    }
}
```

Obrázek 4.5: Tři objekty ve formátu JSON, které budou uloženy v separátních kolekcích. Část a) zobrazuje formát jednoho objektu, který reprezentuje podobu nabídky spolupráce. atribut content bude obsahovat json reprezentaci strukturovaného textu, který bude zobrazen u nabídky spolupráce. Část b) ukazuje v jakém formátu budou záznamy chatu pro zprávy mezi tvůrcem a manažerem. Konkrétně tento objekt je i přiřazen ke konkrétní spolupráci. Pokud chat nemá atribut collaborationId, jedná se o obyčejný chat mezi manažerem a tvůrcem mimo spolupráci. Část c) ukazuje příklad uložené notifikace. Tato konkrétní notifikace je zaslána manažerovi, jakmile nějaký uživatel podá novou žádost o spolupráci v reakci na nabídku spolupráce.

4.4 Návrh Rest API

Návrh RestAPI v podstatě probíhal zkombinováním diagramu případů užití z kapitoly 4.1 a ER diagramu z kapitoly 4.3. Zkombinováním datového modelu s jednotlivými případy užití jsem dostal velmi dobrou představu o tom, jakou strukturu by samotné API mělo mít.

Při návrhu jsem přemýšlel nad dvěma způsoby definování REST API. Buď by se mohlo jednat o víceúčelové endpointy, které by prováděli různé věci. Tím lze např. redukovat počet různých PUT požadavků. Ztrácí se tím ale dokumentovatelnost API. Stejně tak už dané endpointy neslouží pouze jedinému účelu, ale mnoha. Např. pro potvrzení odeslání a potvrzení přijetí produktů by mohl být jeden endpoint, stejně tak by se mohly sloučit se všemi PUT metodami, které se týkají stejné entity. Takový endpoint by pak přijímal vždy celý objekt probíhající spolupráci a celý by jej nahrazoval. Nejen že by prováděl spoustu různých změn, ale také by bylo velice netransparentní, kdy se v systému stala jaká změna. Proto jsem se rozhodl pro vyšší granularitu a rozdělil jsem endpointy na základě případu užití. Vede to sice na jejich vyšší počet a i počet modelů dat, které se budou zasílat na/z backend/u, ale každý endpoint je pak zodpovědný pouze za jedinou věc, což vede na lepší dokumentovatelnost, testovatelnost a celý systém je poté čistší.

REST API bylo rozděleno dle funkcionality – podle toho pro které uživatele jsou určeny a také podle toho, kde se v systému používají. Tímto vzniklo celkem 10 kategorií endpointů, které popisuji v následujícím výčtu:

- **Authentification** – endpointy, které se týkají autentizace jsou zde např. endpointy pro: registraci tvůrce či firmy, přihlášení pomocí jména a hesla, přihlášení pomocí OAuth, potvrzení e-mailu po registraci atd.
- **Chat** – endpointy týkající se funkcionality chatu. Např. : získání všech chatů tvůrce, získání zpráv k chatu, zaslání zprávy atd.
- **Company** – endpointy, které se týkají firmy. Např. : získání detailů o firmě, změna profilového obrázku atd.
- **Creator** – endpointy, které se týkají nastavení účtu tvůrce. Jsou zde také endpointy, které tvůrce používají před započetím spolupráce tzn.: zaslání žádosti o spolupráci, poskytnutí detailu nabídky.
- **Creator_collaboration** – endpointy, které slouží pro zpracování požadavků, které tvůrce provádí v systému řízení spolupráce. Např. nahrání smluveného materiálu, zadání adresy tvůrce atd.
- **SMM** – podobně jako v kategorii Creator, tyto endpointy slouží pro nastavení uživatelského účtu manažera (zkratka SMM – social media marketer) a také jsou zde implementovány operace pro správu nabídek spoluprací.
- **SMM_collaboration** – podobně jako Creator_collaboration, tyto endpointy slouží pro veškeré operace, které provádí manažer během vedení spolupráce
- **Universal_collaboration** – univerzální endpointy, které jsou použity v systému vedení spolupráce. Např. zisk detailu spolupráce
- **General** – obecné, jako např.: zisk kategorií nabídek či zisk notifikací pro konkrétního uživatele.

4.5 Návrh uživatelského rozhraní

Podle [37] je při návrhu aplikací důležité myslet na následující vlastnosti:

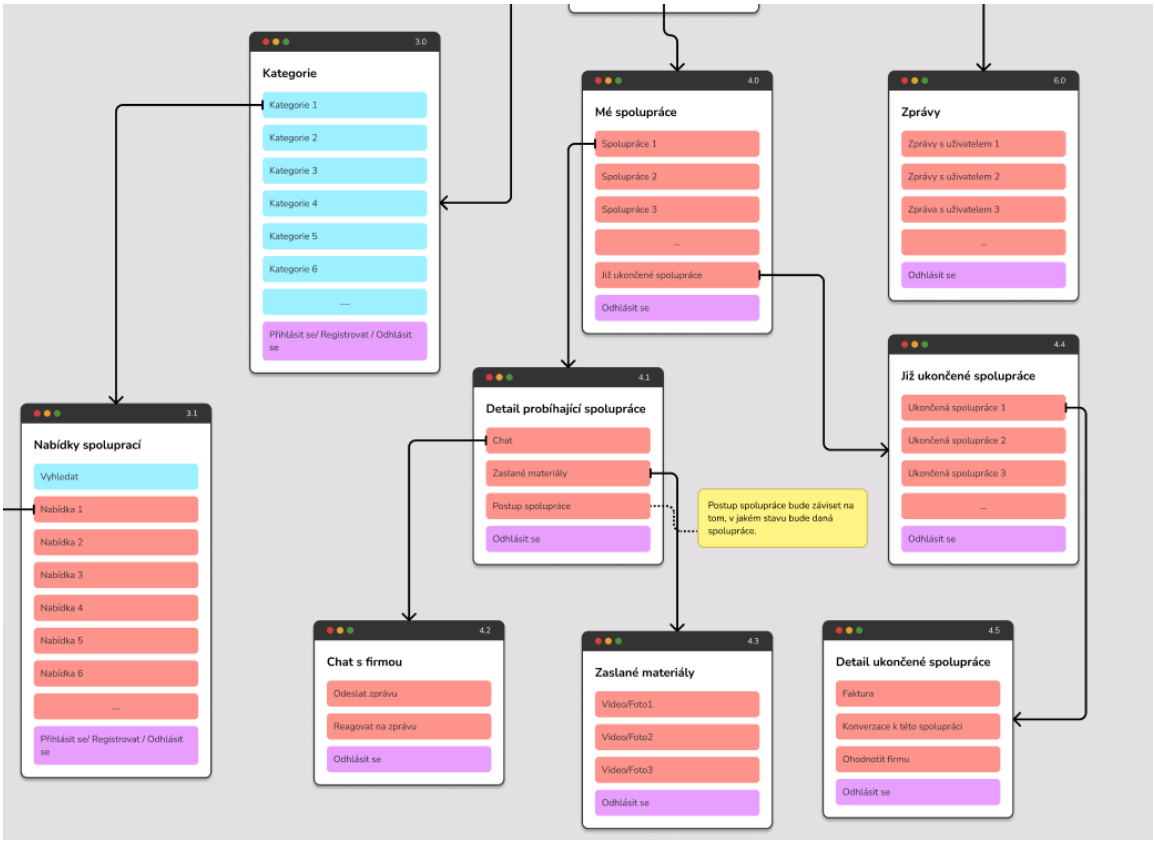
- Použitelnost – dělá aplikace něco, co lidé potřebují?
- Naučitelnost – dokáží se lidé naučit, jak se aplikace používá?
- Zapamatovatelnost – musí se uživatelé znovu učit, jak se aplikace používá, pokud se k ní vrátí?
- Efektivita – dělá aplikace svojí práci za relativně krátkou dobu a s malou snahou uživatelů?
- Žádoucí – chtějí lidé tuto aplikaci?
- Příjemné – je používání aplikace příjemné, nebo dokonce zábavné?

Zároveň podle této knihy jsou uživatelé při používání webových aplikací unáhlení až zbrklí. Návrhář si často při návrhu rozmýšlí, jak by uživatelé postupně měli procházet webovou stránku krok po kroku, ale uživatelé si ji rychle prohlédnou a kliknou na první tlačítko, které jim připadá, že by mohlo dělat jejich požadovanou funkci [37]. Design webové stránky by tak neměl v uživateli vyvolat jakékoliv otázky. Měl by být tzv. sebevysvětlující. Uživatelé by měli sami rozeznat klikatelné objekty od neklikatelných, sami by měli určit, které prvky jsou na stránce důležité a které méně. Důležitější prvky by tak měly být větší nebo jakýmkoliv způsobem výraznější než ostatní: např. použití tučného písma nebo použití výraznější barvy. Zároveň prvky, které spolu souvisí logicky, spolu musí souviset i graficky. Posledním pravidlo hierarchie podle [37] se týká vnořování. Aplikace by graficky měla členit prvky tak, aby uživatelé věděli, které prvky jsou vnořené a které naopak nadřazené.

Návrh aplikace probíhal v několika fázích. První fáze spočívala v rozvržení celé webové aplikace. Z analýzy případů užití jsem navrhl tzv. stránkovou mapu (výřez z mapy lze vidět na obr. 4.6). Stránková mapa ukazuje jednotlivé stránky webové aplikace. Ukazuje z jaké stránky může uživatel přejít na jakou a které různé odkazy budou na stránce k dispozici. Tímto jsem dostal přesnou představu o samotném počtu stránek v celé webové aplikaci, kterých je nakonec přes 20.

Druhou fází návrhu byla tvorba wireframů, které byly tvořeny s respektováním dříve zmíněných principů. Hlavní úlohou těchto jednoduchých modelů je získat představu o finální struktuře každé stránky [35]. Po navržení wireframu jsem přešel k finálním designům, které vždy z wireframu vychází a pouze mu přidají barvy, různé abstraktní detaily a konkrétní obrázky. Wireframy pro tuto aplikaci byly tvořeny detailněji než běžné wireframy, jelikož je pak lehčí z nich vytvořit finální design. Přechod od wireframu k finálnímu návrhu je zpracován na obrázku 4.7. Finální designy různých stránek aplikace jsou zobrazeny v příloze A.

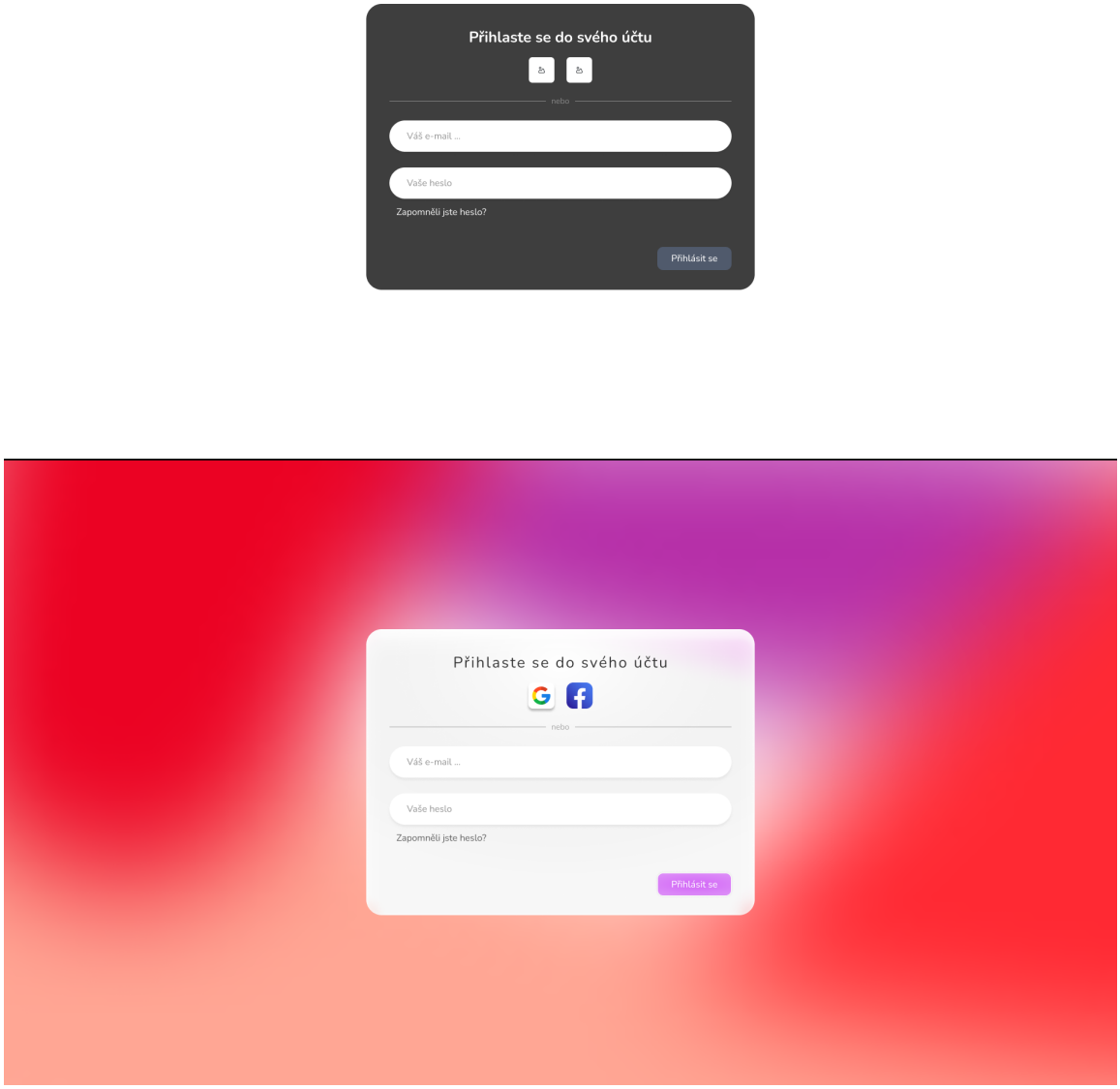
Krom jednotlivých stránek bylo potřeba navrhnout i jednotlivé elementy uživatelského rozhraní jako jsou: ikony, vstupy, tlačítka, odkazy, navigační lišty atd. Návrh tlačítek je zobrazen na obrázku 4.8. Podle [37] je velice důležité odlišit tlačítka od všech ostatních elementů a je také důležité, aby uživatel neměl problém co na webové stránce je klikatelné a co ne. Proto jsem tlačítka navrhl jako velice odlišné od ostatních elementů. Tlačítko může tvořit buďto jen text nebo text s ikonou. Tyto prvky jsou umístěny do středu tlačítka a okolo tohoto středu je velký prostor (padding), aby tlačítko bylo opticky výrazné. Zároveň se v aplikaci používají tlačítka dvou typů – outlined a contained. Tlačítka outlined mají



Obrázek 4.6: Výřez stránkové mapy, která byla použita v rámci návrhu uživatelského rozhraní aplikace. Na výřezu je vždy jeden box jedna stránka, která obsahuje odkazy na další. Přesměrování na jinou stránku je zde zpracováno pomocí šipek. Odkazy mají různé barvy – modré odkazy jsou veřejné a může na ně přejít kdokoliv, červené jsou soukromé, tedy takové, které mohou používat pouze přihlášení uživatelé a fialové jsou ostatní odkazy.

bílé pozadí a jsou ohraničené růžovým okrajem. Naopak contained tlačítka mají výrazné pozadí, které je tvořeno přechodem růžové barvy do modré. Zároveň pro tato tlačítka bylo navrženo několik stavů podle [14]. Tlačítka jsou navržena pro čtyři stavy:

- Enabled – tlačítko, které je volně umístěno na stránce.
- Hovered – tlačítko, nad kterým uživatel umístí kurzor myši.
- Active – tlačítko, na které uživatel klikl.
- Disabled – tlačítko, které je zablokované proti kliknutí (v jazyku HTML je na tlačítku použit atribut disabled)



Obrázek 4.7: Vrchní polovinu obrázku tvoří wireframe, který určoval finální rozložení přihlašovací stránky. Ve spodní polovině obrázku je pak finální návrh přihlašovací stránky.

Obrázek 4.8: Navržená tlačítka, která budou použita napříč celou aplikací ContentCupid

Kapitola 5

Technologie pro vývoj aplikace

Před samotnou implementací navržené aplikace je důležité vybrat adekvátní technologie, které se pro implementaci budou využívat. V této době se již nevybírá pouze konkrétní programovací jazyk, ale také konkrétní framework, který se při implementaci využije. Pro všechny technologie, které jsou vybrány pro vývoj aplikace se používá název Tech stack (zásobník technologií) [67]. Každá webová aplikace se skládá z několika technologií, které jsou na sobě více, či méně závislé a tím tvoří zásobník [30]. Moderní webová aplikace se totiž skládá z několika systémů, datových sad, databází, služeb a vývojových nástrojů. Vývoj aplikace můžeme rozdělit do dvou odvětví: klientská strana aplikace (frontend) a serverová strana aplikace (backend). Dnes, kdy je na výběr z velkého množství technologií, může být velmi složité zvolit konkrétní technologii pro obě tyto odvětví. Výběr totiž záleží na několika různých faktorech [67]. Uvádím jen některé z nich:

1. Velikost a složitost projektu,
2. specifikace a funkcionalita projektu,
3. požadavky na rychlost systému,
4. počet uživatelů, kteří budou systém využívat,
5. čas, za který je třeba aplikaci vyvinout,
6. ecosystém technologického zásobníku – jak dobře jednotlivé technologie spolupracují,
7. kvalifikace vývojářů, kteří budou na projektu pracovat.

Mezi další faktory můžeme zařadit i to, co využívají konkurenti pro vývoj podobných aplikací nebo například i to jaké plány jsou s projektem do budoucna (jestli se bude k webové aplikaci pojit i mobilní aplikace atd.).

5.1 Technologie pro vývoj klientské strany

Klientská strana aplikace je ta, kterou vidí uživatelé a která tvoří vizuální podobu aplikace. S touto vývojovou vrstvou souvisí jen několik technologií webového zásobníku [67]. Zde se převážně využívají jazyky HTML, CSS, Javascript a frameworky, které jsou na těchto jazycích postaveny.

V minulosti vývojáři pro vývoj frontendu webových aplikací využívali pouze holé jazyky HTML, JavaScript a CSS, ale tento trend již není dostačující. V těchto dnech je pro vývoj

k dispozici mnoho různých frameworků a knihoven, které si kladou za cíl práci urychlit, zjednodušit a zpřehlednit. Frameworky a knihovny, které zde zmiňuji jsou postaveny na jazyku JavaScript, který je jedním z nejpoužívanějších jazyků vůbec. Pro tento jazyk je k dispozici více než 50 frameworků [19]. Mezi nejpopulárnější můžeme zařadit knihovnu React.js nebo frameworky Angular.js a Vue.js.

Všechny tři zmíněné technologie se používají pro vývoj tzv. SPA - single page application [61, 77, 3]. Tyto aplikace se jmenují podle toho, že potřebují načíst pouze jedinou HTML stránku pro používání celé aplikace [25]. Tyto aplikace místo načítání nových HTML souborů překreslují obraz pomocí JavaScriptových operací, které upravují DOM strom. Díky tomu se při každém kliknutí na odkaz nemusí stránka znovu celá obnovovat, ale obsah se pouze překreslí.

5.1.1 Knihovna React

Knihovna React byla vytvořena softwarovým inženýrem ve Facebooku v roce 2011 [15]. Na konferenci JSConf v roce 2013 se knihovna stala open-source řešením a připojila se tak ke spoustě dalším knihovnám pro tvorbu webového uživatelského rozhraní – v té době nejvyužívanější byly knihovny jQuery, Angular, Dojo atd. V této době byl React popisován jako Véčko v modelu MVC nebo-li, React komponenty sloužily jako view vrstva javascriptovým aplikacím. Od té doby se začal React hojně využívat – např. společnost Netflix v roce 2015 oznámila, že pro vývoj uživatelského rozhraní používá právě React.

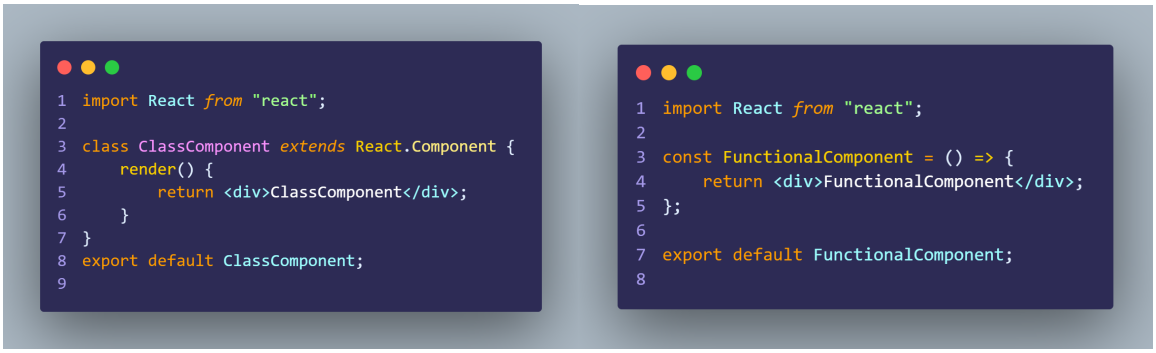
Z této knihovny se časem vyvinuly i další nástroje. Mezi ně můžeme zařadit např. React Native, což je knihovna pro tvorbu mobilních aplikací pomocí knihovny React. Facebook dále vyvinul ReactVR – knihovnu pro tvorbu aplikací virtuální reality. V letech 2015 a 2016 se objevilo velké množství nástrojů jako React Router, Redux a Mobx, které řeší úkoly jako směrování a distribuci dat a správu stavů. React není frameworkem, ale knihovnou, protože se zabývá implementací specifické sady funkcí, nikoliv poskytováním nástroje pro každý případ užití.

Jedním z hlavních důvodů, proč se stala knihovna tak populární je kvůli její jednoduchosti [33]. Její jednoduchost spočívá hlavně v rozšíření jazyka Javascript pomocí JSX (zkratka pro Javascript XML), podobně jako rozšíření XHP pro jazyk PHP. JSX dovoluje psát jazyk HTML přímo v JavaScriptovém kódu a tedy přímo při specifikaci komponenty je jeho reprezentace jednoduše čitelná pro každého, kdo zná jazyk HTML.

Hlavním prvkem knihovny React jsou znovupoužitelné komponenty. Tyto komponenty mají dva způsoby, jak se dají specifikovat [55] – příklad v kódu je na obrázku 5.1:

1. **Třídní komponenty** – díky syntaxi JavaScript ES6 je možné definovat nové komponenty jako třídy, které dědí od třídy `React.Component`. Tyto třídy pak následně specifikují metodu `render`, která určuje, jak se tento komponent vykreslí.

2. **Funkční komponenty** – v tomto případě je komponenta funkce, která navrací JSX strukturu. Oficiální dokumentace knihovny React tento typ komponent upřednostňuje [56] hned z několika důvodů. Funkční komponenty jsou jednodušší na porozumění než třídní [57]. Jsou méně náročné na výpočet, protože se nemusí při každém vykreslování komponentu tvořit nová instance třídy. Funkční komponenty nemají ani žádné speciální metody, které se týkají životní cyklu instance. Ve funkčních komponentách je od verze `16.8` možné využívat tzv. React Hooks (ty jsou popsány dále v této kapitole).



```jsx
import React from "react";

class ClassComponent extends React.Component {
    render() {
        return <div>ClassComponent</div>;
    }
}
export default ClassComponent;
```

```jsx
import React from "react";

const FunctionalComponent = () => {
    return <div>FunctionalComponent</div>;
};

export default FunctionalComponent;
```

Obrázek 5.1: Vlevo je příklad třídní komponenty a vpravo příklad funkční komponenty. Oba tyto příklady produkují stejný výsledek.

Aby mohli být stránky dynamické a mohly se měnit i bez obnovení celé stránky, používá React princip, který se nazývá stav (angl. state) [58]. Další funkcí těchto stavů je také schopnost jednotlivých komponent si svůj vnitřní stav zapamatovat i v průběhu několika překreslení. Např. mějme formulář s několika textovými poli pro vyplnění. Každé toto textové pole se při každém zadání znaku musí překreslit – upravit hodnotu, která je uložena v tomto poli. Kdyby tento formulář byl React komponentou, tak nejpravděpodobněji je implementována pomocí stavů, kdy každé textové pole má připsáno stav, který se v průběhu zadávání mění. Alternativně může mít připsaný formulář stav jen jeden – v tomto případě by šlo o objekt, který by jako atributy měl hodnoty uchované v textových polích.

Stavy se vytváří několika způsoby. Záleží totiž na tom jestli se jedná o třídní nebo funkční komponentu. Každá třída, která dědí od třídy `React.Component` má totiž již definovaný atribut `state`. Pod tímto atributem je většinou uložen objekt, který má několik dalších atributů reprezentující stavy. Pro změnu stavu třídní komponenty je k dispozici metoda `setState`, pomocí které se dají změnit jednotlivé atributy objektu, který je uložen v atributu state. Ve funkčních komponentách se využívá hooku useState. Tato funkce vrací pole o dvou prvcích. Prvním z nich je uložená hodnota a druhým z nich je funkce, pomocí které se stav dá měnit.

```jsx
import React from "react";

class ClassComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = { counter: 0 };
    }
    handleClick = () => {
        this.setState({ counter: this.state.counter + 1 });
    };
    render() {
        return (
            <div>
                <h1>Počítadlo {this.state.counter}</h1>
                <button onClick={this.handleClick}>Increment</button>
            </div>
        );
    }
}
export default ClassComponent;
```

```jsx
import React, { useState } from "react";

const FunctionalComponent = () => {
    const [counter, setCounter] = useState(0);

    const handleClick = () => {
        setCounter(counter + 1);
    };
    return (
        <div>
            <h1>Počítadlo {counter}</h1>
            <button onClick={handleClick}>Increment</button>
        </div>
    );
};

export default FunctionalComponent;
```

Obrázek 5.2: Rozdíl při práci se stavem komponenty při použití třídní (vlevo) a funkční (vpravo) komponenty. Oba kódy produkují stejný výsledek a tedy stránku, kde je tlačítko, které po kliknutí způsobí překreslení části komponenty. Zde je stav jednoduché počítadlo, které se inkrementuje při každém kliknutí.

Dalším velmi využívaným prostředkem Reactu jsou efekty (angl. effects). Dle dokumentace [59] se efekty používají pokud je potřeba nějakým způsobem vystoupit z klasického React kódu. Příkladem může být např. připojení k serveru, volání API atd. Komponenty bez těchto efektů jsou prakticky matematické funkce – přijímají nějaké argumenty a vrací výsledek – pokud jsou argumenty stejné, produkují stejný výsledek. Jakmile se ale do komponent přidají efekty, tak toto již nemusí platit. Samotnou aplikaci totiž mohou ovlivňovat nějaké vnější prostředky – např. API. Mějme komponentu, která zobrazuje informace o uživateli. Tato komponenta nepřijímá žádné argumenty a tedy pokaždé by měla produkovat stejný výsledek, ale kvůli použití efektů tomu tak není. Při prvním vykreslení se totiž může volat nějaké API, které navrátí hodnoty o právě přihlášeném uživateli. Ten se liší podle toho, kdo je právě přihlášený, tedy komponenta vrátí různé výsledky. Příklad takové komponenty s použitím efektů můžete vidět na obrázku 5.3.

```jsx
import React, { useEffect, useState } from "react";

const FunctionalComponent = () => {
    const [currentUser, setCurrentUser] = useState(null);

    useEffect(() => {
        fetch("http:/localhost:8000/api/user-info")
            .then((response) => response.json())
            .then((data) => {
                setCurrentUser(data);
            });
    }, []);
    return (
        <div>
            {currentUser && (
                <>
                    <h1>Jméno: {currentUser.username}</h1>
                    <p>Adresa: {currentUser.adress}</p>
                </>
            )}
        </div>
    );
};

export default FunctionalComponent;
```

Obrázek 5.3: Ukázka použití hooku useEffect, který při prvním vykreslení komponenty získá data z externího API a následně pomocí změny stavu způsobí překreslení komponenty a zobrazí tak jméno a adresu uživatele.

Funkce useEffect je dalším z React hooků. Tato funkce přijímá dva argumenty. Prvním argumentem je funkce, která se provádí pokud je efekt spuštěn. Tato funkce může také navracet tzv. čistící funkci. Ta se používá, pokud např. v useEffect deklarujeme nějaký časovač, který se periodicky opakuje. Tato navrácená funkce slouží pro zastavení opakování tohoto časovače a spustí se v momentě, kdy je komponenta odstraněna z vykreslovacího stromu. Druhým argumentem funkce useEffect jsou tzv. závislosti efektu. V praxi se jedná o pole proměnných. Pokud se nějaká z těchto proměnných změní je efekt spuštěn. Pokud

je pole prázdné, tak se efekt spustí pouze při prvním vykreslení komponenty. Pokud by se tento argument vynechal, tak se efekt bude volat při každém vykreslení komponenty.

Jak už bylo zmíněno `useEffect` a `useState` jsou tzv. React hooky. Tyto hooky jsou funkce, které dovolují přímo v komponentách využít jinou funkčnost – použití stavu nebo efektu [60]. Hooky se dají použít pouze ve funkčních komponentách – tedy i proto React sám doporučuje používat převážně funkční komponenty. Následující pravidla musí být splněna aby hooky fungovaly:

- Hook může být volán pouze ve vrchní vrstvě kódu – nesmí být volán ve smyčkách ani v podmíněných výrazech.
- Hook je možno volat pouze z React komponent. Jediná výjimka je, pokud programátor vytvoří vlastní hook – odtud lze volat jiné hooky.

Virtuální DOM

Když se podíváme, jak jednotlivé knihovny/frameworky vykreslují výsledný HTML kód, setkáme se se dvěma různými technologiemi: inkrementální DOM a virtuální DOM [19]. Inkrementální DOM je popsán v kapitole 5.1.3.

Celá tato podkapitola vychází z oficiální dokumentace knihovny React [53, 54].

Virtuální DOM je struktura v paměti na klientské straně, která uchovává ideální/virtuální reprezentaci uživatelského rozhraní. Tento virtuální strom je poté synchronizován se skutečným stromem DOM, který je využit pro vykreslení webové aplikace. Tento proces synchronizace se nazývá rekonciliace (angl. reconciliation). Tento přístup umožňuje Reactu jeho deklarativní rozhraní, tedy programátor specifikuje v jakém stavu chce, aby jeho vytvářený objekt byl a knihovna se postará o to, aby reálný DOM tento stav odrážel. Součástí virtuálního DOMu jsou také objekty zvané vlákna (angl. fibers), které uchovávají další informace o stromu komponent.

Při vykreslení jednotlivých komponent je použita funkce render, ať už se jedná o komponenty funkční nebo třídní. Funkce render prakticky navrací strom React elementů, které je třeba vykreslit. Při změně podrobností, které vykreslená komponenta přijímá, nebo při změně vnitřního stavu komponenty je funkce render znovu zavolána a navrácený strom se od předchozího může lišit. React potřebuje zjistit, jaká část stromu zůstala zachována a která se změnila, aby mohl efektivně změnit DOM.

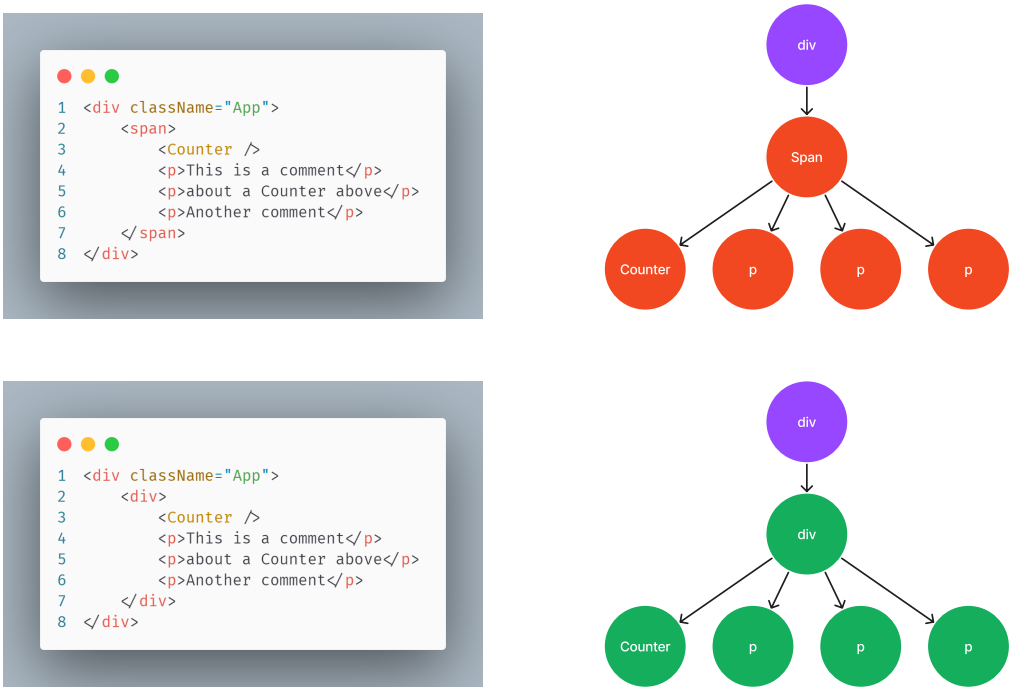
Tento algoritmický problém má několik naivních řešení, které generují minimální posloupnost operací vedoucích k optimální změně stromu. Tyto algoritmy ale příliš vysokou složitost (konkrétně $\mathcal{O}(n^3)$, kde n je počet uzlů stromu). To znamená, že pro zobrazení 1000 elementů by bylo potřeba 1 miliardy porovnání, což je příliš drahé a v praxi nepoužitelné. React implementuje algoritmus s jednoduchou heuristikou, který dává za výsledek algoritmus s lineární časovou složitostí. Heuristika má pouze dva body:

1. Dva elementy různých typů produkují jiný strom.
2. Vývojář může napovídat, které vnořené elementy mohou zůstat beze změny napříč různými vykresleními pomocí atributu key.

React si neuchovává virtuální strom pouze jeden, ale hned dva [33]. První je virtuální strom, který odráží aktuálně zobrazovaný DOM strom a druhý je ten, na kterém se promítají změny např. při změně stavu. Pro spojení těchto dvou stromů se používá tzv. Diffing algoritmus. Po aplikování tohoto algoritmu tedy vznikne výsledný virtuální DOM, který se následně synchronizuje se skutečným DOMem.

Dokumentace Reactu přesně nespecifikuje, jak tento algoritmus funguje, ale zmiňuje několik bodů, které algoritmus řeší [53].

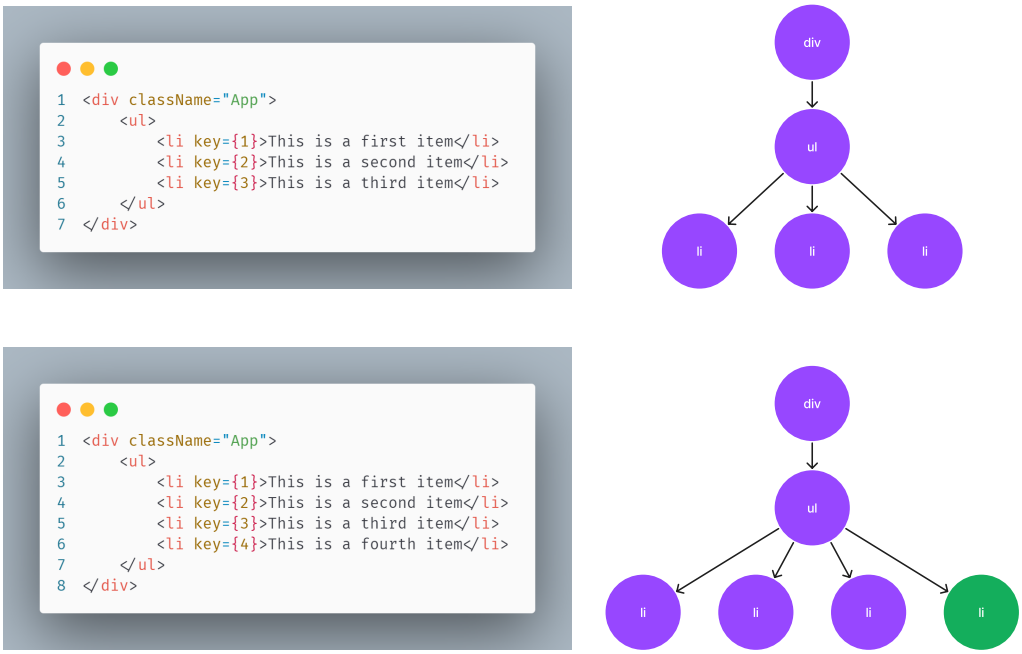
- **Elementy jiného typu** – Pokud se změní typ otcovského uzlu, React kompletně zničí strom, který pod ním byl (ilustrace takové změny je ukázána na obr. 5.4.
- **Elementy stejného typu** - Pokud se spojují dva stromy a rodičovské uzly mají stejné, musí se ještě zkontrolovat atributy daného uzlu – pokud se změnili, tak React změní pouze ty a uzel nijak jinak nemění.
- **Synovské uzly** – Při iterování přes synovské uzly React pouze prochází obě verze (starou a aktualizovanou) zároveň. Pokud narazí na nějakou nesrovnalost (jako např. na obrázku 5.5), tak ji vyřeší přidáním nebo odebraním. Pokud by React procházel seznamy postupně, bylo by více náročné, když by v nové verzi stromu byl přidán prvek na začátek seznamu než, kdyby byl přidán nakonec. To React řeší pomocí key atributu. Poté tedy neporovnává synovské uzly na základě pozice, ale na základě hodnoty klíče.



Obrázek 5.4: Vizualizace virtuálního DOM stromu, kde lze vidět, že při změně rodičovského uzlu se musí kompletně zničit všechny červené uzly (vrchní polovina) a vytvořit se znovu (spodní polovina).

5.1.2 Framework Vue.js

První verze frameworku Vue.js byla implementována Evanem Youem, který v té době pracoval v Google Creative Labs [24]. Potřebovali nástroj pro rychlou tvorbu prototypů spíše než



Obrázek 5.5: Při tvorbě např. neseřazených seznamů React pouze přidává prvky k rodičovským uzlům při vykreslení nově vzniklých. Atribut key pomáhá rozeznat prvky na základě klíče, aby je nemusel algoritmus procházet sekvenčně. Typicky se, ale takto seznamy nevykreslují – většinou se tvoří ve smyčce, která generuje vždy jeden prvek ze seznamu s unikátní hodnotou klíče.

velkých uživatelských rozhraní. V době, kdy tento framework vznikl, byl velmi používaným frameworkem Angular a React.js se teprve začínal rozvíjet a frameworky jako Backbone.js se používali pro velké aplikace s architekturou MVC. Ani jeden z těchto frameworků tedy nebyl dobrou volbou pro rychlé prototypování UI.

Vue.js byl inspirován spousty dobrých vlastností ostatních frameworků jako Angular, Polymer[1] nebo React.js [13]. Z těchto frameworků se inspiroval především těmito vlastnostmi:

- Reaktivní datový systém, který dokáže automaticky aktualizovat uživatelské rozhraní s lehkým enginem založeným na virtuálním DOM. Není tedy třeba žádné velké úsilí pro optimalizaci rychlosti vykreslování.
- Flexibilní deklarování zobrazení pomocí HTML šablon nebo JSX.
- Uživatelské rozhraní založeno na skládání znovupoužitelných komponent.
- Oficiální dodatečné knihovny pro směrování a management stavů jednotlivých komponent.

[1]Polymer je JavaScriptová knihovna vytvořená firmou Google také založena na tvorbě komponent [50].

První publikovaná verze byla vydána v roce 2014, následovala první velká verze 1.0 v říjnu roku 2015. Do konce roku 2015 tato knihovna zaregistrovala 382 tisíc stažení pomocí správce javascriptových knihoven npm [13]. Poslední velkou verzí současnosti je verze 3.0, která byla vydána v roce 2020 [70].

Podle [24] jsou nejdůležitějšími součástmi frameworku následující:

- Vue.js dovoluje jednoduše provázat datové modely s reprezentační vrstvou a také dovoluje jednoduché znovupoužití komponent skrze aplikace.
- Není zapotřebí speciálních modelů, kolekcí nebo registrování událostí objektů. Není zapotřebí žádný speciální syntax ani nikdy nekončící instalace dodatečných závislosti.
- Modely jsou ve Vue reprezentovány pomocí normálních JavaScriptových objektů a mohou být navázány na cokoliv v reprezentační vrstvě – text, input (vstupní pole), třídy, atributy atd.
- První použití frameworku je velice jednoduché, protože existují nástroje, které ulehčují vývoj. S použitím např. `vue-cli`, `Webpack` a `Browserify` je jednoduché vytvořit začáteční šablonu aplikace. Nástroje také zpřístupňují možnost tzv. hot-reloadingu, což znamená, že při změně kódu se tato změna ihned projeví v aplikaci.
- Ve frameworku je možné kompletně separovat View vrstvu od stylů a Javascriptové logiky, ale je jen na programátorovi jestli toho využije.

Framework byl sice ze začátku vytvořen jako nástroj pro vytváření rychlých prototypů, ale postupným vývojem se dostal do fáze, kdy se v něm programují komplexní aplikace [24] a je používán mnoha velkými firmami, jako např. Microsoft, Adobe, Alibaba, Xiaomi a jiné [13].

Programování ve Vue je odlišné od toho v Reactu tím, že jednotlivé komponenty jsou popsaný v souborech s příponou `.vue`. V tomto souboru většinou nalezneme tři části, které specifikují určitou komponentu. Je zde část pro javascriptový kód, šablonu napsanou v jazyce HTML a část pro CSS stylování komponenty.Rozložení tohoto souboru je zobrazeno na obrázku 5.6.

Podobně jako v knihovně React mají programátoři na výběr mezi třídními a funkčními komponentami i ve Vue jsou dva způsoby, jak komponenty reprezentovat. Tyto dva způsoby se nazývají Options a Composition [72].

- Options – při tomto způsobu se logika komponenty definuje pomocí objektu nastavení. Tento objekt poté má speciální metody jako: `data()`, `mounted()`, nebo pomocí speciálních atributů jako např. atribut `methods`.
- Composition – při tomto způsobu se logika komponenty definuje pomocí předem připravených API funkcí. Takto definovaná logika komponenty je uzavřená do značky `script` s atributem `setup`. Tento atribut slouží jako nápověda pro Vue, aby přetransformoval kód tak, aby např. proměnné a funkce na nejvyšší úrovni byly dostupné v šablonové části. Tento způsob programování je doporučován, protože vede k čistějšímu kódu.

Na obrázku 5.7 je také vidět práce s reaktivními stavy v aplikaci. Opět se tu liší způsob implementace podle toho, jakým způsobem jsou komponenty naprogramovány. Pokud je komponenta naprogramována způsobem Options je stav reprezentován v atributech objektu s nastaveními. Proto je zapotřebí při psaní metod, které pracují se stavy, používat

```vue
<script>
  export default {
    data() {
      return {
        hello_text: 'Hello, World!'
      }
    }
  }
</script>

<template>
  <h1>{{ hello_text }}</h1>
</template>

<style>
  h1 {
    font-size: 3rem;
    background-color: red;
    color: white;
  }
</style>
```

Obrázek 5.6: Příklad souboru `.vue`, který obsahuje tři části komponenty. Mezi značkami `script` je javascriptový kód, který specifikuje data, která budou dostupná v šabloně zapsané mezi značkami `template`. Mezi značkami `style` je pak zapsán CSS kód, který specifikuje vzhled komponenty.

klíčové slovo `this` a za tečkou specifikovat jméno stavu. Při použití způsobu Composition je zapotřebí použít funkci `ref()`, která navrací reaktivní a měnitelný objekt. Pomocí parametru této funkce se specifikuje počáteční hodnota stavu. Pro změnu stavu je potřeba použít atribut `value` tohoto objektu.

Jedním z hlavních benefitů Options API je jeho jednoduchost a snadná srozumitelnost [76]. Dále také jasný, deklarativní způsob programování, který je mnoha vývojářům známý. To z něj činí dobrý výběr pro začátečníky, kteří teprve začínají pracovat s Vue. Avšak Options API má některá omezení, která mohou znemožnit jeho použití pro složitější projekty. Tím jak roste složitost komponenty, tím se rozrůstá kód a tím pádem je těžké mít přehled nad celou komponentou, jelikož části kódu, které se týkají stejné funkcionality jsou na různých místech. To může vést k jevu známému jako „exploze možností“, kdy se komponenta stává tak rozsáhlou a obtížnou na údržbu, že se stává neudržitelnou. Další výhodou Compositon API je efektivní kompilace, jelikož kompilovaný „minified“ kód je kratší. Malá nevýhoda Composition API je ta, že byla vyvinuta později s čímž se váže nekompatibilita s verzemi Vue 2.6 a níže. Kvůli větším výhodám Composition API je dále popsán pouze tento přístup.

Mezi další důležité techniky Vue patří tzv. Lifecycle metody – metody životního cyklu. Jsou to metody, které jsou volány v různých částech životního cyklu komponenty. Každá komponenta totiž prochází několika inicializačními kroky během jejího vytváření – např. potřebuje zprovoznit pozorování objektů, zkompilovat šablonu, namontovat instanci do DOM

```vue
<script>
  export default {
    data() {
      return {
        counter: 0
      }
    },
    methods: {
      increment() {
        this.counter++
      }
    }
  }
</script>

<template>
  <div>
    <button @click="increment">Increment</button>
    <p>Counter: {{ counter }}</p>
  </div>
</template>
```

```vue
<script setup>
  import { ref } from 'vue';

  const counter = ref(0);

  function increment() {
    counter.value++;
  }
</script>

<template>
  <div>
    <button @click="increment">Increment</button>
    <p>Counter: {{ counter }}</p>
  </div>
</template>
```

Obrázek 5.7: Rozdíl mezi komponentou naprogramovanou v Options API (vlevo) a Composition API (vpravo)

.

stromu a také aktualizovat DOM strom při změnách [75]. Funkce jsou znázorněny na obrázku 5.9.

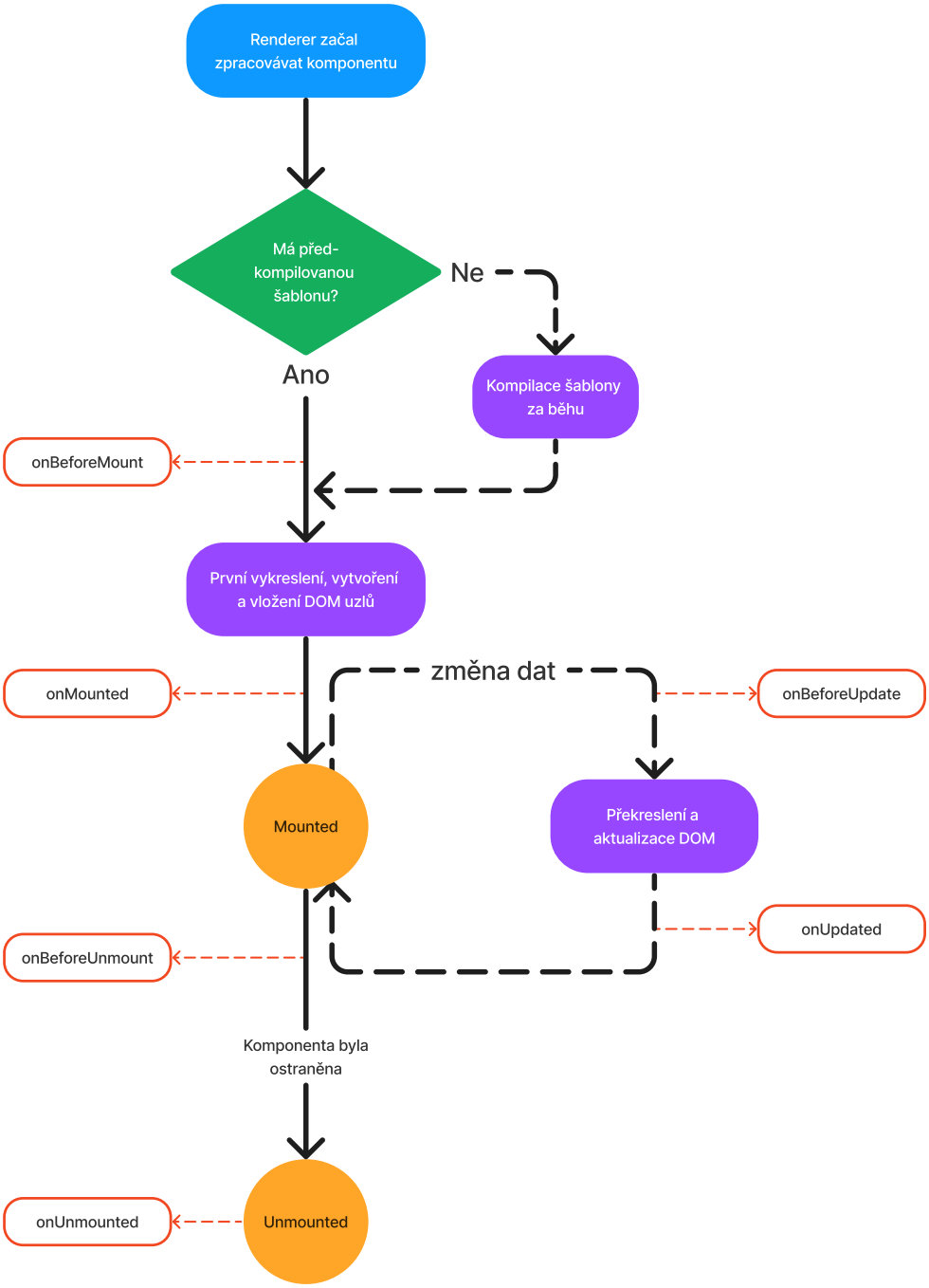
Pro přístup k proměnným se v šablonovacím jazyku používají dvojité složené závorky nebo-li tzv. „double mustache“ závorky [73]. Tento přístup ovšem funguje pouze, pokud hodnotu chceme zobrazit mezi dvěma HTML značkami – v atributech značky se používá styl, kdy napíšeme dvojtečku před jméno atributu. Vue, také podporuje složitější výrazy v dvojitých složených závorkách, jako např. aritmetické operace nad proměnnými a jejich následné zobrazení, výběr hodnoty pomocí ternárního operátoru, volání metod, které navrací zobrazovanou hodnotu atd. V atributeche s dvojtečkou na začátku, je také možné volat funkce – tyto funkce jsou zavolány kdykoliv při aktualizaci komponenty, proto se doporučuje, aby tyto funkce neměly žádné vedlejší efekty – jako např. změna dat, volání asynchronních operací atd.

Vue v šablonovacím jazyku má také speciální direktivy. Jedna již byla zmíněna v předchozím odstavci. Když totiž napíšeme před název atributu dvojtečku, je to zkratka pro direktivu `v-bind` – tato direktiva knihovně Vue říká, že hodnota daného atributu musí být synchronizována s objektem, který je tomuto atributu připsán. Další často používanou direktivou je direktiva `v-for`, pomocí které může programátor jednoduše definovat vykreslení objektů v seznamu nebo v jiném iterovatelném objektu. Direktiva `v-on` připojuje k elementu naslouchač událostí (častěji se využívá zkratka – znak zavináče) jako např. události: click, submit, keyup atd. Další direktivy jsou zkráceně popsány v tabulce 5.1.

Obrázek 5.8: Na obrázku je vidět, jak může být kód roztroušen do různých částí zdrojového souboru. Podbarvené sekce stejnou barvou se týkají jedné funkcionality a jak lze vidět, v Options API je kód v různých částech souboru, kdežto v Composition API je vše, co se týká jedné funkcionality na jednom místě. Obrázek je převzat z [76].

Název direktivy	**Význam direktivy**
v-text	Hodnota direktivy přepíše vnitřní textovou hodnotu značky.
v-html	Hodnota direktivy přepíše vnitřní HTML hodnotu značky.
v-show	Přepíná mezi zobrazením a skrytím objektu.
v-if	Pokud je hodnota direktivy pravdivá, zobrazí element.
v-else-if	Používá se pro ztřetězení s předchozí direktivou.
v-else	Jestliže žádná z direktiv v-if, v-else-if není pravdivá, zobrazí se komponenta s touto direktivou.
v-model	Vytvoří obousměrnou vazbu mezi input elementem a komponentou.
v-slot	Slouží pro pojmenování tzv. slotů, což je ve Vue způsob, jak může rodičovská komponenta předat informace potomkovi.
v-pre	Direktiva, která specifikuje, že daná komponenta se nemá kompilovat.
v-once	Specifikuje, že se element/komponenta má vykreslit pouze jednou a neaktualizovat se dál.
v-memo	Očekává seznam hodnot, které se kontrolují při znovuvykreslování a jejich změna zapříčiní překreslení komponenty.

Tabulka 5.1: Vybrané vestavěné direktivy knihovny Vue [74]



Obrázek 5.9: Diagram znázorňující různé fáze, kterými prochází komponenta knihovny Vue během vytváření, aktualizování a zničení. Obrázek byl převzat z [75].

Vykreslování ve Vue.js

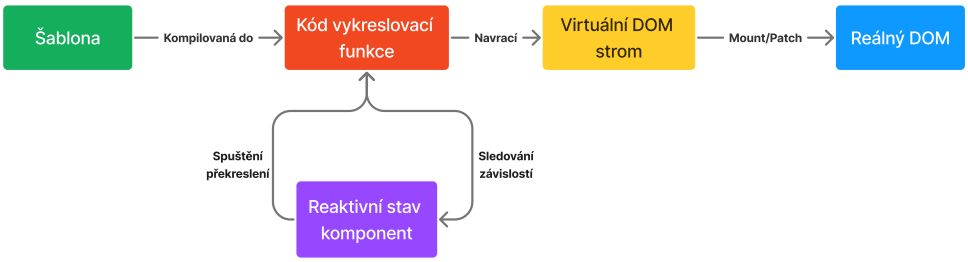
Vykreslování ve Vue.js funguje, podobně jako v Reactu, na principu Virtuálního DOMu [71]. Runtime renderer prochází virtuálním DOM stromem a vytváří z něj skutečný DOM strom. Tento proces se nazývá mount. Pokud máme dvě kopie virtuálního DOM stromu, renderer umí porovnávat tyto dva stromy, při tom porovnává odlišnosti v těchto stromech a aplikuje změny na skutečném DOM stromě. Tento proces se nazývá patch, někdy též známý jako diffing nebo rekonciliace.

Vykreslovací řetězec Vue tedy rozděluje do několika částí:

1. Kompilace – šablony ve Vue jsou kompilovány do vykreslovacích funkcí: funkce, které navrací virtuální DOM stromy. Tento krok je možné udělat předem (build) nebo při běhu pomocí runtime kompilátoru.

2. Mount – runtime renderer invokuje vykreslovací funkce, prochází virtuální DOM strom a vytváří skutečný DOM strom. Tento krok je prováděn jako reaktivní efekt, takže udržuje přehled o všech reaktivních závislostech, které jsou v jednotlivých uzlech použity.

3. Patch – Pokud se změní nějaká ze závislostí z procesu mount, tak se tento proces provádí znovu a vytvoří nový virtuální DOM strom. Runtime renderer poté porovnává jednotlivé změny ve virtuálních stromech a následně aktualizuje skutečný DOM strom.



Obrázek 5.10: Graficky znázorněný vykreslovací řetězec frameworku Vue.js [71].

5.1.3 Framework Angular

První verze frameworku Angular vytvořil zaměstnanec Googlu Miško Hevery. První verze frameworku se jmenovala AngularJS [9]. Podobně jako předchozí zmíněné technologie, AngularJS vznikl aby ulehčil a urychlil repetetivní práci, která při programování webových aplikací často vznikala. Další podobnost s např. knihovnou React je ta, že AngularJS vznikl interně ve firmě Google a až později byl otevřen veřejnosti [27]. Tato verze, ale neobstála konkurenci hlavně v podobě Reactu a proto Google v roce 2016 vydal úplně novou, od jádra přepracovanou verzi, kterou nazval Angular 2 – dnes už pouze Angular. Po verzi Angular 2 přišla verze Angular 4. Dále se již verze zvyšovali a poslední je verze Angular 17.

Angular je framework napsaný v jazyce Typescript [7]. Pomocí Angularu se programuje prakticky jen v Typescriptu. Kvůli tomu, že se Typescript ale stejně překládá do JavaScriptu je možné programovat i v něm. Používání Typescriptu namísto Javascriptu má hned několik výhod. Podle [10] Typescriptové aplikace mají méně zapáchajícího kódu (code smells), což znamená, že kód je celkově kvalitnější. Dále v této studii prokázali, že používání Typescriptu zlepšuje čitelnost kódu. Podle této studie ale používání Typescriptu nemá žádný větší vliv na počet chyb, které se v aplikaci nacházejí.

```
1  import { Component } from '@angular/core'
2
3  @Component({
4      selector: 'book',
5      template: `<h2>{{ title }}</h2>`,
6      style: ['h1 { font-size: 24px; }']
7  })
8  export class BookComponent{
9      title="Book tittle"
10 }
```

Obrázek 5.11: Základní komponenta ve frameworku Angular, která zobrazuje statický obsah, který je specifikován v atributu komponenty. Tato komponenta je vykreslena pokaždé, kdy se v nějaké šabloně vyskytne html značka s názvem book, která je specifikována pomocí atributu selector v dekorátoru třídy.

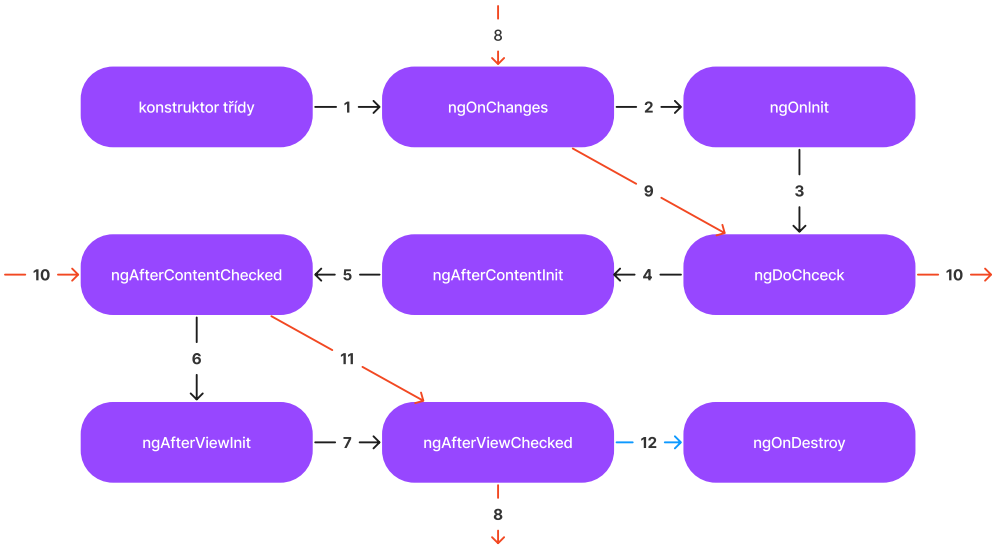
Stejně jako knihovna React nebo framework Vue je i Angular postaven na znovupoužitelných komponentách. Komponenta se skládá z [2]:

- HTML šablony, která deklaruje, co se vykreslí na stránce,
- TypeScriptové třídy, která definuje chování komponenty,
- CSS selektor, který definuje, jak se dá komponenta v šabloně použít,
- a volitelně se můžou specifikovat i CSS styly pro danou komponentu.

Pro definici komponenty je zapotřebí dekorátor zvaný `Component` [7]. Pomocí tohoto dekorátoru je celá komponenta konfigurována – zde se specifikuje jaký HTML element komponenta vytváří, jaká bude jeho vnitřní struktura a také jak bude tento prvek stylován [2]. Parametrem pro tento dekorátor je objekt, kterému se specifikují atributy. Atribut `selector` specifikuje značku, kterou tato komponenta nahradí – je to pojmenování nové značky, na kterou když kompilátor narazí v šablonách ví, kterou komponentu za ní dosadit.

Dalším argumentem je argument `template` nebo `templateUrl`, který specifikuje šablonu komponenty – buďto přímo v TypeScriptovém souboru nebo je zde odkaz na soubor HTML. Dalším argumentem je stylování pomocí argumentů `styles` nebo `styleUrls`, kde se podobně jako v případě šablony mohou přímo definovat CSS styly dané komponenty nebo se zde specifikují odkazy na CSS soubory. Samotná logika komponenty se pak specifikuje v samostatné třídě, kterou dekorátor dekoruje.

Komponenty v tomto frameworku mají také svůj životní cyklus a metody, které se cyklu týkají. Angular definuje několik rozhraní, pomocí kterých, může programátor implementovat vlastní logiku v různých částech životního cyklu komponenty. Mezi základní řadí Angular tyto: `OnInit`, `OnDestroy`, `OnChanges`, `AfterViewInit` [7]. Tyto rozhraní vždy specifikují, kterou metodu musí třída, implementující toto rozhraní, definovat. Všechny rozhraní vždy definují jednu metodu, která vždy začíná řetězcem `ng` a následuje jméno rozhraní. Následnost volání jednotlivých metod životního cyklu je ukázána na obrázku 5.12.



Obrázek 5.12: Graf zobrazuje, jak se jednotlivé metody životního cyklu za sebou volají ve frameworku Angular. Při prvním zpracování komponenty jsou volány metody, které jsou na cestě označeny černými šipkami. Při detekování změny, která vede k překreslení komponenty je následována cesta označená červenými šipkami. Obrázek je převzat z [80].

Jako šablonovací jazyk Angular používá HTML, který je obohacený o pár funkcionalit[7]. Stejně jako React či Vue, Angular používá dvojité složené závorky, nebo-li tzv. „double mustache“ pro vkládání hodnot proměnných komponent do šablon. Pokud by programátor chtěl definovat nějaký standardní HTML atribut pomocí hodnoty proměnné, musí tento atribut ohraničit hranatými závorkami a jako hodnotu specifikovat jméno proměnné komponenty, pro kterou je šablona definována. Pro definování naslouchačů událostí používá Angular jednoduché závorky. Pomocí nich tedy můžeme definovat, co se má stát když uživatel klikne na tlačítku, zapíše nějakou hodnotu do vstupního pole atd. V šabloně je tedy například událost kliknutí specifikována atributem `(click)`.

S použitím komponent se pojí i jejich vnořování. Mezi komponentami může být umožněna komunikace buď jedním nebo oběma směry – od rodičovské komponenty k potomkovi a naopak. Angular podporuje oboustranné datové propojení – to znamená, že přímo podporuje jak komunikaci od rodičovských komponent k těm synovským tak i naopak [7]. Komunikace od rodiče k potomkovi probíhá pomocí tzv. input binding (vstupní propojení). Pomocí něj může rodič svému potomkovi předávat data. Potomek může rodičovské komponentě posílat data pomocí tzv. output binding (výstupní propojení). To funguje na principu vytváření událostí – potomek vytvoří událost a rodič na tuto událost má vytvořený naslouchač.

Inkrementální DOM strom

V kapitole 5.1.1 byl popsán jeden způsob fungování vykreslovacích řetězců. Angular používá odlišný přístup. Hlavním nedostatkem používání Virtuálního DOM stromu je jeho paměťová náročnost. Při aktualizaci DOM stromu se totiž vždy vytváří kopie existujícího stromu. To se děje kvůli tomu, že tento přístup se snaží různé změny provádět po dávkách – tedy ne každá změna vede k vytvoření nového virtuálního DOM stromu, ale vždy se provádí nějaká dávka změn. Kvůli vytváření nových virtuálních DOM stromů je tento přístup paměťově náročný [4]. Inkrementální přístup si klade za cíl redukovat využité množství paměti [31].

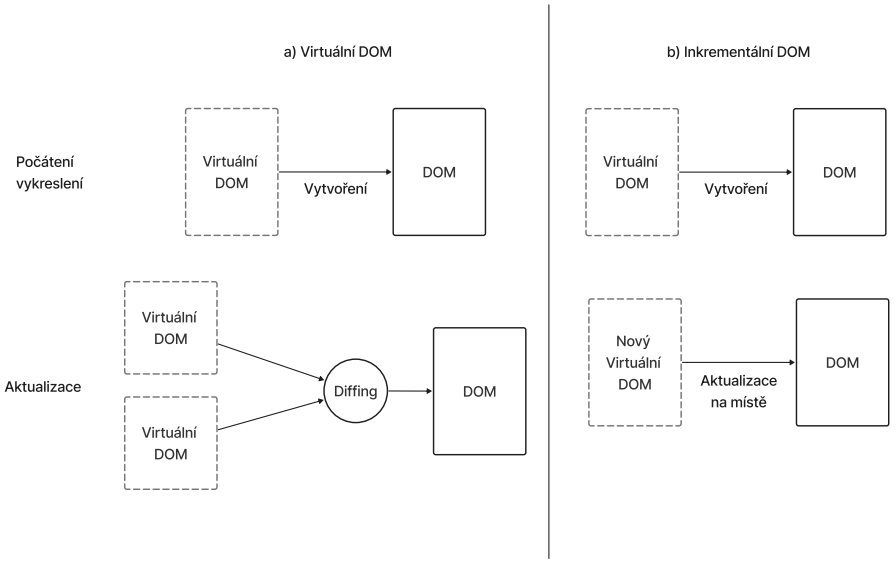
Virtuální DOM strom byl popularizován díky knihovně React firmou Facebook. Inkrementální DOM strom byl vyvinut firmou Google a využívá přístup virtuálního stromu, ale vynechává jeden krok algoritmu nebo spíše jej modifikuje [4].

Při používání čistého virtuálního DOMu je následující postup:

1. Vykresli kompletně nový virtuální DOM strom.
2. Při změnách porovnej nový virtuální DOM strom s poslední známou verzí a změny ve virtuálním DOM promítni do reálného DOM stromu.

Inkrementální DOM strom tento přístup upravuje následujícím způsobem [31].

1. Vykresli kompletně nový virtuální DOM strom.
2. Během vytváření nového DOM stromu, procházej existující strom a zjisti změny během průchodu. Nealokuj žádnou paměť pokud jsi nenarazil na změnu, pokud zde nějaká změna je, změň již existující strom a aplikuj tyto změny i na reálný DOM strom.



Obrázek 5.13: Porovnání tvorby nového stromu a aktualizaci pomocí přístupů virtuálního stromu a inkrementálního stromu. Při prvním vykreslení je třeba vytvořit v obou případech nový virtuální DOM strom celý. Při aktualizaci se ovšem přístupy liší. Při použití čistého Virtuálního DOMu je třeba vytvořit novou verzi virtuálního DOMu a následně tuto novou verzi pomocí Diffing algoritmu (ten je popsán v kapitole 5.1.1) spojit s tou starší. Při použití inkrementálního přístupu není třeba vytvářet nový strom a aktualizace se projeví na místě. Obrázek je převzat z [4].

5.1.4 Porovnání zmíněných technologií

Různé studie porovnávají dříve zmíněné frontendové frameworky na základě spousty různých faktorů. Např. [47] porovnává framework Vue a knihovnu React na základě způsobu překreslení, optimalizace, použitého šablonovacího jazyka, managementu stavů komponent, vývojových prostředí a dalších. Autoři studie také implementovali stejnou aplikaci pomocí obou frameworků – kalkulačku. Na základě této aplikace porovnávali rychlost jednotlivých frameworků pomocí výkonnostní analýzy v prohlížeči Google Chrome. V této studii se projevila jako rychlejší knihovna React.

Další studie [20] porovnává všechny tři dříve zmíněné frameworky a zároveň je porovnává i s čistým JavaScriptem. Studie porovnávala frameworky ve třech různých kategoriích. Prvně byla vytvořena stejná aplikace pomocí Angularu, Reactu, Vue a i pomocí čistého JavaScriptu. Poté se porovnávala výsledná velikost funkčního překompilovaného kódu, také tzv. čas do interaktivity a manipulační čas s DOM. Co se týče velikosti zkompilovaného kódu frameworků je na tom nejhůře Angular se 196 kB, poté React se 135 kB a nejlépe z nich je na tom Vue se 126 kB. Pro porovnání čistý Javascript má pouze 8 kB. Čas do interaktivity je čas, kdy stránka začne akceptovat interakce po manipulaci s DOM. V této kategorií testů byl kupodivu nejrychlejší React s 0.3 sekundami, poté bylo Vue a čistý JavaScript s 0.4 s a poslední Angular s 0.6 s. Poslední testování prováděli na 4 různých DOM operacích: vytvoření, změna jednoho, změna všech a smazání všech. To postupně testovali na 1000, 10000 a 50000 elementech. Výsledkem je, že inkrementální DOM frameworku Angular je nejlepší při změnách za běhu. Naopak při vytváření DOMu se ukázal přístup Vue jako nejefektivnější.

Ve studii [47] kromě analýzy odlišností mezi knihovnou React a Angular prováděli i analýzu oblíbenosti a výkonnosti. Výkonnost měřili na aplikaci, která byla implementována oběma technologiemi. Výkon technologií pak ověřovali pomocí nástroje: Google Lighthouse, kde knihovna React projevila lepší výsledky než Angular. V této studii také porovnávali učící křivku a oblíbenost technologií. V obou kategoriích také vyšla lépe knihovna React.

Pro novou aplikaci v této diplomové práci byl zvolen framework Vue, kvůli jeho oblíbenosti dobré učící křivce a také kvůli dobré struktuře výsledného kódu a možnosti oboustranného datového provázání.

5.2 Technologie pro vývoj backendu

Pro vývoj serverové strany aplikace je volba konkretní technologie mnohem složitější než pro tu klientskou [30]. Vývoj backandové části zahrnuje volbu:

- operačního systému serveru,
- webového serveru,
- databáze/databází,
- backendového programovacího jazyka,
- a backendového frameworku pro rychlejší vývoj.

5.2.1 Jazyky pro vývoj backendu

Mezi nejpoužívanější backendové jazyky v této době řadíme jazyky C#, Java, Python, PHP, JavaScript [67] nebo např. Ruby, Golang a Rust [5].

JavaScript je již 11 let nejpoužívanějším programovacím jazykem [65]. I když je JavaScript hlavně používán na klientské straně webu, používá se i jako serverový jazyk. JavaScript je dynamicky typovaný, objektově orientovaný a interpretovaný jazyk [17]. Dále podporuje asynchronní zpracování a je nezávislý na operačním systému, jelikož jej dokáže interpretovat každý webový prohlížeč. Nevýhodou je ale, že některé webové prohlížeče nepodporují aktuální verzi tohoto jazyka. To znamená, že nějaký kód může v jednom prohlížeči fungovat správně a ve druhém zase nemusí fungovat vůbec.

Python se v žebříčku oblíbenosti umístil na třetím místě [65]. Jednou z největších předností tohoto jazyka je jeho jednoduchost. Ta je zapříčiněna velkou podobností jazyka s normální angličtinou a i proto je Python často vyučován jako první programovací jazyk na mnoha univerzitách [5]. Další předností tohoto jazyka je jeho široká škála použití – od základních skriptů, přes použití v oblasti umělé inteligence až po sofistikované webové aplikace [46]. Jedním z často vytýkaných problémů tohoto programovacího jazyka je jeho rychlost. Jedná se o interpretovaný jazyk, takže je pomalejší než kompilované jazyky jako např. Java. Při pokročilejším webovém vývoji python nabízí i jednoduchou možnost jak naprogramovat asynchronní aplikace. Mezi další přednosti tohoto jazyka se řadí i rychlost vývoje webových aplikací – frameworky, které se při programování webových aplikací používají, velmi usnadňují a zkracují vývoj [34]. Python řadíme také mezi objektově orientované jazyky.

Dalším často používaným jazykem pro vývoj backendu je Java. Java je třídní objektově orientovaný jazyk, který se mimo jiné využívá i pro tvorbu business software nebo mobilních aplikací. Java je známá pro svoji přenositelnost, rychlost a škálovatelnost [46]. Tento jazyk funguje na kterékoliv platformě, která podopruje JVM – java virtual machine. Jednou z negativních stránek tohoto jazyka je jeho složitost. Javovský kód je často složitější a hůře srozumitelný. Také kvůli tomu je učící křivka horší než u předchozích jazyků.

Posledním jazykem, který je zde zmíněn je jazyk C#. Tento jazyk představila firma Microsoft v roce 2000 [5] a i když byl původně navržen pouze pro framework .NET je dnes využíván v mnoha oblastech jako např. herní vývoj. Nejpoužívanější k vývoji backendů je ovšem ASP.NET Core framework. Je to moderní, vysoce výkonný, open-source webový framework, který používá známé návrhové vzory a paradigmata [39]. Stejně jako Java je ale mnohem těžší se tento jazyk/framework naučit (oproti např. Pythonu nebo Javascriptu). Další nevýhodou je, že pokud by programátor nějakým způsobem nechtěl využívat .NET je C# nemožnou volbou [5].

Pro vývoj aplikace v této diplomové práci byl zvolen programovací jazyk Python hned z několika důvodů:

- Za relativně krátkou dobu se pomocí něj dají vytvořit velmi dobře fungující aplikace.
- Jelikož aplikaci nebude využívat obrovské množství uživatelů, nebude jeho slabší výkonnost na obtíž.
- Jedná se o velmi populární jazyk, takže kdyby projekt převzal jiný programátor je tento jazyk nejlepší volbou.
- Dalším určitě důležitým faktorem je to, že sám mám s vývojem v tomto programovacím jazyce velké zkušenosti, což samotný vývoj urychlí ještě více.
- Podporuje jednoduché asynchronní programování.

5.2.2 Frameworky pro tvorbu API v jazyce Python

Jak bylo zmíněno v předchozí kapitole, bude při vývoji API využíván programovací jazyk Python, proto v této kapitole blíže porovnávám vybrané frameworky, které dovolují vyvinout REST API v tomto jazyce. Podle [18] mezi nejoblíbenější frameworky patří Flask, Django, FastAPI a Tornado.

Dříve se backend skládal z databázových modelů, URL adres a views (pohledů), které interagovaly s „frontendovými“ HTML, CSS a Jasvascriptovými šablonami, které ovládali rozložení každé webové stránky [69]. V poslední době se ale spíše používá tzv. „API-first“ přístup. Toto oddělení má několik výhod. První výhodou je nezávislost na frontendových technologiích, které se rychle vyvíjí a mění. Tímto oddělením je prakticky jedno jaká technologie bude výsledek zobrazovat, jelikož bude posílat ty samé požadavky na API. Druhou výhodou je, že tento přístup také umožňuje použití mnoha frontendů napsaných v různých jazycích a frameworcích. Např. Javascript je používán pro webové frontendy, pro Android zařízení se využívá např. programovací jazyk Java, pro iOS aplikace je zase použit programovací jazyk Swift [69]. S tradičním monolitickým přístupem by backendový framework nemohl podporovat tolik typů frontendů. S API-first přístupem, ale mohou všechny tři zmíněné frontendy komunikovat se stejným rozhraním a se stejnou databází. A jelikož z analýzy požadavků vyplynulo, že v budoucnu je zamýšlena i mobilní aplikace, je tento přístup jasnou volbou. Další výhodou je, že tento přístup také umožňuje, aby API bylo využíváno interně, ale i externě [69] – může sloužit jako API pro jednu konkrétní webovou stránku nebo může sloužit jako veřejné API, které bude poskytovat data komukoliv, kdo je potřebuje.

Během plánování, jakou strukturu bude mít API je třeba zvážit následující činnosti [62]:

1. Pochopení případů užití.
2. Vypsat všechny funkce, které bude API poskytovat.
3. Identifikovat všechny platformy, které API bude používat.
4. Plánovat API dlouhodobě, aby jeho podpora a aktualizace byla co nejsnadnější
5. Plánovat verzovací strategii API, aby byla zaručena kontinuální dostupnost API.
6. Naplánovat přístupovou strategii API – autentizace atd.
7. Rozhodnout se jaká dokumentace API bude použita.
8. Pochopit i jak API bude pracovat s jinými druhy médií - např. s videi, obrázky a jinými druhy hypermédií.

Framework Django byl poprvé zpřístupněn v roce 2005 a v této době většina aplikací byla naprogramována pomocí jedné velké monolitické architektury. Tradiční Django bylo postaveno právě na monolitické struktuře s šablonami, ale dnes se spíše využívá pro tvorbu jednoduchých API, které striktně oddělují backend a frontend. Tímto vznikl i framework Django REST, který funguje jako dodatečný plugin k tradičnímu Djangu. Při implementaci pomocí tohoto frameworku je tedy zapotřebí vytvořit aplikaci pomocí obyčejného Djanga a následně doinstalovat dodatečný balík Django REST frameworku. Poté se tento balík přidá do seznam nainstalovaných aplikací, aby o této instalaci Django vědělo a následně je možné používat Django REST framework. Tento framework oproti normálnímu Djangu velmi zjednodušuje přístup k tvorbě API. Pomocí předdefinovaných tříd je možné specifikovat např. do jaké kolekce se má vkládaný objekt přidat, z jaké kolekce se má vybrat

hledaný objekt nebo např. i zkontrolovat práva uživatele, který požadavek posílá. Od verze 3.1 Django začalo podporovat i asynchronní zpracování požadavků – verze byla vypuštěna v druhé půlce roku 2020 [21]. Bohužel, ale do tohoto dne, kdy je práce psána, nebyla přidána podpora asynchronního přístupu do frameworku Django REST.

Framework Flask byl vyvinut open source vývojaři Pocoo v roce 2010 [62]. Hlavní myšlenkou tohoto frameworku je poskytnout dobrý základ všem webovým aplikacím a vše ostatní nechat na dodatečných balících/knihovnách/modulech. Je postaven na dvou hlavních technologiích – Werkzeug a Jinja2. Werkzeug je zodpovědný za směrování, debugování, a WSGI [2]. Jinja2 je použit jako šablonovací jazyk pro tvorbu a generaci HTML dokumentů. Samotný Flask nijak nepodporuje funkčnosti jako: autentizace nebo komunikaci s databází. To vše je třeba dělat pomocí dodatečných knihoven jako např. SQLAlchemy. Flask sice podporuje asynchronní programování, ale v samotné dokumentaci zmiňují, že to nevede k žádnému signifikantnímu vylepšení výkonu [26]. Jelikož je Flask postaven na synchronním WSGI byl asynchronní přístup přidán až později a to způsobem, který neodpovídá zcela čistému asynchronnímu přístupu. Když přijde požadavek na nějaký endpoint, který spravuje asynchronní view, Flask spustí nový cyklus událostí ve vlákně a funkce spouští v něm. Tedy dovoluje použití asynchronních funkcí a klíčových slov `async` a `await`. Kód v rámci jednoho požadavku může sice běžet asynchronně pokud se např. provádí více asynchronních přístupů do databáze během jednoho požadavku, ale počet požadavků, které aplikace dokáže zpracovat v jednom časovém okamžiku zůstává stejný jako při synchronním přístupu.

Nejnovějším frameworkem, který zde budu popisovat je FastAPI, které bylo vyvinuto koncem roku 2018. Je považován za jeden z nejrychlejších Python frameworků, jelikož využívá možností moderního Pythonu. Mezi hlavní vlastnosti tohoto frameworku patří: typová nápověda, asynchronní zpracování a REST architektura [38]. Jelikož Python je dynamicky typovaný, nepodporuje žádnou formu statické kontroly typů. Tomuto se ale FastAPI snaží pomoci tzv. napovídání typů proměnných. Tahle funkčnost byla přidána ve verzi Python 3.5 a dovoluje programátorovi specifikovat typ proměnných, se kterými bude funkce pracovat. Jak již bylo zmíněno neznamená to, že tato vlastnost zamezí chybnému použití proměnných ve funkcích. Podle názvu jde pouze o napovídání – tedy tato funkce pomáhá pouze vývojářům, jelikož většina moderních IDE s ní pracují a dokáží varovat vývojáře během psaní kódu. Toto napovídání typů nejen pomáhá při programování, ale FastAPI má na něm založenou i automatickou generaci OpenAPI dokumentace. Jak již bylo zmíněno FastAPI je postaveno na asynchronním přístupu vytváření API.

Asynchronní programování je technika, kde běží pouze jedno vlákno a dokáže spouštět několik úloh najednou [42]. Tohoto je docíleno pomocí korutin a cyklu událostí. Asynchronní programování je ideální pro vstupně/výstupní operace, jelikož dovolí programu provádět jiné instrukce, a během nich čeká na vyhodnocení nějaké vstupně/výstupní operace. Jakmile tato operace skončí, vlákno se k ní opět vrátí a pokračuje ve vykonávání dál již s připravenými daty. Tato funkcionalita je obsažena ve vestavěném modulu `asyncio` a je to také základ pro ASGI webové aplikační frameworky jako např. právě FastAPI [38].

Dalším zmíněným frameworkem bylo Tornado, které vzniklo v roce 2009. Je tedy o 9 let starší než FastAPI, ale také jeho hlavním cílem byla především rychlost [22]. Podle [12] je ale FastAPI 2x rychlejší. Tento test zkoumal, kolik jednoduchých požadavků dokáží jednotlivé frameworky odbavit během sekundy. Požadavky byly směrovány na výchozí endpoint a tento endpoint navracel krátký dokument ve formátu JSON. Nejen rychlostní srovnání lze nalézt také zde [66]. Je tu zmíněno několik rozdílů mezi frameworky FastAPI a Tornado.

[2] WSGI nebo-li Web Server Gateway Interface je specifikace, která popisuje jak webový server komunikuje s webovými aplikacemi a jak webové aplikace mohou být propojeny pro zpracování jednoho požadavku [23]

Např. typová anotace je dostupná pouze ve FastAPI, stejně tak automatické generování dokumentace, FastAPI podporuje i automatickou dependency injection narozdíl od Tornada a v neposlední řadě je zde porovnána i učící křivka, která je opět lepší v případě FastAPI kvůli mnohem lepší dokumentaci a rostoucí komunitě.

Na základě předchozích srovnání byl pro vývoj aplikace vybrán framework FastAPI z několika důvodů:

- podporuje moderní asynchronní programování,
- jedná se o jeden z nejrychlejších Python frameworků,
- napovídání typů usnadní vývoj a testování,
- automatická dokumentace usnadní testování,
- dobrá dokumentace a rostoucí komunita také urychlí samotný vývoj.

Kapitola 6

Implementace

V předchozí kapitole byly vybrány konkrétní technologie pro vývoj webové aplikace ContentCupid. Pro vývoj jsem tedy použil:

- Framework FastAPI v jazyce Python pro tvorbu asynchronního API,
- framework Vue pro klientskou stranu aplikace (komponenty jsou psány pomocí jazyka TypeScript a stylovány pomocí obyčejného CSS),
- relační databázi MySQL,
- NoSQL dokumentovou databázi MongoDB
- a pro uložení dokumentů, videí, fotografií uživatelů jsem použil cloudové úložiště Amazon S3.

Na serverové straně jsou použity následující knihovny:

- SQLAlchemy – pro relačně-objektové mapování databáze MySQL,
- Pydantic – pro zavedení typů v jazyce Python – zároveň tuto knihovnu používá FastAPI pro generaci dokumentace a pro kontrolu příchozích a odchozích dat,
- Jose – pro vytvoření a dekódování JWT tokenů,
- PyMongo a Motor – pro asynchronní komunikaci s nerelační databází Mongo DB,
- Aioboto3 – pro asynchronní komunikaci s cloudovým úložištěm Amazon S3,
- SMTPlib a email – pro zasílání e-mailových zpráv přes SMTP,
- Socketio – pro navázání a následnou komunikaci protokolem websocket.
- MoviePy, PIL – tyto byly použity pro překreslení obrázku/videa vodoznakem.
- Pytest a requests – pro otestování API pomocí jednotkových testů.

Na klientské straně jsou použity následující knihovny

- Axios – pro komunikaci pomocí HTTP,
- Pinia – pro globální stavový sklad,

- Vue-router – pro směrování pomocí URL v single page aplikaci,
- jwtDecode – pro dekódování access tokenů na klientské straně,
- FbSdkWrapper a GoogleSdkLoaded – pro přihlášení přes externí Oauth pomocí Facebook a Google,
- VuePDF – pro zobrazení náhledů PDF souborů
- Socket.io-client – pro navázání komunikace přes protokol websocket na straně klienta,
- VueTippy – pro zobrazení pomocných titulků při přejetí přes tlačítka.

V následujících kapitolách se věnuji detailnímu popisu implementace aplikace Content Cupid pomocí těchto technologií. Podkapitoly jsou členěny dle využitých technologií a také pomocí rozdělení částí aplikace. Důkladněji jsou zde popsány principy autentizace a manipulace s daty na cloudovém uložišti.

6.1 Autentizace a využití OAuth

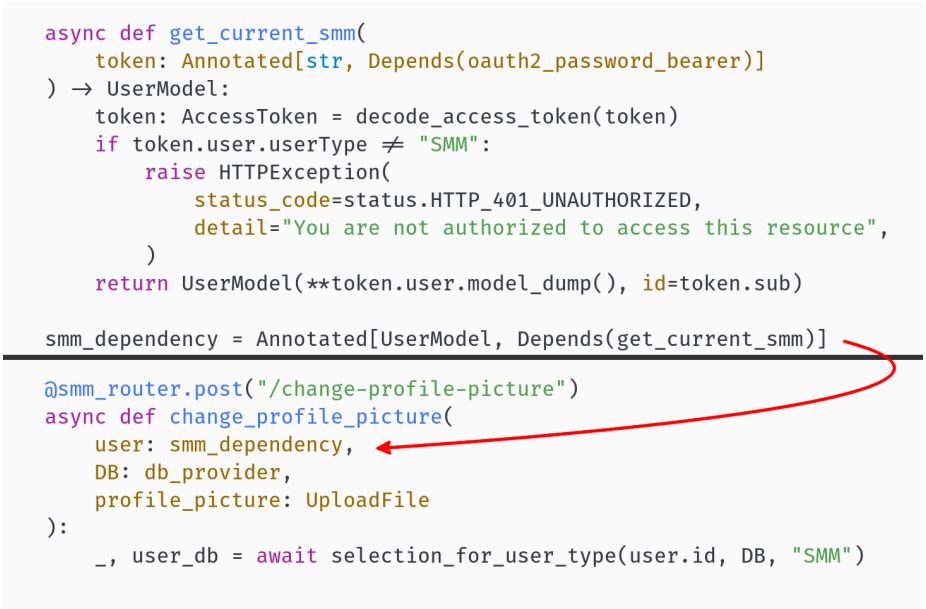
Autentizace v systému probíhá pomocí JWT tokenů, které jsou zasílány s každým požadavkem na API v hlavičce `Authorization` (access token) a v `Cookies` (refresh token). Využívají se zde dva typy tokenů: access token a refresh token. V aplikaci Content Cupid jsou využity přesně tak, jak jsou popsány např. zde [49]. Access token dává jeho držiteli přístup k nějakým chráněným zdrojům. Tyto tokeny mají většinou dobu platnosti kratší (v rámci dní) a nesou nějaká programátorem definovaná data. Naproti tomu refresh tokeny mají dlouhou dobu platnosti (až v rámci měsíců). Po vypršení platnosti access tokenu je refresh token zaslán na API, kde je ověřena jeho validita a podpis a následně je vygenerován nový access token, který se používá dále. Konkrétní struktura access tokenu je ukázána na obrázku 6.1. Refresh token obsahuje mnohem méně dat a to: čas expirace, typ uživatele a jeho identifikátor. Oba typy tokenů jsou podepsány algoritmem se zkratkou HS256, což je zkratka pro HMAC s použitím hashovacího algoritmu SHA256. Toto lze ověřit v hlavičce zasílaných JWT tokenů. Tyto podpisy jsou automaticky validovány s každým požadavkem na API.


```
{
    "exp": 1714497133,
    "sub": "1",
    "user": {
        "email": "user@email.com",
        "name": "Tomas Krivanek",
        "nickname": null,
        "superAdmin": null,
        "userType": "SMM",
        "oauthId": "101529936149827049908",
        "companyCreator": true
    }
}
```


Obrázek 6.1: Data, která jsou zasílána v části payload JWT tokenu typu access.

Byly vyvinuty tři různé funkce pro validaci přístupu k různým endpointům. Tyto funkce jsou navrženy tak, aby odpovídaly různým uživatelským oprávněním, což znamená, že každý

endpoint má přidělenou funkci podle toho, jaké kategorie uživatelů jej mohou využívat. Pokud je endpoint dostupný i pro nepřihlášené uživatele není použita žádná funkce. K volání funkcí dochází automaticky před zahájením jakékoliv operace definované v rámci daného endpointu. Integrace těchto funkcí do jednotlivých endpointů je řešena pomocí techniky dependency injection. K tomu využívám typ Annotated, který mi umožňuje specifikovat, která funkce má být v daném kontextu použita. V případě, že dojde k selhání validace tokenu, API reaguje vrácením chybového kódu 401 Unauthorized, což uživateli signalizuje, že pro přístup k požadovaným datům nemá dostatečná oprávnění.



Obrázek 6.2: Funkce, která je implementována na vrchní polovině obrázku, je použita pro kontrolu zda přihlášený uživatel je manažer. Na spodní polovině lze vidět, jak je použita dependency injection v jednom z endpointů aplikace.

6.1.1 Registrace a přihlášení

Registrace je rozdělena na dva typy: registrace firmy a registrace tvůrců. Při registraci tvůrce musí zadat tyto údaje: Celé jméno, přezdívku, heslo, heslo pro kontrolu, e-mail a datum narození. Již během zadávání do jednotlivých textových polí jsou uživatelé upozorněni, jestliže zadají nějaký nevalidní vstup, např. příliš krátké jméno, příliš slabé heslo nebo pokud jsou uživatelé mladší 18 let. Po zaslání těchto údajů na API je zde zkontrolováno zda již uživatel s tímto e-mailem v databázi existuje. Pokud ano, je navrácen kód 409 Conflict. Jestliže konflikt nenastal je vytvořena nová entita Creator a ta je uložena do databáze. Ještě předtím je vygenerován náhodný šestimístný kód, který je po vytvoření entity v databázi zaslán na zadaný e-mail společně s odkazem pro verifikaci e-mailu.

Po obdržení e-mailu a kliknutí na zaslaný odkaz je uživatel přesměrován na stránku, kde kód musí zadat. Jakmile zadá správný kód, je jeho e-mail potvrzen a zároveň s tímto je uživatel i přihlášen – jsou mu vytvořeny jak refresh tak i access token. Access token je na frontendu uchováván pomocí lokálního úložiště v prohlížeči a zároveň také pomocí stavového skladu, který je naprogramován pomocí knihovny Pinia. Lokální úložiště je použito kvůli zachování persistence při opuštění webové aplikace. Tento access token je poté používán při

každém požadavku na API a to pomocí tzv. interceptorů knihovny Axios. Tyto interceptory se dají použít dvěma způsoby [8]– buďto jsou volány před samotným odesláním požadavku na API nebo jsou volány před přijetím odpovědi. Proto se tyto interceptory dají použít pro automatické připojení access tokenu ke každému požadavku. Tento interceptor je tedy volán před jakýmkoliv odesláním požadavku.

Dalším implementovaný interceptor se spouští při každém přijetí požadavku. Pokud vyprší platnost platnost access tokenu je třeba zaslat refresh token na předem definovaný endpoint. Jelikož jsou refresh tokeny uloženy v tzv. http-only secure cookies[1], není možné ověřit zda-li vůbec nějaký refresh token uživatel má, proto je třeba při každé příchozí odpovědi zkontrolovat zda-li nebyl navrácen chybový kód 401 a až následně poté zažádat o nový access token pomocí refresh tokenu. Zdrojový kód tohoto interceptoru je ukázán na obrázku 6.3.

```
const apiClient: AxiosInstance = axios.create({
    baseURL: BASE_URL,
    withCredentials: true,
    headers: {
        'Content-Type': 'application/json'
    }
})
apiClient.interceptors.response.use(
    (response) => {
        return response
    },
    async (error: AxiosRefreshError) => {
        const originalRequest: RetryAxiosRequestConfig = error.config
        const errMessage = error?.response?.status
        if (errMessage === 401 && !originalRequest._retry) {
            if (originalRequest.url !== '/auth/refresh') {
                originalRequest._retry = true
                const userStore = useUserStore()
                const newToken = await refreshAccessTokenFn() // Need to set the new token

                if (newToken.error || !newToken.data?.accessToken) {
                    return Promise.reject(error)
                }
                userStore.setUserAccessToken(newToken?.data?.accessToken)
                return apiClient(originalRequest)
            }
        }
        return Promise.reject(error)
    }
)
```

Obrázek 6.3: Zdrojový kód ukazuje použití interceptoru typu response v knihovně Axios. Tento interceptor zaručí, že pokud vyprší access token je následně proveden pokus o získání nového. Funkce refreshAccessTokenFn zažádá API o nový a pokud skutečně API nový access token poskytne, je na klientské straně aktualizován access token ve stavovém skladu a zároveň i v lokálním úložišti prohlížeče. Zároveň tento interceptor zopakuje požadavek, který byl v minulosti zamítnut kvůli expiraci access tokenu.

[1]Cookie s atributem HttpOnly není možné získat skrze Javascriptový atribut `Document.cookie` [41]. Je tedy pouze zasílána s požadavky na server, ale nelze ji nijak na klientské straně získat v Javascriptovém kódu. Cookie s atributem Secure je zasílána pouze po šifrovaném kanálu, tedy pouze při použití protokolu HTTPS.

Registrace firmy je velmi podobná registraci uživatele. Jsou pouze požadovány nějaké informace navíc – identifikační číslo, jméno firmy a adresa. Zároveň jsou tedy potřeba i údaje o zakládajícím manažerovi, které jsou podobné, jako když se registruje tvůrce. Opět je zde po registraci použit verifikační kód, který je zaslán na e-mail a následně je třeba e-mail potvrdit, jinak se uživatel nemůže přihlásit. Oba typy uživatelů jsou po přihlášení přesměrováni do svého účtu.

Pro zamezení přístupu na stránky, na které nemá uživatel právo byl použit atribut beforeEnter při definování cest pomocí knihovny Vue-router. Tento atribut zapříčiní, že před přístupem na konkrétní stránku je spuštěna funkce, která je specifikována jako hodnota tohoto atributu. Tím pádem před každým přesměrováním je možné zkontrolovat typ uživatele, který je přihlášen. Zároveň bylo třeba zvlášť kontrolovat nejen typ uživatele, ale i jeho práva na určité odpovědi z API. Tím je myšleno např. přístup na stránku detailu spolupráce jiného uživatele – to je řešeno pomocí manuálního přesměrování. Pokud uživatel ručně zadá do prohlížeče URL adresu na obsah, ke kterému nemá oprávnění, je vždy přístup kontrolován skrze příslušný API endpoint. V případě, že API identifikuje uživatele jako neautorizovaného, vrací chybový kód "Unauthorized", a následně dojde k jeho přesměrování na stránku pojmenovanou "Unauthorized", která mu oznamuje omezení přístupu.

Odhlášení z aplikace probíhá ve dvou krocích. Na straně API je v rámci procesu odhlášení volána metoda delete_cookie s atributem refresh nad objektem response. Tato metoda je do těla endpointové funkce vložena pomocí dependency injection. Díky tomu dojde k smazání http-only cookie, která obsahuje refresh token. Na straně klienta jsou následně vymazány veškeré informace o uživateli ze stavového skladu Pinia a z lokálního úložiště je odstraněn access token, což zajišťuje kompletní odstranění všech identifikačních dat uživatele.

6.1.2 Využití Google a Facebook OAuth

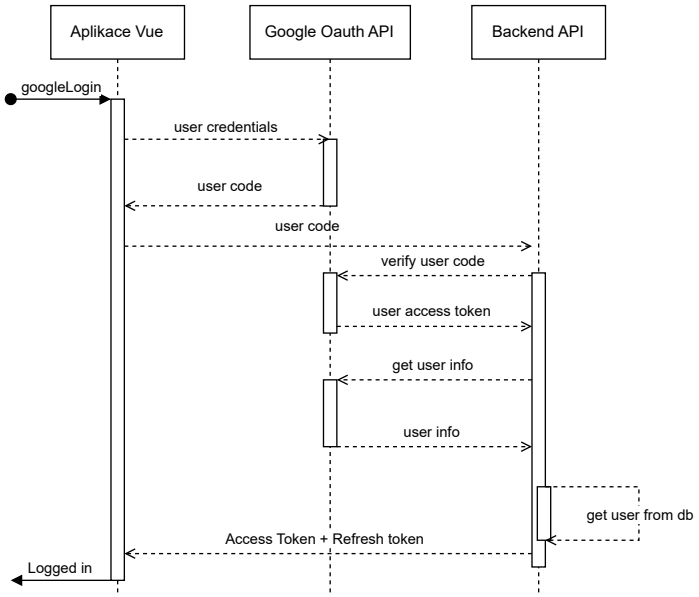
Uživatelé mají možnost využít přihlášení i přes externí služby a to konkrétně přes Google či Facebook. V této kapitole se věnuji převážně přihlášení přes Google, ale přihlášení přes službu Facebook je velice podobné. Grafické znázornění průběhu je zobrazeno na obrázku 6.4. Přihlášení probíhá v několika krocích.

Při prvním kroku komunikuje uživatelská strana pouze s danou přihlašovací službou. Na klientské straně jsou využity dodatečné knihovny. Pro přihlášení přes Google je použita knihovna `vue3-google-login`[2]. Po kliknutí na tlačítko s ikonou Google je spuštěna funkce googleSdkLoaded, která uživatele vyzve ke zvolení účtu právě na platformě Google. To provádí ve vyskakovacím okně mimo aplikaci. Zde uživatel buď zvolí již přihlášený účet nebo se může přihlásit k novému účtu pomocí e-mailové adresy a hesla. Po úspěšném přihlášení je zavolána mnou definovaná funkce googleLoginCallback, která provádí další krok přihlášení.

Pokud byl předchozí krok přihlášení úspěšný, tak služba Google poskytne uživateli tzv. code, který bude sloužit pro ověření uživatele na backendu. Klientská strana, tedy tento kód zasílá na příslušný API endpoint, kde probíhá další ověření. Jakmile API příjme tato data, je třeba ověřit uživatelem poskytnutý kód. Backend asynchronním požadavkem na službu OAuth2 od Googlu zasílá požadavek pro ověření kódu. Pokud je kód správný, je následně navrácen access token, který je dále použit pro získání dodatečných informací o přihlašovaném uživateli: především e-mail a identifikátor ze služby Oauth2. Tyto údaje jsou následně porovnány s uživatelským účtem uloženým v databázi. Pokud uživatel ještě není na službě registrován, je o tomto informována klientská strana a uživatel je přesměrován na regis-

[2]Odkaz na dokumentaci knihovny: `https://devbaji.github.io/vue3-google-login/`

traci, kde musí doplnit další potřebné informace (přezdívka, datum narození atd.). Pokud je uživatel registrován je porovnán identifikátor Oauth2 s tím, který je uložen u příslušné entity v databázi. Pokud se tyto údaje shodují je uživatel již úspěšně přihlášen, tzn. je mu vygenerován nový access a refresh token.



Obrázek 6.4: Sekvenční diagram na kterém je znázorněn průběh přihlášení pomocí Google Oauth2 tak, jak je použit v aplikaci Content Cupid.

Přihlášení přes Facebook funguje podobným způsobem až na dva rozdíly. Po přihlášení přes Oauth službu Facebooku je uživateli rovnou vygenerován access token, který je zaslán na backend. Tento access token je opět ověřen i na straně backendu akorát zde není potřeba provádět dva požadavky na externí Oauth službu, jelikož Facebook poskytuje informace o uživateli již při ověření access tokenu. Další postup je ale obdobný jako u přihlášení přes Google. Pro tento typ přihlášení je využita knihovna `fb-sdk-wrapper`[3]

Aby byl backend schopen komunikovat s externími službami, musely být vygenerovány i autentizační údaje pro API. Tyto údaje byly vygenerovány na příslušných webových stránkách těchto služeb. Pro Google je třeba při zasílání požadavku specifikovat vygenerované `client_id` a `client_secret`. Pro službu poskytovanou Facebookem je třeba specifikovat `app_id` a `app_secret`.

6.2 Manipulace s cloudovým úložištěm

S cloudovým úložištěm přímo komunikují obě strany aplikace: jak backend tak i frontend. Backend pro komunikaci využívá knihovnu Aioboto3 pomocí níž provádí následující operace:

1. Nahrání souborů – v případě, že se jedná o malé soubory (např. profilová fotografie), klientská strana posílá tyto soubory přímo na backend a odtud se soubory nahrávají do cloudového úložiště.

2. Odstranění souborů – pokud klientská strana zašle požadavek na smazání souboru (např. tvůrce chce smazat jeden ze svých propagačních materiálů), je ze serveru pro-

[3]Repozitář knihovny `https://github.com/erikhagreis/fb-sdk-wrapper`

vedena operace pomocí metody delete_object, která přijímá atributy: jméno bucketu v cloudovém uložišti a klíč souboru, který si uživatel přeje smazat.

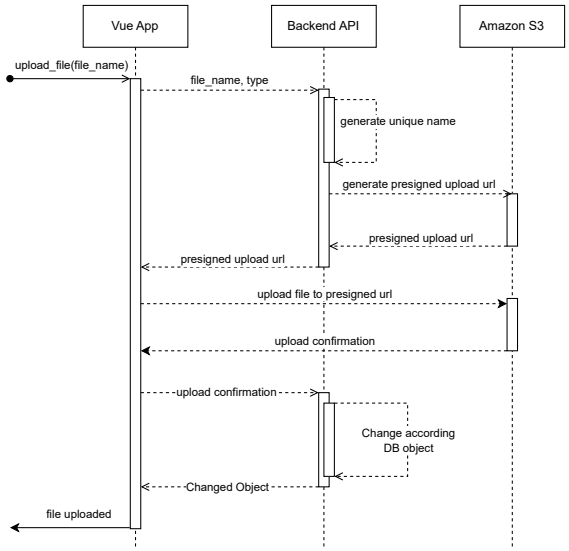
3. Generování předem podepsaných odkazů – jelikož jsou všechny objekty uložené na Amazon S3 soukromé, je zapotřebí mechanismus, který dovolí dočasný přístup k těmto objektům i běžným uživatelům. Pro zajištění tohoto typu přístupu slouží předem podepsané odkazy [1]. V tomto případě API zasílá požadavek na Amazon S3 s žádostí o vygenerování tohoto předem podepsaného odkazu. V požadavku se vždy posílají autentizační údaje backendu a klíč k objektu, pro který je třeba dočasný přístup. Výsledkem je odkaz pro stažení či nahrání objektu, který je platný pouze po specifikovanou dobu. Tyto odkazy se používají napříč celou platformou Content Cupid kdekoliv, kde je třeba umožnit přístup k objektům uloženým v cloudovém úložišti na klientské straně.

Pokud klientská strana potřebuje nahrát nějaký větší soubor např. video, postupuje se následovně (schéma tohoto typu nahrávání je na obrázku 6.5):

1. Z klientské strany jsou na API zaslány informace o souborech – typ a jméno souboru. Zde je vygenerován tzv. předem podepsaný odkaz pro nahrání a tyto odkazy jsou zaslány zpět na klientskou stranu.

2. Následně klientská strana komunikuje a nahrává soubory přímo do cloudového úložiště Amazon S3.

3. Klientská strana potvrdí nahrání souborů na cloudové úložiště pomocí dalšího požadavku na API, kde je vytvořena nová entita do tabulky `creator_info_assets`.



Obrázek 6.5: Sekvenční diagram na kterém je znázorněn průběh nahrávání velkých souborů na cloudové úložiště Amazon S3.

Při jakémkoliv nahrání se upravuje jméno souboru a vytváří se jeho klíč. Tento klíč je sice prezentován a má tvar, jako cesta k souboru, ale v tomto cloudovém uložišti je vždy objekt

uložen pouze jako klíč a jeho hodnota, kde klíčem je právě jméno ve tvaru cesty k souboru a hodnotou je soubor [1]. Následující výčet tedy specifikuje tvar jmen jednotlivých souborů pomocí cest uložených na Amazon S3 bucketu:

- `categories/` – obsahuje ikony kategorií, ty by sice mohli být uloženy přímo v klientské aplikaci, ale jelikož jeden z požadavků na systém byl, že by mělo být možné v budoucnu kategorie upravovat, je třeba zajistit možnost jednoduchého přidání kategorie do databáze i s její ikonou.
- `collab_offers/` – tato složka je dále členěna pomocí identifikátoru nabídky spolupráce a následně obsahuje podsložky: `images/` a `videos/`, které obsahují fotografie a videa přiřazené k určité nabídce.
- `collaborations/` – složka uchovává veškeré soubory ke spolupráci. Podobně jako předchozí složka je dále členěna pomocí identifikátoru probíhající spolupráce. Dále je členěna následovně
  - `{asset_id}` – identifikátor materiálu, který tvůrce musí vytvořit. Těchto podsložek je k jedné spolupráci připsáno většinou více, jelikož většina manažerů během jedné spolupráce specifikuje více požadovaného materiálu
  - `invoice` – v této složce je uložena faktura k dané spolupráci, pokud nějaká existuje.
  - `support` – všechny pomocné soubory, které byly nahrány ke spolupráci.
- `profile_assets/` – zde se uchovávají propagační materiály tvůrců, které jsou zobrazeny na jejich profilu.
- `profile_pics/` – v této složce jsou uchovány profilové fotografie. Zde je složka dále členěna na profilové obrázky tvůrců a manažerů.

I přes rozdělení do podsložek se může stát, že by uživatel chtěl nahrát soubor pod stejným klíčem. Proto je vždy klíč generován následovně:

1. Na základě typu souboru se určí cesta, kam je třeba soubor uložit,
2. dále je vypočten unikátní identifikátor pomocí knihovny `uuid`,
3. na konec tohoto řetězce je připojeno skutečné jméno souboru.

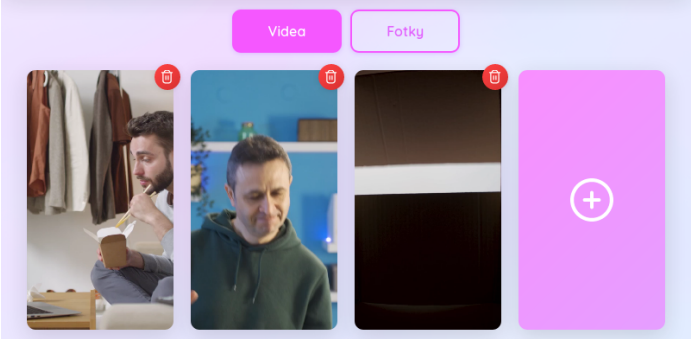
6.3 Systém pro tvůrce

V této kapitole se věnuji klíčovým funkcím systému, které jsou zásadní pro tvůrce obsahu v rámci platformy Content Cupid. Kapitola je rozdělena do dvou hlavních sekcí: Úprava portfolia, Žádost o spolupráci. V každé z těchto sekcí detailně popisuji jak praktické fungování jednotlivých součástí systému. Tyto dvě funkcionality jsou mimo vedení spolupráce zásadní pro každého tvůrce, který bude platformu používat. Pro tvůrce je důležité přizpůsobení jeho portfolia, jelikož je to hlavní zdroj informací o tvůrčích schopnostech na základě kterých budou manažeři vybírat tvůrce pro jejich spolupráce. Důležité je také aby systém nabízel přehledné seznamy nabídek spoluprací, které jsou členěny do kategorií. A poslední důležitou součástí je žádost o spolupráci, ve které uživatel bude schopen specifikovat, proč právě on by měl být vybrán pro konkrétní spolupráci.

Tvůrce může svůj profil přizpůsobit hned několika způsoby. V systému je tvůrce schopen: nastavit profilový obrázek, odkazy na sociální sítě, popsat svůj profil textově a hlavně na svůj profil může nahrát své propagační materiály (ukázková videa a fotografie). Vzhled uživatelského rozhraní profilu tvůrce je na obrázku A.5 v přílohách. Změna profilového obrázku probíhá ve dvou krocích. Nejprve uživatel nahraje svůj nový profilový obrázek a následně potvrdí, že je s podobou spokojen (obrázky jsou totiž zarovnány na střed a oříznuté do kruhu). Pro nahrání souborů je prakticky všude použita víceúčelová komponenta, která je blíže popsána v kapitole 6.6.

Dalším způsobem přizpůsobení profilu je nastavení odkazů na sociální sítě. Každý tvůrce může specifikovat 4 odkazy – na svůj Instagram, LinkedIn nebo TikTok účet a jeden obecný. Tento obecný vzešel z uživatelského testování, kdy jsem se dozvěděl, že spousta tvůrců by si přála mít možnost přesměrovat manažery na své vlastní webové stránky. Po přidání nového odkazu uživatel buď může zkontrolovat, že zadaná adresa směruje na správnou webovou stránku, nebo tento odkaz může smazat a vytvořit nový.

Tvůrce by také měl napsat něco o sobě. Proto je pod základními údaji prostor pro napsání čehokoliv, co konkrétního tvůrce napadne. Poslední způsob, jakým uživatel může své portfolio upravit je poskytnutí svých reprezentativních videí nebo fotografií. Každý tvůrce má nárok na přidání až 4 videí a 4 fotografií (zdůvodnění tohoto omezení je popsáno v kapitole 3.1). Tyto materiály může přidat po kliknutí na kartu se symbolem plus (Obr. 6.6).



Obrázek 6.6: Snímek, který je pořízen z části portfolia tvůrce. Tímto způsobem jsou zobrazena jednotlivá videa a pokud uživatel chce přidat další může tak učinit po kliknutí na kartu se symbolem plus.

Tvůrce dále může zasílat požadavky na konkrétní nabídky spolupráce. Na hlavní stránce webové platformy je odkaz, který směřuje na kategorie. Po kliknutí na jednu z nich se zobrazí nabídky spolupráce z této kategorie (uživatelské rozhraní lze vidět v přílohách na obrázku A.3). Zde jsou vždy zobrazeny nejdůležitější informace o nabídce – jméno, typ odměny a případná výše odměny. Po kliknutí je zobrazen detail spolupráce, tak jak ho specifikoval manažer (popsáno v kapitole6.4.1). Zde může přihlášený tvůrce poslat žádost o tuto spolupráci podobnou formou jakoby posílal e-mail, tzn. specifikuje předmět a text zprávy.

6.4 Systém pro manažery sociálních sítí

Uživatelské rozhraní pro manažery sociálních sítí je vizuálně velmi podobné tomu pro tvůrce. Manažer má však na výběr z následujících podstránek na svém profilu: vaše firma, nabídky spoluprací, probíhající spolupráce a chat. V následujících podkapitolách se věnuji

těm nejdůležitějším součástem, které platforma manažerovi nabízí. Nebudu zde tedy zmiňovat, jak probíhá proces nahrávání profilových fotografií firmy/manažera, jelikož ten je obdobný tomu na profilu tvůrce.

6.4.1 Tvorba nabídek spoluprací

Prvním krokem k započetí spolupráce s konkrétním tvůrcem je vytvoření nabídky spolupráce. Na vytváření těchto nabídek se manažer dostane na podstránce nabídky spoluprací po kliknutí na tlačítko nová nabídka. Manažer je následně přesměrován do systému tvorby nabídky. Vytváření nabídky je rozděleno do dvou vizuálně oddělených částí – v levé části se nachází obecné nastavení a pravou tvoří podoba nabídky (uživatelské rozhraní lze vidět v přílohách na obrázku A.7).

Na levé straně manažer nastavuje: typ produktu, typ odměny, jestliže se jedná o placenou spolupráci, tak i výši odměny, dále počet tvůrců, které na spolupráci hledá a nakonec kategorie nabídky spolupráce. V pravé části Specifikuje název spolupráce a samotný text nabídky. Zde má možnost k nabídce přidat i multimediální obsah - fotografie, videa, která se poté zobrazují ve spodní části nabídky. Při uložení této nabídky je obecné nastavení uloženo do relační databáze a obsah textového editoru i se seznamem připojených multimédií je uložen v databázi MongoDB. Samotný textový editor je naprogramován pomocí knihovny Tiptap[4]. Tato knihovna slouží právě ke tvorbě takovýchto textových editorů a zároveň lze jednoduše obsah zapsaného textu uložit ve formátu json a následně jej načíst zpět do editoru či jej zobrazit běžnému uživateli.

Po uložení nabídky může manažer nabídku zveřejnit. Jakmile tak učiní, mohou tuto nabídku tvůrci nalézt v příslušných kategoriích a zasílat žádosti o tuto spolupráci. Žádosti manažer vidí v přehledu svých nabídek – zde se mu ukazuje kolik žádostí daná nabídka má. Tyto nabídky může manažer přijmout či odmítnout nebo může zahájit chat s daným tvůrcem a blíže se s ním seznámit. Při odmítnutí má možnost specifikovat důvod odmítnutí, ale nemá povinnost toto pole vyplnit. Při přijetí žádosti je založena nová spolupráce mezi manažerem a tvůrcem.

6.5 Systém vedení spolupráce

K tomuto systému má přístup jak manažer, tak tvůrce, mezi kterými byla započata spolupráce. Uživatelské rozhraní (v přílohách na obrázku A.8) se dělí na tři sekce. V pravé části je vždy zobrazen chat s protistranou – to vyplynulo přímo z požadavků od uživatelů, jelikož je to nejpoužívanější část. Tento chat je oddělený a přísluší pouze k této spolupráci. V levé horní části je navigační lišta, která přepíná mezi podstránkami: informace o spolupráci, smlouvy, podpůrné soubory, odevzdávárna a faktura.

Podstránka **informace o spolupráci** nabízí manažerovi možnost sepsat podrobné zadání požadovaných materiálů. Je zde možnost specifikovat obecný popis celé spolupráce (např. kterých produktů se bude spolupráce týkat), ale také zde mohou specifikovat zadání jednotlivých požadovaných materiálů. Při přidání zvolí typ (video, fotografie, audio, ostatní) a následně mají možnost zadat jejich představu o výsledném materiálu. Velmi často však manažeři nechávají prostor fantazii tvůrce a nechají vytvoření materiálu přímo na něj, proto detailní zadání není povinné. Z pohledu tvůrce je tedy na této podstránce vidět, co je jeho úkolem v rámci této spolupráce.

[4]Odkaz na dokumentaci: `https://tiptap.dev/docs/editor/introduction`

Podstránky **smlouvy** slouží pro sdílení smluv mezi oběma účastníky spolupráce. Od potenciálních uživatelů jsem zjistil, že je smlouva uzavřena vždy jen jedna - buďto smlouvu zasílá tvůrce a podepisuje ji manažer nebo naopak. Proto je zde možnost nahrání jediného souboru ve formátu PDF. Jakmile jeden z uživatelů do této sekce nahraje PDF se smlouvou, protistrana si tuto smlouvu může stáhnout a následně nahrát její podepsanou kopii.

Další podstránka **podpůrné soubory** slouží pro sdílení čehokoliv, co se týče spolupráce. Zde uživatelé nejsou vůbec omezeni typem souboru ani ničím jiným, mohou zde nahrávat cokoliv. Protistrana si vždy může uložené soubory stáhnout a ten kdo soubory nahrál je může také smazat.

Další velice důležitou podstránkou je **odevzdávárna**. Ta je rozdělena na tolik sekci, kolik bylo specifikováno materiálů na podstránce informace o spolupráci. Zde tvůrci nahrávají materiál, který jim byl zadán manažerem. Pokud se jedná o barterovou spolupráci jsou soubory nahrávány stejně, jak bylo popsáno v kapitole 6.2. Ovšem pokud se jedná o placenou spolupráci je třeba videa ještě před samotným nahráním na cloudové úložiště dodatečně zpracovat, proto se zde postupuje jinak. Při zasílání videa či fotografie se nežádá o předem podepsaný odkaz ke cloudu, ale tyto soubory se nahrávají přes backend následujícím způsobem:

1. Klientská strana zašle soubor přímo na backend server, který jej dočasně uloží kvůli dalšímu zpracování.
2. V databázi je poznačeno, že probíhá proces přidávání vodoznaku do přijatého souboru.
3. Jelikož proces přidávání vodoznaku do videa či fotografie trvá delší dobu, je využito tzv. BackgroundTasks[5] z knihovny FastAPI. Do těchto BackgroundTasks je tedy přidána funkce, která provede přidání vodoznaku buďto do videa nebo do fotografie.
4. Server následně zasílá odpověď na původní požadavek.
5. Systém dopočítá nový soubor (vytvoří vodoznak a zhorší kvalitu videa) a uloží jej.
6. Následně oba soubory nahraje na cloudové úložiště a uloží jejich klíče k příslušné databázové entitě.
7. V posledním kroku smaže dočasné soubory.

Pokud se tedy jedná o placenou spolupráci je důležité, aby se k uživateli s rolí manažera nikdy nedostali záběry v plné kvalitě, proto jsou vždy manažerovi zasílány pouze materiály s vodoznakem. Jakmile tvůrce nahraje soubor k zadání je manažerovi umožněno tento materiál buďto zamítnout či přijmout. Při zamítnutí má povinnost specifikovat důvod zamítnutí, kam manažeři většinou píší, co by potřebovali zlepšit. Po přijetí všech materiálu spolupráce je tvůrci zobrazeno tlačítko, kterým může tvůrce odemknout materiál v plné kvalitě a bez vodoznaku. Tento krok se u barterové spolupráce nemusí provádět, jelikož zde má přístup k původním materiálům manažer již od okamžiku nahrání tvůrcem.

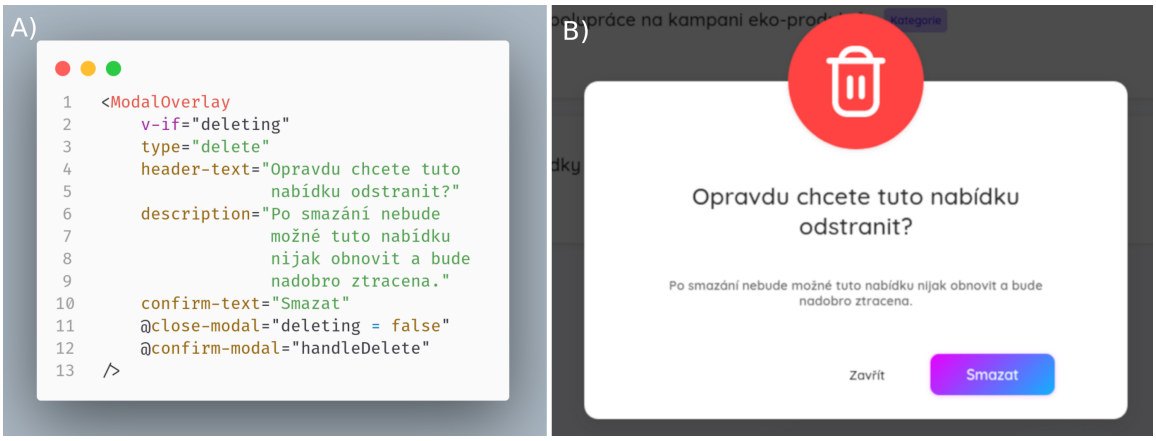
Na poslední podstránce **faktura** má tvůrce možnost nahrát fakturu za placenou spolupráci. Samozřejmě ji má možnost smazat a nahrát novou případně si ji může pro kontrolu stáhnout. Manažer zde má pouze možnost stáhnutí faktury.

[5]BackgroundTasks (úlohy v pozadí) se používají, pokud je třeba aby server něco vypočítal až po zpracování požadavku [52].

6.6 Víceúčelové komponenty

Napříč celou klientskou částí aplikace se používá několik komponent opakovaně. V této kapitole popisuji ty nejpoužívanější a zároveň nejzajímavější. Znovupoužitelnost těchto komponent spočívá především ve využití tzv. props, tedy atributů, které lze předat od rodičovského elementu k synovskému. Dále také využívají tzv. emits, což zajišťuje komunikaci druhým směrem

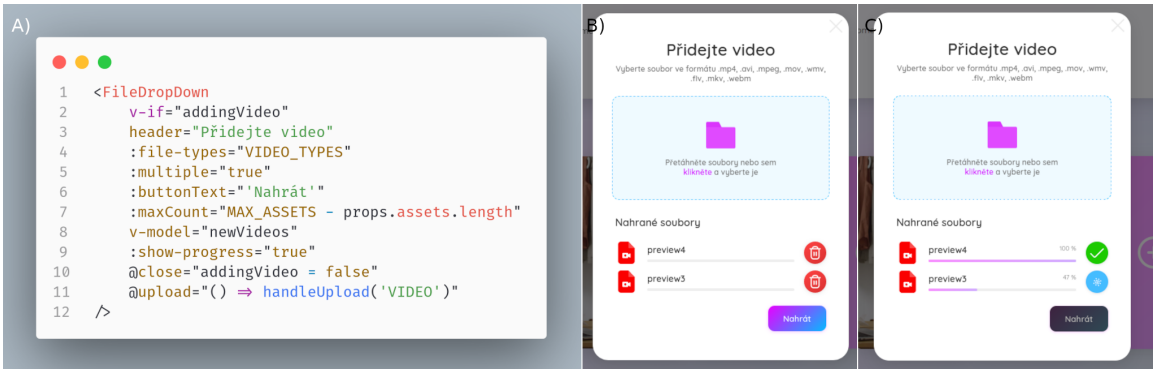
Nejpoužívanější komponentou v celé aplikaci je komponenta `ModalOverlay`, která slouží pro potvrzení nějaké uživatelské akce, např. zaslání žádosti o spolupráci, odstranění souboru z podpůrných souborů, odstranění propagačního materiálu tvůrce (návrh této komponenty je v přílohách na obrázku A.9). Tato komponenta má nastavitelný: nadpis, popis, typ, text zobrazený v potvrzujícím i v zamítacím tlačítku a nakonec atribut, který specifikuje zda-li má zablokováno zavření potvrzení po kliknutí na potvrzovací tlačítko. Díky těmto atributům je tato komponenta plně nastavitelná. Její použití je zobrazeno na obrázku 6.7. Na tomto obrázku lze i vidět použití již zmíněných tzv. emits, které se ke komponentě přiřazují pomocí znaku zavináče. Komponenta prakticky vytvoří událost, na kterou si rodičovská komponenta vytvoří naslouchač. Po vypuštění této události je spuštěna funkce specifikovaná jako hodnota tohoto naslouchače. Díky této technice je tedy jednoduché specifikovat libovolnou funkcionalitu a u této komponenty je to použito na spuštění určité funkce po kliknutí na tlačítko zavřít či potvrdit.



Obrázek 6.7: Vlevo je zobrazeno použití komponenty `ModalOverlay` ve zdrojovém kódu a vpravo je zobrazeno, co tento kód produkuje na webové stránce.

Další často používaná komponenta je `FileDropDown` (obr. 6.8). Ta je použita pro nahrávání souborů jak už na backend aplikace či do cloudového úložiště. Tato komponenta slouží jak k samotnému zvolení souborů, tak i zároveň ukazuje přehled o aktuálně zvolených souborech. Komponenta je nastavitelná díky následujícím atributům:

- `header` – specifikuje text hlavního titulku,
- `fileTypes` – povolené koncovky souborů, které má komponenta přijímat (interně se překládají na skutečné typy, např. `['.pdf']`, je přeloženo na skutečný typ, který má nahraný sobory, tedy: `['application/pdf']`),
- `maxCount` – maximální počet nahrávaných souborů,
- `showProgress` – specifikuje zda má komponenta ukazovat průběh nahrávání,



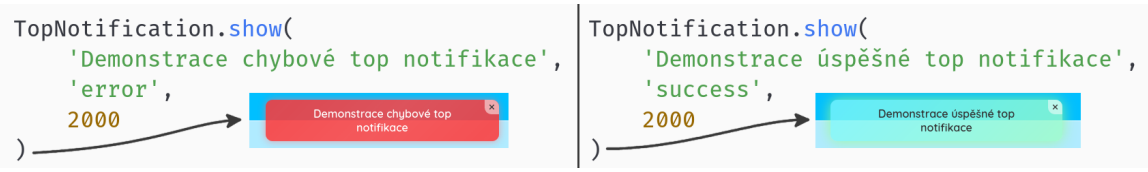
Obrázek 6.8: A) Použití komponenty FileDropDown ve zdrojovém kódu, B) komponenta v uživatelském rozhraní aplikace, C) komponenta během spuštění nahrávání souborů.

Dále je zde použita direktiva `v-model`, která automaticky synchronizuje stavovou proměnnou. V tomto případě je to seznam souborů, které následně budou nahrány na libovolnou adresu. Stejně jako v předchozí komponentě jsou zde naslouchače pro potvrzení a zamítnutí akce. Při zobrazení nahrávání je využito přidaných atributů k vestavěnému datovému typu File: `uploading`, `progress`. Při započetí nahrávání souboru je nastaven atribut `uploading` na `true` a dále se postupně aktualizuje hodnota `progress`, která nabývá hodnot od 0 do 100. Pro získání informace o průběhu nahrávání je použit atribut konfiguračního objektu, který příjmá metoda put knihovny axios (zdrojový kód obecné funkce pro nahrání souboru je na obr. 6.9).

```
static async uploadFileWithProgress(
    file: File,
    presignedUrl: string,
    onUploadProgress: (progress: Number) => void
): Promise<ServiceOutput<undefined>> {
    try {
        const response = await axios.put(presignedUrl, file, {
            headers: {
                'Content-Type': 'multipart/form-data'
            },
            onUploadProgress: (event: AxiosProgressEvent) => {
                if (!event.progress || !event.total) return 0
                const progress = Math.round((event.loaded * 100) / event.total)
                onUploadProgress(progress)
            }
        })
        return { error: response.status !== 200 }
    } catch (error: any) {
        return handleError(error, [])
    }
}
```

Obrázek 6.9: Zdrojový kód metody, která nahrává soubor na zadanou url adresu. Při nahrávání souboru lze specifikovat funkci `unUploadProgress` v parametru metody, která informuje o celkovém průběhu nahrávání v procentech. Díky této funkci lze aktualizovat stav nahráván ve stavové proměnné a tím pádem i v uživatelském rozhraní.

I když se nejedná o komponentu, velmi často použitým prvkem byly tzv. TopNotifications. Tento prvek sloužil pro informování uživatele o úspěchu či neúspěchu nějaké operace.



Obrázek 6.10: Ukázka použití třídy `TopNotification` a její metody show.

Nejedná se o komponentu napsanou ve Vue nýbrž o funkci napsanou v jazyce Typescript. Tímto stylem byl tento prvek naprogramován, kvůli pohodlnějšímu způsobu použití – tyto notifikace se totiž objeví ihned po zavolaní metody `show`. Ta přijímá tři parametry: text zprávy, typ zprávy (zda se jedná o úspěch či chybu) a nepovinný atribut, který specifikuje délku zobrazení notifikace v milisekundách. Je důležité podotknout, že tento typ notifikací se liší od těch, které jsou popsány v kapitole 6.7, jelikož TopNotifications, nejsou nikde perzistentně uloženy – pouze se zobrazí ve vrchní části obrazovky a po určitém čase zmizí.

Dalšími důležitými komponentami jsou `AmazonImage` a `AmazonVideo`, které slouží pro zobrazení fotografie či videa na webové stránce z cloudového uložiště Amazon S3. Jelikož předem podepsané odkazy propůjčují přístup pouze na omezenou dobu, může se stát, že je na stránce zažádáno o objekt s již prošlým předem podepsaným odkazem. Proto v těchto komponentách je implementován mechanismus, který při vypršení přístupu, zažádá backend o nový odkaz.

Další komponenty, které jsou použity napříč celou aplikací jsou např.:

- `AsyncIcon` – podobný jako `AmazonImage` a `AmazonVideo`, ale tento prvek je použit pro načtení SVG ikon. Zde je použita direktiva v-html, aby bylo možné dodatečně stylovat danou ikonu (např. nastavovat atribut fill).
- `TopNavbar` – Navigační lišta, která se zobrazuje na domovské stránce, na stránce kategorií a na seznamu nabídek spoluprací.
- `LocalLoader` – Tato komponenta zobrazí načítací animaci na určitém místě.
- `FullscreenLoader` – Oproti předchozí komponentě, je tato použita pro překreslení celého uživatelského rozhranní a zobrazení načítání přes celou uživatelovu obrazovku
- `StyledInput` – Prvek `input` s definovanými styly
- `StyledTextarea` – Prvek `textarea` s definovanými styly

6.7 Chat a notifikace

Velmi důležitou součástí platformy je možnost zasílání zpráv, která je zde řešena pomocí živého chatu A.6. Chat lze na platformě najít na dvou místech – přímo na profilu uživatele nebo v systému vedení spolupráce. Na obou místech je použita stejná komponenta. To jak jsou chaty uloženy bylo popsáno v kapitole 4.3, každopádně připomenu, že jsou uloženy v nerelační databázi MongoDB, aby byla zaručena persistence dat. Na uložení či zobrazení není nic neobvyklého. Při posílání zprávy je vložen nový objekt do pole zpráv – je jí připsáno pořadové číslo jako identifikátor a pokud by uživatel na druhé straně komunikace obnovil stránku, zpráva by se mu zobrazila.

Pro to, aby bylo možné živě aktualizovat obsah stránky protistrany bylo potřeba navázat spojení přes protokol WebSocket. K tomu byla použita knihovna Socket.io, jak v jazyce

Python tak i v jazyce JavaScript. Tato knihovna poskytuje možnost jak navázat komunikaci mezi klientem a serverem, která je oboustranná, založená na událostech a také rychlá [64]. Na klientské straně je připojení udržování pomocí globálního stavového skladu Pinia a díky tomu je jednoduché odkudkoliv přidávat naslouchače a tím pádem ovlivňovat lokální stavy komponent. Po připojení klientské strany k serveru je zaslán access token, který je na serveru ověřen a poté je uživatel uložen do slovníku všech aktuálně připojených uživatelů. Zde je uložen i se svým `SID`, které se v knihovně Socket.io používá pro směrování událostí. Po tomto připojení má tedy server možnost kdykoliv vypustit událost, na kterou bude reagovat naslouchač na klientské straně.

Proces odeslání a přijetí zprávy probíhá následujícím způsobem:

1. Odesílatel – Zasílá zprávu pomocí HTTP požadavku POST ve kterém je specifikováno: od koho zpráva je, komu se má zobrazit, obsah zprávy a čas odeslání.

2. Server – příjme tento požadavek a uloží tuto zprávu do databáze MongoDB a zároveň vytvoří notifikaci o nové zprávě pro adresáta.

3. Server – jestliže adresát není v aktuálně přihlášených uživatelích přes Socket.io, server pouze potvrdí odeslání zprávy odesílateli.

4. Server – jestliže adresát je připojený pomocí Socket.io, server vypustí událost `chat` a `notification` a směruje je adresátovi pomocí jeho SID.

5. Adresát – pokud adresát není na stránce s chaty, reaguje pouze na příchozí notifikaci.

6. Adresát – pokud je na stránce s chaty, zachytí událost `chat` a pokud má adresát aktuálně otevřený chat s odesílatelem, je mu ihned zpráva zobrazena. Jinak je tato zpráva přidána do levé části (aktualizován počet nepřečtených zpráv v levé části uživatelského rozhraní).

Notifikace fungují obdobným způsobem: na klientské straně je registrován naslouchač událostí `notifications` a při přijetí nové notifikace je zobrazena v levém dolním rohu obrazovky. Pokud na ní uživatel neklikne, lze ji nalézt pod ikonou zvonečku na uživatelském profilu (notifikace lze vidět v přílohách na obrázku A.5). Díky atributu, který určuje typ notifikace a atributu `additional_info` je možné uživatele nasměrovat přímo na místo, odkud notifikace pochází.

Kapitola 7

Testování

Testování bylo rozděleno do dvou částí: testování API a testování uživatelského rozhraní s uživateli. V následujících podkapitolách popisuji, jak jsem oba tyto typy testování prováděl a co jsem k testování využil.

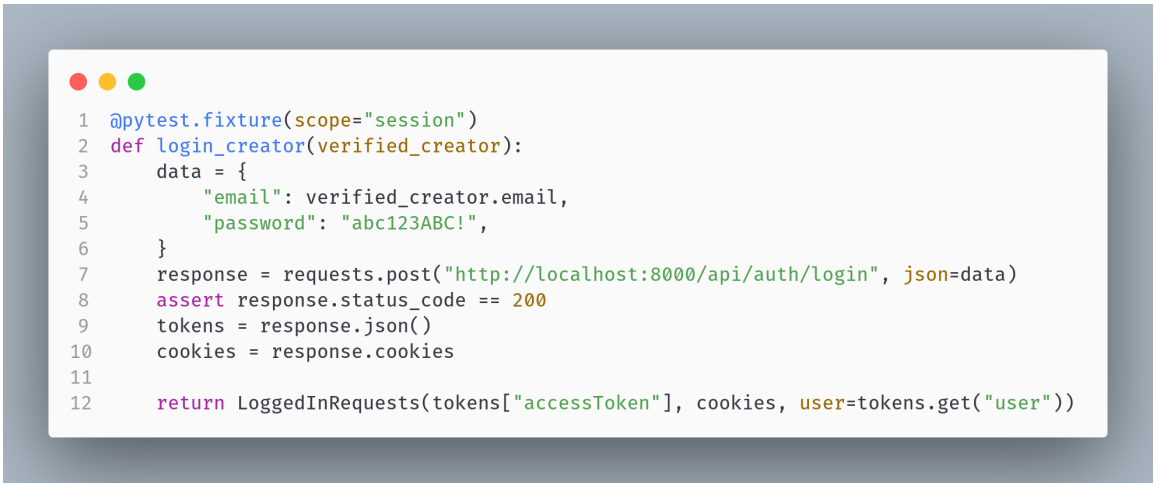
7.1 Testování API

Testování správného fungování API bylo provedeno pomocí jednotkových testů. Tyto testy jsou napsány za pomocí knihovny pytest a také knihovny requests, která slouží pro zasílání HTTP požadavků. Celkově je pomocí knihovny pytest při spuštění provedeno 283 testů, které ověřují funkčnost všech endpointů, které jsou na API dostupné. Pokud se nejedná o endpoint, který je veřejně dostupný má vždy test následující strukturu:

1. Testování zda-li je uživateli, který není přihlášen, odepřen přístup a je mu navrácen HTTP stavový kód 401.
2. Testování zda-li uživateli s jinou rolí je také odepřen přístup a navrácen HTTP stavový kód 401.
3. Testování zda-li uživatel se stejnou rolí nemá přístup ke zdrojům jiného uživatele. Zde se stavový kód může lišit dle endpointu, ale většinou se jedná o stavové kódy `401`, `403` či `404`.
4. Testování jestli je navrácen správný stavový kód při chybném tvaru dat (neexistující identifikátory v databázi atd.).
5. Testování správného fungování při zaslání všech potřebných dat ve správném tvaru.

Prakticky každý endpoint má testy napsané v samostatném souboru. Výjimky tvoří zejména endpointy, které jsou těsně spjaté. To jsou například ty, které slouží pro nahrání nějakého souboru do cloudu – generování předem podepsaných odkazů a následné potvrzení nahrání se musí testovat zároveň. Dále jsou tyto soubory členěny do složek, které jsou pojmenovány stejně, jako soubor, který obsahuje implementaci samotných testovaných funkcí. Všechny tyto složky jsou poté seskupeny do složky `testing`.

Pomocí skriptu `tester.py` jsou následně všechny tyto testy spuštěny a zároveň PyTest načte i konfigurační soubor `conftest.py`, ve kterém jsou další potřebné funkce pro testování. Jsou zde naprogramovány tzv. fixtures [29]. Tato technika umožňuje do jednotlivých testů opakovaně vkládat jakýkoliv kontext – např. předdefinovat data v databázi,



```python
@pytest.fixture(scope="session")
def login_creator(verified_creator):
    data = {
        "email": verified_creator.email,
        "password": "abc123ABC!",
    }
    response = requests.post("http://localhost:8000/api/auth/login", json=data)
    assert response.status_code == 200
    tokens = response.json()
    cookies = response.cookies

    return LoggedInRequests(tokens["accessToken"], cookies, user=tokens.get("user"))
```

Obrázek 7.1: Zdrojový kód jedné z použitých fixtures během testování API. Fixture je použita pro zasílání požadavků pod uživatelem s rolí tvůrce. Zde je také vidět, že v parametrech je specifikovaná jiná fixture, která před spuštěním této, vytvoří uživatele a vloží jej do databáze. Následně je tedy uživatel přihlášen pomocí e-mailu a hesla. Fixture poskytuje objekt třídy `LoggedInRequests`, která poskytuje základní HTTP metody pro komunikaci s přiděleným access a refresh tokenem.

přihlášení uživatele atd. Příklad fixture, která do jednotlivých testů přidává kontext přihlášeného tvůrce, je zobrazena na obrázku 7.1. Na obrázku lze také vidět, že fixtures se dají vkládat nejen do testů, ale i do dalších fixtures. Zde je to například použito k tomu, že před přihlášením uživatele je vytvořen již verifikovaný účet tvůrce, ke kterému se následně funkce `login_creator` přihlásí pomocí HTTP požadavku s přihlašovacími údaji. Pak jsou z odpovědi využity data o uživateli, access a refresh token, které jsou předány třídě `LoggedInRequests`. Tím je vytvořen nový objekt této třídy, která implementuje standardní HTTP metody – pouze k požadavkům vždy přidává již zmíněné tokeny.

Kontext se pomocí fixtures může tvořit v závislosti na parametru `scope`, který definuje kdy má být kontext vytvořen a kdy má být provedeno tzv. čištění kontextu, angl. cleanup (např. odstranění přihlášeného uživatele). Uvádím zde pro příklad tři možnosti použití parametru scope:

1. `scope='function'` – kontext se tvoří na začátku každého jednotkového testu a na konci něj je proveden cleanup,
2. `scope='class'` – kontext je tvořen pro každou třídu testů,
3. `scope='session'` – kontext se tvoří při spuštění testů a cleanup je provede až po dokončení všech testů.

Např. pro tvorbu přihlášených uživatelů byla použita hodnota atributu `scope='session'`, ale pro většinu dalších testů bylo potřeba tvořit kontext pro každý text zvlášť, aby byla zachována nezávislost jednotlivých testů.

7.2 Testování uživatelského rozhraní

V minulé kapitole bylo popsáno testování API pomocí jednotkových testů. Tato kapitola popisuje, jak byla aplikace testována s potenciálními uživateli aplikace, od kterých jsem původně sesbíral i požadavky na tuto aplikaci. Testování probíhalo ze dvou důvodů:

- **Testování použitelnosti** – tento typ testování slouží pro ověření, že uživatelské rozhraní je pro uživatele intuitivní a atraktivní.
- **Ověření fungování frontendu** – zároveň při testování použitelnosti aplikace je zkoumáno, že aplikace i na klientské straně funguje správně.

Testování aplikace jsem prováděl jak v průběhu implementace, tak i v konečné fázi. V průběhu implementace byl především kladen důraz na přehlednost uživatelského rozhraní, tedy nešlo o klasické testování, spíše se jednalo o zpětnou vazbu, která byla nutná pro navržení přehledného uživatelského rozhraní. Rozdíl mezi původně navrženým a finálním uživatelským rozhraním je zobrazen na obrázku 7.2.

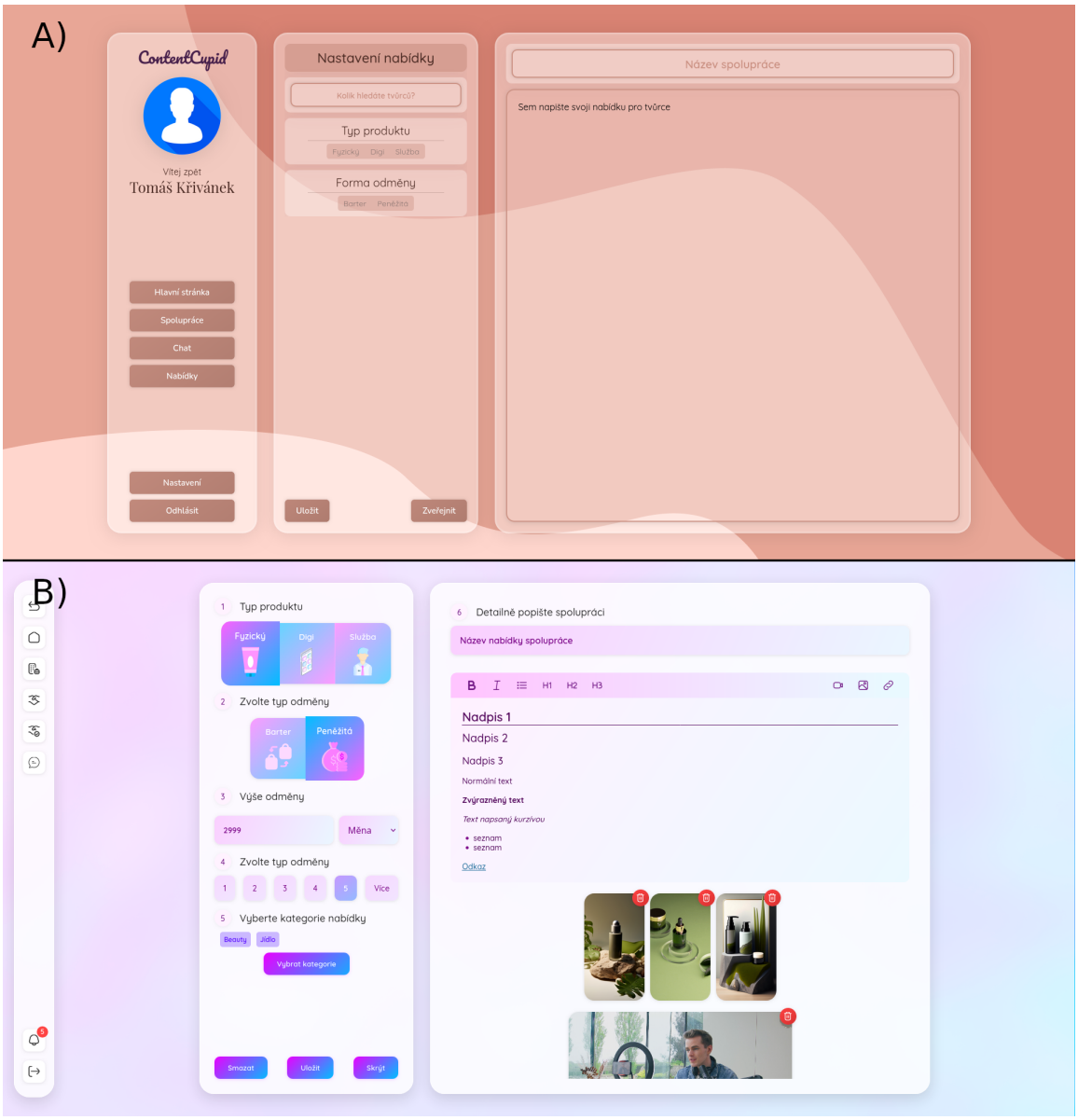
Testování použitelnosti bylo inspirováno knihou Steva Kruga [36], která vysvětluje důležitost, principy a způsob vedení tohoto testování. K této knize je také spousta příkladných materiálů dostupných online[1], mezi kterými jsou dostupné např. testovací scénář, vzor souhlasu s nahráváním sezení atd. Toto testování je založeno na pozorování uživatelů s novým uživatelským rozhraním a je kladen důraz na popis aplikace očima uživatele. Během testování uživatelé popisují jak se systémem interagují a zdůvodňují proč dělají určité kroky. Během sezení je pořizován audio záznam rozhovoru a video záznam obrazovky, na které je prováděno testování aplikace. Uživatelé při testování mají k dispozici vzorové vstupy, aby se neztrácel čas např. vymýšlením nabídky spolupráce nebo vybíráním profilového obrázku – ty mají během testování připravené.

Testování použitelnosti aplikace Content Cupid probíhalo ve 4 krocích:

1. Seznámení s průběhem testování – v této fázi bylo uživateli vysvětleno, jak testování probíhá. Je seznámen s principem testování a co se od něj očekává. Právě zde bylo uživatelům vysvětleno, že by měli popisovat, co, kde ve webové aplikaci provádějí a proč se k tomu rozhodli.
2. Stručný popis aplikace – i když uživatelé byli předem seznámeni s účelem aplikace, bylo důležité jim krátce nastínit co tato webová aplikace přináší a k čemu bude použita.
3. Provedení testovacího scénáře/ů – zde již probíhalo samostatné testování aplikace.
4. Zhodnocení, otázky, připomínky – uživatelé po testování ještě mohly klást otázky či říci připomínky k uživatelskému rozhraní. Zde měli prostor specifikovat např. nápady na zlepšení aplikace a funkčnosti.

Hlavní části testování je tedy krok číslo 3, kde uživatel provádí akce, které jsou specifikovány testovacím scénářem (příklady testovacích scénářů jsou v příloze B). Pro přehledné zobrazení těchto scénářů, byla navržena i samostatná jednoduchá webová stránka, která na základě popisu v souboru ve formátu JSON, vygeneruje přehledný seznam činností. Tento seznam byl následně použit během testování jako opora.

[1] `https://sensible.com/download-files/`



Obrázek 7.2: Rozdíl mezi původním návrhem uživatelského rozhraní (obrázek A) a finálním návrhem uživatelského rozhraní (obrázek B).

Aplikace byla opakovaně testována 5 tvůrci a 2 manažery sociálních sítí. Každý z uživatelů prováděl několik testovacích scénářů, které se týkali jeho role v systému. S těmito uživateli bylo testování vedeno při osobním setkání nebo online pomocí videohovoru a sdílení obrazovky. Jednotlivé úkoly jsem jim sdělil a nalezené nedostatky systému jsem si zapisoval. Díky záznamu obrazovky jsem mohl dále analyzovat nedostatky, které se týkali např. rozložení tlačítek na jednotlivých stránkách. Před každým testováním byl systém naplněn daty tak, aby systém nepůsobil prázdně – bylo vytvořeno několik uživatelů s různými rolemi, dále několik nabídek spoluprací nebo také žádosti na tyto nabídky.

7.3 Výsledky testování

Během testování API byla odhalena spousta nedostatků. Většina z nich se týkala pouze okrajových podmínek, ale našly se i takové chyby, které velmi narušovali práva uživatelů. Tyto chyby vzešli ze skutečnosti, že jsem zároveň implementoval jak backend tak frontend. Na frontendu aplikace je takovéto narušení prakticky nemožné, jelikož zde uživatel nemá možnost specifikovat přesné parametry požadavku. Pokud by ale útočník manuálně specifikoval přesný HTTP požadavek, mohl by využít některých slabin, které se mi během testování podařilo objevit. Uvádím několik chyb, které byly objeveny během testování API a následně opraveny:

- O žádosti tvůrce mohl rozhodovat i manažer, který nevytvořil nabídku,
- tvůrci měli možnost žádat i o nezveřejněné nabídky,
- při načtení detailu nabídky se v odpovědi na požadavek zasílali i informaci o žádostech jiných tvůrců,
- při načítání nových zpráv z chatu byla odhalena chyba, kdy při specifikování špatného identifikátoru zprávy aplikace spadla.

Během testování uživatelského rozhraní byl nalezen nedostatek hned v počátku vývoje. Tento nedostatek již byl zmíněn dříve a vedl k přepracování designu. Prvním důvodem přepracování uživatelského rozhraní byly špatně zvolené barvy, které uživatelům nepřipomínali prostředí sociálních sítí – proto bylo následně zvoleno barevné schéma založené na modré a růžové barvě. Dalším a důležitějším nedostatkem bylo to, že uživatelé už jen na domovské stránce svého profilu byly přehlceni informacemi – jelikož se zde objevovali různé sekce aplikace do jedné. Kvůli tomu byla aplikace rozčleněna na více podstránek. A posledním důvodem byla kombinace dvou předchozích. Kombinace hnědých odstínů a velkého množství informací na jedné stránce dala za výsledek, že uživatelé často nevěděli, kde se co v aplikaci nachází a kde by jejich požadovanou funkci hledali.

Během testů použitelnosti byli objeveny následující nedostatky a chyby:

- při prvních testech se neodesílal potvrzovací e-mail – to bylo zapříčiněno zastaralými informacemi v konfiguraci souboru,
- při zadání správného kódu při verifikaci účtu byl uživatel přesměrován na stránku "Unauthorized", kvůli přehození operací – aktualizování stavové proměnné a přesměrování,
- po přihlášení do účtu se nedokončilo načítání propagačních materiálů tvůrce – pro tato data je totiž potřeba separátní požadavek na API a při přihlášení se neposílal identifikátor tvůrce, který tento požadavek potřebuje jako parametr,

- při nezadání názvu nabídky spolupráce nebyla ukazována chybová hláška,
- při kliknutí na notifikaci, která uživateli oznamovala přijetí jeho žádosti o spolupráci, nebyl přesměrován na konkrétní stránku. To bylo zapříčiněno špatným pojmenováním atributu v databázi MongoDB,
- při kliknutí na notifikaci, která odkazovala na stránku, kde se již uživatel nacházel, nedošlo k načtení nových údajů z databáze. Např. pokud manažer již byl v systému vedení spolupráce s tvůrcem a tento tvůrce nahrál nějaký materiál. Manažer byl sice přesměrován na správnou sekci systému vedení spolupráce, ale nahraný materiál se nezobrazil – to bylo opraveno dodatečným dotazem na API, při obdržení notifikace,
- další změnou bylo smazání možnosti upravit externí odkaz na profilu tvůrce, jelikož všichni uživatelé u testování této funkcionality nepoužili možnost upravit externí odkaz, ale smazali jej a přidali nový. Tlačítko upravit by pak bylo zbytečné a překáželo by,
- mnoho uživatelů také zmátli vstupy při registraci, kdy první vstupní pole sloužilo pro celé jméno a druhé pro přezdívku – uživatelé sami od sebe předpokládali, že křestní jméno a příjmení bude v kolonkách zvlášť a tak své křestní jméno psali do pole pro celé jméno a příjmení psali do pole pro přezdívku. Této chybě je nyní předcházeno tím, že místo vstupního pole pro celé jméno je při registraci požadováno křestní jméno a příjmení zvlášť.
- Během nahrávání profilové fotografie byli uživatelé zmatení a čekali na nahrání fotografie i když měli správně přejít na další krok, kde profilovou fotografii potvrdí. To bylo způsobeno tím, že i při nahrávání profilové fotografie se pod jménem nahrávaného souboru zobrazoval prostor pro zobrazení průběhu nahrávání. Jakmile se tento prvek odstranil, uživatelé již přecházeli na další krok sami.
- Tvůrci také při prohledávání nabídek spoluprací chtěli vyhledávat podle názvu firmy – to bylo po testování přidáno.

Všechny nalezené chyby, jak během testování API, tak během testování s uživateli byli následně opraveny a znovu otestovány. Z finálního testování použitelnosti webové aplikace vzešlo i několik nových nápadů na rozšíření aplikace, které budou popsány v následující kapitole.

Kapitola 8

Budoucí rozšíření systému

I když aplikace aplikace splňuje všechny uživatelské požadavky, tak již během testování během implementace a i při finálním testování byla navržena možná rozšíření platformy. Webová aplikace i po odevzdání této práce bude dále iterativně zlepšována a obohacena o další funkcionality zmíněné v této kapitole.

První funkcionalitou, která byla navržena byla možnost manažerů kontaktovat přímo určité tvůrce. Pokud by manažer měl přehled o profilech tvůrců mohl by si vybrat konkrétního z nich a pomocí chatu jej kontaktovat. Po vzájemné domluvě by mohl manažer specifikovat detaily nové spolupráce a odstartovat ji jen na základě komunikace s konkrétním tvůrcem bez použití nabídek spolupráce. Dalším rozšířením by mohlo být i žádost opačným směrem – tedy manažeři by měli možnost pouze zvolit, že jejich firma hledá nové tvůrce, ale nespecifikovali by konkrétní nabídku. Tvůrci by pak mohli konkrétní firmu a manažera kontaktovat přímo a domluvit se na vzájemné spolupráci.

Vzájemná domluva se týká i částky, kterou za danou spolupráci tvůrci obdrží. Dalším budoucím rozšířením aplikace by tedy byla možnost specifikovat odměnu dohodou – tedy tvůrce by se s manažerem na ceně domluvili. Manažer by při specifikaci nabídky spolupráce zvolil možnost placené spolupráce, ale specifikoval by např. rozpočet, který má k dispozici nebo by nespecifikoval vůbec nic a cena by tak závisela pouze na dohodě s konkrétním tvůrcem.

Další rozšíření se týká také spoluprací. Často se stává, že jeden tvůrce vytváří materiály pro jednu firmu opakovaně (např. každý měsíc poskytuje firmě 3 videa). V systému by měla být možnost řídit i takovéto formy dlouhodobé spolupráce. Zde by se dal využít již stávající systém vedení spoluprací, pouze by zde manažer měl možnost např. začít novou spolupráci na základě stávající nebo by mohl nastavit v jakých intervalech by se měla spolupráce opakovat (např. každý měsíc).

Tvůrci i firmy by měli mít také možnost vzájemně se hodnotit. Na této funkcionalitě se již pracuje (byla zakomponována do návrhu uživatelského rozhraní i do návrhu databázového schématu). Tvůrci by mohli po dokončení spolupráce firmu ohodnotit a popsat např. rychlost komunikace, počet požadovaných oprav poskytnutých materiálu atd. Manažeři by zase mohli hodnotit poskytnuté materiály, jak byli spokojení s výsledky spolupráce a např. jak dlouho tvůrce požadované materiály tvořil.

Dalším velmi důležitým rozšířením webové aplikace by spočívalo v přidání administračního systému, jelikož ten v aplikaci implementován není. Administrátoři by zde měli přehled o registrovaných uživatelích, nabídkách spoluprací i jejich průběhu. Měli by možnost dohlížet na celý systém – blokovat uživatele, kteří porušují pravidla platformy atd. Dále by zde

měla být možnost i dodatečně potvrdit tvůrce na základě jeho poskytnutých materiálů – zvýšila by se tak úroveň profesionality tvůrců, kteří by platformu používali.

Dalším technickým vylepšením aplikace by určitě bylo šifrování zpráv, které si mezi sebou zasílají uživatelé prostřednictvím chatu. Zprávy se v současné verzi aplikace ukládají v MongoDB v otevřené podobě, což není ideální vůči důvěrnosti uživatelů.

Posledním navrženým rozšířením by byla mobilní aplikace na operační systémy Apple iOS a Android. Tato mobilní aplikace by byla spíše zaměřená na tvůrce, jelikož manažeři ve většině případů pracují na počítači. Mobilní aplikace by tak tvůrcům usnadnila hledání spoluprací i jejich samostatný průběh. Měli by zde, stejně jako ve webové aplikaci přehled o nabídkách spoluprací a mohli by o ně žádat. Následně by zde byla možnost sledovat své probíhající spolupráce a jelikož většina tvůrců tvoří svůj materiál pomocí mobilních zařízení, usnadnilo by to i samotné nahrávání na platformu. Zároveň by při např. natáčení videa pro spolupráci měli specifikaci přímo u sebe.

Kapitola 9

Závěr

Hlavním cílem této diplomové práce bylo navrhnout a dále implementovat systém, který bude pomáhat tvůrcům uživatelsky generovaného obsahu a firmám. Aplikace nabízí možnost manažerům sociálních sítí firem specifikovat nabídky spolupráce pomocí jednoduchého textového editoru s možností dodatečného nastavení parametrů spolupráce. Tvůrci na tyto nabídky mohou zasílat žádosti. Aby manažeři měli možnost srovnaní jednotlivých žádostí, mají možnost nahlédnout do portfolia tvůrce, které si předem sám vytváří. Zde má možnost specifikovat, jak textový popis, odkazy na jejich sociální sítě a hlavně na svém portfoliu mohou prezentovat svou již odvedenou práci, tedy ukázat videa a fotografie, které již tvůrci v minulosti vytvořili. Na základě těchto poskytnutých informací si manažeři vybírají konkrétní tvůrce na spolupráci. Hlavní součástí platformy je systém vedení spolupráce, který má za úkol oběma typům uživatelů usnadnit a urychlit průběh založené spolupráce. Zde manažeři specifikují požadavky na vytvářený materiál tvůrcem a následně tento materiál mají možnost odmítnout s výhradami či jej přijmout. Platforma také dodává jednoduchou ochranu tvůrce při placených spolupracích. Aby se zamezilo krádeži poskytnutých materiálů je u videí a fotografií použit vodoznak, který tvůrcem poskytnutý materiál chrání.

Pro vývoj klientské strany aplikace byly porovnány dva frameworky Angular a Vue a knihovna React. Z nich po detailním porovnání byl vybrán framework Vue. Pro implementaci REST API byl vybrán framework FastAPI. Aplikace dále bude využívat databázové systémy MySQL a MongoDB. Kromě databázových systémů bude využívat i cloudové úložiště Amazon S3, kvůli potřebné škálovatelnosti, jelikož se v aplikaci bude vyskytovat velké množství multimediálního obsahu.

Tyto technologie byly navrženy po detailním sběru a analýze požadavků od potenciálních uživatelů tohoto systému. Požadavky jsem sbíral pomocí předem sepsaných otázek na osobních a on-line setkáních s potenciálními uživateli. Kromě konkrétních technologií byla navržena i struktura databáze pomocí ER diagramu. Dále bylo navrženo uživatelské rozhraní pomocí wireframů a jejich následnou úpravou je vytvořen finální grafický návrh.

Během i po implementaci celého systému byla aplikace řádně otestována. Funkčnost API byla ověřena pomocí jednotkových testů, pro které byla využita knihovna Pytest. Při testování s uživateli byl kladen důraz na testování použitelnosti systému a na jeho správné fungování.

V poslední části technické zprávy byla popsána možná rozšíření webové aplikace. Tyto rozšíření byly sesbírány od potenciálních uživatelů během testování a většina z nich se týká nových forem vedení a zakládání spoluprací. Dalším navrženým rozšířením by byla mobilní aplikace, která by byla více zaměřena na potřeby tvůrce, který mobilní telefon velmi často využívá při pořizování požadovaných materiálů.

Výsledkem je tedy plně funkční a řádně otestovaná aplikace Content Cupid, která slouží pro propojení tvůrců uživatelsky generovaného obsahu a manažerů sociálních sítí. Aplikace bude v budoucnu doplněna o další funkcionality, které byly zmíněny v kapitole 8. Po zapracování těchto rozšíření bude publikována na doméně contentcupid.cz, která je již teď předem zakoupená.

Literatura

[1] Amazon Web Services. *Amazon Simple Storage Service User Guide*. Amazon Web Services, 2024 [cit. 2023-04-25]. Dostupné z: https://docs.aws.amazon.com/s3/.

[2] *Angular: Angular components overview*. [cit. 2023-10-12]. Dostupné z: https://angular.io/guide/component-overview.

[3] *Angular: Introduction to Angular concepts*. [cit. 2024-01-10]. Dostupné z: https://angular.io/guide/architecture.

[4] Ast, M. Incremental DOM for Web Components. *Studierendensymposium Informatik 2016 der TU Chemnitz*. 2016.

[5] *Back4app: What Are The Top 10 Backend Programming Languages in 2023?* [cit. 2024-01-13]. Dostupné z: https://blog.back4app.com/backend-programming-languages/#Top_10_Backend_Programming_Languages.

[6] Bahtar, A. Z. a Muda, M. The Impact of User – Generated Content (UGC) on Product Reviews towards Online Purchasing – A Conceptual Framework. *Procedia Economics and Finance*. 1. vyd. 2016, sv. 37, č. 1, s. 337–342. DOI: https://doi.org/10.1016/S2212-5671(16)30134-4. ISSN 2212-5671. The Fifth International Conference on Marketing and Retailing (5th INCOMaR) 2015. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2212567116301344.

[7] Bampakos, A. a Deeleman, P. *Learning Angular*. 4. vyd. Birmingham, England: Packt Publishing, únor 2023.

[8] Barisberkemalkoc. *Axios Interceptor*. 6. srpna 2023 [cit. 2024-04-24]. Dostupné z: https://medium.com/@barisberkemalkoc/axios-interceptor-intelligent-db46653b7303.

[9] Bhaskar, A. a A.E, M. An Interpretation and Anatomization of Angular: A Google Web Framework. In: *International Research Journal of Engineering and Technology (IRJET)*. 2020, sv. 7, č. 5, s. 7613–7169.

[10] Bogner, J. a Merkel, M. To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and Typescript Applications on GitHub. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2022, s. 658–669. MSR '22. DOI: 10.1145/3524842.3528454. ISBN 9781450393034. Dostupné z: https://doi.org/10.1145/3524842.3528454.

[11] Bouchard, N. *What Is a UGC Creator? Here is How To Become One* [online]. Červen 2023 [cit. 2023-08-21]. Dostupné z: https://www.theleap.co/blog/how-to-become-ugc-content-creator/.

[12] Buczkowski, M. *Python REST frameworks performance comparison.* 10. srpna 2020 [cit. 2024-01-15]. Dostupné z: https://www.grandmetric.com/python-rest-frameworks-performance-comparison/.

[13] Chau, G. *Vue.js 2 Web Development Projects: Learn Vue.js by building 6 web apps.* Paperback. Packt Publishing, listopad 2017. 398 s. ISBN 978-1787127463.

[14] Chechique, E. *Designing button states: Tutorial and best practices.* 8. listopadu 2023 [cit. 2024-04-23]. Dostupné z: https://blog.logrocket.com/ux-design/designing-button-states/.

[15] Chinnathambi, K. *Learning React.* 1. vyd. Addison-Wesley Professional, 2016. ISBN 0134546318.

[16] Chodvadiya, S. *Building a Real-Time Chat Application with FastAPI and WebSocket.* 22. července 2022 [cit. 2024-01-20]. Dostupné z: https://medium.com/@chodvadiyasaurabh/building-a-real-time-chat-application-with-fastapi-and-websocket-9965778e97be.

[17] Jacobs, R. *Couchbase: The Top 8 Best Languages for Backend Development.* 28. listopadu 2022 [cit. 2024-01-13]. Dostupné z: https://www.couchbase.com/blog/backend-languages/.

[18] *Developer Ecosystem: Python.* 2023 [cit. 2024-01-14]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2022/python/.

[19] Diniz Junior, R. N., Figueiredo, C. C. L., De S.Russo, G., Bahiense Junior, M. R. G., Arbex, M. V. et al. Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In: *2022 XVLIII Latin American Computer Conference (CLEI).* 2022, s. 1–9. DOI: 10.1109/CLEI56649.2022.9959901.

[20] Diniz Junior, R. N., Figueiredo, C. C. L., De S.Russo, G., Bahiense Junior, M. R. G., Arbex, M. V. et al. Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In: *2022 XVLIII Latin American Computer Conference (CLEI).* 2022, s. 1–9. DOI: 10.1109/CLEI56649.2022.9959901.

[21] *Django 3.1 release notes.* 4. srpna 2023 [cit. 2024-01-14]. Dostupné z: https://docs.djangoproject.com/en/5.0/releases/3.1/.

[22] Dory, M., Parrish, A. a Berg, B. *Introduction to Tornado: Modern Web Applications with Python.* Paperback. O'Reilly Media, duben 2012. 142 s. ISBN 978-1449309077.

[23] Eby, P. J. *PEP 3333.* 26. září 2010 [cit. 2024-01-15]. Dostupné z: https://peps.python.org/pep-3333/.

[24] Filipova, O. *Learning Vue.js 2: Learn how to build amazing and complex reactive web applications easily with Vue.js*. Paperback. Packt Publishing, prosinec 2016. 334 s. ISBN 978-1786469946.

[25] Flanagan, D. *JavaScript: The definitive guide*. 6. vyd. Sebastopol, CA: O'Reilly Media, květen 2011.

[26] *Flask: Using async and await*. 22. května 2021 [cit. 2024-01-15]. Dostupné z: https://flask.palletsprojects.com/en/latest/async-await/.

[27] Geetha, G., Mittal, M., Prasad, K. M. a Ponsam, J. G. Interpretation and Analysis of Angular Framework. In: *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*. 2022, s. 1–6. DOI: 10.1109/ICPECTS56089.2022.10047474.

[28] Geyser, W. *22 Best UGC Platforms to Help Boost your content marketing* [online]. Srpen 2023 [cit. 2023-08-21]. Dostupné z: https://influencermarketinghub.com/user-generated-content-platforms/.

[29] holger krekel and pytest-dev team. *About fixtures*. 2024 [cit. 2024-04-29]. Dostupné z: https://docs.pytest.org/en/latest/explanation/fixtures.html.

[30] Horiachko, A. *How to Choose the Best Technology Stack for Web Application Development: 10 Helpful Tips* [online]. Březen 2023 [cit. 2023-09-27]. Dostupné z: https://www.softermii.com/blog/10-tips-in-choosing-the-best-tech-stack-for-your-web-application.

[31] Ubl, M. *Introducing Incremental DOM*. 9. července 2015 [cit. 2024-01-11]. Dostupné z: https://medium.com/google-developers/introducing-incremental-dom-e98f79ce2c5f.

[32] Kleppmann, M. *Designing Data-Intensive Applications*. Beijing: O'Reilly, 2017. ISBN 978-1-4493-7332-0. Dostupné z: https://www.safaribooksonline.com/library/view/designing-data-intensive-applications/9781491903063/.

[33] Komperla, V., Pratiba, D., Ghuli, P. a Pattar, R. React: A detailed survey. *Indonesian Journal of Electrical Engineering and Computer Science*. Institute of Advanced Engineering and Science. červen 2022, sv. 26, č. 3, s. 1710. DOI: 10.11591/ijeecs.v26.i3.pp1710-1717. Dostupné z: https://doi.org/10.11591/ijeecs.v26.i3.pp1710-1717.

[34] Korsun, J. *Djangostars: Python for Web Development: Pros and Cons*. 25. prosince 2023 [cit. 2024-01-13]. Dostupné z: https://djangostars.com/blog/python-web-development/.

[35] Koďousková, B. *Co je wireframe webu, proč ho potřebujete a jak ho vytvořit?* 8. října 2023 [cit. 2024-01-24]. Dostupné z: https://medium.com/@chodvadiyasaurabh/building-a-real-time-chat-application-with-fastapi-and-websocket-9965778e97be.

[36] Krug, S. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. 1st. USA: New Riders Publishing, 2009. ISBN 0321657292.

[37] Krug, S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3rd. USA: New Riders Publishing, 2014. ISBN 0321965515.

[38] Lathkar, M. *High-Performance Web Apps with FastAPI: The Asynchronous Web Framework Based on Modern Python*. Apress, 2023. ISBN 9781484291788. Dostupné z: http://dx.doi.org/10.1007/978-1-4842-9178-8.

[39] Lock, A. *ASP. NET core in Action*. Simon and Schuster, 2023.

[40] McNally, M. B., Trosow, S. E., Wong, L., Whippey, C., Burkell, J. et al. User-generated online content 2: Policy implications. *First Monday*. 1. vyd. Jun. 2012, sv. 17, č. 6. DOI: 10.5210/fm.v17i6.3913. Dostupné z: https://firstmonday.org/ojs/index.php/fm/article/view/3913.

[41] MDN Web Docs. *HTTP Cookies*. 2024 [cit. 2024-04-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

[42] Mirakyan, M. *Multithreading VS Multiprocessing VS Asyncio in Python*. 22. března 2022 [cit. 2024-01-15]. Dostupné z: https://peps.python.org/pep-0484/.

[43] Morris, J. *11 Powerful UGC Platforms to Supercharge Your Content Marketing Strategy in 2023* [online]. Srpen 2023 [cit. 2023-08-21]. Dostupné z: https://taggbox.com/blog/ugc-platforms/.

[44] Naem, M. a Okafor, S. User-Generated Content and Consumer Brand Engagement. In: University of Worcester. *Leveraging Computer-Mediated Marketing Environments*. IGI Global, 2019, s. 193–220. DOI: 10.4018/978-1-5225-7344-9.ch009. ISSN 2327-5502. Dostupné z: https://doi.org/10.4018/978-1-5225-7344-9.ch009.

[45] Nealson, K. *The Internet Has Previously Let You Buy; Now It's Letting You Shop* [online]. Září 2021 [cit. 2023-08-21]. Dostupné z: https://www.forbes.com/sites/forbesbusinesscouncil/2021/09/02/the-internet-has-previously-let-you-buy-now-its-letting-you-shop/.

[46] Norton, A. *A Comparison of popular backend programming languages like Node.js, Python and Java*. 26. dubna 2023 [cit. 2024-01-13]. Dostupné z: https://www.linkedin.com/pulse/comparison-popular-backend-programming-languages-like-aria-norton.

[47] Novac, C. M., Novac, O. C., Sferle, R. M., Gordan, M. I., BUJDOSÓ, G. et al. Comparative study of some applications made in the Vue.js and React.js frameworks. In: *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*. 2021, s. 1–4. DOI: 10.1109/EMES52337.2021.9484149.

[48] *OECD Information Technology Outlook 2006*. OECD, říjen 2006. Dostupné z: https://doi.org/10.1787/it_outlook-2006-en.

[49] Peyrott, S. E. *JWT Handbook*. Auth0 Inc., 2016.

[50] *Polymer library*. 2020 [cit. 2024-01-15]. Dostupné z: https://polymer-library.polymer-project.org/.

[51] Pressman, R. *Software Engineering: A Practitioner's Approach*. 7. vyd. USA: McGraw-Hill, Inc., 2009. ISBN 0073375977.

[52] Ramírez, S. *Background Tasks*. 13. dubna 2024. Dostupné z: https://fastapi.tiangolo.com/tutorial/background-tasks/.

[53] *React: Reconciliation*. [cit. 2023-10-05]. Dostupné z: https://legacy.reactjs.org/docs/reconciliation.html.

[54] *React: Virtual DOM and Internals*. [cit. 2023-10-05]. Dostupné z: https://legacy.reactjs.org/docs/faq-internals.html.

[55] *React: Components and props*. [cit. 2023-10-05]. Dostupné z: https://legacy.reactjs.org/docs/components-and-props.html.

[56] *React: Component*. [cit. 2023-10-05]. Dostupné z: https://react.dev/reference/react/Component.

[57] *Geeksforgeeks: Why it is Recommended to use Functional Components over Class Components ?* [cit. 2023-10-05]. Dostupné z: https://www.geeksforgeeks.org/why-it-is-recommended-to-use-functional-components-over-class-components/.

[58] *React: State: A Component's Memory*. [cit. 2023-10-11]. Dostupné z: https://react.dev/learn/state-a-components-memory.

[59] *React: Synchronizing with Effects*. [cit. 2023-10-11]. Dostupné z: https://react.dev/learn/synchronizing-with-effects.

[60] *React: Hooks at a Glance*. [cit. 2023-10-11]. Dostupné z: https://legacy.reactjs.org/docs/hooks-intro.html.

[61] *React: Glossary of React Terms*. [cit. 2024-01-10]. Dostupné z: https://legacy.reactjs.org/docs/glossary.html#single-page-application.

[62] Relan, K. *Building REST APIs with Flask: Create Python Web Services with MySQL*. Apress, 2019. ISBN 9781484250228. Dostupné z: http://dx.doi.org/10.1007/978-1-4842-5022-8.

[63] Shneiderman, B., Preece, J. a Pirolli, P. Realizing the Value of Social Media Requires Innovative Computing Research. *Commun. ACM*. 1. vyd. New York, NY, USA: Association for Computing Machinery. sep 2011, sv. 54, č. 9, s. 34–37. DOI: 10.1145/1995376.1995389. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/1995376.1995389.

[64] Socket.io. *Socket.io Documentation*. 2024 [cit. 2024-04-26]. Dostupné z: https://socket.io/docs/v4/.

[65] *StackOverflow: Developer Survey 2023*. [cit. 2024-01-13]. Dostupné z: https://survey.stackoverflow.co/2023/#most-popular-technologies-language.

[66] Tetka, A. *Tornado vs. FastAPI: Why We Made the Switch*. 23. dubna 2023 [cit. 2024-01-15]. Dostupné z: https://dzone.com/articles/comparing-tornado-and-fastapi-and-why-we-made-the.

[67] Tomych, I. *10 Tips To Choose Tech Stack For Web App Development* [online]. Prosinec 2022 [cit. 2023-09-27]. Dostupné z: https://dashdevs.com/blog/10-tips-to-choose-tech-stack-for-web-app-development/.

[68] Trazey. *My first month as a UGC creator: income breakdown July 2022* [online]. Červenec 2022 [cit. 2023-08-21]. Dostupné z: https://realskindiaries.com/ugc-creator-income-breakdown-july-2022/.

[69] Vincent, W. *Django for APIs: Build web APIs with Python and Django*. WelcomeToCode, 2022. Dostupné z: https://books.google.cz/books?id=0VxwDwAAQBAJ.

[70] *Endoflife.date: Vue*. [cit. 2023-10-11]. Dostupné z: https://endoflife.date/vue.

[71] *Vue.js: Rendering mechanism*. [cit. 2023-10-18]. Dostupné z: https://https://vuejs.org/guide/extras/rendering-mechanism.html.

[72] *Vue.js: API Styles*. [cit. 2023-10-18]. Dostupné z: https://vuejs.org/guide/introduction.html#api-styles.

[73] *Vue.js: Creating a Vue Application*. [cit. 2024-01-10]. Dostupné z: https://vuejs.org/guide/essentials/template-syntax.html.

[74] *Vue.js: Built-in Directives*. [cit. 2024-01-10]. Dostupné z: https://vuejs.org/api/built-in-directives.html.

[75] *Vue.js: Lifecycle Hooks*. [cit. 2024-01-10]. Dostupné z: https://vuejs.org/guide/essentials/lifecycle.html.

[76] Allotey, C. *VueSchool: Options API vs Composition API*. Leden 2023 [cit. 2024-01-10]. Dostupné z: https://vueschool.io/articles/vuejs-tutorials/options-api-vs-composition-api/.

[77] *Vue.j: Ways of Using Vue*. [cit. 2024-01-10]. Dostupné z: https://vuejs.org/guide/extras/ways-of-using-vue.

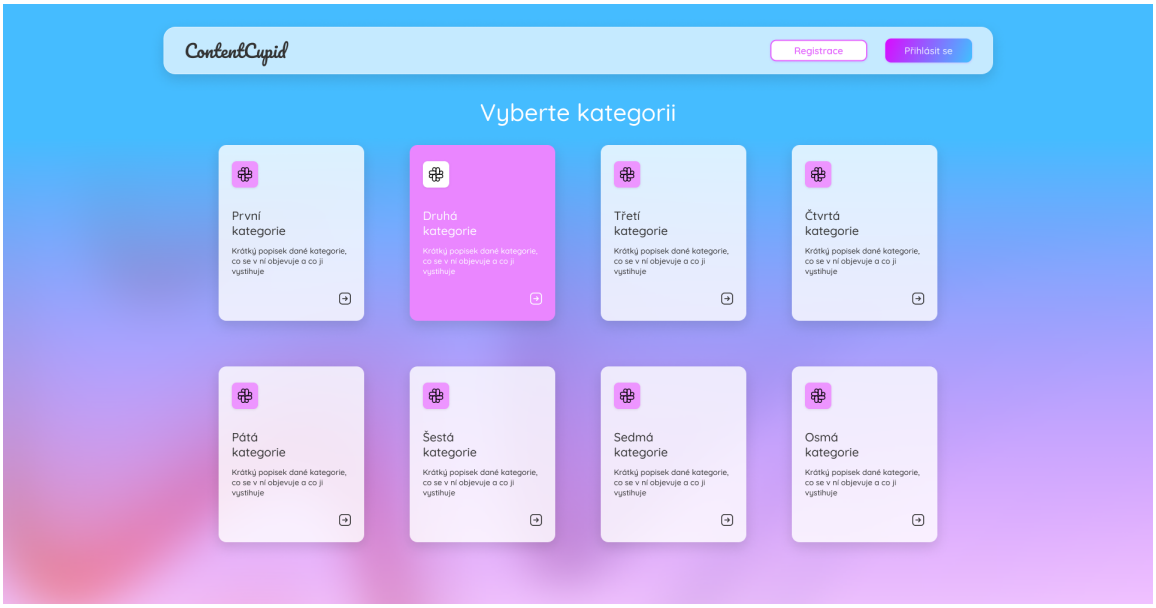
[78] Wiegers, K. E. a Beatty, J. *Software Requirements 3*. USA: Microsoft Press, 2013. ISBN 0735679665.

[79] Wikipedie. *Word of Mouth marketing — Wikipedie: Otevřená encyklopedie*. 2021. [Online; navštíveno 20. 08. 2023]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Word_of_Mouth_marketing&oldid=20356123.

[80] Zeljko, M. *Zeljko: Angular Lifecycle Hooks: A Deep Dive into Component Lifecycle*. 21. května 2023 [cit. 2023-10-12]. Dostupné z: https://miloszeljko.com/angular-lifecycle-hooks/.

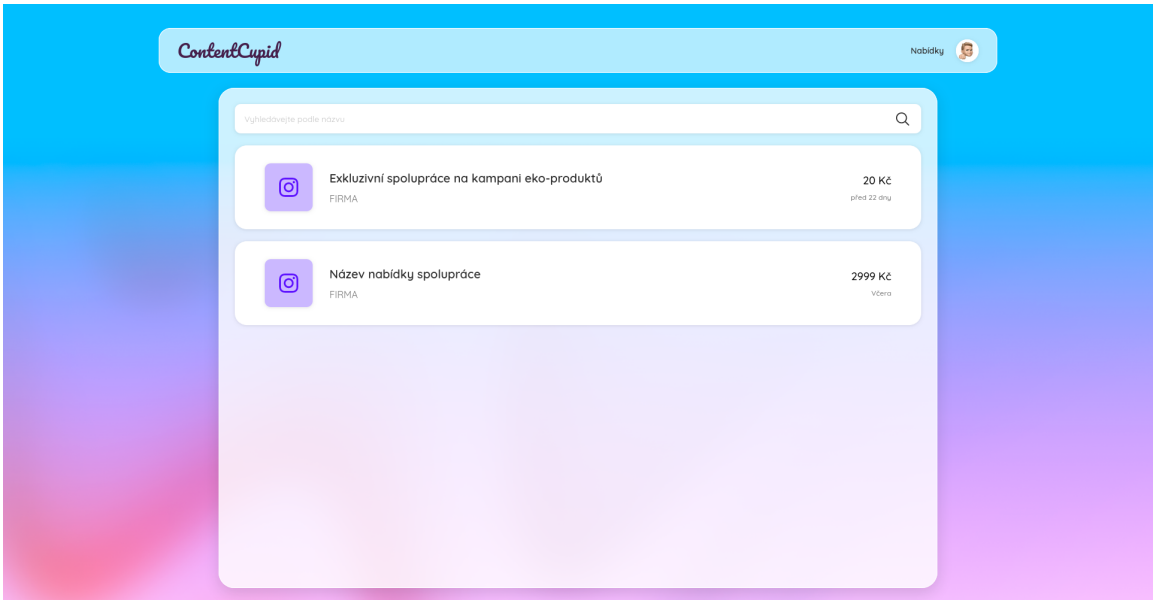
Příloha A

Návrhy uživatelského rozhraní

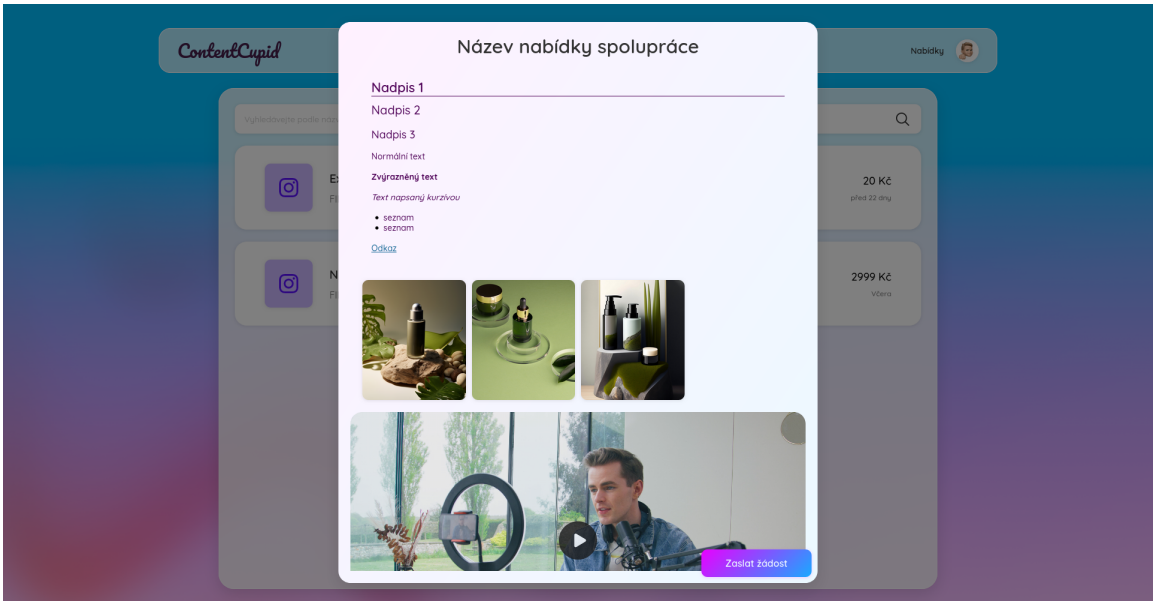


Obrázek A.1: Stránka, která zobrazuje kategorie, do kterých budou rozřazeny jednotlivé nabídky spolupráce

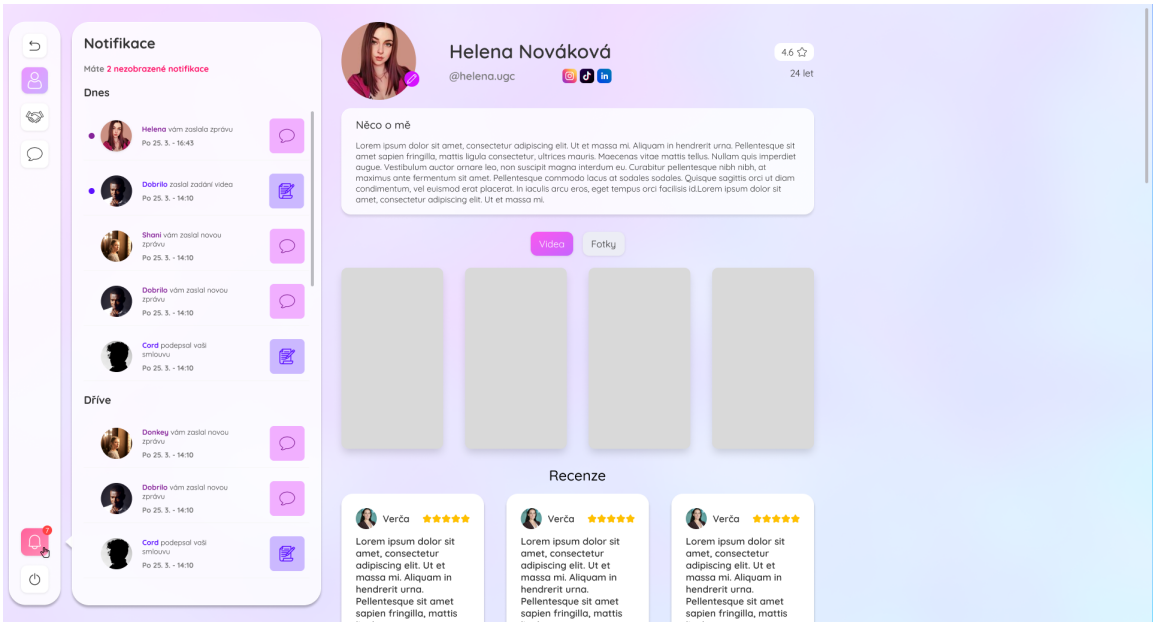
Obrázek A.2: Návrh navigační lišty, která bude umístěna ve vrchní části obrazovky.



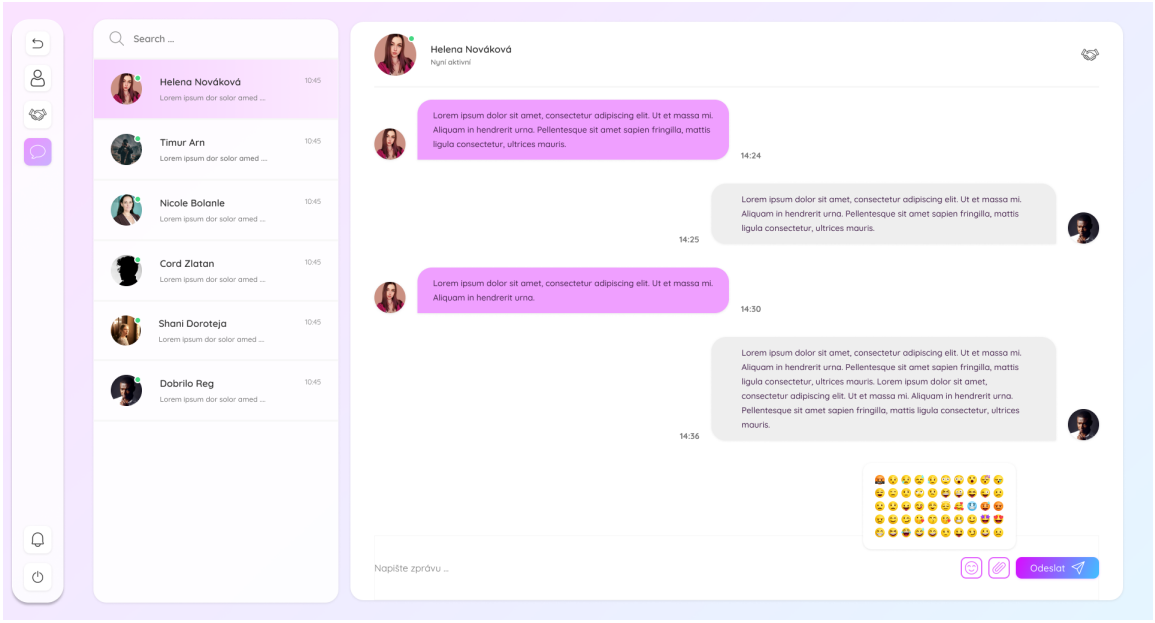
Obrázek A.3: Návrh uživatelského rozhraní, kde tvůrce uvidí jednotlivé nabídky spoluprací a jejich základní detaily.



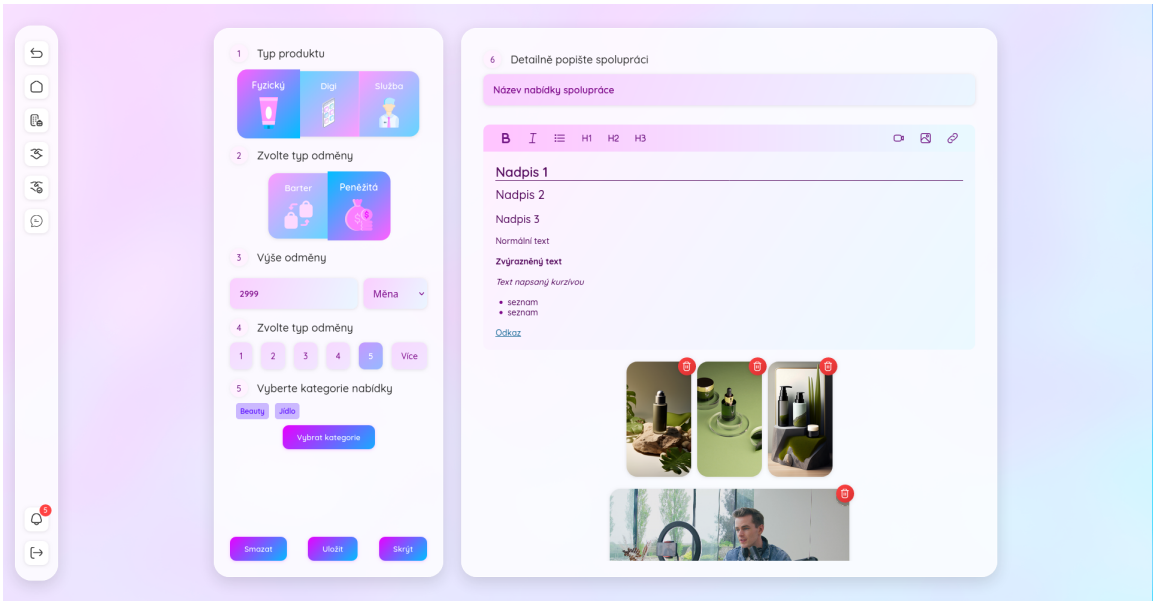
Obrázek A.4: Po kliknutí na jednu z nabídek spolupráce se zobrazí detail, kde uživatel uvidí strukturovaný text a další podrobnosti, které specifikoval manažer.



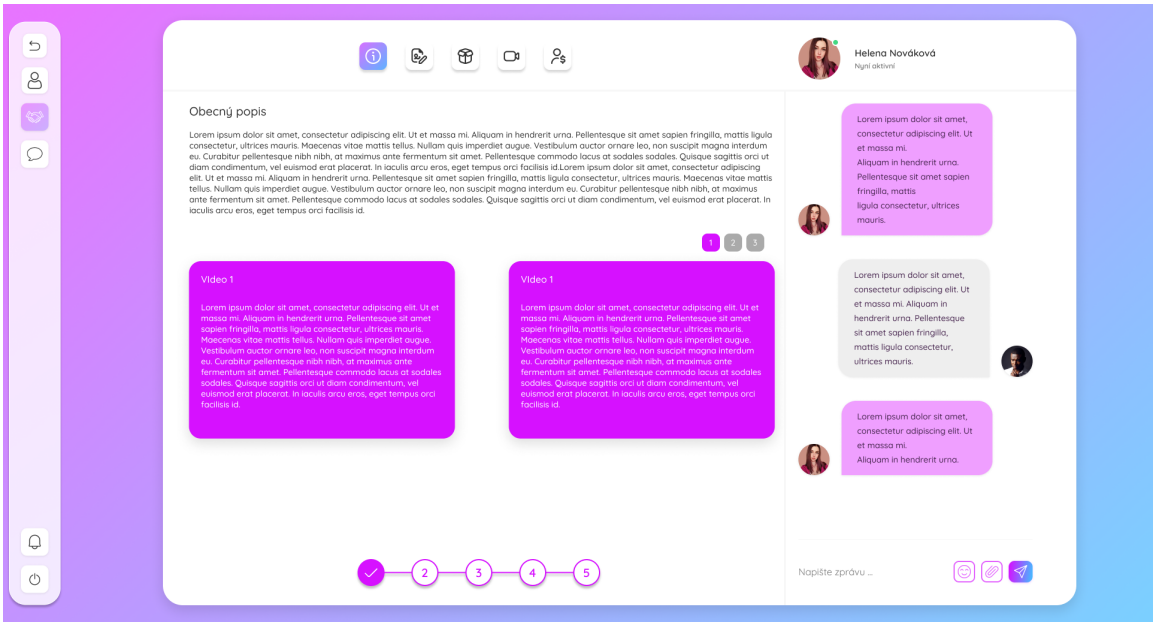
Obrázek A.5: Navržený profil pro uživatele s rolí tvůrce. Na stránce jsou zobrazeny základní údaje o uživateli: profilový obrázek, jméno, přezdívka, věk a průměrné hodnocení od ostatních. Pod tímto je zobrazen popis, který specifikuje tvůrce. Dále jsou zde reprezentační videa a fotografie mezi kterými bude možné přepínat pomocí dvojic tlačítek. Ve spodní části budou zobrazeny recenze tvůrce. V levé části obrazovky jsou zobrazeny i poslední notifikace. Nezobrazené notifikace jsou zvýrazněny barevnou tečkou vlevo.



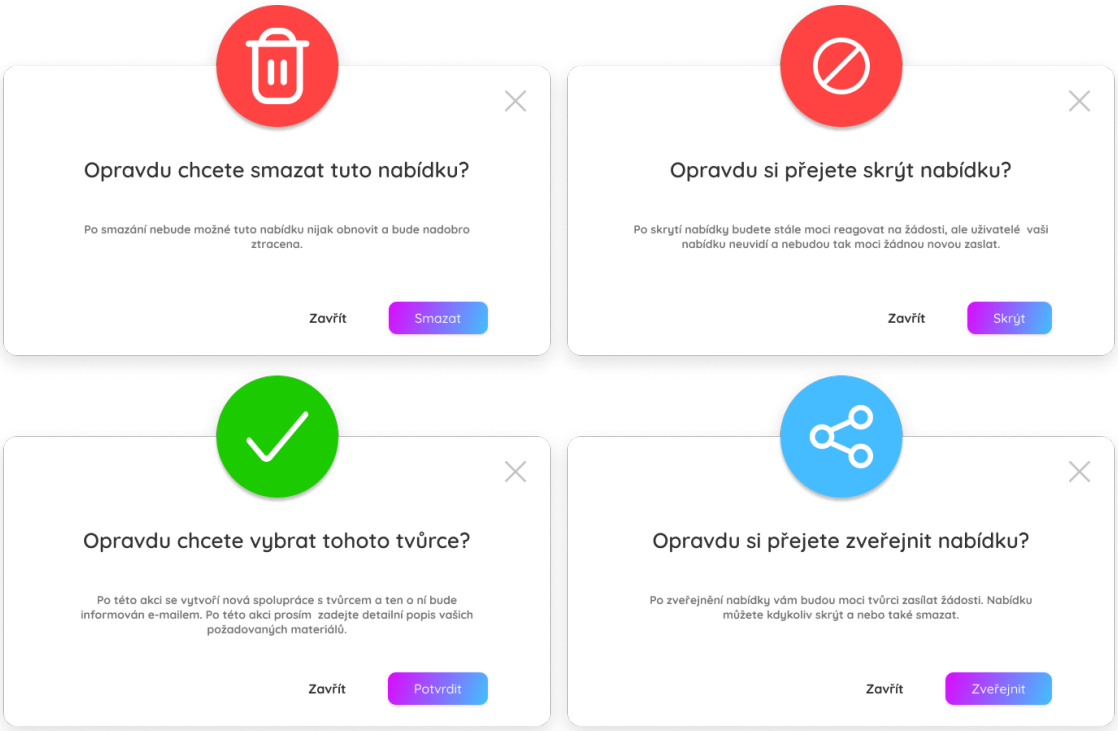
Obrázek A.6: Navržená stránka pro chat. V levé části jsou zobrazeni uživatelé, se kterými si přihlášený uživatel píše. Ti jsou seřazeni podle času poslední zprávy. V pravé části je již detail otevřeného chatu s několika zprávami.



Obrázek A.7: Tuto stránku využívají manažeři sociálních sítí, kteří tvoří nabídky spolupráce. V levé části je obecné nastavení nabídky. Zde si mohou manažeři zvolit typ produktu, typ odměny a také její výši, pokud se jedná o placenou spolupráci. Dále je zde možnost nastavit také počet tvůrců, kteří budou pro spolupráci hledáni a kategorii kam nabídka spadá. V pravé části je textový editor, který manažerům nabízí možnost text nabídky různě strukturovat. Mají zde také možnost připojit fotografie či videa.



Obrázek A.8: Jedna ze stránek v systému vedení spolupráce, kde manažeři mohou specifikovat obecný popis spolupráce – např. o které produkty se jedná. Dále zde specifikují zadání jednotlivých požadavků na materiál od tvůrce. V pravé části je vždy viditelný chat nehledě na to, na jaké podstránce právě teď uživatel je.



Obrázek A.9: Návrh jedné z víceúčelových komponent, tato konkrétně slouží k potvrzení akce.

Příloha B

Testovací scénáře

Pro potřeby testování byla naprogramována jednoduchá webová stránka, která zobrazovala informace z předem napsaného JSON souboru. Jelikož tyto testovací scénáře byly na webové stránce vertikálně dlouhé uvádím zde jejich podobu ve formě číslovaného seznamu. Příklad jednoho z kratších scénářů na webové stránce je zobrazen na obrázku B.1.

Testovací scénář pro ověření normální registrace pomocí e-mailu a hesla a následné přizpůsobení portfolia tvůrce:

1. Registrujte se jako tvůrce.
2. Ověřte si svůj účet.
3. Nahrajte novou profilovou fotografii .
4. Na své portfolio nahrajte své propagační materiály (fotografie i videa).
5. Jeden z nahraných propagačních materiálů nahraďte jiným.
6. Změňte popisek svého účtu.
7. Přidejte si na svůj účet 4 externí odkazy a následně jeden odkaz upravte, aby směřoval jinam.
8. Po provedení všech předchozích kroků se odhlaste.

Druhý testovací scénář byl zaměřen na přihlášení přes externí službu Google a testoval také tvorbu nabídek spolupráce:

1. Přihlaste se pomocí Google účtu do profilu manažera – pokud ještě nemáte účet doplňte potřebné údaje pro registrace a následně verifikujte svůj e-mail.
2. Vytvořte novou nabídku spolupráce. Specifikace, byla přiložena v předpřipraveném souboru.
3. Uložte nabídku a následně k ní připojte soubory (videa a fotografie).
4. Jednu fotografii smažte.
5. Následně tuto nabídku zveřejněte a následně ji skryjte.
6. Tuto nově vytvořenou nabídku smažte a vytvořte novou.

7. Zveřejněte ji a nechte ji v systému jako zveřejněnou.
8. Po provedení všech předchozích kroků se odhlaste.

Třetí test je zobrazen na obrázku B.1.
Čtvrtý test testoval přihlašování přes Facebook a zasílání žádostí o spolupráci:

1. Přihlaste se pomocí Facebook účtu do profilu manažera – pokud ještě nemáte účet doplňte potřebné údaje pro registrace a následně verifikujte svůj e-mail.
2. Projděte si nabídky spoluprací, které jsou v různých kategoriích.
3. Prohlédněte si detail nabídek, které vás zaujaly.
4. Následně na některé zašlete žádost o spolupráci.
5. Po provedení všech předchozích kroků se odhlaste.

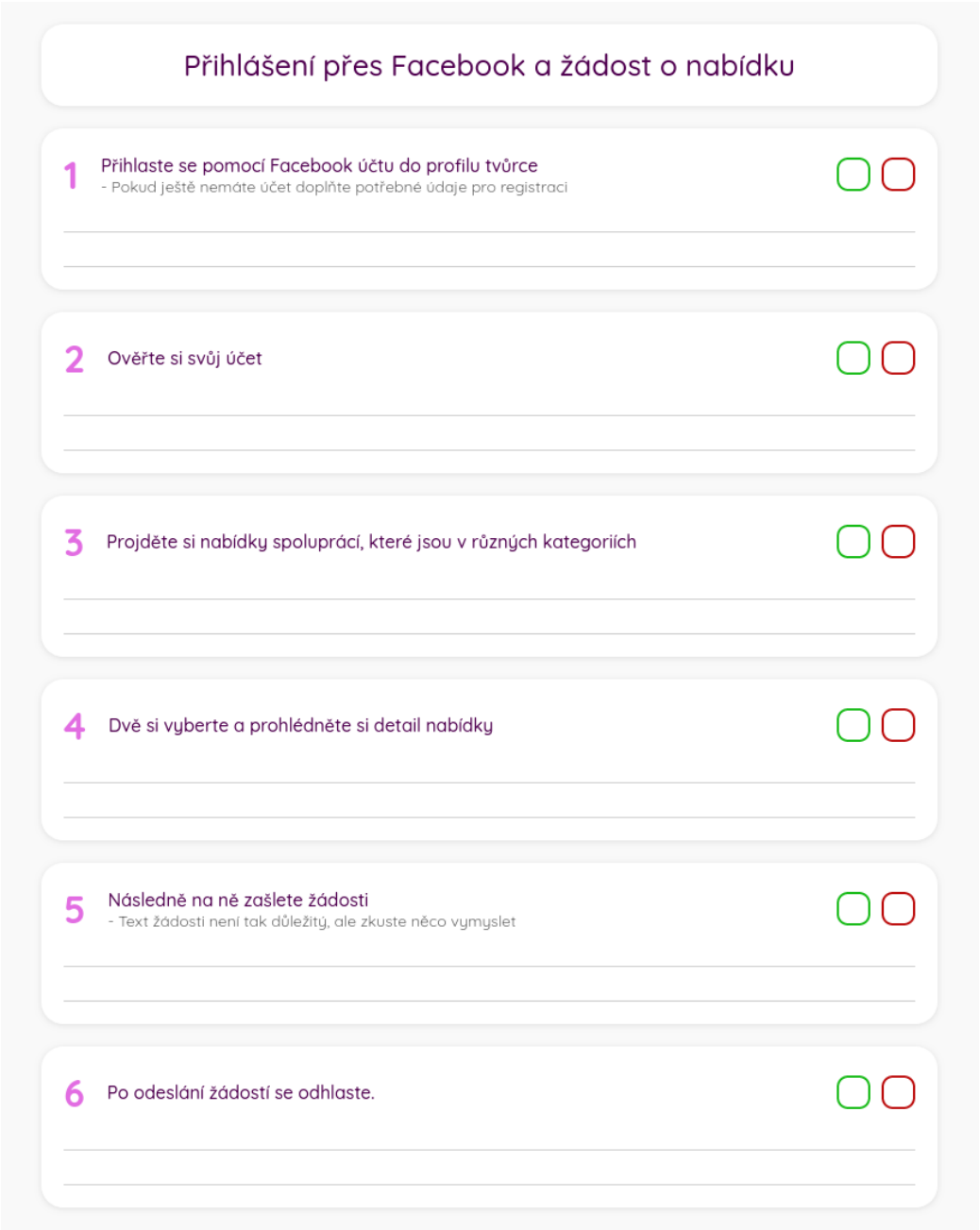
Pátý test ověřil funkčnost vedení spolupráce z pohledu manažera:

1. Přihlaste se k účtu s e-mailem: test@smm.com a heslem: Test123Test.
2. Projděte si tento účet.
3. Vyberte si jednu nabídku spolupráce a projděte si žádosti tvůrců.
4. Jednu z žádostí odmítněte (není zapotřebí psát důvod zamítnutí).
5. Zkuste si s jedním z tvůrců založit chat a napište mu zprávu.
6. Vraťte se zpět na otevřené žádosti a jednu z nich přijměte.
7. Po započetí spolupráce specifikujte požadavky na tvůrce.
8. Zkuste změnit jeden z požadavků na materiál.
9. Podívejte se zda tvůrce již nenahrál smlouvu ke spolupráci. Pokud zde smlouvu nevidíte – nahrajte ji vy.
10. Vyčkejte na nahrání materiálů od tvůrce.
11. Jakmile tvůrce nahraje všechen materiál, jeden zamítněte a ostatní přijměte.
12. Jakmile tvůrce nahraje opravený materiál, přijměte ho.
13. Vyčkejte na odemčení materiálů od tvůrce a následně si je stáhněte.
14. Po provedení předchozích kroků se odhlaste.

Šestý test ověřil funkčnost vedení spolupráce z pohledu tvůrce:

1. Přihlaste se k účtu s e-mailem: test@creator.com a heslem: Test123Test.
2. Přejděte na detail probíhající spolupráce, kterou nalaznete na svém profilu.
3. Projděte si zadání všech požadovaných materiálů.
4. Nahrajte smlouvu k dané spolupráci.

5. Nahrajte všechny požadované materiály a ověřte si, že jsou všechny materiály úspěšně nahrané.
6. Vyčkejte na zpětnou vazbu od manažera.
7. Zamítnuté materiály nahraďte jinými.
8. Vyčkejte na zpětnou vazbu na nové materiály.
9. Mezitím můžete nahrát fakturu do příslušné sekce spolupráce.
10. Jakmile budete mít všechny materiály schválené, dokončete spolupráci odemčením materiálů.
11. Po provedení předchozích kroků se odhlaste.



Obrázek B.1: Snímek testovacího scénáře, který byl použit přímo při testování použitelnosti.

Příloha C

Popis přiloženého paměťového média

Paměťové médium obsahuje veškeré zdrojové kódy aplikace, jak aplikaci ve Vue, tak i API, které je naprogramováno v Pythonu pomocí FastAPI. Paměťové médium obsahuje následující soubory

FrontendApp Složka obsahuje veškeré zdrojové soubory frontendu aplikace Content Cupid. Obsahuje také `README.md`, podle kterého lze frontend aplikace zprovoznit.

BackendApp Složka obsahuje veškeré zdrojové soubory backendu aplikace Content Cupid. Zprovoznění backendu je složitější, jelikož pro správné fungování API je třeba spustit databáze, založit účty na Oauth2 službách atd. Veškerý postup pro zprovoznění i této části řešení je také v souboru `README.md` v této složce.

DipText.zip Zdrojové soubory, které byly použity pro vygenerování PDF verze diplomové práce (společně se všemi obrázky, které jsou v práci použity).

xkriva29_dp.pdf Diplomová práce ve formátu PDF.