F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Backend aplikace pro textové konzultace

Volodymyr Semenyug
Otevřená informatika, specializace Software

Únor 2023, Květen 2023
Vedoucí práce: Ing. Jindřich Prokop

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Semenyug** Jméno: **Volodymyr** Osobní číslo: **503165**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Backend aplikace pro textové konzultace

Název bakalářské práce anglicky:

Text Chat Consultations Application Backend

Pokyny pro vypracování:

1) Seznamte se s požadavky aplikace pro textové konzultace veřejnosti s Národní linkou pro odvykání a sběr dialogových dat. Zakládní funkční požadavky aplikace jsou:
1a) Aplikace musí umožňovat správu uživatelů-konzultantů, jejich autentifikaci a autorizaci jednotlivých akcí podle přidělených rolí.
1b) Aplikace musí zprostředkovávat textový dialog mezi konzultantem a zájemcem o pomoc (klientem) v reálném čase včetně napojení na dialogový model umělé inteligence pro návrhy odpovědí.
1c) Aplikace musí umožňovat správu dialogů.
1d) Aplikace musí umožnit plánování konzultací.
1e) Aplikace musí podporovat autentifikaci klientů skrze jednorázová sezení a odkazy zaslané např. do emailové schránky.
1f) Aplikace musí umožnit správu karet klientů.
1g) Aplikace musí podporovat export karet klientů a souvisejících statistik.
2) Zvolte technologie pro zajištění serverové části aplikace s ohledem na napojení externích služeb (AI generování textů, ukládání anonymizovaných dat).
3) Navrhněte vhodný databázový model a strukturu API.
4) Zvolte si vhodný způsob dokumentace API.
5) Implementujte navrženou aplikaci a zdokumentujte API.

Seznam doporučené literatury:

1) Allamaraju, S. (2010). RESTful Web Services Cookbook. O'Reilly Media.
2) Date, C. J. (2003). An introduction to database systems: 8th ed. Pearson.
3) Webber, J., Parastatidis, S., & Robinson, I. (2010). REST in practice: Hypermedia and systems architecture. O'Reilly Media.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jindřich Prokop Analýza a interpretace biomedicínských dat FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.01.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Jindřich Prokop podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

Poděkování / Prohlášení

Děkuji své rodině a svým kamarádům za víru a pomoc, kterou mně vždycky byli ochotni poskytnout.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. 05. 2023

.....

Abstrakt / Abstract

Cílem práce je vytvoření webové aplikace pro odborníky pomáhající kuřákům odvyknout kouření přes realtime chat. Jádrem aplikace je chat s klientem v reálném čase integrující přes API neurální model pro automatický návrh odpovědí. Další funkcionality se odvíjí od tohoto základního cíle: přihlašování odborníků, jednoduchá správa klientů, zobrazení historie rozhovorů a dalších informací během probíhající konverzace, možnost interních poznámek, případně plánovaní sezení.

Klíčová slova: odvykání kouření, webová aplikace, backend, API.

The aim of the work is to create a web application for professionals helping smokers to quit smoking via real-time chat. The core of the application is a integrating chat with the client via API neural model for automatic response suggestion. Other functionalities are derived from this basic goal: login specialists, simple clients management, viewing interview history and other information during an ongoing conversation, internal notes or scheduling a consultation.

Keywords: smoking cessation, web app, backend, API..

Title translation: Text Chat Consultations Application Backend

Obsah /

/ Obrázky

Kapitola 1
Úvod

Nezdravé zlozvyky, jako jsou kouření a nadměrná konzumace alkoholu, představují pro lidi, kteří se snaží žít zdravým životním stylem v dnešní dynamické společnosti, velkou výzvu. Stále více lidí se obrací na odborníky s žádostí o pomoc při odvykání těmto návykům kvůli jejich negativním účinkům na fyzické i duševní zdraví. Webové aplikace se staly praktickou platformou pro propojení lidí s pomocí, jíž potřebují, a to v důsledku technologických zlepšení a širokého využívání internetu.

Cílem této bakalářské práce je navrhnout a vytvořit backend webové aplikace pro ty, kteří chtějí přestat kouřit, konzumovat alkohol nebo odvyknout jiným podobným zlozvykům. Tato webová aplikace funguje jako digitální prostředek pro klienty, kteří se mohou v reálném čase zapojit do chatových sezení s odbornými poradci, a vytvořit tak podpůrné prostředí pro jejich cestu ke zdravému životnímu stylu.

Webová aplikace byla rozdělena na dvě nezávislé části, z nichž každá byla přidělena specializovanému členovi týmu. Já jsem pracoval na backendu a zaměřil jsem se na logiku na straně serveru, ukládání dat a architekturu komunikace v reálném čase. Současně byla frontendová část aplikace, která zahrnuje návrh uživatelského rozhraní, interaktivitu na straně klienta a vizuální prezentaci, přidělena mému kolegovi Fabianu Bodnářovi.

Požadavkem od Národní linky pro odvykání, pro kterou tato aplikace byla vyvinuta, bylo vytvořit spolehlivý systém, který umožní bezproblémovou a bezpečnou komunikaci mezi konzultanty a klienty. Systém má funkce zahrnující komunikaci v reálném čase, autentizaci uživatelů, ukládání a vyhledávání dat. Naším cílem bylo vytvořit intuitivní a uživatelsky přívětivou platformu, která umožní klientům aktivně se podílet na jejich rehabilitačním procesu, a to za použití moderních webových technologií a frameworků.

Ještě před tím, než přejdeme přímo k popisu systému, chtěl bych uvést statistické údaje o problematice závislostí v současné společností.

1.1 Statistika

V moderním světě se stále častěji u lidí vyskytují fyzické a psychické závislosti, které představují obrovskou výzvu pro jednotlivce, rodiny a komunity. Fyzická závislost je definována jako závislost na látkách, jako jsou drogy nebo alkohol, při níž si tělo vytvoří toleranci a trpí abstinenčními příznaky, když se užívání sníží nebo přeruší. Tento druh závislosti může mít vážné zdravotní následky a k zotavení z ní může být zapotřebí lékařské pomoci.

Psychologická závislost je naopak charakterizována nutkavou touhou nebo závislostí na určitých návycích nebo činnostech, jako jsou hazardní hry, počítačové hry nebo sociální média. Uspokojující a posilující účinky těchto závislostí vedou ke ztrátě kontroly a nežádoucím důsledkům v různých oblastech života. Psychické závislosti mohou ničit vztahy, potlačovat produktivitu a poškozovat duševní zdraví a celkovou pohodu.

Podle Světové zdravotnické organizace (SZO) je kouření tabáku jednou z hlavních příčin úmrtí. Odhaduje se, že na světě je více než 1,1 miliardy kuřáků, přičemž většina z nich se soustřeďuje v zemích s nízkými a středními příjmy.[1]

Kouření způsobuje řadu zdravotních problémů, včetně rakoviny plic, krku a úst, onemocnění dýchacích cest (např. chronickou obstrukční plicní nemoc - CHOPN), srdečních chorob, mrtvice a dalších závažných onemocnění. Podle údajů Centra pro kontrolu a prevenci nemocí (CDC) kouření každoročně zabije přibližně 8 milionů lidí.[2]

Konzumace alkoholu je každoročně příčinou 3 milionů úmrtí na celém světě, stejně jako poškození a špatného zdravotního stavu milionů dalších lidí. Na celosvětové zátěži nemocemi se škodlivá konzumace alkoholu podílí 7,1% u mužů a 2,2% u žen. Alkohol je hlavní příčinou předčasných úmrtí a invalidity u osob ve věku 15 až 49 let a představuje 10% všech úmrtí v tomto věkovém rozmezí.[3]

1.1.1 Česká republika

Podle nejnovějších statistik Ministerstva zdravotnictví Českés republiky, vydaných v roce 2020[4], činí počet kuřáků starších 15 let v ČR 24,9% z celkové populace (v roce 2019 bylo v republice přibližně 8,8 milionu obyvatel starších 15 let[5]), což je zhruba 2,2 milionu kuřáků. Oproti statistikám z roku 2018 je patrný pokles o 4%.

Nadměrná konzumace alkoholu je přičinou smrtí 6500 lidí v ČR ročně[6], což tvoří přibližně 6% celkové úmrtnosti. Alkohol způsobuje různá zdravotní rizika, např. je velmi škodlivý pro trávicí a kardiovaskulární systém.[7]

Kapitola 2
Technologie

Úspěch a efektivita každého softwarového projektu jsou silně závislé na pečlivém výběru a použití příslušné technologie. V této kapitole se podíváme na základní technologie použité při tvorbě našeho softwarového projektu, přičemž se zaměříme na zvolený programovací jazyk, frameworky, databáze a další podpůrné nástroje. Rád bych poskytl náhled na to, jak tato technologická rozhodnutí ovlivnila návrh projektu, jeho funkčnost a celkový výkon, a to tak, že se ponořím do logiky, která za výběrem těchto technologií stojí.

2.1 Programovácí jazyk

Výběr vhodného programovacího jazyka je důležitou součástí jakéhokoliv softwarového projektu. Zvolený programovací jazyk má vliv nejen na proces vývoje, ale také na škálovatelnost, udržovatelnost a kompatibilitu projektu. Různé jazyky podprojí různé frameworky, každý je v něčem lepší a v něčem horší. Vedoucím práce bylo doporučeno 3 programovácí jazyky: Python, PHP a Rust, ze kterých jsem se rozhodl pro Python.

2.1.1 Python

Python[1] je flexibilní a silný programovací jazyk, jehož obliba v oblasti vývoje webových stránek roste.[8] Je skvělou volbou pro vývoj webových aplikací díky své jednoduchosti, čitelnosti a rozsáhlé podpoře knihoven. Výhody jazyka Python při vývoji webových aplikací jsou dány jeho expresivní syntaxí, rozsáhlým ekosystémem a důrazem na efektivitu a udržovatelnost kódu.[9]

Čitelnost jazyka Python je jednou z jeho hlavních výhod. Syntaxe jazyka je jasná a jednoduchá, podobná běžné angličtině, což usnadňuje pochopení a tvorbu kódu. Tento faktor zlepšuje spolupráci vývojářů a usnadňuje údržbu a řešení problémů. Důraz jazyka Python na čitelnost kódu také přispívá ke zkrácení vývojových cyklů, což umožňuje webovým vývojářům konstruovat efektivnější aplikace.[10]

Další velkou výhodou pro vývoj webových aplikací je obrovský ekosystém nástrojů a frameworků Pythonu. Frameworky jako Django a Flask poskytují organizovaný přístup k vývoji webových aplikací, včetně možností, jako je směrování URL, zpracování formulářů, abstrakce databáze a bezpečnostní ochrany. Tyto frameworky umožňují vývojářům soustředit se na logiku aplikace spíše než na šablonovitý kód tím, že nabízejí předem připravené komponenty a automatizují typické činnosti. Rozsáhlý ekosystém knihoven jazyka Python navíc obsahuje moduly pro různé aplikace, od zpracování dat až po strojové učení, a poskytuje tak vývojářům rozmanitou sadu nástrojů a zdrojů pro zlepšení možností webových aplikací.[11]

[1] `https://www.python.org/`

2.1.2 PHP

PHP[2], programovací jazyk orientovaný na stranu serveru, je již několik desetiletí základem pro vývoj webových stránek[12]. Pohání velkou část internetu, zejména systémy pro správu obsahu (CMS) a platformy pro elektronické obchodování. Navzdory svému rozsáhlému používání popularita jazyka PHP ve srovnání s jazykem Python a dalšími jazyky z různých důvodů klesá.

Jednou z příčin poklesu popularity je jeho pověst, že je náročnější na učení než Python. Syntaxe jazyka PHP a nejednotné konvence pojmenování mohou být pro nováčky obtížně pochopitelné. Kromě toho měly starší verze jazyka PHP bezpečnostní chyby a architektonická omezení, což způsobovalo problémy s kvalitou kódu a jeho udržovatelností. Přestože tyto problémy byly s vydáním dalších verzí a frameworků do značné míry vyřešeny, dojem přetrvává a ovlivnil preference vývojářů.[13]

2.1.3 Rust

Rust[3] je systémový programovací jazyk uznávaný pro svůj důraz na efektivitu, bezpečnost a souběžnost. Díky svému potenciálu získává větší zájem v oblasti vývoje webových aplikací.[14] V současnosti je však z různých důvodů méně populární než Python.

Opět problémem, který přispívá k menší popularitě jazyka Rust při vývoji webových aplikací, je jeho větší náročnost na učení v porovnání s jazykem Python. Programovací jazyk Rust se vyznačuje úzkou paměťovou bezpečností, což může být náročné pro vývojáře, kteří nemají zkušenosti s nízkoúrovňovými principy programování. Důraz, který jazyk klade na prevenci běžných programátorských chyb a problémů souvisejících s pamětí, vyžaduje důkladnější pochopení a dodržování jeho standardů. Zatímco pro systémové programování je jazyk Rust kvůli jeho zaměření na bezpečnost skvělou volbou, může od jeho využití pro projekty vývoje webových aplikací některé vývojáře odradit.[15]

Přestože jazyk Rust není při vývoji webových aplikací tak populární jako jazyk Python, jeho jedinečné vlastnosti, včetně paměťové bezpečnosti, výkonu a souběžnosti, jej činí ideálním pro aplikace, u kterých tyto vlastností jsou nejžádanější. S tím, jak ekosystém jazyka Rust dozrává a vyvíjejí se další nástroje a frameworky specifické pro web, může jeho využití při tvorbě webových stránek časem vzrůst.

2.2 Framework

Framework v programování je předem definovaná struktura nebo sada nástrojů, která slouží jako základ pro konstrukci softwarových aplikací. Poskytuje standardizovaný způsob psaní, organizace a údržby kódu tím, že nabízí knihovny, moduly a šablony pro běžné funkce. Frameworky často specifikují architekturu, návrhové vzory a osvědčené postupy, které mohou vývojáři použít k vytvoření efektivnějších aplikací.

Jedním z hlavních důvodů, proč se frameworky používají, je zjednodušení procesu vývoje. Framework minimalizuje potřebu vývojářů vytvářet kód od nuly pro typické úlohy, jako je připojení k databázi, ověřování uživatelů, směrování a ověřování vstupů. To nejen šetří čas a úsilí, ale také podporuje jednotnost a opakované použití kódu mezi projekty.[16]

Frameworky také podporují modulární přístup k vývoji aplikací a umožňují vývojářům rozdělit kód na menší, lépe zvládnutelné části. Tato modularita zlepšuje strukturu

[2] https://www.php.net/
[3] https://www.rust-lang.org/

kódu, udržovatelnost a zapojení členů týmu. Frameworky často zavádějí určité návrhové vzory, například Model-View-Controller (MVC), což napomáhá oddělení odpovědností a dosažení přehlednosti kódu.

Frameworky také poskytují určitý stupeň abstrakce, který chrání vývojáře před technologickými detaily nižší úrovně. Provádějí náročné úkoly, jako je zpracování požadavků HTTP, správa relací a implementace bezpečnostních opatření, a umožňují tak vývojářům soustředit se na obchodní logiku svých aplikací.

2.2.1 Framework vs. knihovna

Frameworky i knihovny jsou významnými součástmi při vývoji softwaru, ačkoli jejich účel, rozsah a úroveň abstrakce se liší. Následují základní rozdíly mezi frameworky a knihovnami:

- Framework je ucelená struktura nebo soubor nástrojů pro vývoj aplikací. Poskytuje předem definovanou architekturu, návrhové vzory a zásady, které je třeba dodržovat. Naproti tomu knihovna je kolekce opakovaně použitelného kódu a funkcí, které mohou vývojáři používat ve svých projektech. Mají být univerzální a modulární a umožňují vývojářům selektivně používat a začleňovat do svého softwaru konkrétní funkce.
- Frameworky mají vyšší úroveň abstrakce než knihovny. K vývoji aplikací přistupují komplexně a vytvářejí obecnou strukturu a běh aplikace. Naproti tomu knihovny poskytují specializované funkce, které mohou vývojáři využívat jako stavební kameny ve svých projektech. Knihovny se soustředí na konkrétní problémy nebo funkce, ale nedefinují obecný návrh nebo organizaci programu.
- Framework řídí průběh vykonávání aplikace a poskytuje specifické prostředky pro zpracování požadavků, událostí a dalších akcí na úrovni aplikace. Naproti tomu knihovny nevnucují programu předem definovaný postup. Vývojáři mají větší kontrolu nad průběhem provádění a mohou si vybrat, kdy a jakým způsobem použijí určité funkce knihovny.

Python nabízí širokou škálu frameworků pro vývoj webových aplikací. Mezi nejoblíbenější patří Django a Flask.[17]

2.2.2 Django

Django[4], vysokoúrovňový webový framework Python, je známý svým přístupem `batteries-included` (`se vším všude`) , který vývojářům poskytuje kompletní sadu nástrojů a funkcí pro rychlou tvorbu kvalitních online aplikací. Vrstva ORM (Object-Relational Mapping) frameworku umožňuje vývojářům propojit se s databázemi pomocí tříd a metod jazyka Python a abstrahuje od složitostí dotazů SQL. To usnadňuje práci s databázemi, udržování integrity dat a provádění typických operací, jako je vyhledávání, filtrování a aktualizace informací.

Další důležitou vlastností Djanga je důraz na bezpečnost. Je vybaven vestavěnou ochranou proti typickým zranitelnostem online, jako jsou XSS[5] (cross-site scripting), CSRF[6] (cross-site request forgery) a SQL injection[7]. Django disponuje bezpečnostními funkcemi, jako je automatické escapování HTML, validace tokenů formulářů a parametrizované dotazy, které snižují riziko narušení bezpečnosti a zároveň vývojářům usnadňují vytváření bezpečných online aplikací.

[4] `https://www.djangoproject.com/`
[5] `https://owasp.org/www-community/attacks/xss/`
[6] `https://www.synopsys.com/glossary/what-is-csrf.html`
[7] `https://portswigger.net/web-security/sql-injection`

Django má navíc živou a aktivní komunitu, která přispívá k jeho neustálému vývoji a dostupnosti balíčků třetích stran. Komunita Django pravidelně distribuuje aktualizace, opravy chyb a nové funkce, čímž zajišťuje, že framework zůstává v souladu s moderními standardy vývoje webových aplikací. Obrovský ekosystém balíčků třetích stran, známých jako `Django packages`[8], navíc nabízí široký výběr doplňků a rozšíření, které rozvíjejí možnosti frameworku Django a přizpůsobují se jedinečným požadavkům projektů.

2.2.3 Flask

Flask[9], lehký a přizpůsobitelný webový framework Python, je známý pro své snadné použití a minimalistický design. Vývojářům poskytuje základy potřebné k vytváření webových aplikací bez nežádoucích omezení nebo závislostí. Flask využívá přístup `mikro` frameworku, který vývojářům poskytuje větší flexibilitu a nezávislost na architektuře a designu jejich projektů.

Jednou z hlavních výhod Flasku je jeho jednoduchost a snadné používání. Jeho malá kódová základna a jednoduché rozhraní API umožňují vývojářům začít rychle pracovat. Flask klade důraz na jednoduchost a čitelnost a zároveň využívá přirozené možnosti a idiomy jazyka Python, takže je framework přístupný jak začínajícím, tak zkušeným vývojářům. Snadné použití Flasku se vztahuje i na jeho systém směrování, který vývojářům umožňuje vytvářet trasy URL a doprovodné funkce zobrazení s malým množstvím zaváděcího kódu.

Dalším charakteristickým prvkem Flasku je jeho rozšiřitelnost. I když Flask poskytuje pouze základní možnosti, lze jej snadno vylepšit pomocí rozšíření Flask. Tato rozšíření pokrývají integraci databází, zpracování formulářů, ověřování a další funkce. Vývojáři si mohou vybrat a integrovat jen ta rozšíření, která potřebují, což vede k úspornému a přizpůsobenému vývojovému prostředí. Díky své přizpůsobivosti je Flask vynikající alternativou pro malé až středně velké aplikace nebo scénáře vyžadující minimalistický přístup.

2.2.4 Django vs. Flask

Volba mezi Django a Flask pro vývoj webových aplikací je závislá na potřebách projektu, preferencích vývojářů a požadované rovnováze mezi jednoduchostí a funkčností. Existuje však několik klíčových důvodů, proč jsem se rozhodl používat Django místo Flasku:[18]

- Django je díky své bohaté funkční výbavě vhodnější pro větší a složitější aplikace, které vyžadují plnohodnotný framework se standardizovanými komponentami.
- Django má výrazný přístup ke konfiguraci založený na konvencích. Tento framework a vynucená pravidla umožňují vývojářům rychle začít pracovat, udržovat konzistenci kódu mezi projekty a hladce komunikovat v rámci týmu.
- Škálovatelnost Djanga a jeho schopnost spravovat webové stránky a aplikace s velkým provozem jsou dobře známé. Má funkce, jako je sdružování databázových připojení, ukládání do cache a podpora distribuovaných systémů.

2.3 Databázový systém

Databáze jsou v softwarovém inženýrství nezbytné, protože poskytují systematické a organizované prostředky pro ukládání, zpracování a vyhledávání dat. Databáze se v

[8] `https://djangopackages.org/`
[9] `https://flask.palletsprojects.com/en/2.3.x/`

softwarovém inženýrství používají k průběžnému ukládání a správě velkých objemů organizovaných nebo nestrukturovaných dat, což programům umožňuje efektivnější přístup k informacím a jejich změnu.

Jednou z hlavních výhod databází je jejich schopnost udržovat integritu a konzistenci dat. Aby bylo zaručeno, že data zůstanou správná a platná, používají databáze postupy, jako jsou transakce a omezení. Transakce umožňují považovat mnoho databázových operací za jedinou atomickou jednotku a zaručují, že budou provedeny buď všechny změny, nebo žádná z nich. Omezení primárního a cizího klíče například zavádějí datové vztahy a standardy integrity a zabraňují uložení nekonzistentních nebo nesprávných dat.

Databáze navíc umožňují efektivní vyhledávání a získávání dat. Nabízejí výkonné dotazovací jazyky, jako je SQL (Structured Query Language), které umožňují vývojářům navrhovat složité dotazy pro získání konkrétních dat z databází. Ke zvýšení rychlosti operací vyhledávání dat se používají metody indexování a optimalizační taktiky, jako je zdokonalení dotazů a ukládání do mezipaměti, což aplikacím umožňuje snadno spravovat obrovské soubory dat.

Existují různé druhy databázových systémů, které se liší způsobem ukládáním dat, účelem jejich využití a místem provozu. Mezi nejpoužívanější patří relační, NoSQL, hierarchické a objektově-orientované databáze.

2.3.1 Relační databáze

Relační databáze jsou jedným z nejběžnějších druhů databází používaných při vývoji softwaru.[19]. Jsou postaveny na relačním modelu, který rozděluje data do tabulek s řádky a sloupci. Každá tabulka představuje entitu a vazby mezi nimi jsou definovány klíči a cizími klíči. Relační databáze mají různé vlastnosti, díky nimž jsou oblíbené a vhodné pro nejrůznější aplikace.

Jednou z významných vlastností relačních databází je jejich schopnost zajistit integritu a konzistenci dat. Díky omezením, jako jsou primární klíče, cizí klíče a referenční integrita, relační databáze zaručují, že jsou zachovány datové vztahy a že jsou uchovávána pouze platná a konzistentní data. Snižuje se tak možnost poškození dat a zajišťuje se kvalita a spolehlivost jejich uložení.[10]

SQL (Structured Query Language) je univerzální a sofistikovaný dotazovací jazyk, který poskytují relační databáze. SQL umožňuje vývojářům rychle získávat, měnit a analyzovat databázová data. Deklarativní styl jazyka SQL umožňuje vývojářům soustředit se na poskytnutí požadované sady dat a ne se starat o to, jak je získat.

Relační databáze se často používají v celé řadě oblastí a aplikací. Lze je nalézt v transakčních systémech, jako je bankovnictví a elektronické obchodování, kde je integrita a konzistence dat klíčová. Relační databáze se používají také v systémech založených na datech, systémech pro správu obsahu, systémech pro řízení vztahů se zákazníky (CRM) a v řadě dalších odvětví, kde je vyžadováno strukturované a organizované ukládání dat. Systémy pro správu relačních databází (RDBMS), jako jsou MySQL[11], PostgreSQL[12] a Oracle[13], mají rozvinutý ekosystém, který poskytuje spolehlivá a škálovatelná řešení pro správu velkého množství dat a umožňuje provoz vysoce výkonných aplikací.

Protože s relačními databázemi jsem měl největší zkušennost, rozhodl jsem se, že i v rámci tohoto projektu použiji pro hlavní úložiště dat technologii, se kterou jsem

[10] https://www.techtarget.com/searchdatamanagement/definition/relational-database
[11] https://www.mysql.com/
[12] https://www.postgresql.org/
[13] https://www.oracle.com/cz/database/

pracoval nejvíc. A jelikož ve většině projektů, kterých jsem se zúčastnil, byla použitá databáze PostgreSQL, a byla osobně doporučená vedoucím práce, můj výběr padl na ni.

2.3.2 NoSQL databáze

Databáze NoSQL (Not Only SQL) jsou speciální kategorií databázových struktur, které se liší od standardních relačních databází. Na rozdíl od nich nespoléhají na přísnou strukturu tabulek a spojení. Místo toho poskytují flexibilní datovou architekturu pro ukládání a vyhledávání nestrukturovaných, polostrukturovaných a strukturovaných dat. Databáze NoSQL mají řadu výhod a využívají se v různých oblastech.[20]

Jednou z hlavních výhod databází NoSQL je jejich schopnost spravovat obrovské množství dat a zároveň být škálovatelné. Protože databáze NoSQL jsou distribuované, mohou růst horizontálně přidáváním dalších serverů, aby se přizpůsobily rostoucím objemům dat. Jsou vhodné pro aplikace s často se měnícími a neočekávanými požadavky na data.[21]

Další výhodou databází NoSQL je jejich vysoká rychlost a minimální latence. Databáze NoSQL umožňují rychlejší vyhledávání a ukládání dat, protože se vyhýbají složitým relačním spojováním a omezením schémat. Využívají datové modely, které jsou přizpůsobeny pro určité případy použití a umožňují efektivní vzory přístupu k datům, jako jsou databáze klíč-hodnota, sloupcové, dokumentově orientované nebo grafové databáze. Databáze NoSQL vynikají v aplikacích vyžadujících vysokorychlostní příjem dat, jejich rychlé zpracování a analýzu v reálném čase.

Databáze NoSQL nacházejí uplatnění v různých oblastech. Příkladem jejich využití je analýza velkých objemů dat, online aplikace v reálném čase, systémy pro správu obsahu, platformy sociálních médií a systémy internetu věcí. Jsou ideální pro scénáře zahrnující dynamická, nestrukturovaná nebo částečně strukturovaná data a také pro scénáře vyžadující horizontální škálovatelnost a vynikající výkon. Cassandra[14], Redis[15] a Apache HBase[16] jsou příklady populárních databází NoSQL, z nichž každá je zaměřena na odlišné případy použití a typy dat.

Pro účely této aplikace byla použita NoSQL databáze Redis pro správu komunikace mezi jednotlivými uživatelemi. Více najdete v kapitolách o komunikačním protokolu a použitých knihovnách.

2.3.3 Objektově-orientované databáze

Objektově orientované databáze (OODBMS) jsou typem systému správy databází, který ukládá data jako objekty stejným způsobem, jakým se objekty používají v objektově orientovaném programování. OODBMS umožňují přímou reprezentaci a manipulaci se složitými datovými strukturami, takže jsou ideální pro aplikace se komplikovanějšími interakcemi s daty.

Jednou z jejich hlavních výhod je schopnost spravovat složité datové struktury a propojení, což jim umožňuje umístit objekty přímo do databáze při zachování jejich vazeb a chování. Díky tomu je reprezentace realističtější a přirozeněji zobrazuje reálné prvky a jejich vztahy. Zapouzdření[17], dědičnost[18] a polymorfismus [19], které jsou základ-

[14] `https://cassandra.apache.org/_/index.html`
[15] `https://redis.io/`
[16] `https://hbase.apache.org/`
[17] Zapouzdření - skrytí vnitřního stavu objektu
[18] Dědičnost - princip v OOP, který umožňuje vytvářet hierarchickou strukturu objektu
[19] Polimorfismus - reprezentace objektu jednoho typu pomocí objektu jiného typu

ními principy objektově orientovaného programování, jsou podporovány OODBMS, což umožňuje univerzální a rozšiřitelné datové modely.

Objektově orientované databáze se často používají v systémech počítačového navrhování (CAD), multimediálních aplikacích, vědeckých a technických aplikacích a složitých systémech náročných na data. Přestože však OODBMS mají v některých případech výhody, nejsou tak široce využívány jako relační databáze kvůli své složitosti, nedostatečné standardizaci a preferenci relačních databázových systémů ze strany průmyslu.[22]. Příkladami objektově-orientovaných datobazových systému jsou ObjectDB[20], Versant Object Database[21] a Objectivity/DB[22].

2.3.4 Hierarchické databáze

Hierarchické databáze (HDBMS) jsou druhem databází, které organizují data ve stromové struktuře se záznamy s vazbami rodič-dítě. Každý nadřazený záznam v hierarchické databázi může obsahovat několik podřízených záznamů, čímž vzniká hierarchie dat.[23]. Příkladem hierarchické databáze je Neo4j[23].

HDBMS jsou nejvhodnější pro případ existence hierarchie mezi prvky. Proto se často používají např. při modelování sítí nebo souborových systémů. Díky jejich grafové struktuře, lze na nich použít existující algoritmy vyhledávání, řazení apod.

Hierarchické databáze však nemusí být vhodné pro případy vyžadující složité datové vazby nebo flexibilní možnosti dotazování. Proto obecně vytlačily hierarchické databáze relační databáze a další moderní databázové architektury, které nabízejí lepší flexibilitu a škálovatelnost.[24]

2.4 Komunikační protokol

Komunikační protokoly jsou při vývoji webu nezbytné, protože umožňují vzájemnou komunikaci různých komponent a systémů. Pravidla a standardy pro sdílení dat prostřednictvím sítí jsou definovány komunikačními mechanismy, které zajišťují správný a spolehlivý přenos dat. Při vývoji webu se používá řada komunikačních protokolů, z nichž každý má své vlastní výhody a cíle.

Jelikož hlavním požadavkem aplikace bylo umožnit komunikaci mezi klientem a kontultantem v reálném čase, měl jsem na výběr mezi několiká alternativami, které tuto vlastnost splňují.

2.4.1 WebSocket protokol

WebSocket je komunikační protokol, který umožňuje navázat plně duplexní komunikační kanály prostřednictvím jediného připojení TCP. Překračuje omezení běžných dotazů HTTP[24], které jsou obvykle založeny na přístupu požadavek-odpověď. WebSocket umožňuje interaktivní komunikaci v reálném čase tím, že klientům a serverům umožňuje kdykoli si navzájem posílat zprávy.

Minimální režie protokolu WebSocket je jednou z jeho hlavních vlastností. Na rozdíl od typických dotazů HTTP, které vyžadují odesílání hlaviček s každým požadavkem, WebSocket udržuje trvalé spojení, čímž minimalizuje náklady na zahajování a uzavírání spojení pro každou konverzaci. Díky své efektivitě je WebSocket vhodný pro aplikace

[20] https://www.objectdb.com/
[21] https://www.actian.com/data-management/nosql-object-database/
[22] https://objectivity.com/
[23] https://neo4j.com/
[24] https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol

vyžadující častou a rychlou výměnu dat, jako jsou chatovací aplikace nebo nástroje pro spolupráci.[25]

WebSocket je systém pro doručování zpráv, který běží nad protokolem TCP (Transmission Control Protocol)[25]. Zaručuje, že zprávy odeslané z jednoho koncového bodu budou druhým koncovým bodem přijaty ve stejném pořadí, čímž se zachová integrita a konzistence dat. Tato spolehlivost je rozhodující pro aplikace v reálném čase, kde pořadí zpráv je kritické.

Kromě toho, že je WebSocket obousměrný, poskytuje podprotokoly, které umožňují libovolné formáty zpráv a rozšíření. Tyto podprotokoly umožňují vývojářům vytvářet nad WebSocketem protokoly specifické pro jednotlivé aplikace, čímž ještě více rozšiřují jeho možnosti a přizpůsobivost.[26]

WebSocket je celkově vynikající protokol pro interaktivní online komunikaci v reálném čase. Díky své minimální režii, spolehlivosti a rozsáhlé podpoře je skvělou volbou pro vývoj aplikací, které vyžadují rychlou aktualizaci a obousměrnou komunikaci mezi klienty a servery. Kvůli těmto vlastnostem jsem se rozhodnul použít právě jeho.

2.4.2 Polling

Polling je komunikační strategie pro vývoj webových stránek, která umožňuje aktualizace mezi klientem a serverem téměř v reálném čase. Na rozdíl od WebSocket zahrnuje časté odesílání dotazů klientem na server, kde se zjišťují změny. Klient v pravidelných intervalech odesílá serveru požadavek s dotazem, zda jsou k dispozici nějaké nové informace. Server buď vrátí čerstvá data, nebo odpověď `žádná nová data`.

Krátký nebo pravidelný polling jsou populárními druhy pollingu. Při krátkém dotazování klient posílá požadavek na server v pravidelných intervalech, například každých několik sekund nebo minut. Server zjistí, zda jsou k dispozici čerstvá data, a pokud ano, odpoví aktualizovanými informacemi. Pokud nejsou nalezena žádná nová data, server vrátí prázdnou odpověď. Klient pak postup opakuje a v předem stanoveném intervalu odešle další požadavek.

Ačkoli dotazování může poskytovat informace téměř v reálném čase, má několik nevýhod. Jednou z významných nevýhod je zvýšená zátěž serveru způsobená častými dotazy klientů, i když nejsou k dispozici žádná nová data. To může zatěžovat server a snižovat jeho škálovatelnost. Kromě toho může dotazování způsobovat zpoždění, protože klient musí čekat na další období dotazování, aby získal informace.[27]

2.4.3 Long polling

Long polling je přístup, který umožňuje aktualizace mezi klientem a serverem téměř v reálném čase. Long polling se od běžného pollingu liší tím, že klient odešle požadavek na server a ponechá jej otevřený, dokud se neobjeví nová data. Místo čekání na další období pollingu může server odpovědět okamžitě, jakmile získá k dispozici nová data. Aby bylo možné provést případné aktualizace, může server ponechat spojení otevřené po definovanou dobu prodlevy.

Long polling má tu výhodu, že snižuje množství prázdných nebo zbytečných odpovědí ze serveru. Namísto toho, aby klient neustále odesílal dotazy a dostával prázdné odpovědi, když nejsou k dispozici žádná nová data, server udržuje spojení a odpovídá pouze tehdy, když má co odeslat. Ve srovnání s běžným pollingem se tak snižuje celkové zatížení serveru a zvyšuje se efektivita.[28]

[25] `https://www.ietf.org/rfc/rfc793.txt`

2.5 Architektura API

Rozhraní API (Application Programming Interface) je soubor pravidel, protokolů a nástrojů, které umožňují různým softwarovým programům vzájemně komunikovat a spolupracovat. Umožňuje vývojářům přistupovat k funkcím softwarového programu nebo služby, aniž by museli rozumět jejich vnitřnímu fungování.

API vytvářejí standardizovanou metodu pro sdílení dat a provádění úloh různými softwarovými komponentami. Definují strukturu a syntaxi požadavků a odpovědí, stejně jako dostupné funkce, metody a koncové body. Lze je použít k získání informací ze vzdáleného serveru, ke sdělení informací službě nebo k provádění určitých činností a procesů. Často se používají při vývoji webových aplikací, mobilních aplikací a integraci softwarových systémů a umožňují vývojářům rozšiřovat stávající služby a vytvářet nové aplikace.[29]

Mezi nejoblíbenější architektonické styly API patří RPC, SOAP, REST (který jsem se rozhodl používat) a GraphQL. Některé z nich již ztratily svou dřívější popularitu a byly nahrazeny modernějšími styly, ale dodnes lze nalézt projekty, které je využívají.

2.5.1 RPC

RPC (Remote Procedure Call) je technika meziprocesové komunikace, která umožňuje softwaru spuštěnému v jednom počítači volat proceduru nebo funkci v jiném počítači nebo procesu. Nabízí standardizované prostředky pro provádění vzdálených volání funkcí a umožňuje distribuovaným systémům snadnou interakci a spolupráci. Volající software v RPC odešle zprávu s požadavkem vzdálenému systému, který pak spustí požadovanou proceduru a poskytne výsledek volajícímu.

Protokol RPC zapouzdřuje složitost vzdálené komunikace a poskytuje vývojářům známé procedurální rozhraní. Umožňuje jim vyvolávat vzdálené operace stejným způsobem, jakým se vyvolávají lokální funkce, aniž by se museli starat o specifika základního síťového připojení. RPC pokrývá povinnosti zahrnující serializaci zpráv, přenos po síti a předávání argumentů a výsledků funkcí. RPC usnadňuje vytváření distribuovaných systémů zapouzdřením těchto funkcí a podporuje kompatibilitu mezi různými platformami a programovacími jazyky.

Protokol RPC byl široce využíván v různých oborech, například v aplikacích klient-server, distribuovaných výpočetních systémech a architekturách mikroslužeb. Umožňuje hladkou integraci komponent nebo služeb napříč více stroji nebo procesy, což z něj činí nezbytnou technologii pro vývoj škálovatelných a distribuovaných systémů. Ačkoli je RPC jednoduchou a efektivní technikou pro komunikaci mezi procesy, přináší určité obtíže, jako jsou kontrola síťové latence, řešení chyb a zajištění kompatibility různých verzí téhož rozhraní. RPC však nadále zůstává základní technologií v softwarovém inženýrství, která umožňuje efektivní a transparentní komunikaci mezi distribuovanými komponentami. [30]

2.5.2 SOAP

SOAP (Simple Object Access Protocol) je komunikační systém založený na XML[26], který umožňuje síťovým aplikacím vyměňovat si strukturovaná a typizovaná data. Umožňuje aplikacím komunikovat konzistentním způsobem napříč platformami a operačními systémy. SOAP specifikuje soubor pravidel a požadavků na konstrukci zpráv, strukturu zpráv a jejich zpracování. Podporuje řadu síťových protokolů, včetně HTTP,

[26] XML - eXtensible Markup Language

SMTP a TCP/IP, a představuje tak univerzální alternativu pro meziprocesovou komunikaci.

Zprávy SOAP jsou obvykle formátovány jako dokumenty XML s obálkou, hlavičkami a tělem. Obálka obsahuje informace o původu a cíli zprávy a případná další metadata a zapouzdřuje celou zprávu. V hlavičkách zprávy mohou být zahrnuty nepovinné informace, jako jsou bezpečnostní tokeny, ověřovací pověření nebo pokyny pro směrování zprávy. Vlastní užitečné zatížení je obsaženo v těle zprávy SOAP a může být v libovolném formátu XML. Zpráva také může obsahovat popis chyb, který se doplňuje do podčásti těla.

SOAP se často používá v podnikových aplikacích a online službách. Poskytuje standardizovaný a přizpůsobitelný rámec pro vývoj distribuovaných systémů schopných komunikovat napříč několika platformami a jazyky. SOAP zahrnuje složitější funkce, jako je zabezpečení na úrovni zpráv, správa transakcí a spolehlivost, a je tak vhodný pro aplikace vyžadující robustnost a spolehlivost výměny zpráv. Ve srovnání s jinými lehkými možnostmi, jako jsou rozhraní API RESTful, však lze za nevýhodu považovat rozsáhlost a složitost zpráv SOAP. Navzdory tomu je SOAP stále významnou technologií v softwarovém inženýrství pro usnadnění součinnosti a komunikace mezi aplikacemi na podnikové úrovni.[31]

2.5.3 REST

REST (Representational State Transfer) je architektonické paradigma pro vytváření síťových aplikací, konkrétně webových služeb, které klade důraz na jednoduchost, škálovatelnost a používání běžných protokolů HTTP. Rozhraní API REST jsou založena na myšlence zdrojů, které jsou jednoznačně identifikovatelné pomocí adres URL (Uniform Resource Locators). Klienti mohou s těmito zdroji v architektuře RESTful komunikovat prostřednictvím HTTP requestů s využitím běžných metod, jako jsou GET, POST, PUT a DELETE.

Bezstavovost je základním konceptem REST, který znamená, že každý požadavek klienta na server by měl poskytnout všechny informace potřebné k tomu, aby server požadavek pochopil a provedl. Protože server neukládá pro klienta žádná data relace, je tento návrh lépe škálovatelný a umožňuje lepší ukládání do mezipaměti a vyrovnávání zátěže. Rozhraní API RESTful také používají standardní kódy odpovědí HTTP k označení stavu požadavku, což umožňuje konzistentní a snadný způsob správy problémů a komunikace s klienty.[32]

REST si získal oblibu díky snadnému použití a kompatibilitě se stávajícími webovými technologiemi. Podporuje volné propojení klientů a serverů, což jim umožňuje nezávislý růst. Rozhraní API REST jsou nezávislá na jazyku, což umožňuje součinnost mezi platformami. RESTful API jsou snadno pochopitelná a lze s nimi snadno pracovat, protože používají standardní metody a stavové kódy HTTP, což zjednodušuje postupy vývoje a integrace.[33]

2.5.4 GraphQL

GraphQL je dotazovací jazyk a běhové prostředí API, které klientům umožňuje požadovat a získávat konkrétní data ze serveru. Na rozdíl od standardních rozhraní RESTful API, kde server diktuje strukturu a obsah odpovědi, GraphQL dává klientovi kontrolu nad formou a hloubkou dat, která si přeje získat v rámci jednoho požadavku. Tato metoda zabraňuje nadměrnému a nedostatečnému načítání dat, což vede k efektivnějšímu a optimálnějšímu získávání dat.[27]

[27] `https://graphql.org/`

Jednou z hlavních výhod GraphQL je jeho schopnost vytvořit samo-dokumentující rozhraní API. Vývojáři mohou specifikovat typy, vztahy a možnosti rozhraní API stručným a lidsky čitelným způsobem pomocí jazyka pro definici schémat GraphQL. Díky tomu mohou zákazníci i vývojáři snadno pochopit dostupná data a činnosti rozhraní.

Díky své flexibilitě a architektuře zaměřené na vývojáře si jazyk GraphQL získal mezi nimi oblibu. Klienti ji mohou používat ke kombinování dat z mnoha zdrojů, sestavování složitých dotazů a získávání předvídatelných odpovědí. Systém typů v jazyce GraphQL navíc umožňuje výkonné typování a validaci, což usnadňuje odhalování problémů a zajišťuje konzistenci dat. I když má jazyk GraphQL určité výhody, může ve srovnání s jednoduššími možnostmi, jako je například REST, zvyšovat složitost, zejména pokud jde o ukládání do mezipaměti, zneplatňování mezipaměti a změny prováděné v reálném čase. Na druhou stranu se jazyk GraphQL vyvinul ve výkonný nástroj pro vývoj rozhraní API, který vývojářům umožňuje vytvářet rychlé a přizpůsobitelné aplikace založené na datech.[34]

2.6 Použité knihovny a balíčky

Na závěr bych rád v rychlostí probral seznam použitých knihoven, které mi velice oolehčily proces vývoje.

- environ - knihovna umožňující konfigurovat proměnné prostředí. Je použita hlavně pro konfigurování připojení k databázi.[28]
- Django Rest Framework - nástroj pro vytváření Web API v Django. Zahrnuje v sobě také autentizační a autorizační metody, serializaci, validaci atd.[29]
- djoser - knihovna poskytující možnosti přihlášení se, registrace, odhlášení, změnu hesla a aktivaci účtu.[30]
- django-cors-headers - knihovna, která je určená pro správu domén, portů atd., kterým bude umožněno předavat zdroje (např. skripty, obrázky apod).[31]
- daphne[32] - server s podporou protokolů HTTP a WebSocket, který je nadstavbou nad Django Channels.[33]
- psycopg2-binary - PostgreSQL databázový adaptér pro Python. Binární verze nepotřebuje kompilátor a extérní knihovny. [34].
- asgiref - balíček obsahuje standardní ASGI (Asynchronous Server Gateway Interface) knihovny. Používá se hlavně převod synchronních funkcí na asynchronní.[35]
- channels - umožňuje používat WebSocket a jiné protokoly s podporou dlouhodobého zachování otevřeného kanálu pro posílání zpráv.[36]
- channels-redis - umožňuje používat Redis pro vrtsvu komunikace mezi uživateli.[37]
- redis - Python rozhraní pro Redis `klíč-hodnota` úložiště. [38]

[28] https://pypi.org/project/python-environ/
[29] https://www.django-rest-framework.org/
[30] https://pypi.org/project/djoser/
[31] https://pypi.org/project/django-cors-headers/
[32] https://pypi.org/project/daphne/
[33] https://channels.readthedocs.io/en/stable/
[34] https://pypi.org/project/psycopg2-binary/
[35] https://pypi.org/project/asgiref/
[36] https://channels.readthedocs.io/en/stable/introduction.html
[37] https://pypi.org/project/channels-redis/
[38] https://pypi.org/project/redis/

Kapitola 3
Byznys část projektu

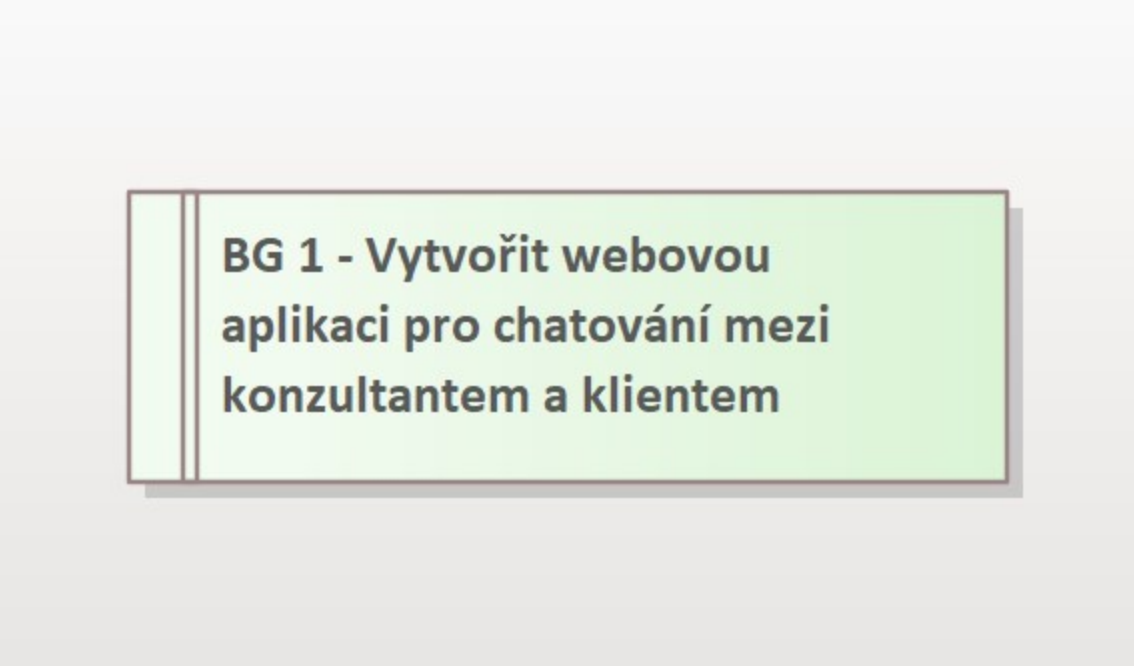
Byznysové požadavky v softwarovém inženýrství se týkají konkrétních potřeb a cílů organizace, které určují vývoj softwarového řešení. Tyto specifikace definují funkce, vlastnosti a schopnosti, které musí aplikace mít, aby splňovala obchodní cíle a potřeby zainteresovaných stran. Obchodní požadavky jsou klíčové pro řízení celého procesu vývoje softwaru, od počátečního plánování a analýzy až po konečnou implementaci a dodání.

Byznysové potřeby se často odvíjejí od strategických cílů a činností organizace. Zachycují cíle a priority podniku na vysoké úrovni, jako je zvýšení efektivity, zlepšení spokojenosti klientů nebo snížení nákladů. Tyto požadavky jsou prezentovány netechnickým způsobem a zaměřují se spíše na to, co by měl softwarový systém poskytovat, než na to, jak by měl být implementován. Aby bylo zaručeno, že výsledné softwarové řešení odpovídá vizi a cílům organizace, musí být obchodní požadavky pečlivě sestaveny a zdokumentovány.

Pro efektivní definování byznys požadavků vedou softwaroví inženýři podrobné diskuse a konzultace s celou řadou zainteresovaných stran, včetně obchodních analytiků, projektových manažerů, koncových uživatelů a dalších relevantních stran. Na základě těchto interakcí tým softwarových inženýrů získává hluboké znalosti o pracovních postupech a bolestivých oblastech organizace. Tyto znalosti se následně promění v sadu dobře definovaných obchodních požadavků, které slouží jako plán pro proces vývoje softwaru. Obchodní požadavky slouží jako spojovací článek mezi byznys doménou a technickou implementací a zajišťují, že všechny zúčastněné strany mají jasnou a společnou znalost požadovaného softwarového řešení[35].

3.1 Byznys cíle

Obchodní cíle softwarového projektu odpovídají na otázku: `Čeho chceme dosáhnout?` Cíle projektu sestavují zainteresované strany, pro které je software vyvíjen. Vzhledem k tomu, že tento projekt lze klasifikovat jako menší, získali jsme jeden cíl od Národní linky pro odvykání (NLO):



Obrázek 3.1. Byznys cíle projektu

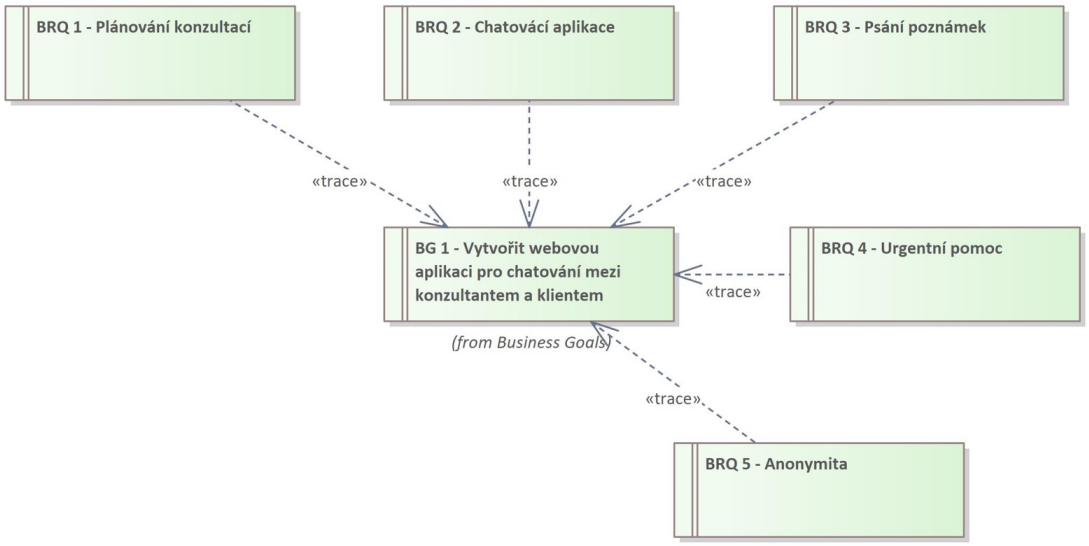
Pod tímto cílem je myšleno propojit pracovníka Národní linky pro odvykání a klienta (závislého na nikotinu, alkoholu atd.) prostřednictvím chatovácích místností. Po celou dobu fungování měla linka pouze telefonní linku, kam se klienti mohli obrátit v případě potřeby.

3.2 Byznys požadavky

Na rozdíl od byznys cílů projektu určují byznys požadavky důvody a přínosy realizovaných částí. Požadavky vztahují se ke konkrétním cílům a jsou vyjádřeny ve formě:

`Jako <role> potřebuji <cíl/přání>, protože <přínos>.`[36]

Požadavky, které jsme od NLO obdrželi, jsou zobrazeny na následujícím obrázku.



Obrázek 3.2. Byznys požadavky projektu

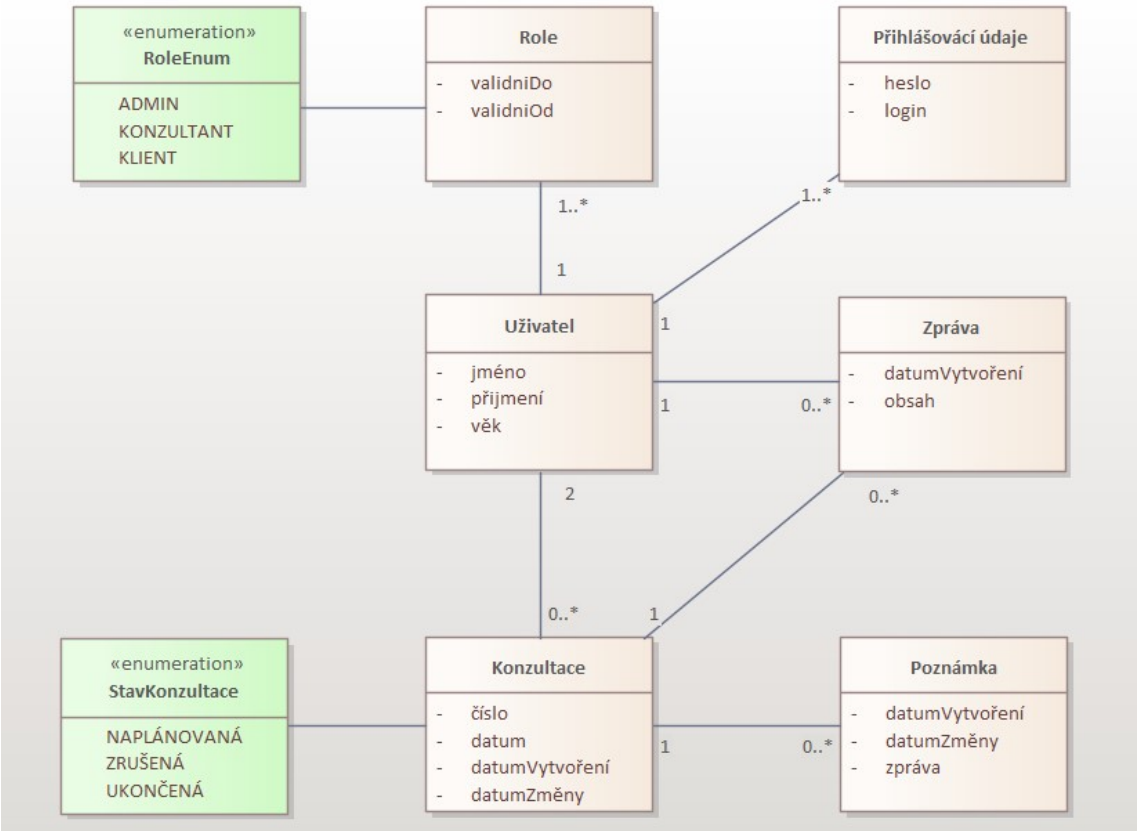
Postupně projdeme jednotlivé požadavky:

- BRQ 1 - Plánování konzultací
  - Jako pracovník NLO potřebuji mít možnost naplánovat, rušit a editovat jednotlivé konzultace, protože se chci zabývat jedním klientem v konkrétním čase.
  - Jako klient NLO chci mít možnost domluvit si čas konzultace, protože chci mít jistotu, že mi bude věnován čas a pomoc.
- BRQ 2 - Chatovácí aplikace
  - Jako pracovník NLO chci mít chatovácí aplikaci, protože chci svým klientům poskytnout možnost kontaktování linky nejenom prostřednictvím telefonní linky.
  - Jako klient NLO chci mít možnost kontaktovat linku prostřednictvím chatu, protože to usnadní celý proces komunikace a přinese další výhody, například větší anonymitu.
- BRQ 3 - Psaní poznámek
  - Jako pracovník NLO chci mít možnost vytvářer a editovat si během konzultace poznámky, protože potřebuji shromažďovat informace o klientovi pro další konzultace..
- BRQ 4 - Urgentní pomoc
  - Jako klient NLO potřebuji mít možnost napsat konzultantovi, i když jsem neměl domluvenou konzultaci, protože mohu potřebovat urgentní pomoc ze strány linky.
  - Jako pracovník NLO potřebuji mít možnost chatovat s klientem, i když nebyla naplánována konzultace, protože klient může potřebovat urgentní pomoc.
- BRQ 5 - Anonyymita
  - Jako klient chci mít možnost napsat konzultantovi, aniž bych se musel v systému zaregistrovat, protože je možné, že nebudu systém používat nadále.

3.3 Byznysový doménový model

Byznysový doménový model je konceptuální reprezentací nezbytných entit, jejich vazeb a chování uvnitř společnosti. BDM zapouzdřuje operace, procesy a data, se kterými byznys pracuje, a je organizovaným vizuálním představením, které přináší zainteresovaným osobám, softwarovým inženýrům a vývojářům společné pochopení systému.

Cílem BDM je vytvořit společnou strukturu pro diskuze a vývoj softwaru, který bude odpovídat potřebám a požadavkům obchodu. Pomocí něj lze lépe identifikovat klíčové části domény, jejich charakteristiky a vztahy mezi nimi. Při vytváření BDM, vývojář může pochopit, jak probíhají jednotlivé operace a zpracování dat v požadovaném systému, což pomahá správnému návrhu architektury, která bude odpovídat cílům obchodu.



Obrázek 3.3. Byznysový doménový model

Jelikož v dalších částech popíšu systém detailněji, nebudu zde uvádět význam každého objektu a jejich propojení.

Kapitola 4
Use-Case model

Model případu užití je základním nástrojem softwarového inženýrství, který pomáhá zachytit a reprezentovat funkční potřeby softwarového systému z pohledu uživatelů. Nabízí strukturovanou metodu rozpoznávání a dokumentování interakcí mezi uživateli (označovanými jako aktéři) a samotným systémem. Model případů užití ukazuje různé způsoby interakce uživatelů se systémem za účelem dosažení určitých cílů, což umožňuje vývojářům efektivně navrhovat a vyvíjet software, který splňuje požadavky uživatelů. Model obsahuje tzv. aktéra - abstraktní entitu, která jednotlivý případ užití spouští. Pro každy use case je doporučeno definovat scenář, který podrobně popisuje, jak se případ užití vyvíjí v určité situaci, včetně procesů, akcí a očekávaných reakcí systému. [37]

Jednou z hlavních motivací pro použití Use-Case modelu je pochopit a prozkoumat chování a možnosti systému z pohledu uživatele. Umožňuje vývojářům softwaru odhalit základní funkce a vlastnosti požadované systémem pomocí definování a dokumentování jednotlivých `use case`, které představují konkrétní akce nebo transakce prováděné aktéry. Toto pochopení slouží jako základ pro návrhová rozhodnutí, definování hranic systému a stanovení priorit vývojového postupu.

Přístup založený na případech užití navíc zlepšuje komunikaci mezi zúčastněnými stranami, jako jsou obchodní analytici, softwaroví inženýři a klienti. Poskytuje společný jazyk, překlenuje propast mezi technickými a netechnickými zúčastněnými stranami a umožňuje jim sdílet znalosti o zamýšleném fungování systému. V průběhu celého životního cyklu vývoje softwaru poskytují případy užití jasný a jednoduchý popis chování systému, což zúčastněným stranám usnadňuje ověřování a revizi požadavků, odhalování potenciálních problémů a případná řešení.

Model případu užití je také důležitý při definování rozsahu a hranic systému. Určením aktérů a jejich interakcí se systémem mohou softwaroví inženýři definovat funkční a behaviorální hranice softwaru. Pomáhá to zabránit rozšiřování rozsahu a udržuje úsilí při vývoji softwaru zaměřené na důležité funkce, které přinášejí spotřebitelům požadovanou hodnotu.[38]

Slouží také jako základ pro tvorbu testovacích scénářů a hodnocení fungování systému. Softwaroví inženýři mohou sestavením testů na základě modelu ověřit, zda systém splňuje daná kritéria a funguje tak, jak je zamýšleno, v reálných podmínkách.

Celkově lze říci, že model případu užití je v softwarovém inženýrství silným nástrojem pro pochopení, zdokumentování a vysvětlení funkčnosti systému z pohledu uživatele. Pomáhá při zachycení požadavků, stanovení hranic systému, usnadnění efektivní komunikace a konstrukci kompletních testovacích případů.

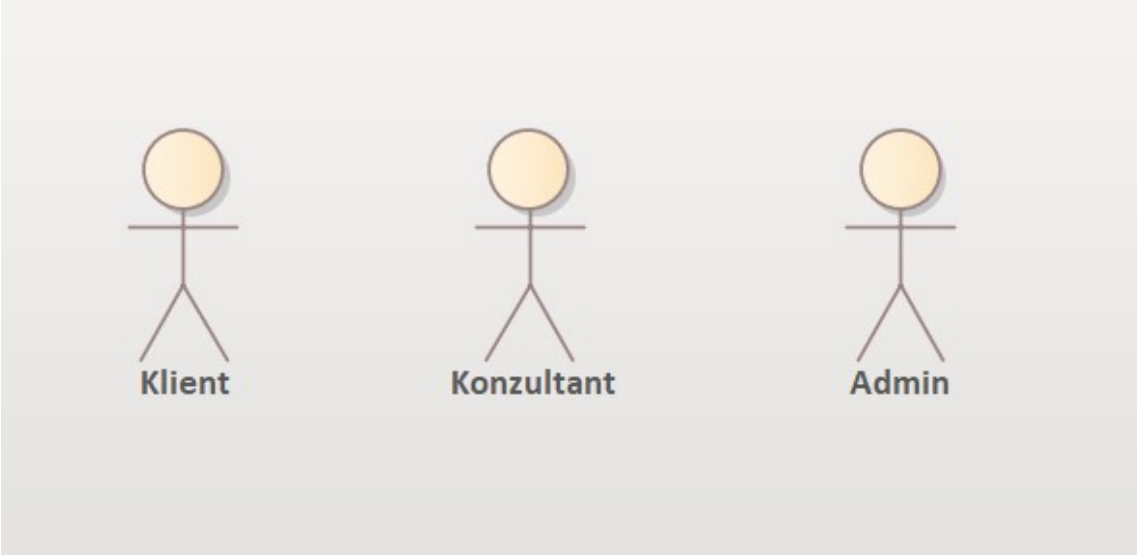
4.1 Aktéři

V modelu případů užití představují aktéři četné subjekty, které přímo interagují se systémem. Jako aktéři mohou vystupovat jednotlivci, externí systémy nebo dokonce jiné softwarové programy. Jsou to osoby nebo role, které se zapojují do případů užití.

Aktéři se dělí na dva typy: primární a sekundární aktéry. Primární aktéři jsou uživatelé nebo zainteresované strany, které se systémem přímo spolupracují za účelem dosažení stanovených cílů nebo úkolů. Zahajují a spouštějí případy užití a hrají důležitou roli při určování chování a fungování systému. Například v aplikaci e-obchodu může být hlavním aktérem spotřebitel, který se systémem komunikuje za účelem prohlížení položek, zadávání objednávek a placení.

Sekundární aktéři jsou naopak subjekty, které jsou do případů užití systému zapojeny nepřímo. Mohou systému dodávat data nebo služby nebo od něj data přijímat, ale nejsou hlavními spotřebiteli. Externí systémy, databáze nebo služby třetích stran jsou příkladem sekundárních aktérů, s nimiž systém komunikuje při provádění určitých funkcí. Tito účastníci hrají v celkovém fungování systému podpůrnou roli.[39]

Protože náš systém nevyužívá služeb třetích stran, máme mezi aktéry pouze primární uživatele systému, a to:

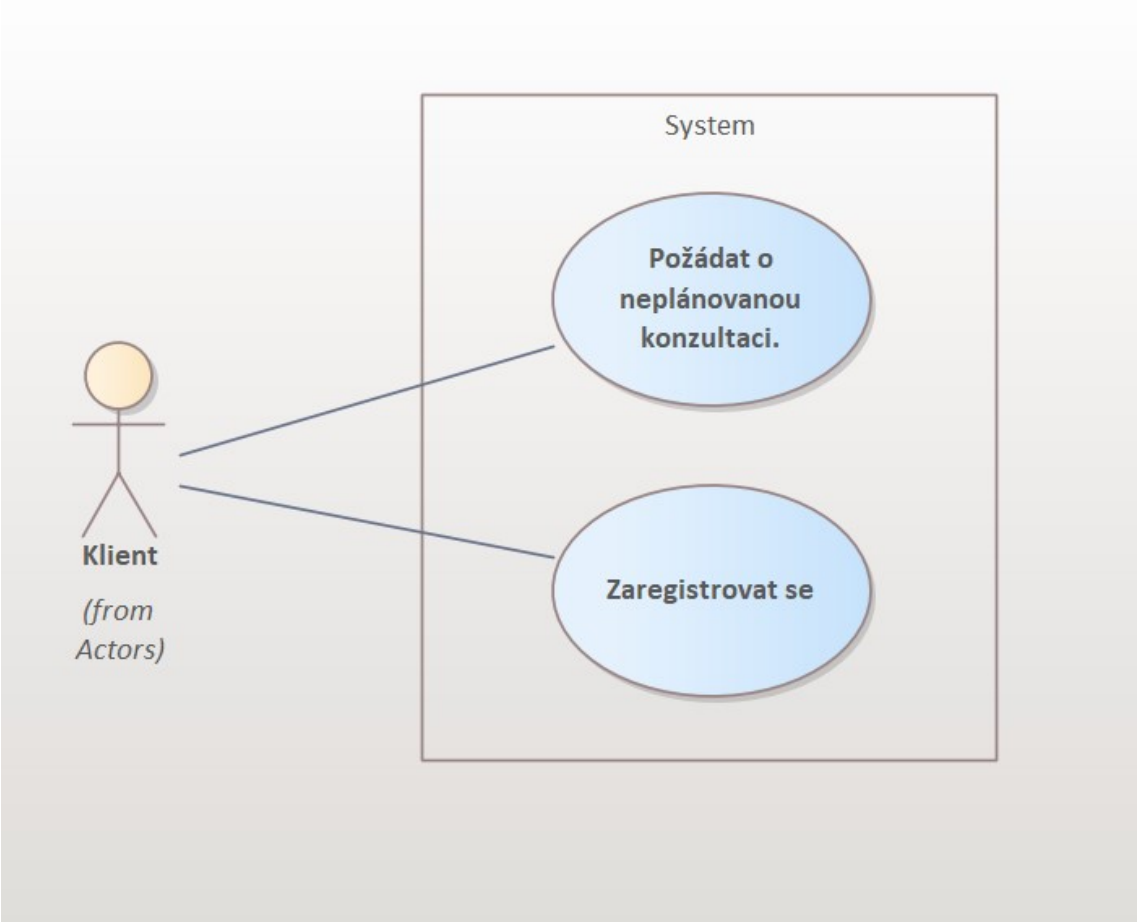


Obrázek 4.1. Aktéři

Každý aktér má svou roli a reprezentuje určitou skupinu uživatelů. Postupně probereme každého aktéra.

4.1.1 Klient

Aktér, který reprezentuje klienta NLO, který požádal o pomoc v odvykání závislostí na nikotinu, alkoholu nebo jiných věcech, které jsou fyzicky nebo psychicky návykové.

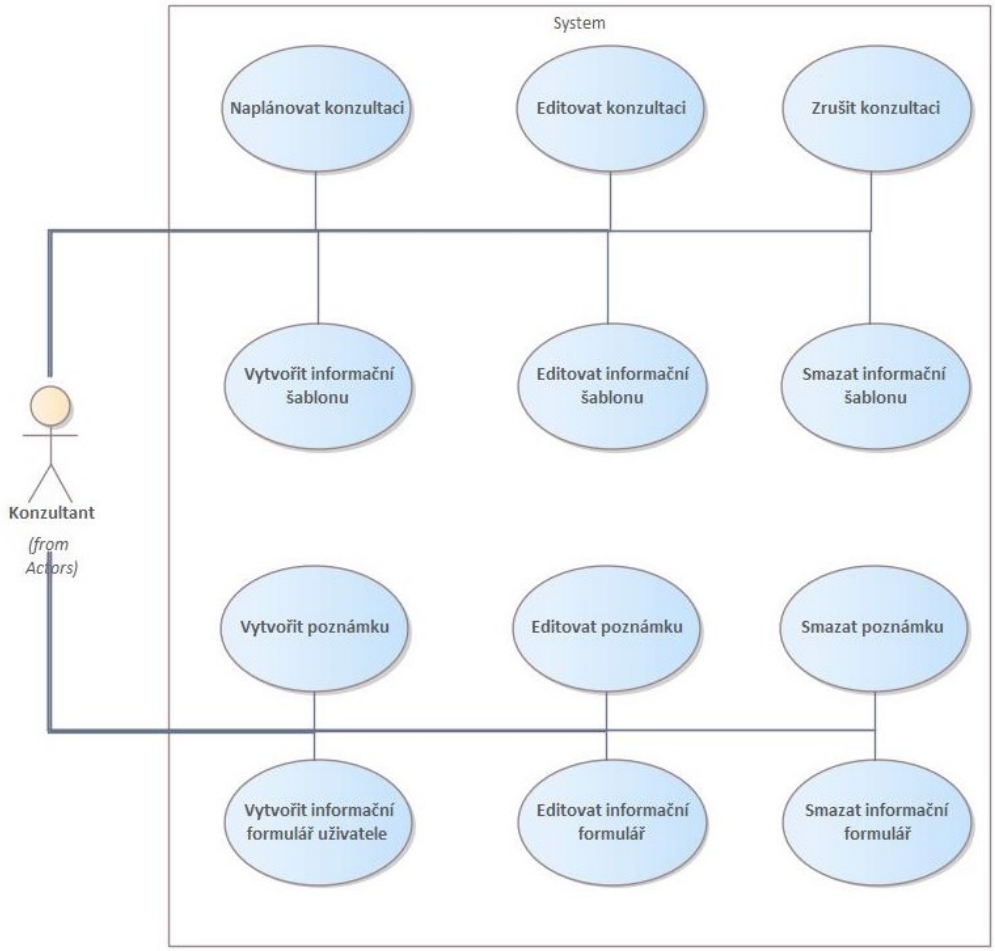


Obrázek 4.2. Jednotlivé případy užití klienta

- Požádat o neplánovanou konzultaci - klient, který potřebuje urgentní pomoc může požádat o neplánovanou konzultaci. Systém mu zobrazí, jak dlouho by klient musel čekat, až někdo z konzultantů bude volný. Scénář:
  1. Klient požadá o neplánovanou konzultaci.
  2. Systém zobrazí uživateli frontu čekajících klientů ve tvaru počtu klientů a přibližný čas čekání.
  3. Klient potvrdí, zda bude chtít do fronty vstoupit.
  4. Systém přidá klienta do fronty.
- Zaregistrovat se - klient se může v systému zaregistrovat. Pro registraci bude potřebovat zadat e-mail a heslo. Pro potvrzení e-mailu uživatel obdrží potvrzovácí kód, který bude muset zadat v registračním formuláři. Scénář:
  1. Systém zobrazí uživateli registrační okno.
  2. Uživatel vyplní formulář a potvrdí registraci.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém odešle potvrzovácí kód na zadaný e-mail] ELSE [Systém zobrazí chybové hlášky o špatně zadaných polích]

4.1.2 Konzultant

Aktér, který reprezentuje pracovníka NLO, již pomahá klientům s odvykáním.



Obrázek 4.3. Jednotlivé případy užití konzultanta

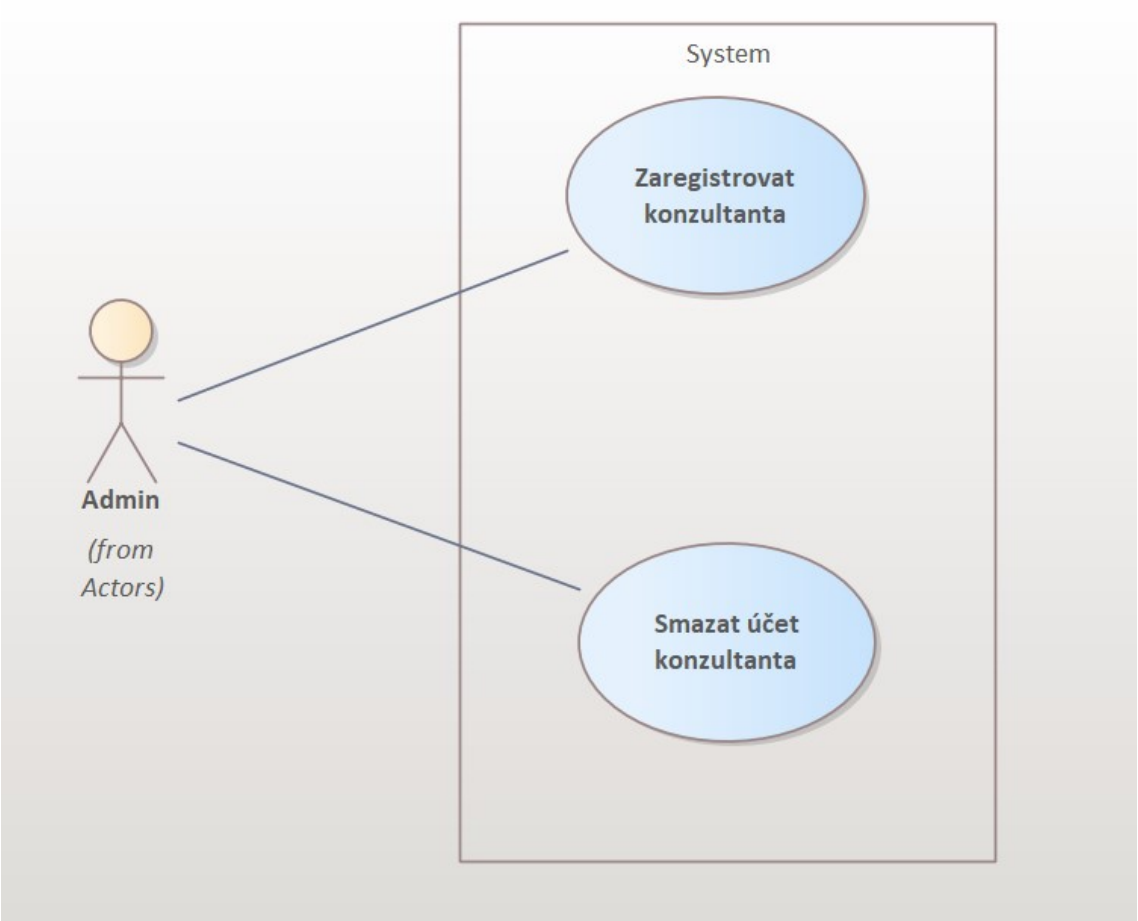
- Naplánovat konzultaci - konzultanti mohou v systému vytvářet konzultace a plánovat je. Scénář:
  1. Systém zobrazí konzultantovi okno pro vytváření nové konzultaci.
  2. Konzultant vyplní údaje a potvrdí je.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém uloží novou konzultaci] ELSE [Systém zobrazí chybové hlašky u špatně zadaných polí]
- Editovat konzultaci - konzultanti mohou editovat jednotlivé konzultace, např. měnit termín konání konzultace. Scénář:
  1. Systém zobrazí konzultantovi vybranou konzultaci.
  2. Konzultant vyplní údaje a potvrdí uložení.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém uloží novou konzultaci] ELSE [Systém zobrazí chybové hlašky u špatně zadaných polí]
- Zrušit konverzaci - v případě potřeby konzultanti mohou rušit jednotlivé konzultace. Scénář:
  1. Systém zobrazí konzultantovi vybranou konzultaci.
  2. Konzultant zruší konzultaci a potvrdí akci.
  3. Systém nastaví stav konzultace na `ZRUŠENA`.
- Vytvořit informační šablonu - konzultanti mohou v systému vytvářet šablony pro informační formuláře. Scénář:

1. Systém zobrazí konzultantovi formulář pro vytvoření informační šablony.
2. Konzultant nadefinuje potřebná políčka (dá název a upřesní typ) a potvrdí uložení šablony.
3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém vytvoří šablonu] ELSE [Systém zobrazí chybové hlašky u špatně vyplněných polí]

- Editovat informační šablonu - konzultanti mohou informační šablony měnit, např. přidat nebo smazat pole. Scénář:
  1. Systém zobrazí konzultantovi informační šablonu.
  2. Konzultant provede potřebné změny v šablone (např. přidá nebo odstraní pole, příp. pozmění typ existujícího pole) a potvrdí uložení změn.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém uloží aktualizovanou šablonu] ELSE [Systém zobrazí chybové hlašky u špatně vyplněných polí]
- Smazat informační šablonu - konzultanti v případě potřeby mohou celou šablonu smazat. Scénář:
  1. Systém zobrazí konzultantovi potvrzovácí okno o mazání šablony.
  2. Konzultant potvrdí mazání.
  3. Systém smaže šablonu a všechny informační formuláře, které používají tutu šablonu.
- Vytvořit poznámku - Konzultanti mohou psát poznámky k jednotlivým konzultacím. Scénář:
  1. Systém zobrazí konzultantovi okno pro vytváření poznámky.
  2. Konzultant napíše poznámku a uloží ji.
  3. Systém uloží poznámku.
- Editovat poznámku - konzultanti mohou poznámky měnit. Scénář:
  1. Systém zobrazí konzultantovi poznámku ke konzultaci.
  2. Konzultant změní obsah požnámky a uloží ji.
  3. Systém uloží editovanou pozámku.
- Smazat poznámku - v případě potřeby konzultanti mohou poznámku i smazat. Scénář:
  1. Systém zobrazí konzultantovi poznámku ke konzultaci.
  2. Konzultant smaže poznámku a potvrdí mazání.
  3. Systém smaže poznámku.
- Vytvořit informační formulář uživatele - konzultanti mohou vytvořit nový formulář v účtu klienta podle nějaké existující šablony. Scénář:
  1. Systém zobrazí konzultantovi názvy existujících šablon.
  2. Konzultant vybere jednu z nabízených šablon a potvrdí vytvoření.
  3. Systém uloží v účtu klienta nový formulář.
  4. Systém zobrazí konzultantovi formulář.
- Editovat informační formulář - konzultanti mohou informační formuláře měnit, např. vyplňovat jednotlivá políčka. Scénář:
  1. Systém ukaže konzultantovi informační formulář.
  2. Konzultant pozmění ve formuláři potřebná pole a potvrdí uložení změn.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém uloží aktualizovaný formulář] ELSE [Systém zobrazí chybové hlašky u špatně vyplněných polí]
- Smazat informační formulář - konzultant může smazat informační formulář pro konkrétního uživatele. Scénář:
  1. Systém zobrazí konzultantovi potvrzovácí okno o mazání formuláře.
  2. Konzultant potvrdí mazání.
  3. Systém smaže formulář v účtu uživatele.

4.1.3 Administrátor

Aktér reprezentuje administrátora (dále jenom admin) systému se všemi právy.

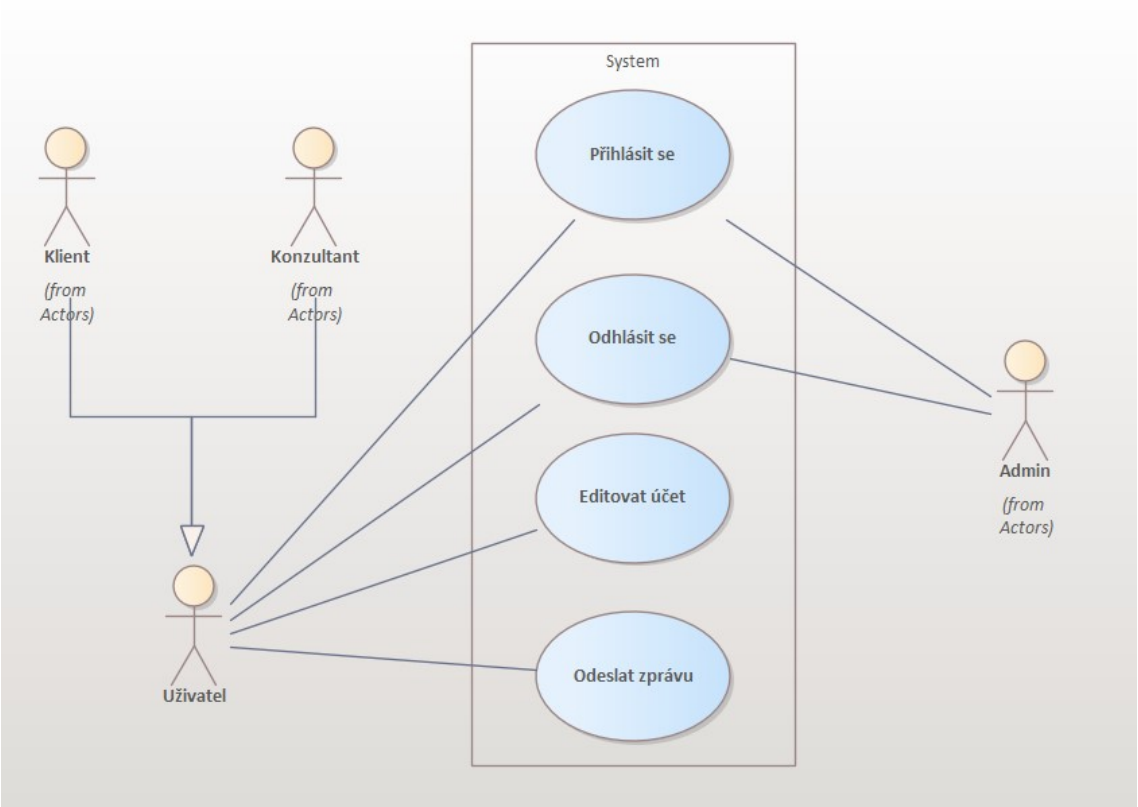


Obrázek 4.4. Jednotlivé případy užití admina

- Zaregistrovat konzultanta - admin systému může vytvářet účty pro konzultanty. Scénář:
  1. Systém zobrazí adminovi formulář pro registraci konzultanta.
  2. Admin vyplní pole e-mail a heslo.
  3. Systém vytvoří účet se zadanými údaje.
- Smazat účet konzultanta - admin může smazat jakýkoliv účet konzultanta NLO. Scénář:
  1. Systém zobrazí adminovi účet konzultanta.
  2. Admin smaže účet a potvrdí operaci.
  3. Systém smaže účet.

4.1.4 Případy užití pro všechny role

Kromě případů užití přiřazených přímo jedné z rolí existuje v systému několik dalších use casů, které mohou být spouštěny různými aktéry.



Obrázek 4.5. Obecne případy užití

- Přihlásit se - registrovaný uživatel se může do systému přihlásit. Pro přihlášení bude muset zadat e-mail a heslo, které zadaval při registraci. Scénář:
  1. Systém zobrazí uživateli přihlašovácí okno.
  2. Uživatel vyplní formulář.
  3. IF [Systém zkontoluje správnost zadaných údajů] THEN [Systém přesměruje uživatele na hlávní stránku] ELSE [Systém zobrazí chybové hlášky u špatně vyplněných polí]
- Odhlásit se - Přihlášený uživatel se může ze systému odhlásit. Scénář:
  1. Uživatel potvrdí odhlašení ze systému.
  2. Systém odhlasí uživatele a přesměruje ho na hlávní stránku.
- Editovat účet - uživatelé mohou měnit svůj účet. Konzultanti navíc mohou měnit účty jiných uživatelů. Scénář:
  1. Systém zobrazí uživateli okno pro editaci účtu.
  2. Uživatel vyplní formulář.
  3. IF [Systém zkontroluje správnost zadaných údajů] THEN [Systém uloží změny] ELSE [Systém zobrazí chybové hlášky u špatně vyplněných polí]
- Odeslat zprávu - přihlášený uživatel může posílat zprávy jiným uživatelům.
  1. Uživatel napíše zprávu a potvrdí odeslání.
  2. IF [Systém zkontroluje délku zprávy] THEN [Systém odešle zprávu cílovému uživateli] ELSE [Systém zobrazí chybovou hlášku o přesahnuté délce zprávy]

Kapitola 5
Technická část projektu

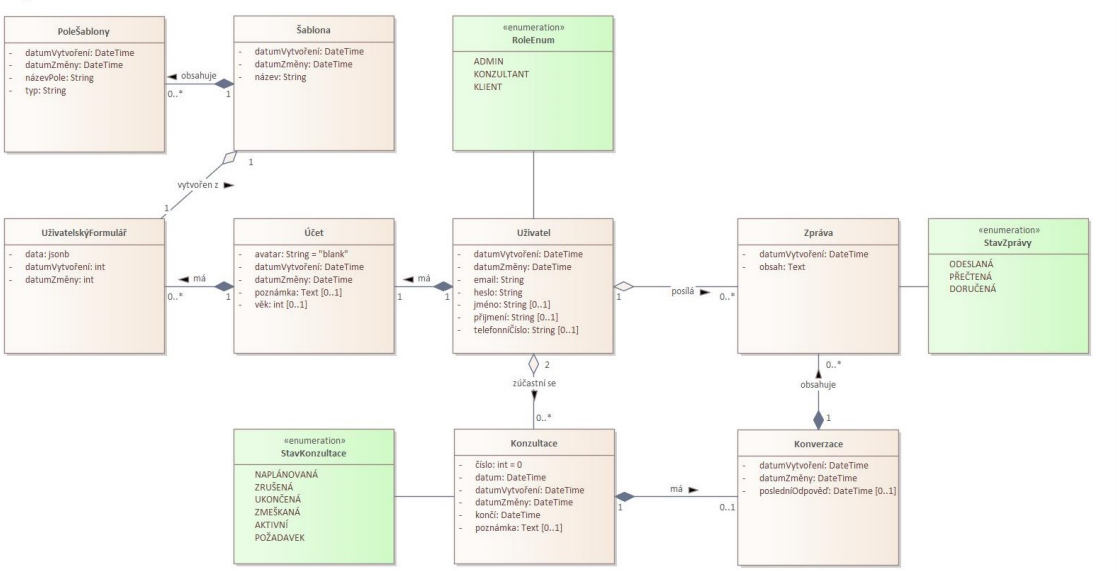
V této kapitole si projdeme architekturu aplikace z technického hlediska. Začneme diagramem tříd, pak plynule přejdeme k softwarovým požadavkům projektu, nadefinujeme komponenty, z nichž se aplikace skládá, a na závěr si ukážeme diagram nasazení.

5.1 Analytický doménový model

Analytický model domény je reprezentací problémové oblasti nebo předmětu, který řeší softwarový systém. Zaměřuje se na zachycení klíčových pojmů, pravidel a souvislostí v problémové doméně a slouží jako základ pro návrh a implementaci softwarových řešení. Analytický model domény jde hlouběji do konkrétního řešeného problému nebo odvětví než vysokoúrovňový model business domény a nabízí důkladnější a přesnější znalost problémové oblasti.

Zatímco model byznys domény se zaměřuje na obchodní operace a cíle organizace na vysoké úrovni, model analytické domény se soustředí na konkrétní problém a složitost jeho řešení. Zkoumá danou oblast problému z technického hlediska a bere v úvahu i takové aspekty, jako jsou datové struktury, algoritmy a chování systému. Cílem analytického modelu domény je transformovat reálný problém do strukturované a uspořádané reprezentace, která bude řídit proces vývoje softwaru.[40]

Z hlediska hloubky informací a technologické koncentrace se model analytické domény liší od modelu byznysu. Model obchodní domény poskytuje přehled o činnosti, cílech a zainteresovaných stranách organizace s důrazem na funkční požadavky a interakce mezi uživatelem a systémem. Model analytické domény naproti tomu proniká hlouběji do problémové oblasti a rozkládá ji na menší, lépe zvládnutelné komponenty. Identifikuje entity, vlastnosti, vazby a chování jedinečné pro daný problém, což umožňuje důkladnější studium a návrh softwarového řešení.



Obrázek 5.1. Analytický doménový model

Detalněji projdeme přes všechny třídy modelu:

- Uživatel - abstráktní uživatel systému. Obsahuje následující atributy:
  - datumVytvoření [Typ: DateTime] - datum vytvoření objektu, což odpovídá datu registrace uživatele.
  - datumZměny [Typ: DateTime] - datum poslední změny objektu.
  - email [Typ: String] - e-mailová adresa, kterou uživatel zadal při registraci.
  - heslo [Typ: String] - heslo, které uživatel zadal při registraci.
  - jméno [Typ: String]- křestní jméno uživatele. Nepovinný atribut.
  - přijmení [Typ: String]- přijmení uživatele. Nepovinný atribut.
  - telefonníČíslo [Typ: String] - telefonní číslo uživatele. Nepovinný atribut.
  - uživatel také obsahuje informace o své roli, která odpovídá jedné z hodnot RoleEnum. Defaultně je nastavená role `KLIENT`.
- Účet - uživatelský účet s dopňující informací o uživateli. Atributy:
  - avatar [Typ: String] - název avataru, který si může zvolit uživatel. Defaultní hodnota je `blank`.
  - datumVytvoření a datumZměny mají stejný význam jako ve třídě Uživatel, a nebudu zmíňovat o nich v násldujících třídách.
  - poznámka [Typ: Text] - Poznámka k uživateli, kterou může přidat konzultant. Nepovinný atribut.
  - věk [Typ: Integer] - věk uživatele. Nepovinný atribut.
- Šablona - třída reprezentující šablonu uživatelských formulářů. Atributy:
  - název [Typ: String] - název šablony (např. User Info).
- PoleŠablony - Třída reprezentující jedno pole v šabloně
  - typ [Typ: String] - Typ hodnoty, který se bude validovat na FE.
  - názevPole [Typ: String] - název pole, který uvádíme pro jednodušší identifikaci.
- Konzultace - konzultace mezi klientem a pracovníkem Národní linky pro odvýkaní (konzultantem). Atributy:
  - číslo [Typ: Integer] - číslo konzultace s klientem. Defaultní hodnota je 0.
  - datum [Typ: DateTime] - datum a čas naplánované konzultace.
  - poznámka [Typ: Text] - poznámka ke konzultaci, kterou muže vyplnit konzultant. Nepovinný atribut.

  - stav - odpovídá jedné z hodnot výčtového typu `StavKonzultace`. Defaultně je nastaven na `NAPLÁNOVANÁ`.

- Konverzace - třída reprezentující samotnou konverzaci mezi klientem a konzultantem. Atributy:
  - posledníOdpověď [Typ: DateTime] - datum a čas poslední odpovědi v rámci konverzace. Nepovinný atribut.

- Zpráva - zpráva, kterou posílá uživatel jinému uživateli v rámci konverzace.
  - obsah - obsah (`body`) zprávy. Je ohraničen 300 znaky.
  - stav - odpovídá jedné z hodnot výčtového typu `StavZprávy`. Defaultně je nastaven na `ODESLANÁ`.

5.2 Softwarové požadavky

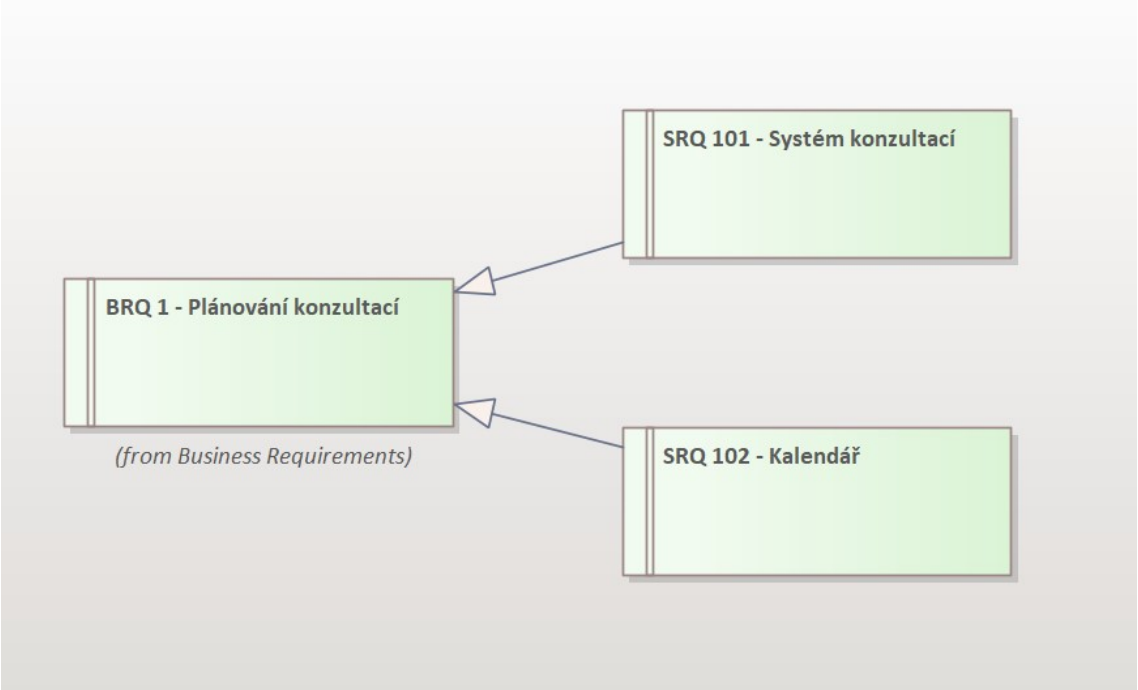
Softwarové požadavky jsou návody a definice toho, co musí softwarový systém dělat. Definují funkce, vlastnosti a atributy, které by měl program mít, aby splnil očekávání zainteresovaných stran a obchodní cíle. Softwarové požadavky slouží jako základ pro návrh, vývoj a testování softwarového systému[41].

Na rozdíl od byznys požadavků, které se zaměřují na cíle a záměry organizace na vysoké úrovni, softwarové požadavky se zabývají technickou složitostí toho, jak by měl systém chovat a fungovat. Ve srovnání s obchodními požadavky jsou podrobnější, přesnější a kvantifikovatelnější. Softwarové požadavky stanovují očekávané chování systému, specifikují vstupy a výstupy a nastiňují případná omezení nebo kritéria kvality, která musí software splňovat.

Softwarové požadavky se obvykle vytvářejí na základě obchodních požadavků a překládají se do technického jazyka. Odrážejí konkrétní a realizovatelné vlastnosti systému, které mu umožňují podporovat zamýšlené obchodní operace. Softwaroví inženýři úzce spolupracují se zúčastněnými stranami při získávání, posuzování a zaznamenávání softwarových požadavků, přičemž dbají na to, aby odpovídaly obchodním cílům a zároveň zohledňovaly technologickou proveditelnost a omezení.

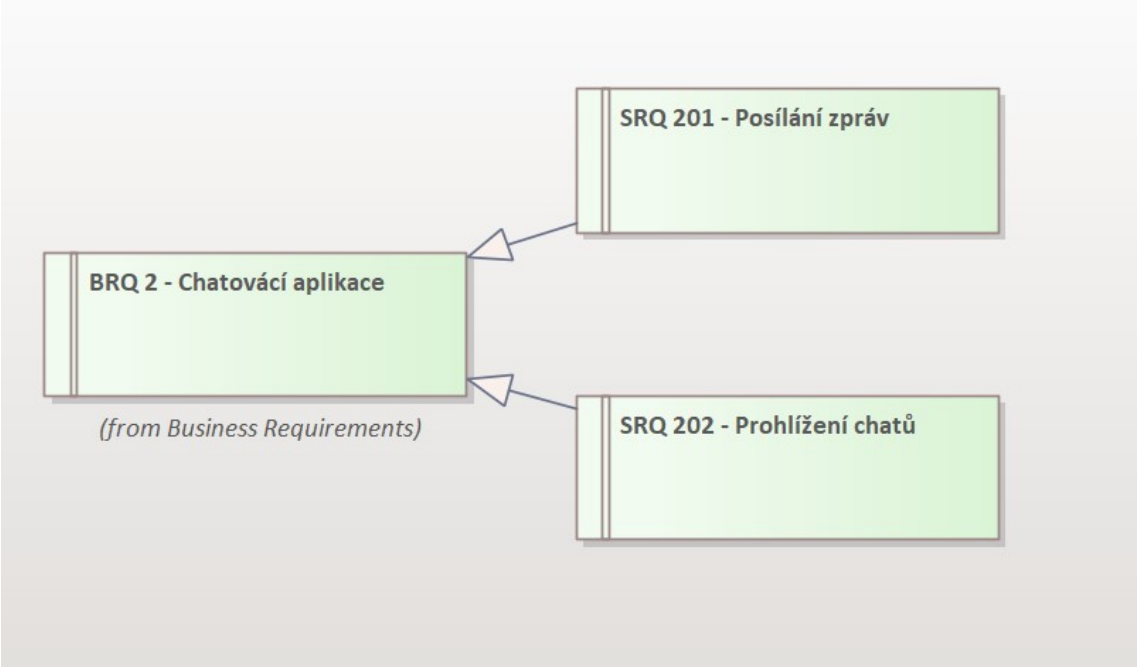
Softwarové požadavky jsou nezbytné pro řízení procesu vývoje softwaru. Slouží k rozhodování o návrhu, vývoji softwarových systémů a ověřování jejich funkčnosti. Softwaroví inženýři mohou zaručit, že hotové řešení splní očekávání zúčastněných stran a bude odpovídat specifikovaným potřebám, pokud explicitně nastíní požadavky na software. Také slouží jako referenční bod pro testování, ověřování a operace údržby v průběhu celého životního cyklu vývoje softwaru a zajišťují, že systém zůstane v souladu se zamýšlenou funkčností a cíli.

Jelikož softwarové požadavky jsou navázané na byznys požadvky, doporučuji zopakovat jednotlivé BRQ, které najdete na v odpovídající kapitole. Pro pojmenování jednotlivých softwarových požadavků jsem se rozhodl použít format `číslo byznys požadavku` hned za kterým následuje číslo samotného SRQ, např. SRQ 101 znamená, že se jedná o první SRQ v pořadí, který je navazán na BRQ 1.



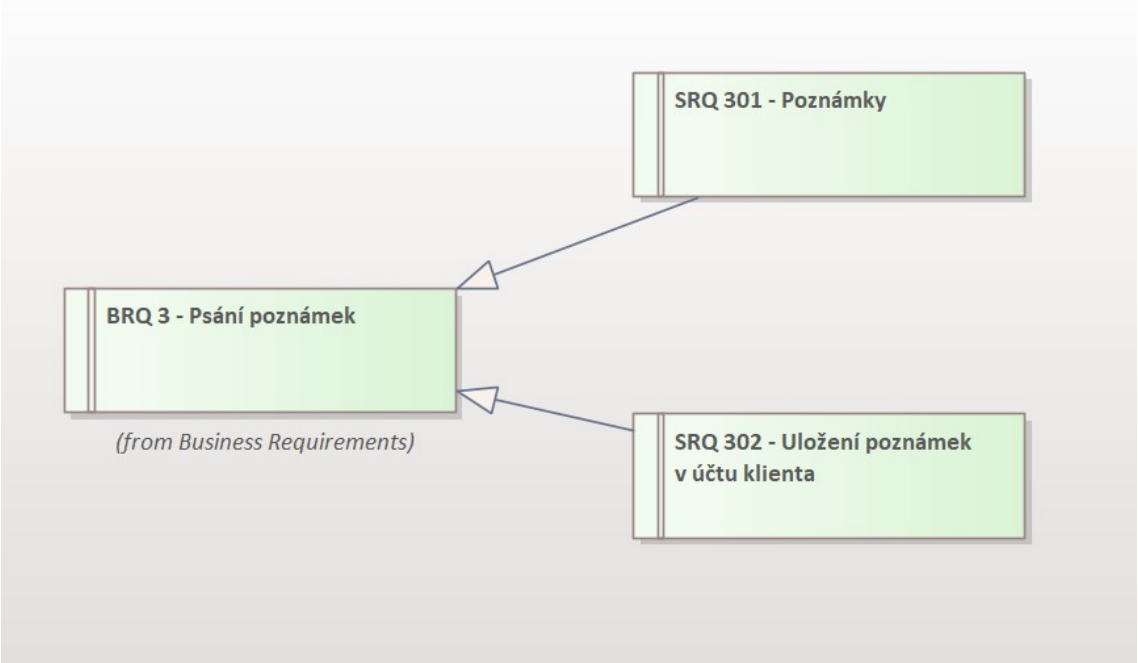
Obrázek 5.2. Softwarové požadavky navázané na BRQ 1

- SRQ 101 `Systém konzultací` - systém umožní pracovníkům linky vytvářet, editovat a rušit konzultace.
- SRQ 102 `Kalednář` - systém umožní uživatelům prohlížet naplánované údalosti.



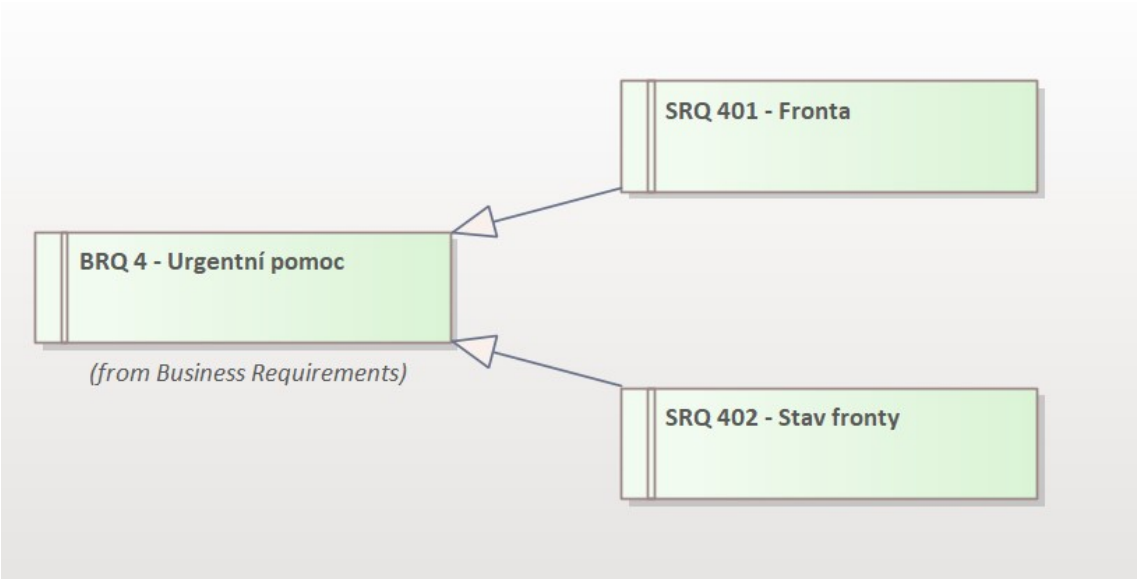
Obrázek 5.3. Softwarové požadavky navázané na BRQ 2

- SRQ 201 `Posílání zpráv` - systém umožní uživatelům posílat mezi sebou zprávy ve tvaru chatu. Samotné posílání je realizované pomocí komunikačního protokolu WebSokets (viz. více v kapitolě o použitých technologiích).
- SRQ 202 `Prohlížení chatů` - systém umožní uživatelovi prohlížet existující chaty, kterých se zúčastnil.



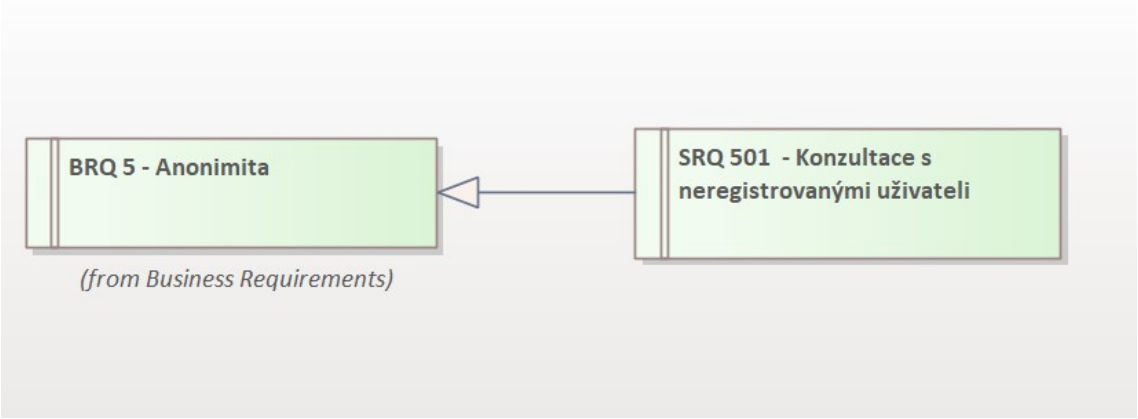
Obrázek 5.4. Softwarové požadavky navázané na BRQ 3

- SRQ 301 `Poznámky` - systém umožní konzultantům vytvářet, editovat a mazat poznámky během konzultací.
- SRQ 302 `Uložení poznámek v účtu uživatele` - systém umožní konzultantům uložit vytvořené v konzultaci poznámky v účtu klienta.



Obrázek 5.5. Softwarové požadavky navázané na BRQ 4

- SRQ 401 `Fronta` - systém umožní automatické vytváření a kontrolu fronty nadcházejících klientů.
- SRQ 402 `Stav fronty` - systém umožní prohížet stav fronty, a bude ukázovat, kolik lidí je před klientem, a kolik by přibližně klient musel čekat. Také umožní klientum obsadit místo ve frontě.



Obrázek 5.6. Softwarové požadavky navázané na BRQ 5

- SRQ 501 `Konzultace s neregistrovanými uživateli` - systém umožní klientům požádat o konzultaci, aniž by museli se před tím zaregistrovat.

5.3 Komponenty systému

Diagram komponent je vizuální znázornění hierarchické struktury a vztahů komponent softwarového systému, které se používá v softwarovém inženýrství. Je to jeden z diagramů jazyka UML (Unified Modeling Language), který se používá k zobrazení a zdokumentování architektury systému. Diagram komponent popisuje jednotlivé součásti a jejich vzájemnou interakci na vysoké úrovni.

V kontextu diagramu komponent jsou komponenty modulární části softwaru, které zapouzdřují funkce a mohou být nasazeny a nahrazeny samostatně. Lze je znázornit jako obdélníky s názvem komponenty uvnitř. Diagramy komponent používají konektory pro znázornění interakcí a závislostí mezi komponentami, jako jsou vztahy závislosti, asociační vazby a vztahy realizace rozhraní.[42]

Primárně tento druh diagramu pomohý softwarovým inženýrům a vývojářům pochopit, jak jednotlivé komponenty vzájemně spolupracují a jak se podílejí na zajištění zamýšlené funkčnosti systému.

Diagramy komponent pomáhají pochopit modulární strukturu systému, zjednodušují opakované použití komponent a pomáhají při návrhu a implementaci softwarového systému. Usnadňují identifikaci možných míst integrace a umožňují lepší řízení složitosti rozdělením systému na zvládnutelné a ucelené komponenty. Vzhledem k tomu, že různé týmy mohou pracovat na nezávislých komponentách s přesně definovanými rozhraními, mohou diagramy komponent sloužit k odhalení možností paralelního vývoje.

Mohou také poskytnout přehled o nasazení komponent tím, že ukazují, jak jsou komponenty distribuovány mezi fyzickými nebo virtuálními uzly. Tyto informace pomáhají pochopit architekturu nasazení systému a také umožnit naplnění požadavků na škálovatelnost, dostupnost a výkon.

Při vytvoření diagramu komponent je výhodné nejdříve definovat tzv. API katalog, který zobrazuje dostupná rozhraní API v systému. Poskytuje přehled rozhraní API, jejich funkcí a vazeb. Katalogový diagram API slouží vývojářům jako reference, která jim umožňuje snadno objevit a pochopit dostupná API a jejich funkce. Podporuje opakované použití rozhraní tím, že umožňuje vývojářům používat existující API namísto toho, aby je znovu vytvářeli, což šetří čas a úsilí při vývoji softwaru.

Diagram katalogu API slouží jako vizuální reference pro sestavení úplnějšího a přesnějšího diagramu komponent. Zaručuje, že diagram komponent přesně zobrazuje závis-

losti a vazby, které existují mezi jendotlivými komponentami a rozhraními API, která používají. Diagram katalogu API je užitečnou referencí v průběhu celého procesu tvorby diagramu komponent.

5.3.1 API katalog

Vzhledem k tomu, že API, které budu dále ukazovat a popisovat, přímo reprezentují implementované modely a kontroléry, rozhodl jsem se použít angličtinu jako jazyk pro názvy API, operace a atributy, která lépe odpovídá kódu.

Obrázek 5.7. Diagram API konzultací, konverzací a uživatelů

- ConsultationControllerAPI - kontrolér, který přijímá požadavky na obržení/vytvoření/změnu/mazání konzultací. Jednotlivé operace:
  - createConsultation - metoda na vytvoření nové konzultace. Vstupním parametrem je struktura konzultace. Chování se liší v závislosti na uživatelské roli: pokud o vytvoření konzultaci požádá klient, konzultace se vytvoří se statusem `POŽADAVEK` (viz. ADM); pokud o vytvoření požádá konzultant, konzultace se vytvoří se statusem `NAPLÁNOVANÁ`. V případě chyby metoda vyvolá výjimku s popisem důvodu (toto chování je realizované pro všechny API). Vrací status requestu a samotný objekt.
  - deleteConsultationById - metoda, která maže konzultaci podle ID v požadavku. Je vyžadována autorizace a opravnění na mazání. Vrací status požadavku.
  - getAllConsultations - metoda která vrací konzultace na základě role uživatele. Pokud o konzultaci požádal klient, metoda vrátí pouze ty konzultace, které se vztahují k tomuto klientovi; v případě, že o konzultace požádá konzultant, metoda vrátí všechny existující konzultace. Je vyžadována autorizace.
  - getConsultationById - metoda, která vrací konkrétní konzultaci podle ID v requestu. Pokud o konzultaci požádá klient, který ke konzultaci nevztahuje, metoda vyvolá výjimku s odmítnutým oprávněním. Je vyžadována autorizace.
  - updateConsultationById - metoda, která aktualizuje konzultaci. Vstupními parametry jsou ID konzultace a nové údaje o ní. Je vyžadována autorizace a opravnění na editaci. Vrací obnovené údaje o konzultaci a status požadavku.

- ModelAPI - abstraktní API nad modelem. Operace (protože API jednotlivých modelů jsou abstrakcí nad tím, co poskytuje framework, operace, které vystavuje každé API, jsou stejné a jediný rozdíl je jenom v typech vstupních/výstupních parametrů, tedy nemá cenu popísovat operace pro každý model):
  - all - vrací seznam všech objektů podle typu.
  - save - vytváří objekt na základě předaných údajů. Nevrací žádná data.
  - delete - smaže objekt podle ID. Vrací počet smazaných objektů a slovník s počtem smazaných referenci podle jednotlivých typů objektu.
  - get - vrací objekt podle ID.
  - update - aktualizuje objekt podle předaného ID a údajů o něm. Vrací počet aktualizovaných záznamů, které ovlivnila operace.
- ConversationControllerAPI - kontrolér, který přijímá požadavky na vytvoření/obdržení konverzací. Operace:
  - createConversation - metoda na vytvoření nové konverzace. Vstupním parametrém je ID konzultace, pro kterou bude vytvořená konverzace. Vrací status requestu.
  - get - metoda, která vrací konverzaci a jeji obsah podle předaného ID konzultace.
- UserControllerAPI - kontrolér, který přijímá požadavky na obržení/změnu uživatelských účtů.
  - delete - metoda, která smaže veškerá data o uživateli podle ID v požadavku. Mazání dat je realizované jenom pro účty konzultantu. Je vyžadována autorizace a opravnění na mazání. Vrací status požadavku.
  - getAllProfiles - metoda, která vrací účty uživatelů a jejich údaje (např. e-mail, jméno atd.). Je vyžadována autorizace. Chování pro klienta: klient vidí jenom na účty konzultantů; chování pro konzultanta: konzultant vidí na všechny účty.
  - getProfileById - metoda, která vrací účet uživatele a jeho údaje podle ID v requestu. Je vyžadována autorizace.
  - updateProfileById - metoda, která aktualizuje účet uživatele a jeho údaje. Vstupními parametry jsou ID uživatele a nové údaje o profilu/uživateli. Je vyžadována autorizace. Na rozdíl od konzultantů, klient může měnit jenom vlastní údaje.

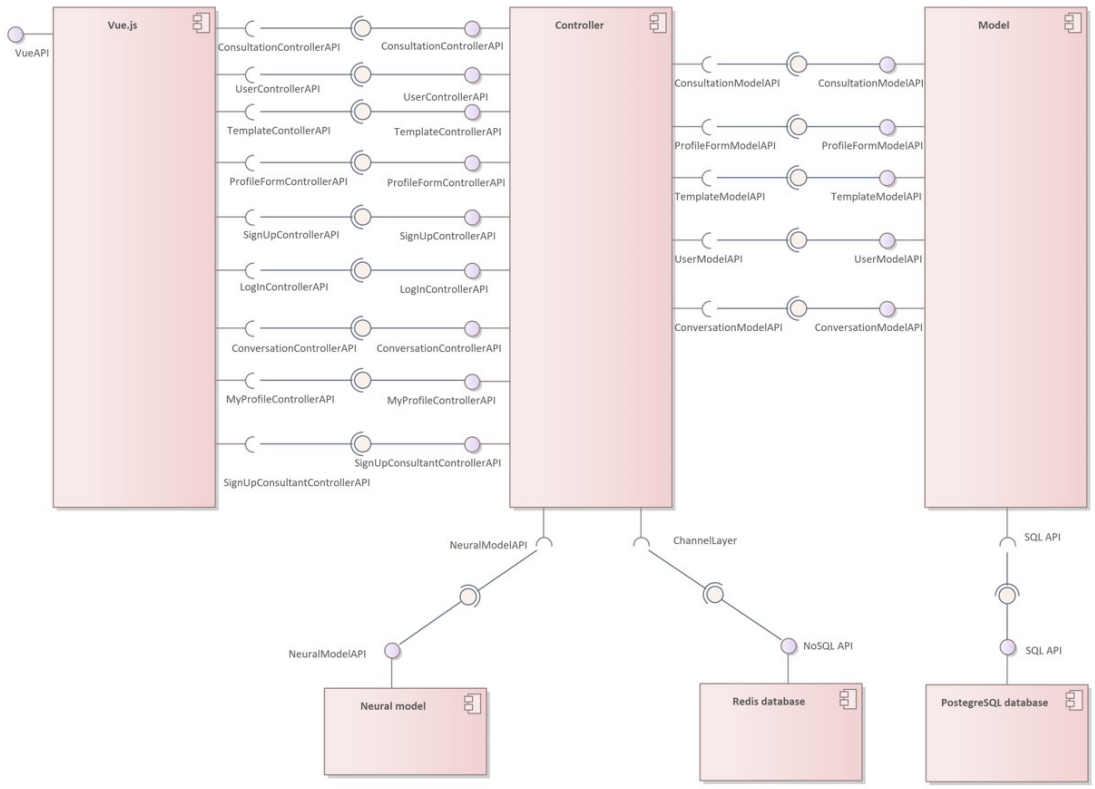
Obrázek 5.8. Diagram API šablon, uživatelských formulářů, přihlášení, registrace a přesměrování na vlastní účet

- TemplateControllerAPI - kontrolér, který přijímá požadavky na obdržení/vytvoření/změnu/mazání šablon informačních formulářů (např. formulář obsahující detailní informace o uživateli). Není přístupný klientům.
  - createTemplate - metoda na vytvoření nové šablony. Vstupním parametrem je struktura šablony (viz. ADM). Je vyžadována autorizace. Vrací status requestu a novou šablonu.
  - deleteTemplateById - metoda, která maže šablonu podle ID v požadavku. Je vyžadována autorizace. Vrací ID smazané šablony.
  - getAllTemplates - Metoda, která vrací existující šablony formulářů. Je vyžadována autorizace.
  - getTemplateById - metoda, která vrací šablonu formuláře podle ID v požadavku. Je vyžadována autorizace.
  - updateTemplateByID - metoda, která aktualizuje šablonu. Vstupními parametry jsou ID šablony a nové údaje o ní. Je vyžadována autorizace. Vrací obnovené údaje o šabloně.
- ProfileFormController - kontrolér, který přijímá požadavky na obdržení/vytvoření/změnu/mazání informačních formulářů pro konkrétního uživatele (např. formulář obsahující detailní informace o uživateli). Není přístupný klientům.
  - createProfileForm - metoda na vytvoření nového uživalského formuláře. Vstupním parametrem jsou ID uživatele a ID nebo název šablony, podle které se bude vytvářet formulář. Je vyžadována autorizace. Vrací status requestu a strukturu formuláře.
  - deleteProfileFormById - metoda, která maže uživatelský formulář podle ID v požadavku. Je vyžadována autorizace.

- generateCSV - metoda, která podle zadaných ID uživatele a uživatelkého formuláře vygeneruje CSV (Comma-separated values) soubor.
- getProfileFormById - metoda, která vrací existující uživatelský formulář podle ID v požadavku. Je vyžadována autorizace.
- getProfileForms - metoda, která vrací existující uživatelské formuláře. Je vyžadována autorizace.
- updateProfileFormById - metoda, která aktualizuje uživatelský formulář. Vstupními parametry jsou ID uživatele, ID formuláře a nové údaje o něm. Je vyžadována autorizace. Vrací obnovené údaje o šabloně.

- LogInControllerAPI - kontrolér, který přijímá požadavky na přihlášení. Požadovanými atributy jsou uživatelské jméno a heslo. Metoda vrací autentizační token.
- SignUpControllerAPI - kontrolér přijímající žádost o regestraci. Požadovanými atributy jsou uživatelský e-mail, heslo a opakované heslo. Metoda vrací objekt obsahující stav requestu a datový objekt, který obsahuje uživatelské jméno vytvořeného uživatele.
- MyProfileControllerAPI - kontrolér přijímající požadavek o přesměrování na vlastní účet. Jediným atributem je HTTP request, ze kterého se dotahuje informace o uživateli. Metoda vrací data o účtu uživatele. V případě chyby metoda vyvolá výjimku s popisem důvodu.
- SignUpConsultantControllerAPI - kontrolér přijímající žádost o regestraci konzultantů. Požadovanými atributy jsou uživatelské jméno, heslo a opakované heslo. Je vyžadována autorizace. Metoda vrací objekt obsahující stav requestu a datový objekt, který obsahuje uživatelské jméno vytvořeného uživatele. Je přístupen pouze pro administrátory systému.

5.3.2 Diagram komponent



Obrázek 5.9. Diagram komponent

- Vue.js - komponenta, která reprezentuje frontend (tj. část aplikace, se kterou uživatel interaguje).
- Controller - komponenta, která představuje část aplikace komunikující s frontendem (přijímá požadavky a odesílá odpovědi prostřednictvím tzv. 'end-pointů').
- Model - komponenta, která představuje část aplikace, jež manipuluje s daty a komunikuje s databází pro vytváření/čtení/aktualizaci/mazání (tzv. CRUD operace).
- Redis database - NoSQL databáze s otevřeným zdrojovým kódem. Je použita pro potřeby komunikace mezi uživateli.
- PostgreSQL database - Relační databáze s otevřeným zdrojrvým kódem. Je použita pro ukládání dat.
- Neural model - komponenta, jež má na starosti automaticky generování odpovědí na zprávy klienta.

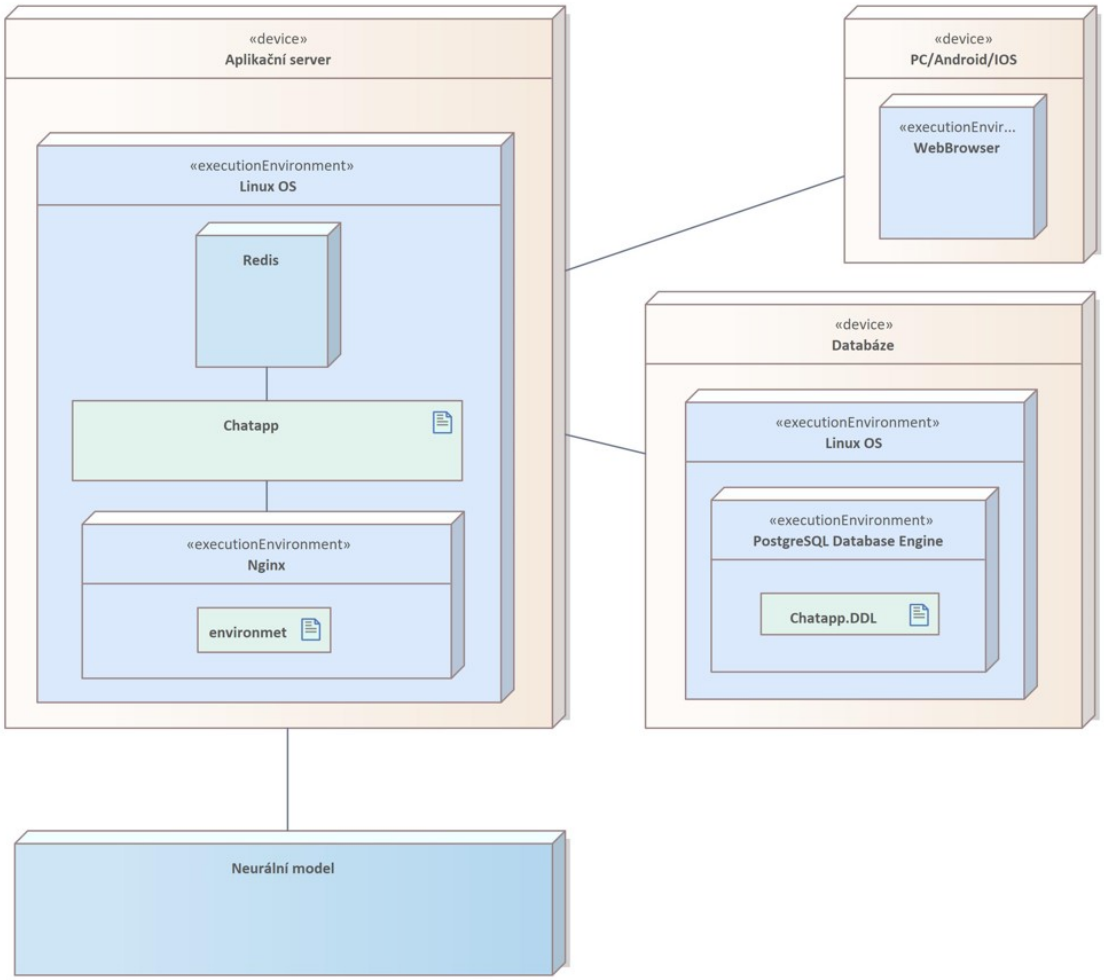
5.4 Diagram nasazení

Diagram nasazení je vizuální znázornění fyzického rozmístění softwarových komponent a jejich interakcí uvnitř systému. Ukazuje nasazení artefaktů, jako jsou komponenty, uzly a prostředí pro provádění na hardwarové infrastruktuře nebo platformách. Diagram nasazení znázorňuje přidělení a konfiguraci softwarových a hardwarových prostředků v architektuře běhu systému.[1]

Hlavním cílem diagramu nasazení je pomoci pochopit fyzické vlastnosti softwarového systému a zaručit úspěšné nasazení a integraci softwarových komponent do cílového prostředí. Pomáhá při identifikaci možných úzkých míst, problémů se škálovatelností a potřeb komunikace mezi různými komponentami a uzly. Diagramy nasazení mohou také pomoci při plánování nasazení systému a stanovení nejlepšího způsobu přidělení softwarových komponent hardwarovým prostředkům.[43]

Diagram nasazení napomáhá komunikaci a spolupráci mezi zúčastněnými stranami, jako jsou vývojáři softwaru, správci systému a týmy infrastruktury. Také usnadňuje koordinaci činností mezi různými týmy zapojenými do vývoje, nasazení a údržby softwarového systému.

[1] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/



Obrázek 5.10. Diagram nasazení

- Application server - kontejner, který udržuje životní cyklus aplikace, přijímá požadavky klientů, zpracovává obchodní logiku a vrací výsledky klientům. [2]
- Database - místo, kam se ukladají data.
- Device - zařízení, přes které se připojuje uživatel.
- Neurální model - je externí části aplikace, se kterou komunikuje BE.

[2] `https://www.gartner.com/en/information-technology/glossary/application-server`

Kapitola 6
Závěr

Závěrem bych rád napsal, že si myslím, že téměř všechny požadavky byly splněny. Jedinou výjimkou je připojení neurálního modelu, který by měl automaticky generovat odpověďi na zprávy klienta. Tuto část se mi bohužel nepodařilo splnit, protože API pro komunikaci s modelem nebylo v době odevzdání práce vedoucím vystaveno.

Jinak byla práce z mého pohledu splněna. Na začátku jsem vytvořil databázový model, který jsem postupně doplňoval podle měnících se požadavků. Poté jsem vytvořil API pro komunikaci s FE a zároveň jsem začal pracovat na části, která umožňuje komunikaci mezi dvěma uživateli. V průběhu psaní kódu jsem se snažil API zdokumentovat, přesněji popsat existující koncové body, strukturu dotazů a odpovědí atd.

Práce mi přišla nesmírně užitečná. Od samostatného projektu, v němž jsem si vybral technologie, které jsem později používal, jsem vstoupil do zcela nového světa programování, protože jsem neměl s vývojem aplikací v Pythonu žádné zvláštní zkušenosti. Jsem rád, že jsem se mohl podílet na projektu, který může pomoci mnoha lidem v jejich snaze překonat závislostí.

Literatura

[1] *Statistika SZO ohledně kouření.*
https://www.who.int/news-room/fact-sheets/detail/tobacco.

[2] *Statistika CDC ohledně kouření.*
https://www.cdc.gov/tobacco/data_statistics/fact_sheets/fast_facts/diseases-and-death.html.

[3] *Statistika SZO ohledně konzumaci alkoholu.*
https://www.who.int/health-topics/alcohol.

[4] *Užívání tabáku v ČR.*
https://www.mzcr.cz/tiskove-centrum-mz/vysledky-narodniho-vyzkumu-szu-o-uzivani-tabaku-v-cr-potvrdily-ze-ceskych-kuraku-ubyva-povedomi-o-zdravotnich-rizicich-koureni-se-zvysilo/.

[5] *Populace ČR v roce 2019 od ČSU.*
https://www.czso.cz/csu/czso/vekove-slozeni-obyvatelstva-2019.

[6] *Úmrtnost vlivem alkoholu v ČR.*
https://www.vlada.cz/cz/ppov/protidrogova-politika/media/1-5-milionu-lidi-ma-v-cr-nakroceno-k-zavislosti-na-alkoholu--pomoci-maji-i-nove-narodni-stranky-alkohol-skodi-cz-167685/.

[7] *Zdravotní problémy způsobené nadměrnou konzumací alkoholu.*
https://www.alkohol-skodi.cz/fakta/zdravi/.

[8] *Přičiny růstu popularity jazyku Python.*
https://github.blog/2023-03-02-why-python-keeps-growing-explained.

[9] *Domény, v nichž je Python skvělou volbou.*
https://www.python.org/about/apps/.

[10] *Vyhody použití Pythonu.*
https://djangostars.com/blog/python-web-development/.

[11] *Vyhody použití Pythonu při vývoji webových stránek.*
https://www.geeksforgeeks.org/why-to-use-python-for-web-development/.

[12] *Popularita jazyku PHP.*
https://kinsta.com/blog/is-php-dead/.

[13] *Proč PHP už není nejlepší volbou.*
https://devm.io/php/php-tiobe-sept-2019-162096.

[14] *Růst popularity jazyku RUST.*
https://thenewstack.io/what-rust-brings-to-frontend-and-web-development/.

[15] *Porovnávání jazyků Python a Rust.*
https://kinsta.com/blog/rust-vs-python/.

[16] *Výhody použití frameworků.*
https://erpsolutions.oodles.io/blog/web-development-frameworks-benefits/.

[17] *Průzkum technologií nabizených jazykem Python.*
https://lp.jetbrains.com/python-developers-survey-2021/.

[18] *Porovnání frameworků Flask a Django.*
https://www.simplilearn.com/flask-vs-django-article.

[19] *Průzkum nejpoužívanějších technologií v softwarovém inženýrství v roce 2022.*
https://survey.stackoverflow.co/2022/#technology.

[20] *Co jsou NoSQL databáze?*
https://www.mongodb.com/nosql-explained.

[21] *Výhody použití NoSQL databází oproti SQL.*
https://www.coursera.org/articles/nosql-vs-sql.

[22] *Co jsou OODBMS?*
https://www.ionos.com/digitalguide/hosting/technical-matters/object-oriented-databases/.

[23] *Co jsou hierarchické databáze?*
https://www.heavy.ai/technical-glossary/hierarchical-database.

[24] *Hierarchické databáze, jejich výhody a nevýhody.*
https://databasetown.com/hierarchical-database/.

[25] *Odlišnosti mezi protokolem WebSocket a HTTP.*
https://www.wallarm.com/what/websocket-vs-http-how-are-these-2-different.

[26] *Rozšiření prokolu WebSocket.*
https://docs.wso2.com/display/ESB500/WebSocket+to+WebSocket+Integration+using+Subprotocols.

[27] *Implementace HTTP Pollingu.*
https://www.abhinavpandey.dev/blog/polling.

[28] *Porovnání protokolů WebSocket, Pollingu a Long-Pollingu.*
https://medium.com/geekculture/ajax-polling-vs-long-polling-vs-websockets-vs-server-sent-events-e0d65033c9ba.

[29] *Co je API?*
https://aws.amazon.com/ru/what-is/api/.

[30] *Co je RPC?*
https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC.

[31] *Porovnání architektur SOAP a REST.*
https://www.atatus.com/blog/soap-vs-rest-whats-the-difference/.

[32] *Tři nejlepší vlastnosti RESTful API.*
https://www.mulesoft.com/resources/api/top-3-benefits-of-rest-apis.

[33] *Proč REST je nejpopularnější?*
https://www.serviceobjects.com/articles-whitepapers/why-rest-popular/.

[34] *Výhody a nevýhody GraphQL.*
https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/.

[35] *Důležitost dokumentování byznys požadavků a cílů.*
https://softwaredominos.com/home/software-design-development-articles/business-requirements-an-essential-guide-to-definition-and-application-in-it-projects/.

[36] *Přednáška "Specefikace požadavků" z předmětu Softwarové inženýrství.*
https://moodle.fel.cvut.cz/pluginfile.php/350314/course/section/61830/SpecifikacePozadavkuprednaska2a.pdf.

[37] *Co je model případů užití?*
https://www.utm.mx/~caff/doc/OpenUPWeb/openup/guidances/concepts/use_case_model_CD178AF9.html.

[38] *Přednáška "Případy užití" z předmězu Softwarové inženýrství.*
https://moodle.fel.cvut.cz/pluginfile.php/350314/course/section/61830/Přednáška3_PripadyUzitiKomplet_SIN_ZS_2020.pdf.

[39] *Jednotlivé části Use-Case modelu.*
https://www.educative.io/answers/what-is-a-use-case-diagram.

[40] *Přednáška "UML diagramy tříd" z předmětu Softwarové inženýrství.*
https://moodle.fel.cvut.cz/pluginfile.php/350314/course/section/61830/Prednaska4_Tridy_a_Stavy_SIN_2021.pdf.

[41] *Jak správně definovat softwarové požadavky.*
https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document.

[42] *Modelování diagramu komponent.*
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/.

[43] *Modelování diagramu nasazení.*
https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-deployment.

Příloha A
Zkratky a symboly

A.1 Zkratky

ADM	Analysis Domain Model - Analitický Doménový Model.
API	Application Programming Interface.
ASGI	Asynchronous Server Gateway Interface
BDM	Business Domain Model - Byznysový Domémový Model.
BE	Backend
BRQ	Business Requirement - Byznys požadavek.
CAD	Computer aided design
CDC	Centrum pro kontrolu a prevenci nemocí.
CMS	Content Management System - Systém pro správu obsahu
CHOPN	Chronická obstrukční plicní nemoc.
CRM	Customer Relationship Management - Řízení vztahů se zákazníky
CRUD	Create, Read, Update, Delete
ČR	Česká republika
CSV	Comma-separated values
FE	Frontend
HTTP	HyperText Transport Protocol
MVC	Model-View-Controller
NLO	Národní linka pro odvykání.
OODBMS	Object-Oriented Database Management System
ORM	Object-Relational Mapping
RDBMS	Relation Database Management System
REST	Representational State Transfer
RPC	Remote Procedure Call
SQL	Structured Query Language
SRQ	Software Requirement - Softwarový požadavek.
SOAP	Simple Object Access Protocol
SZO	Světová zdravotnická organizace.