# Návrh a tvorba progresivní webové aplikace pro boj se syndromem vyhoření

Jiří Šrytr

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Jiří Šrytr** |
| Osobní číslo: | **A19064** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Softwarové inženýrství** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Návrh a tvorba progresivní webové aplikace pro boj se syndromem vyhoření** |
| Téma práce anglicky: | **Design and Creation of a Progressive Web Application for Fighting Against Burnout** |

## Zásady pro vypracování

1. V teoretické části nastudujte a popište webové technologie, které budou využity k praktické implementaci aplikace.
2. Definujte funkční a nefunkční požadavky na webovou aplikaci pro boj se syndromem vyhoření.
3. Navrhněte informační architekturu (Wireframes) webové aplikace a popište datový model a uživatelské scénáře.
4. Popište strukturu vytvořené aplikace, strukturu databáze, důležité části implementace a zejména způsob nasazení standardu PWA (Progressive Web Apps).
5. Věnujte se nasazení aplikace na produkční server a základnímu zabezpečení.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:


1. Getting Started. React –A JavaScript library for building user interfaces [online]. c2021 [cit. 2021-11-21]. Dostupné z: https://reactjs.org/docs/getting-started.html
2. Introduction to Apollo Client – Client (React) – Apollo GraphQL Docs. Apollo GraphQL [online]. [cit. 2021-11-21]. Dostupné z: https://www.apollographql.com/docs/react/
3. Documentation | Node.js. Node.js [online]. c2021 [cit. 2021-11-21]. Dostupné z: https://nodejs.org/en/docs/
4. MongoDB Documentation. MongoDB: the application data platform | MongoDB [online]. c2021 [cit. 2021-11-21]. Dostupné z: https://docs.mongodb.com/
5. CASCIARO, Mario a Luciano MAMMINO. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. Third Edition. Birmingham, United Kingdom: Packt Publishing, 2020. ISBN 9781839214110.
6. MICHÁLEK, Martin. Vzhůru do (responzivního) webdesignu. Verze 1.1. Praha: vlastním nákladem autora, 2017. ISBN 978-80-88253-00-6.
7. SHEPPARD, Dennis. Beginning Progressive Web App Development: Creating a Native App Experience on the Web. Verze 1.1. Berkeley, CA: Apress, 2017. ISBN 978-148-4230-893.
8. FOWLER, Martin. Destilované UML. Praha: Grada, 2009. Knihovna programátora (Grada). ISBN 978-80-247-2062-3.


Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

**prof. Mgr. Roman Jašek, Ph.D., DBA** v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

**Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Jiří Šrytr, v. r.
podpis studenta

## ABSTRAKT

Cílem bakalářské práce je navrhnout a vytvořit aplikaci pro boj se syndromem vyhoření. Hlavním konceptem je propojení lidí procházejících si syndromem vyhoření s dobrovolníky, kteří mají s tímto problémem zkušenost a s profesionálními terapeuty. Aplikace nabídne vstupní test, který uživateli předá informaci o jeho psychickém stavu. Po vykonání testu si uživatel může vytvořit účet, který mu poskytne možnost s terapeuty či dobrovolníky komunikovat. Dále pro uživatele budou dostupné další funkce jako je denní dotazník či odborné materiály ve formě článků, které bude možné filtrovat tak, aby se uživateli zobrazovali pouze články relevantní k výsledku vstupního testu. Teoretická část se bude zabývat popisem progresivních webových aplikací a technologií využitých k implementaci aplikace. Praktická část bude obsahovat souhrn funkčních a nefunkčních požadavků, které by aplikace měla splňovat, návrh informační architektury a navazujících drátěných modelů (wireframes), popis datového modelu, diagram případů užití a s ním spjaté uživatelské scénáře. Text práce se dále bude věnovat struktuře aplikace a databáze a důležitým částem implementace, kde budou popsány jednotlivé části aplikace a jejich provedení na straně klienta a serveru. Nakonec bude popsáno základní zabezpečení aplikace a nasazení na server.



Klíčová slova: progresivní webová aplikace, webové technologie, Next.js, GraphQL, syndrom vyhoření



## ABSTRACT

The aim of the bachelor thesis is to design and create an application to fight burnout. The main concept is to connect people with burnout with volunteers who have experience with this problem and with professional therapists. The application offers an entrance test, which provides the user with information about his mental state. After taking the test, the user can create an account that gives him the opportunity to communicate with therapists or volunteers. In addition, other functions will be available for users, such as a daily questionnaire or professional materials in the form of articles, which will be possible to filter so that only articles relevant to the result of the entrance test are displayed to the user. The theoretical part will deal with the description of progressive web applications and technologies used to implement the application. The practical part will contain a summary of functional and non-functional requirements that the application should meet, design of

information architecture and related wireframes, description of the data model, use case diagram and associated user scenarios. The text of the thesis will also focus on the structure of the application and database and important parts of the implementation, where the individual parts of the application and their implementation on the client and server side will be described. Finally, the basic security of the application and deployment to the server will be described.

Keywords: progressive web application, web technologies, Next.js, GraphQL, burnout syndrome

Tímto bych chtěl poděkovat Ing. Radku Valovi, Ph.D. za poskytování cenných rad a promptních odpovědí na veškeré mé dotazy během vedení mé bakalářské práce. Poděkování také patří celému týmu Nevyhasni, bez kterého by tento projekt vůbec nevznikl.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH

# ÚVOD

V dnešní době se lidé stále častěji setkávají s problém, který je schopný člověka kompletně vykolejit z typického každodenního života. Tímto problémem je syndrom vyhoření.

Podle odhadů odborníků z Psychiatrické kliniky 1. lékařské fakulty Univerzity Karlovy se se syndromem vyhoření potká každý pátý Čech. [58] Přesto, že se tento problém vyskytuje v české populaci tak často, neexistuje mnoho organizací, které by se jím zabývaly.

V rámci nově vznikající neziskové organizace, pojmenované Nevyhasni, vzniká stejnojmenná progresivní webová aplikace pro boj se syndromem vyhoření.

Hlavní myšlenkou a funkcí aplikace je propojení lidí trpících syndromem vyhoření nebo lidí, kteří z něho mají obavy se zkušenými dobrovolníky, kteří si syndromem vyhoření v minulosti již prošli a také s profesionálními terapeuty. Propojení bude realizováno pomocí komunikačního rozhraní, které bude implementováno přímo v aplikaci.

V rámci teoretické části se práce nejprve zaměří na popis problematiky progresivních webových aplikací, jejich vlastností a technologií, které se typicky využívají k jejich implementaci. Také zde budou popsány jednotlivé technologie, které následně budou použity pro reálnou implementaci aplikace Nevyhasni.

Praktická část se bude věnovat samotnému návrhu a vývoji progresivní webové aplikace. Zpočátku popíše obecné zadání projektu a motivaci k jeho vyhotovení. Poté naváže definicí funkčních a nefunkčních požadavků, které by aplikace měla v budoucnu splňovat. Dále bude věnována pozornost návrhu aplikace, počínaje navržením informační architektury webové aplikace a navazujícím drátěným modelem. Také bude popsán datový model aplikace, diagram případů užití a související uživatelské scénáře, které budou demonstrovat tok událostí během provádění jednotlivých případů užití.

V rámci vytvořené aplikace bude popsána adresářová struktura, komunikace klientské a serverové části aplikace a struktura databáze s podrobným vysvětlením. Následně bude práce popisovat důležité a zajímavé části aplikace a způsob nasazení standardu PWA. Nakonec se práce věnuje nasazení aplikace na server a jejímu základnímu zabezpečení.

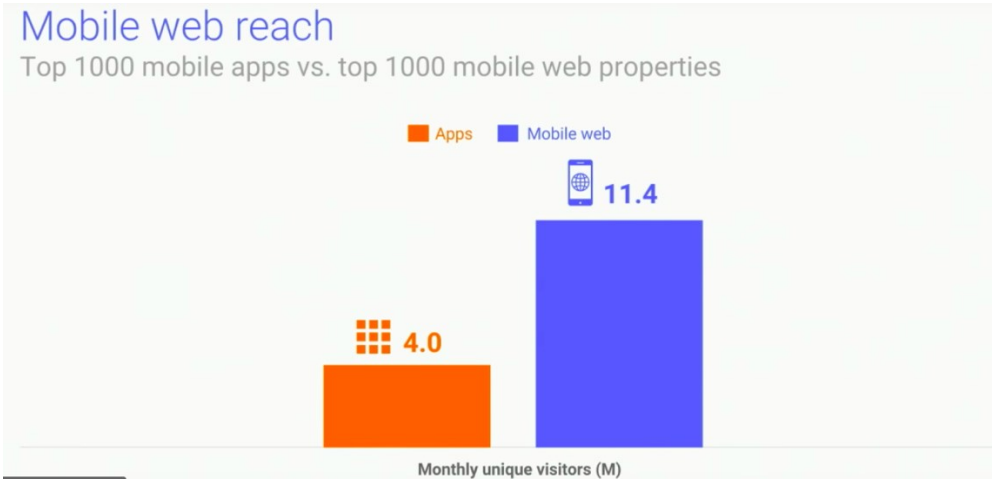
# I. TEORETICKÁ ČÁST

# 1 PROGRESIVNÍ WEBOVÉ APLIKACE

Progresivní webové aplikace, jsou webové aplikace, využívající různé technologie a vzory, díky kterým jsou schopné těžit z výhod webových i nativních aplikací. Kupříkladu mohou běžet bez připojení k internetu, mají přístup k hardwaru zařízení a dokážou uživateli zasílat notifikace. Zároveň se stále jedná o webové prostředí a není tedy nutné aplikace instalovat. [1]

## 1.1 Webové aplikace a jejich výhody

V dnešní době už má téměř každý člověk nějaké chytré zařízení. Dříve obyčejné telefony, které sloužily primárně k hovorům a zasílání SMS zpráv, dnes díky mobilním aplikacím výrazně zasahují do našich životů. Mobilních aplikací existují milióny, téměř cokoliv si dokážeme představit, již existuje na Google Play Store nebo App Store. V tom je ale zásadní problém, mezi takovým množstvím aplikací je mobilní aplikace, oproti těm webovým, složitě objevitelná. Navíc je nutností mobilní aplikaci nainstalovat. Webovou aplikaci nemusí uživatel instalovat, v podstatě se „nainstaluje“ tím, že ji zobrazí. Důsledek těchto výhod je vidět v grafu na obrázku č. 1, který zobrazuje rozdíl v počtu nových návštěvníků za měsíc mezi webovými a mobilními aplikacemi. [2]

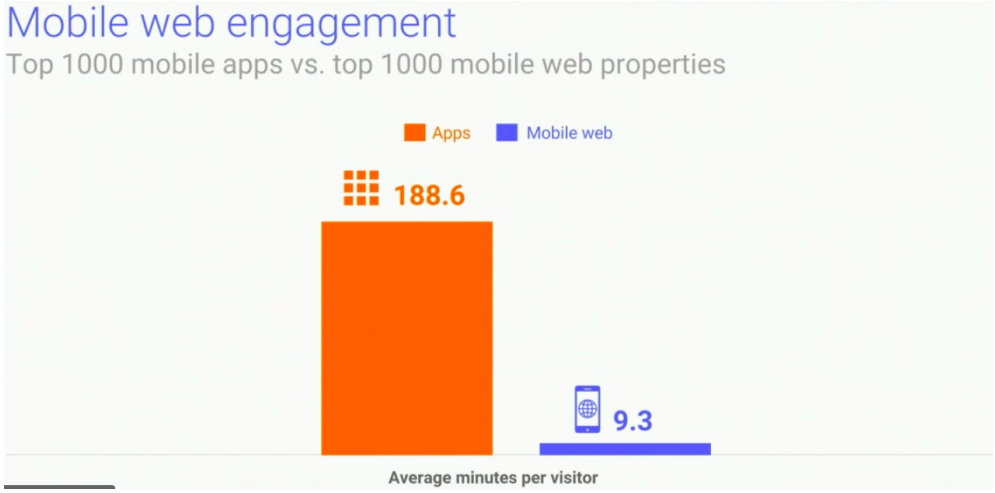



Obrázek 1 Noví návštěvníci za měsíc [2]

Web je všudypřítomná platforma napříč různými zařízeními a operačními systémy. Webové aplikace jsou nezávislé na konkrétním zařízení a uživatelé k nim tedy mohou přistupovat z jakéhokoliv zařízení s připojením k internetu. Je možné je jednoduše vyhledávat a s kýmkoliv sdílet. Další obrovskou výhodou je fakt, že webové aplikace jsou uživateli vždy zobrazovány v aktuální verzi a není nutné instalovat žádné aktualizace. [3]

## 1.2 Nativní aplikace a jejich výhody

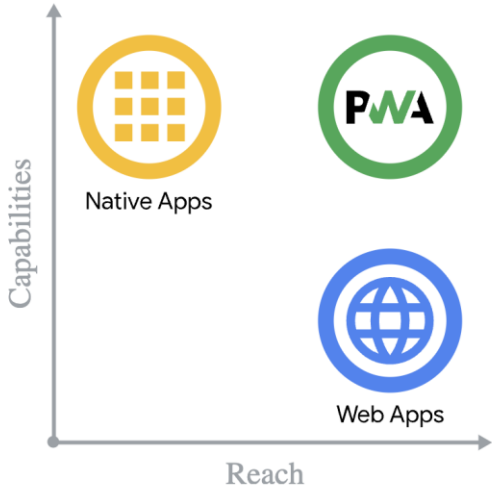
Na druhou stranu, jak ukazuje graf na obrázku č. 2, díky lepší integraci s operačními systémy a s tím spjatou uživatelskou přívětivost, stráví uživatelé v nativních aplikacích mnohem více času než v aplikacích webových. Uživatelé se k nativním aplikacím také mohou vrátit kliknutím na ikonu na ploše svého zařízení. Zabere to méně času než web vyhledávat pomocí vyhledávače nebo zadávat přímo URL adresu. [2] Nativní aplikace do jisté míry fungují bez připojení k internetu, mohou číst a zapisovat soubory z lokálního souborového systému, manipulovat s daty uloženými v zařízení jako jsou kontakty, kalendář a podobně a mají přístup k připojenému hardwaru. S těmito možnostmi nativní aplikace působí, že jsou součástí zařízení, na kterém běží. [3]




Obrázek 2 Průměrná doba strávená v aplikaci na návštěvníka [2]

## 1.3 Spojení výhod webových a nativních aplikací = PWA

Cílem progresivních webových aplikací je kombinovat výhody webových a nativních aplikací, tedy získávat co nejvíce nových uživatelů a zároveň jim poskytnout co nejlepší uživatelskou zkušenost a udržet je tak v aplikaci delší dobu. [4]




Obrázek 3 Kombinace výhod webových a nativních aplikací [3]

## 1.4 HISTORIE

Již před vznikem konceptu progresivních webových aplikací webové technologie implementovaly některé funkce nativních aplikací. Když se webová platforma začala vyvíjet, zjistilo se, že není vhodná pouze pro dokumenty, ale také pro aplikace. [4]

V roce 2000 byla vytvořena technologie *XMLHTTPRequest*, která umožnila získat data z URL adresy bez nutnosti znovunačtení celé stránky. Další technologií, která přispěla světu webu byl *AJAX*. Umožňoval webovým aplikacím odesílat a načítat data ze serveru asynchronně aniž by přerušovaly zobrazování a chování stávající stránky. [5]

Velkým průlomem byl příchod tzv. responsivního web designu. Termín responsivní web design, který popisuje přístup k návrhu webu a soubor osvědčených postupů používaných pro tvorbu layoutu vymyslel roku 2010 Ethan Marcotte. [6]

### 1.4.1 Jak chtěl původně Apple vytvářet aplikace pro iPhony?

Několik let před vznikem PWA, v roce 2007 tehdejší CEO společnosti Apple Inc. Steve Jobs představil nový způsob tvorby aplikací. Aplikace měly být webové, tedy měly běžet v prohlížeči a měly vypadat a chovat se přesně jako nativní aplikace. Nicméně toto se nepodařilo a později téhož roku Apple vydal SDK pro vytváření nativních iOS aplikací a následně roku 2008 App Store. [5]

### 1.4.2 Google představuje PWA

V roce 2015 inženýr Google Chrome Alex Russel a designérka Frances Berriman vymysleli termín progresivní webové aplikace, který popisuje aplikace využívající výhody funkcí moderních prohlížečů. [5]

## 1.5 VLASTNOSTI

Softwarový inženýr Alex Russel a designérka Frances Berriman definovali následující vlastnosti, které by progresivní webové aplikace měly mít. [5]

### 1.5.1 Responsivita

Zobrazení aplikace je optimalizováno pro různé druhy zařízení. Zobrazovaný obsah by tedy měl být přehledný a čitelný na mobilních telefonech, tabletech, noteboocích atd. [10]

### 1.5.2 Nezávislost na konektivitě

Aplikaci je pomocí technologie *Service Workers* umožněno zobrazovat nějaký obsah i bez připojení k internetu. Může se jednat o stránku, která uživateli dává zpětnou vazbu ohledně jeho připojení, v lepším případě by se měla načíst cachovaná data z minulé návštěvy webu, kdy byl uživatel k internetu připojený. [10]

### 1.5.3 App-like interakce

Při používání a navigování aplikací má uživatel pocit, jako by používal nativní aplikaci. [10]

### 1.5.4 „Fresh“

Aplikace je stále aktuální díky procesu aktualizace *Service Workeru*. [10]

### 1.5.5 Zabezpečenost

Aplikace je obsluhována pouze skrze HTTPS, tedy šifrovaný protokol, který zabraňuje odposlouchávání. [10] Pokud aplikace poběží na nezabezpečeném HTTP protokolu nebo nebude mít platný certifikát, nebude fungovat technologie *Service Worker*. [11]

### 1.5.6 Objevitelnost

PWA je identifikovatelná jako aplikace díky W3C manifestu, který vyhledávačům umožňuje aplikaci jednoduše najít. [10]

### 1.5.7 Znovuzapojitelnost uživatele

Umožňují snazší znovuzapojení uživatele k používání aplikace díky funkcím jako jsou *push notifikace*. *Push notifikace* jsou současně bohužel dostupné pouze pro platformu Android, na iOS stále implementovány nebyly. [10]

### 1.5.8 Instalovatelnost

Aplikace poskytuje uživateli možnost uložit si ji na domovskou obrazovku zařízení, aniž by si ji museli do svého zařízení stahovat z App Store či Google Play Store. [10]

### 1.5.9 Odkazovatelnost

Aplikaci je možné jednoduše sdílet pomocí URL bez nutnosti instalace jako klasickou webovou stránku či aplikaci. [10]

## 1.6 Technologie použité v PWA

Právě progresivní technologie umožňují doplnit webové aplikace o funkcionalitu nativních aplikací. Jádrem všech progresivních webových aplikací jsou tzv. *Service Workery*.

### 1.6.1 Service Worker

Technologie Service Worker umožňuje spouštět kód v pozadí, i když daná stránka nemá možnost komunikovat se serverem. Může se jednat o situace, kdy uživatel nemá připojení k internetu či ani nemá danou stránku otevřenou v okně svého prohlížeče. [11]

Service Worker se návštěvníkovi stránky nainstaluje do jeho prohlížeče. Prohlížeč si následně zjistí, na jaké události má Service Worker reagovat. Po vykonání určité události Service Worker spustí této události připsanou část kódu, která provede danou operaci. [11]

Jednou z takových událostí je událost *fetch*, která spouští kód pokaždé, kdy stránka požádá server o nějaká data. Může se jednat o HTML stránku, soubor s kaskádovými styly, obrázek apod. Toto se dá využít právě pro zobrazování aplikace bez připojení k internetu. Service Worker si může po události *fetch* uložit dané soubory do cache paměti. Při další návštěvě aplikace v offline režimu pak Service Worker načte soubory právě z cache paměti a uživatel může do jisté míry s aplikací interagovat. [11]

Dalšími velmi užitečnými událostmi jsou *sync* a *online*. Díky těmto událostem Service Worker pozná, že se uživatel připojil k internetu, a tedy přešel do online režimu.

Následně může načíst aktualizace ze serveru či odeslat data, která se v aplikaci nashromáždila, když byl uživatel v offline režimu. [11]

Díky Service Workerům je možné webové aplikace také obohatit o možnost zasílání push notifikací. Service Worker může návštěvníka požádat o povolení pro zasílání push notifikací. V případě povolení pak dokáže server odesílat zprávy do uživatelova zařízení i když zrovna nemá danou aplikaci otevřenou. [11]

### 1.6.2 Manifest

Důležitou součástí progresivních webových aplikací je webový aplikační manifest zapisovaný ve formátu JSON. V tomto JSON souboru vývojáři dané aplikace definují její metadata. Právě díky manifestu je umožněno aplikaci uložit na domovskou obrazovku. [12]

```
{
  "short_name": "Název aplikace",
  "name": "Název aplikace: jak se dnes máte?",
  "icons": [
    {
      "src": "img/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "img/icon–512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/index.html",
  "background_color": "#EEEEEE",
  "display": "standalone",
  "theme_color": "#999999"
}
```

Kód 1 Příklad implementace manifest.json souboru (vlastní kód)

## 1.7 ÚSPĚŠNÉ PŘECHODY SPOLEČNOSTÍ NA PWA

Efektivitu progresivních webových aplikací dokazuje spousta společností světových rozměrů, které právě PWA zvolili jako svoje řešení, které se velmi vyplatilo. Jejich aplikace nabyly vyšší míry prokliku, získaly vyšší počet konverzí, prodloužily dobu strávenou uživatelem v aplikaci a narostl i například počet vyhledávání. [7]

### 1.7.1 Twitter

Roku 2017 společnost Twitter, Inc. vydala na trh lehčí variantu jejich aplikace, vhodnou pro regiony s nestabilním a pomalým internetovým připojením. Pro vytvoření se rozhodla použít právě progresivní webové technologie. [8] Nově vzniklá aplikace Twitter Lite velikostně odpovídala 3% velikosti nativní aplikace pro Android. Podařilo se zvýšit počet navštívených stránek na relaci o 65%, počet odeslaných Tweetů o 75% a množství návštěvníků, kteří si po zapnutí zobrazí pouze jednu stránku se snížil o 20%. Díky implementovanému vyzvání uživatele k uložení Twitter Lite na plochu zařízení, Twitter zaznamenal 250 000 unikátních uživatelů denně, kteří se do aplikace dostali právě ze své domovské obrazovky v průměru 4krát denně. [9]

### 1.7.2 Forbes

Americký magazín Forbes, se rozhodl, že v době, kdy spousta uživatelů čte zprávy na svých mobilních zařízeních, je potřeba přijít s co nejefektivnějším řešením. Po implementaci progresivních webových technologií se čas načítání snížil na 1 vteřinu z původních 10 vteřin, návštěvy na uživatele vzrostly o 43%, hloubka scrollování se ztrojnásobila a délka průměrné návštěvy se zvýšila o 100%.

### 1.7.3 Tinder

Společnost Match Group Holdings II, LLC v roce 2017 vydala odlehčenou variantu jejich aplikace Tinder založenou na progresivních webových technologiích. Progresivní webová aplikace Tinderu snížila velikost na pouhých 2.8 MB z původních 30 MB velikosti nativní aplikace. Tinder navíc zaznamenal zvýšenou aktivitu uživatelů. Ukázalo se, že uživatelé PWA varianty jsou aktivnější a tráví v aplikaci více času než uživatelé nativní aplikace. [7]

# 2 NERELAČNÍ DATABÁZE

Nerelační databáze, také často nazývány jako NoSQL databáze, jsou databáze, které nepoužívají tabulkové schéma se sloupci a řádky jako většina konvenčních databázových systémů. [13] Data tedy nejsou ukládána v tabulkách s předem pevně definovanou strukturou, díky čemuž lze kombinovat různé struktury a datový model velice jednoduše škálovat a to i během vývoje bez nutnosti úprav schématu databáze. Jelikož nerelační databáze mohou pracovat i s klasickými SQL dotazy, název NoSQL bývá vysvětlován spíše jako Not Only SQL. [14]

## 2.1 Typy nerelačních databází

NoSQL je pojem zastřešující mnoho systémů, které používají datový model, který se strukturou liší od klasických tabulek (relací) používaných v SQL. Každý typ NoSQL databáze však funguje na odlišném principu. [15]

### 2.1.1 Dokumentové databáze

Dokumentové databáze jsou jedním z nejoblíbenějších typů nerelačních databází mezi vývojáři. Jak lze poznat z názvu, data jsou zde ukládána v dokumentech. Obvykle se jedná o dokumenty typu JSON, BSON nebo XML. Dokumenty jsou tedy ukládány ve formátu, který je velmi podobný datovým objektům, se kterými se pracuje na úrovni aplikace. Dokumentové databáze tedy oproti těm, založeným na jazyku SQL, které často vyžadují úpravy při pohybu dat mezi aplikací a samotnou databází, přináší v tomto ohledu výhodu. [15]

Záznam se v dokumentové databázi označuje jako dokument. Dokument popisují atributy, kterým se říká pole. Hodnoty polí v dokumentu mohou obsahovat různé datové typy jako jsou čísla, řetězce, datumy, ale i datové struktury jako jsou pole a objekty. [16]

```
{
   "_id": "10",
   "jmeno": "Jan",
   "prijmeni": "Novák",
   "email": "karel@novak.cz",
   "telefon": "+420777888999",
   "kancelar": "R21/333",
   "ustav": "Ústav informatiky a umělé inteligence"
}
```

Kód 2 Příklad dokumentu e formátu JSON (vlastní kód)

Skupina záznamů, tedy dokumentů, se označuje jako kolekce. V kolekcích se obvykle shlukují dokumenty s podobným obsahem, díky flexibilitě dokumentových databází a NoSQL databází obecně to však není pravidlem. Kolekce tedy může shromažďovat i dokumenty s odlišnými poli. [16]

### 2.1.2 Databáze typu klíč-hodnota

Úložiště typu klíč-hodnota je nejjednodušší typ nerelačních databází. Jednotlivé položky jsou v databázi ukládány v páru jako klíč a jeho hodnota, kde klíč představuje nějaký atribut (např. email). Tyto databáze jsou vhodné pro velké datasety s velmi jednoduchou strukturou. Může se např. jednat o košík na internetových obchodech či o uživatelský účet. [15]

### 2.1.3 Sloupcové databáze

Zatímco typické databáze většinou ukládají data po řádcích, sloupcové databáze je ukládají po sloupcích. Díky tomu je v případě potřeby přistoupení pouze k určitým sloupcům možné číst přímo jejich data bez nutnosti procházení ostatních právě nepotřebných položek z daného řádku. Toto se dá velmi dobře uplatnit v případech, kdy je potřeba agregovat velké množství dat v jednom sloupci. Proto se sloupcové databáze využívají pro analytické úlohy, které oproti konvenčním databázím, dokážou spočítat velmi rychle. [15]

### 2.1.4 Grafové databáze

Grafové databáze jsou databáze založené na jedné z oblastí diskrétní matematiky, a to na teorii grafů. Struktura grafové databáze a grafů obecně se skládá z množiny vrcholů a hran. Vrcholy pak reprezentují entity, kterými mohou být například lidé, firmy nebo jakékoliv jiné objekty. Hrany, také nazývány jako vztahy, zde představují čáry, které jednotlivé vrcholy spojují, a právě tím mezi nimi definují vztahy. Dále grafové databáze obsahují vlastnosti, které jsou asociované s uzly. [17]

Grafové databáze dokážou efektivně vyhledávat spojení mezi jednotlivými datovými prvky (vrcholy). V tomto ohledu mají oproti konvenčním relačním databázím, kde je nutné spojovat více tabulek, výhodu. [15]

V praxi jsou většinou grafové databáze používány spolu s jinými tradičnějšími databázemi. Typickým příkladem použití grafových databází mohou být například sociální sítě. [15]

# 3 ROZBOR TECHNOLOGIÍ VYUŽITÝCH K IMPLEMENTACI APLIKACE

V této části budou popsány veškeré technologie, které budou následně využity pro implementaci výsledné aplikace. Jedná se především programovací jazyk JavaScript a jeho nadstavbu TypeScript vyvinutou a spravovanou společností Microsoft. V rámci JavaScriptu budou také popsány použité JavaScriptové knihovny a běhové prostředí Node.js, které umožňuje spouštět JavaScriptový kód mimo webový prohlížeč a slouží primárně pro tvorbu serverové části webových aplikací. Dále bude popsán dotazovací jazyk pro tvorbu moderních API GraphQL, technologie Apollo Client a Server, databáze MongoDB a v neposlední řadě kaskádové styly CSS, jejich preprocesory a efektivní způsob jejich zápisu.

## 3.1 Javascript a odvozené technologie

JavaScript je skriptovací, interpretovaný a objektově orientovaný jazyk, který primárně slouží jako skriptovací jazyk pro webové stránky a aplikace, kde je interpretovaný právě v prohlížeči uživatele. [18] V dnešní době se ale stále častěji používá i mimo prohlížečové prostředí. Pomocí různých frameworků a knihoven je v něm možné vyvíjet nativně vypadající mobilní a desktopové aplikace. Obrovskou výhodou je pak možnost spustit stejnou aplikaci na různých platformách bez větších úprav kódu.

JavaScript je dynamicky typovaný jazyk, což v praxi například znamená, že do proměnné, která prvně obsahovala celočíselný datový typ může být později přiřazen řetězec.

```
let promenna = 17
console.log(promenna) // Výstupem bude číslo 17

promenna = "Ahoj světe!"
console.log(promenna) // Výstupem bude řetězec "Ahoj světe!"
```

Kód 3 Příklad dynamického typování jazyka JavaScript (vlastní kód)

Oproti většině jiných objektově orientovaných jazyků, JavaScript není založen na typickém konceptu třída-instance, ale disponuje možnostmi, které tuto funkcionalitu částečně zastupují. [19] Jedná se o prototypování, kde objekty dědí metody a vlastnosti od jiných objektů namísto od tříd. V JavaScriptu může v podstatě každá funkce vystupovat jako třída jejíž instance se poté vytvoří klíčovým slovem *new*. [20] Pro přehlednější a jednodušší

syntaxi byly v rámci ECMA2015 (ES6) přidány nové možnosti zápisu OOP velmi podobné těm z klasických objektově orientovaných jazyků. ECMA2015 například obsahuje klíčové slovo *class*, stále se ale jedná o typickou funkci.

```
class Osoba {
    constructor(jmeno, prijmeni, vek) {
        this.jmeno = jmeno
        this.prijmeni = prijmeni
        this.vek = vek
    }

    show () {
        console.log(`
            Osoba: ${ this.jmeno } ${ this.prijmeni },
            věk: ${ this.vek }
        `)
    }
}

class Student extends Osoba {
    constructor(jmeno, prijmeni, vek, skola, rocnik) {
        super(jmeno, prijmeni, vek)

        this.skola = skola
        this.rocnik = rocnik
    }

    show () {
        console.log(`
            Student: ${ this.jmeno } ${ this.prijmeni },
            věk: ${ this.vek },
            škola: ${ this.skola },
            ročník: ${ this.rocnik }.
        `)
    }
}

let osoba = new Osoba("Karel", "Novák", 21)
let student = new Student("Jaroslav", "Novotný", 22, "UTB", 3)

osoba.show()
student.show()
```

Kód 4 Ukázka základního OOP se syntaxí ECMA2015 (vlastní kód)

JavaScriptový kód je možné zapisovat procedurálně, objektově orientovaně, ale díky faktu, že jeho funkce jsou tzv. *first-class*, je možné uplatit i funkcionální programování.

S first-class funkcemi je možné manipulovat jako s každou jinou proměnnou. To například umožňuje předávat funkci jako argument jiné funkce nebo přiřazovat funkci jako hodnotu proměnné. [21]

```
// Funkce uložená v proměnné
const funkce = () => {
    console.log("Ahoj!")
}

funkce()

// Funkce jako parametr
const osloveni = () => {
    return "Vážený pane/paní "
}

const osloveniOsoby = (vypisVety, osoba) => {
    console.log(vypisVety() + osoba)
}

osloveniOsoby(osloveni, "Nováková")
```

Kód 5 Ukázka mechanismu first-class funkcí v JavaScriptu (vlastní kód)

V ukázkách JavaScriptového kódu výše lze vidět, že nikde není použit středník. Středník ale v tomto jazyce stejně jako v jiných programovacích jazycích, jako je například C/C++, C#, PHP atd., odděluje nebo ukončuje jednotlivé příkazy. JavaScriptový interpret středníky při spuštění doplní sám (*Automatic Semicolon Insertion*), tudíž je možnost zapisovat středníky čistě na preferencích daného programátora.

### 3.1.1 TypeScript

TypeScript je open-source programovací jazyk vyvinutý a spravovaný firmou Microsoft. Jedná se o nadstavbu jazyka JavaScript, což v praxi také znamená, že funkční JavaScriptový kód bude fungovat i v TypeScriptu. TypeScript není možné spouštět přímo v prohlížeči či v jiných běhových prostředích, ale používá *source-to-source* kompilátor, nazývaný také jako *transpiler*. [22] Transpiler zdrojový kód napsaný v TypeScriptu přeloží do JavaScriptového zdrojového kódu, kterému již interpret rozumí.

Oproti klasickému JavaScriptu je v TypeScriptu možné zapisovat staticky typované proměnné, což se odráží i v názvu tohoto jazyka. To znamená, že programátor může stanovit

jaké hodnoty smí daná proměnná obsahovat. TypeScript poté nedovolí zkompilovat kód ve kterém by například byl do staticky typované proměnné celočíselného datového typu přiřazen textový řetězec. TypeScriptový transpiler by tento problém zachytil již během kompilace (*compile-time*), místo toho, aby bylo nutné kód spouštět a chybu detekovat až při běhu (*run-time*). Tuto chybu vývojář uvidí již ve svém editoru, za předpokladu, že touto funkcionalitou tento editor disponuje.

```
let cislo: number = 5
let retezec: string = "Ahoj"
```

Kód 6 Statické typování v TypeScriptu (vlastní kód)


Obrázek 4 Chybová hláška v editoru při přiřazení špatné hodnoty do proměnné

Kromě možnosti staticky typovat proměnné TypeScript umožňuje i vytvářet rozhraní (*interface*), které se typicky využívají i v jiných objektově orientovaných jazycích jako je například C#. Rozhraní umožňuje programátorovi vytvářet struktury, které po přiřazení dané proměnné či třídě musí být dodrženy. V případě, že není dodržena daná struktura objektu, je opět zobrazena chybová hláška v editoru. Toto je v praxi velice užitečné při tvorbě velkých aplikací, kde je nutné dodržet pevnou strukturu projektu a udržet kód jasný a čitelný pro další vývojáře z týmu.

```
// Rozhraní bod
interface Bod {
    x: number
    y: number
}
```

```
// Proměnná, která je staticky typovaná jako Bod
let bod: Bod = { x: 5, y: 6 }

// Funkce, která přebírá jako parametr pouze Bod
const ukazBod = (bod: Bod): void => {
    console.log(`Bod: [${ bod.x }, ${ bod.y }]`)
}

ukazBod(bod)
```

Kód 7 Ukázka rozhraní v TypeScriptu (vlastní kód)

### 3.1.2 Node.js

Node.js je open source multiplatformní běhové prostředí, které je primárně určené pro vývoj serverové části webových aplikací. [25]

V minulosti již existovala technologie, která umožňovala spouštět JavaScriptový kód na straně serveru. Jednalo se o prostředí od firmy NetScape zvané LiveWire, které dokázalo vytvářet dynamické webové stránky právě pomocí JavaScriptu. V tuto dobu ale JavaScript nebyl tak pokročilým jazykem jako je dnes a LiveWire se tedy mezi vývojáři neuchytil. [26]

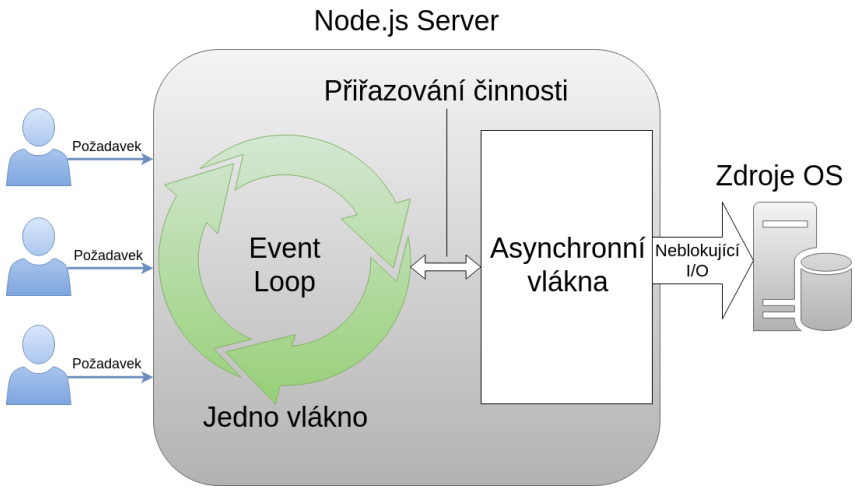
Později s příchodem Webu 2.0, kdy na internetu začali lidé ve velkém množství vytvářet, sdílet a konzumovat obsah, vznikaly čím dál tím pokročilejší webové aplikace, a s tím spjaté velké nároky na JavaScript. JavaScript se postupně začal vyvíjet do podoby, která je vhodná pro použití i na velkých projektech. [26]

S většími nároky na JavaScript a na něm stavěné aplikace se tvůrci populárních prohlížečů snažili vylepšovat JavaScript interprety tak, aby podporovaly všechny nové funkce JavaScriptu a uživateli běžel v prohlížeči rychle. Společnost Google v tuto dobu vyvinula interpret Chrome V8, který byl vydán jako open source v rámci projektu The Chromium. [26]

Novější a pro vývoj přívětivější verze JavaScriptu a propracovaný interpret Chrome V8 přinesl nové možnosti, a tak vznikl i Node.js, který vývojářům umožnil programovat celé projekty (klientskou i serverovou část) v jednom jazyce, tedy v JavaScriptu.

Stejně jako u prohlížeče Google Chrome je i u Node.js jádrem interpret Chrome V8, který je zde ale rozšířený o vrstvu C++ kódu, která umožňuje vykonávaným skriptům přistupovat k souborům nebo síťovým funkcím. [27] Node.js je řešený jako single-thread, tedy pracuje pouze v jednom vlákně. Node.js řeší jednotlivé požadavky asynchronně, což

znamená, že nový požadavek nečeká na dokončení jiného. Pilířem architektury Node.js je smyčka událostí (event loop), která neustále přijímá uživatelské požadavky, ve formě událostí je přiděluje jednotlivým zdrojům a v mezičase asynchronně zpracovává další. [28]




Obrázek 5 Architektura Node.js serveru

Technologie Node.js je díky svým vlastnostem vhodná pro tvorbu aplikací, které očekávají velké množství požadavků tedy například real-time aplikace, které implementují komunikační rozhraní, tedy například pro sociální sítě. Node.js naopak není vhodný pro aplikace, které provádějí velké množství složitých výpočtů a jsou tedy náročné na CPU. [28]

### 3.1.3 Express.js

Express.js je open source Node.js Framework, který slouží primárně k tvorbě API či jako middleware. Oproti vývoji v čistém Node.js výrazně vývojářům ulehčuje práci, eliminuje nutnost psaní konfiguračního kódu a umožňuje tak vývojáři věnovat většinu své pozornosti na programování business logiky aplikace. [54] Na technologii Express.js staví mnoho dalších populárních Node.js frameworků jako je NestJS, Sails či Feathers. [55]

```
const express = require('express')
const cookieParser = require('cookie-parser')

const port = 4000      // Port na kterém bude aplikace naslouchat
const app = express()  // Inicializace Express aplikace

// Obsluha GET požadavku na route /
app.get('/', (req, res) => {
   // Zde může být obslužný kód, který například bude získávat data z databáze a vracet je v odpovědi
klientovi
   res.send('Vítejte v aplikaci.')
})
```

```
// Použití middlewaru třetí strany pomocí Express.js use metody
app.use(cookieParser())

// Spuštění serveru
app.listen(port, () => console.log('Aplikace byla spuštěna na portu ' + port))
```

Kód 8 Jednoduchý příklad použití framworku Express.js (vlastní kód)

V kódu uvedeném výše lze vidět jakým způsobem se za pomocí frameworku Express.js řeší HTTP požadavky, aplikuje middleware a spouští server na daném portu.

### 3.1.4 React

React je jedna z nejpopulárnějších JavaScriptových knihoven pro tvorbu uživatelských rozhraní. Jedná se open source projekt vyvíjený firmou Meta (dříve Facebook) a komunitou jednotlivých vývojářů.

**Komponenty** v Reactu se dají tvořit dvěma způsoby, a to pomocí tříd anebo aktuálněji pomocí funkcí. Funkční komponenty jsou vlastně jednoduché funkce, které přijímají vlastnosti (props) jako vstup a jejich výstupem je tzv. *JSX (více níže)*. Oproti komponentám zapsaných pomocí tříd mají obvykle funkční komponenty kratší a přehlednější kód.

```
const MojeKomponenta = () => {
   return (
      <div className="komponenta">
         <h1>Ahoj světe!</h1>
         <p>Jak se dnes daří?</p>
      </div>
   )
}

export default MojeKomponenta
```

Kód 9 React komponenta vytvořená pomocí funkce (vlastní kód)

**JSX** neboli JavaScript Syntax Extension je rozšíření syntaxe JavaScriptu, které se zápisem podobá značkovacímu jazyku HTML a slouží pro tvorbu struktury komponent. V kódu uvedeném výše funkce MojeKomponenta vrací hodnotu, která na první pohled

vypadá jako klasické HTML, avšak jedná se právě o JSX. Kód zapsaný pomocí JSX se překládá do klasického JavaScriptu (viz následující ukázka kódu).

```
React.createElement(
   "div",
   {
      className: "komponenta"
   },
   React.createElement("h1", null, "Ahoj světe!"),
   React.createElement("p", null, "Jak se dnes daří?")
);
```

Kód 10 JavaScriptová podoba JSX kódu (vlastní kód)

**Stavy** neboli *states* v Reactu slouží pro ukládání dat, která se budou v při běhu aplikace měnit. V komponentách zapsaných pomocí funkcí se pro uchovávání a přepisování stavů používá useState hook.

```
const Pocitadlo = () => {
   const [pocitadlo, setPocitadlo] = useState(0)

   return (
      <div>
         <button onClick={ () => setPocitadlo(pocitadlo + 1) }>
            Inkrementuj
         </button>
         <p>hodnota: { pocitadlo }</p>
      </div>
   )
}
```

Kód 11 React komponenta se stavem zapsaném pomocí useState hooku (vlastní kód)

**Props** neboli vlastnosti slouží k předávání hodnoty z komponenty do komponenty. V praxi je velice vhodné použití například u komponenty v podobě hlavičky stránky, která je vizuálně v každé sekci webu stejná, ale má jiný obsah. V různých sekcích webu se poté tato komponenta importuje a skrze props se do ní vkládá různý obsah.

```
const Pocitadlo = ({ inkrementacniHodnota }) => {
   const [pocitadlo, setPocitadlo] = useState(0)
```

```
  return (
    <div>
      <button onClick={ () => setPocitadlo(pocitadlo + inkrementacniHodnota) }>
        Inkrementuj
      </button>
      <p>hodnota: { pocitadlo }</p>
    </div>
  )
}
```

Kód 12 React kompontenta s využitím props (vlastní kód)

```
<Pocitadlo inkrementacniHodnota={ 10 } />
```

Kód 13 Importovaná komponenta s hodnotou předanou do props (vlastní kód)

### 3.1.5 Next.js

Next.js je open source webový framework postavený na technologiích React a Node.js. Oproti klasickým React aplikacím, které umožňují vykreslit obsah stránky či aplikace pouze na straně klienta, Next.js poskytuje možnost obsah předkreslit na straně serveru a poslat jej již připravený do prohlížeče uživatele. Vykreslováním neboli renderováním se rozumí proces, kdy se převádí kód napsaný v Reactu do HTML struktury uživatelského rozhraní aplikace. Ve standardní React aplikaci prohlížeč od serveru obdrží pouze prázdné HTML připravené na vykreslování jednotlivých komponent a JavaScriptové instrukce podle kterých se tyto komponenty budou vykreslovat. Oproti tomu Next.js defaultně předkreslí každou stránku předem na serveru místo toho, aby se vše řešilo v prohlížeči uživatele. [31]

Právě díky tomu, že jsou jednotlivé stránky předdefinovány ještě před zobrazením na straně klienta je může jakýkoli vyhledávač procházet a tedy indexovat. Next.js je proto vhodným nástrojem pro tvorbu webových aplikací se znamenitým SEO výkonem. [32]

Next.js nabízí tři možnosti renderování, a to Server-Side Rendering, dále jen SSR, Client-Side Rendering, dále jen CSR, a Static Site Generation, dále jen SSG. SSR a SSG bývá také označováno jako již zmiňované předkreslování, protože převod React kódu do HTML je proveden ještě před tím, než je výsledek zaslán klientovi. [31]

SSG stejně jako SSR generuje HTML na straně serveru. Rozdíl oproti SSR je v tom, že SSG renderuje obsah na serveru pouze jednou, a to při sestavení aplikace. HTML je poté

uložené v CDN a znovu použito pro každý požadavek. [31] CDN ukládá statické soubory (např. HTML, obrázky apod.) na více místech po celém světě. Při požadavku je poté obsah ze CDN doručen uživateli z geograficky nejbližšího místa. [33]

Typickým příkladem použití SSR je fukce *getServerSideProps*, která při každém požadavku předkreslí stránku s daty, které tato funkce vrací v podobě objektu zvaného props. [42]

```
function Stranka({ data }) {
  // Next.js stránka ve které se budou získaná data vykreslovat
}

// Tato funkce bude zavolána s každým požadavkem
export async function getServerSideProps() {
  // Získání dat z externího API
  const res = await fetch(`https://.../data`)
  const data = await res.json()

  // Přenesení dat do stránky skrze props (vlastnosti)
  return { props: { data } }
}

export default Stranka
```

Kód 14 Next.js příklad funkce getServerSideProps [42]

### 3.1.6 Plugin next-pwa

Next-pwa je javascriptový plugin usnadňující tvorbu progresivních webových aplikací pomocí technologie Next.js. Plugin staví primárně na knihovně Workbox od společnosti Google, která obsahuje sadu osvědčených postupů pro práci se service workery. [56]

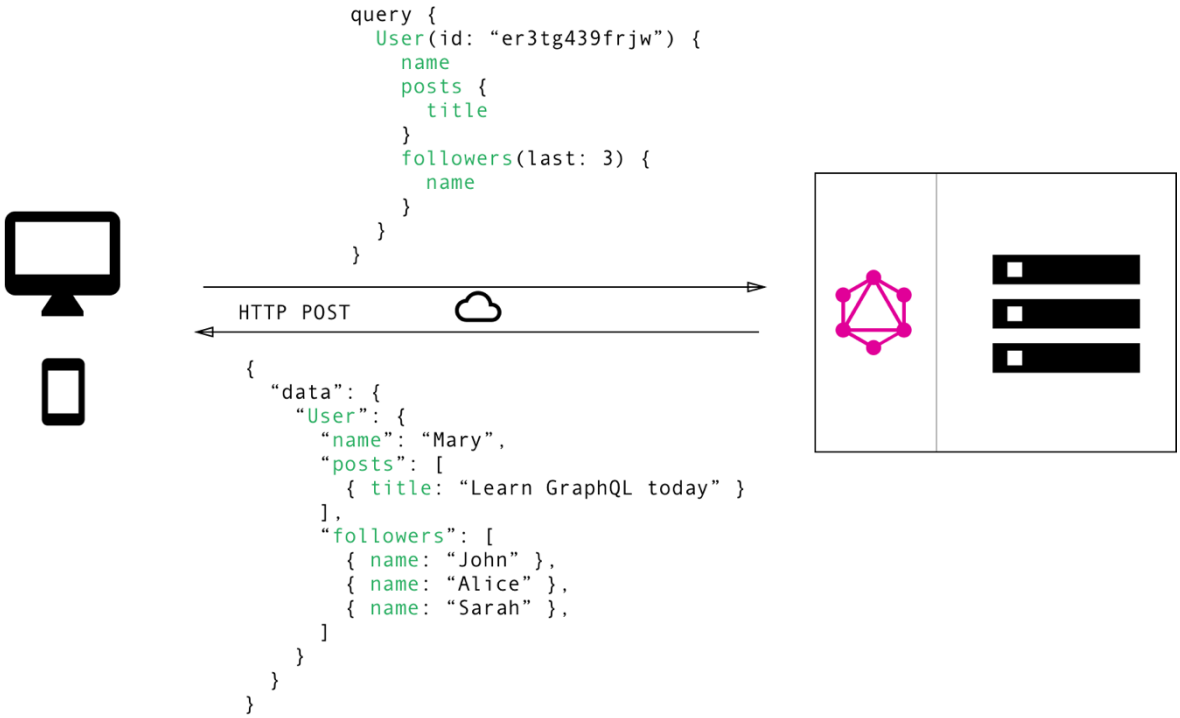
## 3.2 GraphQL

GraphQL je technologie vyvinutá společností Meta (dříve Facebook) v roce 2012 v souvislosti s vývojem nové nativní aplikace Facebook. V roce 2015 byla tato technologie otevřena veřejnosti v podobě open source. GraphQL popisuje způsob jakým komunikuje strana serveru se stranou klienta. Ze strany klienta je GraphQL dotazovacím jazykem, který umožňuje definovat strukturu dotazu. Ze strany serveru je GraphQL běhové prostředí, které tyto dotazy řeší a vrací požadovaná data.

### 3.2.1 Rozdíly oproti technologii REST

Oproti technologii REST, GraphQL využívá pouze jednoho endpointu a lze tedy získat data z vícero zdrojů jedním dotazem. Získávaná data lze tímto dotazem strukturovat, kdežto v případě RESTu jsou obdržena veškerá data, která daný endpoint vrací. Tento rozdíl je demonstrovaný na konkrétním scénáři ve dvou následujících obrázcích, kde na obrázku č. 6 je k řešení využita technologie REST a na obrázku č. 7 řeší problematiku technologie GraphQL.


Obrázek 6 Získávání dat pomocí technologie REST [34]




Obrázek 7 Získávání dat pomocí dotazovacího jazyka GraphQL [34]

### 3.2.2 Schéma a typy

GraphQL server používá schéma k popisu struktury dostupných dat. Základním komponentem tohoto schématu jsou typy, které představují objekty s určitými poli označovanými jako fields. U všech polí objektů je nutné specifikovat návratovou hodnotu. Schéma také přesně definuje, které dotazy a mutace je na straně klienta možné spouštět. [36]

```
type Uzivatel {
   id: ID!              # Nemůže být null
   zivotopis: String    # Může být null
   pratele: [Uzivatel]  # Pole uživatelů
}
```

Kód 15 Příklad GraphQL typu (vlastní kód)

### 3.2.3 Queries

GraphQL poskytuje speciální typ Query, který slouží jako vstupní bod dotazů, tedy definuje, jaké dotazy mohou být na straně klienta spouštěny. Každé pole typu Query definuje název, návratovou hodnotu a případně parametry daného dotazu. [36]

```
type Query {
   uzivatel(id: ID!): Uzivatel     # Vrací uživatele dle zadeného ID nebo null
   seznamUzivatelu: [Uzivatel]     # Vrací seznam uživatelů nebp null
}
```

Kód 16 Příklad GraphQL typu Query (vlastní kód)

GraphQL dotaz ze strany klienta by pak vypadal například následovně.

```
query SeznamUzivatelu {
   seznamUzivatelu {
      id
      zivotopis
   }
}
```

Kód 17 Příklad GraphQL dotazu ze strany klienta (vlastní kód)

### 3.2.4 Mutations

GraphQL dále nabízí další speciální typ, a to Mutation, který je svou strukturou a účelem podobný typu Query. Rozdíl je v tom, že typ Query definuje operace pro čtení, kdežto typ Mutation definuje operace pro zápis.

```
type Mutation {
   vytvorUzivatele(id: ID!, zivotopis: String): Uzivatel
}
```

Kód 18 Příklad GraphQL typu Mutation (vlastní kód)

Příklad klientem odesílané GraphQL mutace by pak mohl vypadat takto:

```
mutation VytvorUzivatele($id: ID!, $zivotopis: String) {
   vytvorUzivatele (id: $id, zivotopis: $zivotopis) {
      id
      zivotopis
   }
}
```

Kód 19 Příklad GraphQL mutace ze strany klienta (vlastní kód)

### 3.2.5 Resolvers

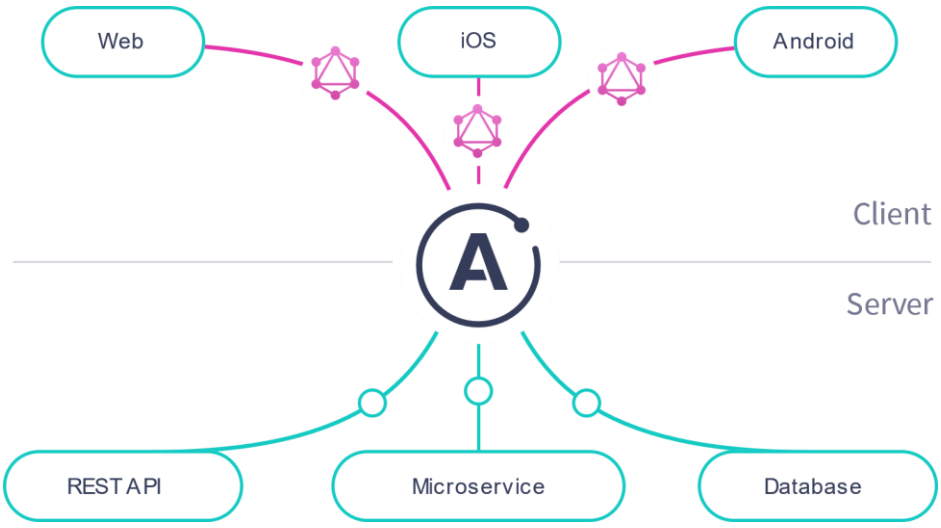
V předchozím textu byly uvedeny příklady jednotlivých dotazů a mutací, GraphQL server ale ještě potřebuje vědět jak a odkud se budou data brát, proto je nutné nadefinovat kolekci tzv. resolverů. Kdykoliv se klientská část dotáže na určité pole, resolver pro něj získá požadovaná data z příslušného zdroje (obvykle se jedná o databázi). Resolver je tedy funkce, jež je zodpovědná za naplnění dat dotazovaných polí. [37]

```
const seznamUzivatelu = async () => {
    try {
        const uzivatele = await Uzivatel.find()
        return uzivatele
    } catch(err) {
        throw new Error(err)
    }
}
```

Kód 20 Příklad GraphQL resolveru pro získání seznamu uživatelů (vlastní kód)

## 3.3 Apollo

Apollo je GraphQL platforma pro tvorbu komunikační vrstvy, která pomáhá řídit tok dat mezi klientskými částmi aplikací a serverovými službami. [38]




Obrázek 8 Vizualizace technologie Apollo [38]

### 3.3.1 Apollo Client

Apollo Client je knihovna pomocí které lze provádět GraphQL operace, ukládat jejich výsledky do mezipaměti a spravovat stav neboli state aplikace. Oficiální dokumentace Apollo Client nabízí tři varianty této knihovny, a to pro JavaScript, pro Swift a také pro jazyk Kotlin, je ale dostupná spousta dalších komunitou vyvíjených variant. Apollo Client umožňuje implementovat GraphQL dotazy přímo v komponentách uživatelského rozhraní a poté, v momentě přijetí výsledků od serveru, tyto komponenty automaticky aktualizovat. [38]

### 3.3.2 Apollo Server

Ke zpracování klientem zaslaných GraphQL operací je nutné implementovat serverovou službu. K implementaci takové služby je možné využít technologie Apollo Server. [38] Jedná se o open source GraphQL server, který je kompatibilní s jakýmkoliv GraphQL klientem, včetně výše zmiňovaného Apollo Client. Pomocí Apollo serveru je možné definovat GraphQL schéma, které specifikuje všechny dostupné typy a pole a kolekci resolverů, které určují jak a z jakých zdrojů dat naplnit každé pole definovaného schématu. [38] Pro vytvoření a následné spuštění serveru Apollo je nezbytné udělat instanci třídy *ApolloServer*, která jako parametr přijímá objekt, jehož dvěma základními vlastnostmi jsou právě nadefinované typy, tedy schéma, a kolekce resolverů. Server je poté možné spustit pomocí metody *listen*.

```
const typeDefs = gql`
    type Uzivatel {
        id: ID!
        zivotopis: String
        pratele: [Uzivatel]
    }
    type Query {
        seznamUzivatelu: [Uzivatel]
    }
`
const resolvers = {
    Query: {
        seznamUzivatelu: () => uzivatele
    }
}
const server = new ApolloServer({ typeDefs: typeDefs, resolvers: resolvers })
server.listen({ port: 5000 })
```

Kód 21 Příklad jednoduché implementace serveru Apollo (vlastní kód)

## 3.4 Databáze MongoDB

MongoDB je open source dokumentová databáze vyvíjená stejnojmennou společností MongoDB, Inc. Namísto ukládání dat do tabulek, jako tomu je u SQL databází, je každý záznam uložen v MongoDB jako dokument typu BSON, tedy v binární podobě formátu JSON. Oproti klasickému JSON formátu může BSON nést více různých datových typů. Aplikace poté mohou data z MongoDB získávat ve standardním formátu JSON, který je velice podobný datovým objektům používaným na úrovni aplikace. [49] Dokumenty jsou organizovány do takzvaných kolekcí. Na rozdíl od relací, předdefinované schéma pro kolekce není povinné, díky čemuž je možné upravovat datové struktury použité v databázi bez nutnosti provádění komplexních databázových migrací. Také je umožněno ukládat spolu související data na jednom místě, což poskytuje výhodu v rychlosti čtení těchto dat, protože není nutné spojovat entity jako tomu je u tabulek relačních databází. [50]

Společnost MongoDB, Inc. také nabízí Database as a Service službu, *MongoDB Atlas,* která umožňuje nastavit, nasadit a škálovat databázi bez nutnosti správy fyzického hardwaru, aktualizace softwaru a zjišťování podrobností o konfiguraci výkonu.

MongoDB navíc nabízí efektivní možnosti škálování. Když se obsah databáze rozroste natolik, že již nestačí dosavadní možnosti serveru, je nutné ji nějakým způsobem škálovat. Škálovat je možné dvěma základními způsoby, vertikálně anebo horizontálně. Vertikálním škálováním se rozumí navyšování výkonu a paměti jednoho stroje, na kterém databáze běží. Toto má ale značné nevýhody, výkon a paměť serveru nelze navyšovat do nekonečna a cena za přidaný hardware prudce narůstá. Horizontální škálování je přístup, kdy jsou místo navyšování výkonu na jednom stroji přidány další stroje. [51] Rozdělení dat mezi vícero strojů může být u konvenčních relačních databází velmi obtížné, kvůli vztahům mezi daty. U nerelačních databází je toto jednodušší, protože kolekce jsou samostatné a nejsou relačně propojeny. [52] V MongoDB je horizontálního škálování dosaženo pomocí tzv. shardingu a sad replik (*replica sets*). Sharding rozkládá data mezi více uzlů, tedy každý uzel obsahuje podmnožinu celkových dat. Replica sets fungují podobně jako sharding s tím rozdílem, že datová sada je duplikovaná. Replica sets tedy slouží pro vytváření redundantních dat za účelem zefektivnění a zabezpečení jejich dostupnosti. To je zvláště užitečné v případě selhání databázového serveru. [53]

### 3.4.1 Knihovna Mongoose

Mongoose je knihovna pro objektově dokumentové mapování MongoDB a je distribuovaná jako npm balíček. Objektově dokumentové mapování (ODM) je alternativa objektově relačního mapování (ORM) používaného při práci s klasickými relačními databázemi. [45] Cílem Mongoose a ODM či ORM obecně je synchronizace mezi používanými objekty na úrovni aplikace a jejich reprezentací v daném databázovém systému. [46]

## 3.5 Kaskádové styly

Kaskádové styly neboli Cascading Style Sheets (CSS) umožňují definovat vzhled HTML stránek. HTML dokument si lze představit jako jakousi kostru webu či aplikace, CSS pak zastává roli prezentační vrstvy, tedy vizuálně obohacuje kostru webu. CSS lze využít k základním úpravám, jako je například změna velikosti či barvy textu, barvy pozadí atp. Moderní CSS ale umožňuje pokročilé úpravy včetně definování layoutu webu či vytvoření animovaných prvků.

Pomocí CSS lze nadefinovat blok pravidel, které budou uplatněny pro určitou část webu nebo aplikace vybranou pomocí CSS selektoru. Selektorem může být například samotný HTML element, identifikátor či třída. V kódu 21 lze vidět, kde selektorem je HTML element *h1* a třída *box*. Pravidla zapsaná v bloku selektoru *h1* budou v tomto případě uplatněna pro všechny elementy *h1* a pravidla zapsaná v bloku selektoru *.box* budou uplatněna pro všechny elementy se třídou *box*.

```
h1 {
  font-size: 4rem;
  font-weight: bold;
  color: white;
}

.box {
  background-color: gray;
  border: 2px solid black;
}
```

Kód 22 Příklad kódu CSS (vlastní kód)

Pro připojení CSS k HTML dokumentu je možné přistupovat vícero způsoby. CSS může být definováno přímo v atributu *style* daného elementu, kde tedy není nutné definovat

selektor, ale zapíšou se přímo vlastnosti pro tento element. Tento přístup však přináší nevýhody v podobě slučování obsahu dokumentu s jeho vzhledem a velký nárůst objemu kódu. Dalším způsobem jsou interní styly, kde se CSS zapisuje přímo do hlavičky HTML dokumentu v elementu *style*. V tomto případě dochází k opětovanému načítání CSS při každé návštěvě dané stránky. Posledním a nejvhodnějším způsobem jsou externí styly, kde se CSS zapisuje do separátního .css souboru, který se následně připojí k HTML dokumentu pomocí značky *link* vložené do hlavičky. Externí .css soubor je pak možné využívat ve vícero HTML dokumentech a zamezit tak psaní redundantního CSS kódu. Navíc externí soubor je možné ukládat do paměti cache, což má za následek potenciální zrychlení načítání webu či aplikace.

### 3.5.1 CSS preprocesory

CSS preprocesory jsou nástroje postavené nad CSS, které umožňují efektivnější psaní CSS kódu. Například obohacují základní CSS o další užitečné funkce jako jsou podmínky, cykly, zanořování a dědičnost selektorů atd. Prohlížeče neumí kód preprocesorů zpracovávat a je tedy nutné na webový server nainstalovat kompilátor, který přeloží kód daného preprocesoru do čistého CSS kódu anebo kompilovat kód preprocesoru v rámci vývojového prostředí při vývoji a na server nahrávat již zkompilovaný CSS kód. Mezi známe preprocesory patří například SASS, LESS či Stylus. [44]

### 3.5.2 Metodiky psaní CSS kódu

Při velkém množství CSS kódu často vývojáři narazí na problém s jeho správou, kód je často redundantní a nepřehledný. Pro psaní kaskádových stylů v průběhu let vznikla spousta různých metodik, které tyto se tyto problémy snaží řešit. [47]

Mezi tyto metodiky se řadí například OOCSS, tedy objektově orientované CSS. Základní myšlenkou a cílem OOCSS je zapouzdření stylů do samostatných objektů. Při psaní CSS kódu s touto metodikou by měl být nezávislý vzhled na struktuře, tedy do CSS selektorů by neměly být vkládány HTML elementy. Například místo selektoru *button* by bylo vhodnější použít třídu *.btn*. Dále by se do CSS neměla promítat struktura HTML dokumentu. To znamená, že například namísto selektoru *.header .btn* je efektivnější použít třídu *.btn-main.* OOCSS je základem většiny dnes používaných metodik pro psaní CSS kódu. [47]

Další metodikou je BEM (Block, Element, Modifier), jejíž název již napovídá, jakým způsobem je zde zapisován CSS kód. BEM se skládá ze tří typů selektorů, a to z bloku,

elementu a modifikátoru. Blok zastává roli nezávislé a znovupoužitelné komponenty uživatelského rozhraní. Blok se v CSS zapisuje ve formátu *.nazev-bloku*. Blok obsahuje další prvky, jimiž jsou právě již zmiňované elementy. Element nelze v uživatelském rozhraní použít samostatně a je tedy závislý právě na daném bloku. Zápis elementu začíná názvem bloku doplněným názvem elementu, který je oddělený dvěma podtržítky, tedy *.nazev-bloku__nazev-elementu*. Pro tvorbu variant komponent a elementů slouží modifikátor. Příkladem využití modifikátoru může být aktivní položka v navigaci. Modifikátor se připisuje k bloku či elementu připsáním názvu modifikátoru odděleným dvěma pomlčkami, tedy *.nazev-bloku--nazev-modifikatoru* v případě modifikace bloku a *.nazev-bloku__nazev-elementu--nazev-modifikatoru* v případě modifikace elementu. [48]

```
.nav {
  /* Pravidla společná pro všechny navigace */
}

.nav--secondary {
  /* Měnící se pravidla pro tuto modifikaci */
}

.nav__item {
  /* Pravidla pro tento element */
}

.nav__item--active {
  /* Měnící se pravidla pro tuto modifikaci elementu */
}
```

Kód 23 Ukázka zápisu BEM [48]

## 3.6 JSON Web Token (JWT)

JSON Web Token, zkráceně JWT, je standard (RFC 7519), který představuje bezpečný způsob přenosu dat mezi více stranami. Přenášeným informacím lze důvěřovat, protože jsou ověřeny digitálním podpisem. JWT je JSON objekt, který se skládá ze tří částí oddělenými tečkou, a to z hlavičky, dat (payload) a podpisu. Hlavička nese informaci o typu tokenu a používaném podepisovacím algoritmu jako je HMAC, SHA256 apod. Druhá část tokenu, tedy payload, drží přenášená data. Poslední částí tokenu je tedy digitální podpis vytvořený pomocí zvolené hashovací funkce, kde prvním parametrem je řetězec, který obsahuje zřetězenou zakódovanou hlavičku a paylaod pomocí base64 a druhým parametrem je

hashovací klíč. [39] Příklad výpočtu digitálního podpisu je pro přehlednost uvedený níže v kódu č. 19.

```
HMACSHA256(
   base64UrlEncode(header) + "." +
   base64UrlEncode(payload),
   "klíč"
)
```

Kód 24 Příklad výpočtu digitálního podpisu [39]

Typickým příkladem použití JWT je autorizace. Jakmile se uživatel přihlásí, na serveru mu je vytvořen token, který je poslán zpět a uložen v klientské části aplikace. Při další komunikaci uživatele se serverem je tento token poté využit pro ověření identity uživatele a jeho přístupových práv k daným operacím (může se jednat např. vytvoření příspěvku, úpravu uživatelského profilu apod.).

## 3.7 Správce balíčků NPM

Node Package Manager, zkráceně NPM, je správce JavaScriptových balíčků a je instalový společně s technologií Node.js. NPM se skládá ze dvou částí, a to z rozhraní příkazového řádku a z robustní online databáze balíčků. [40]

Instalace jednotlivých balíčků se provádí pomocí jednoduchého příkazu *npm install <název balíčku>*, nainstalovaný balíček je poté dostupný v adresáři *node_modules*. V kořenovém adresáři jsou pak dva JSON soubory, a to *package.json*, který obsahuje základní informace o aplikaci a seznam potřebných balíčků a *package-lock.json*, který nese informace o verzích nainstalovaných balíčků. [41]

# II. PRAKTICKÁ ČÁST

# 4 SBĚR POŽADAVKŮ

Praktická část této práce se zabývá návrhem a vývojem softwarové aplikace. Jednou z prvních fází vývoje softwarových produktů je právě sběr požadavků. V této fázi je kladen důraz na komunikaci se zadavatelem projektu či jinými zainteresovanými osobami za účelem získání jejich požadavků na vyvíjený produkt. Požadavky jsou specifikací toho, co by mělo být v aplikaci implementováno. Požadavky dále také popisují, jak by se měl systém chovat. [23]

V případě aplikace Nevyhasni, kde je zadavatelem samotný tým stojící za tímto projektem, se v rámci několika konzultací realizoval sběr požadavků. Některé požadavky, především na komponentu se vstupním psychologickým testem, byly definovány po konzultaci s odborníky na danou problematiku.

Důležitost této vývojové fáze dokládá mnoha studií, které ukázaly značnou nákladovost chyb způsobených při definování požadavků. [24] Obecně platí, že čím dříve se na chybný požadavek narazí, tím je jeho úprava levnější. Pokud se chybný požadavek dostane do pozdějších fází vývoje, může způsobit velké problémy, které můžou vést až k neúspěšným nebo opuštěným projektům.

## 4.1 OBECNÉ ZADÁNÍ PROJEKTU

Ještě před sběrem detailních požadavků na funkcionalitu softwarového produktu je potřeba mít nějaké obecné zadání, které poslouží jako základní stavební kámen celého projektu a od kterého se dá v následujících fázích vývoje odrazit.

### 4.1.1 Motivace

Syndrom vyhoření se v naší populaci vyskytuje stále častěji. Aplikace Nevyhasni poskytne uživateli efektivní způsoby, jak syndromu vyhoření předejít nebo mu s ním pomůže bojovat.

### 4.1.2 Popis projektu

Aplikace Nevyhasni bude poskytovat následující základní prvky:

1) uživatelské účty
    a. role uživatele (vyhořelý)
    b. role dobrovolníka či terapeuta
2) vstupní test

3) denní dotazník pro uživatele
4) komunikace mezi uživatelem a dobrovolníkem/terapeutem
5) odborné materiály

Ad 1) Aplikace bude obsahovat uživatelské účty. Uživatelské účty budou rozděleny na dvě role, a to na uživatele, který trpí syndromem vyhoření nebo na něj má podezření a na terapeuta či dobrovolníka.

Ad 2) Aplikace bude obsahovat vstupní test. Provedením vstupního testu bude novému uživateli poskytnuta informace o jeho zdravotním stavu. Dle jeho výsledku mu bude doporučen vhodný postup.

Ad 3) Aplikace bude obsahovat denní dotazník ve formě deníku. Aplikace bude uživatele vyzívat k jeho vyplnění pomocí push notifikací, ale uživatel se k dotazníku kdykoliv bude moct vrátit přímo v rozhraní aplikace. Data získaná z dotazníku budou sloužit jako přehled vývoje zdravotního stavu jak pro uživatele, tak i pro jeho terapeuta. Na konci každého týdne bude danému terapeutovi zaslán e-mail obsahující přehled o psychickém stavu daného uživatele a jeho vývoji.

Ad 4) Aplikace bude obsahovat komunikační rozhraní, které bude uživateli umožňovat komunikovat s jeho terapeutem či zkušenými dobrovolníky, kteří mají se syndromem vyhoření zkušenost. Odeslané zprávy se budou ukládat do databáze a budou tedy daným uživatelům stále dostupné.

Ad 5) Aplikace bude poskytovat uživatelům různé materiály, především odborné články. Vlastní články budou moct vytvářet psychologičtí odborníci, ale i zkušení dobrovolníci. Jimi přidané články však budou podléhat schvalovacímu procesu ze strany pověřených osob z týmu Nevyhasni. Materiály také půjde přiřadit do dané kategorie, pomocí které je budou uživatelé schopni filtrovat.

## 4.2 FUNKČNÍ POŽADAVKY

Funkční požadavky popisují funkcionalitu budoucí aplikace. Zachycují, co musí vyvíjený produkt umět a jak se má v daných případech chovat, tedy jak bude chování mezi vstupy a výstupy.

### 4.2.1 Základní funkční požadavky

V následující tabulce budou vypsány základní funkční požadavky. Od zde popsaných požadavků se odvíjejí další požadavky na specifické části aplikace, které budou specifikovány v rámci této kapitoly (4.2) níže.

Tabulka 1 Základní funkční požadavky

| ID požadavku | Požadavek |
|---|---|
| FP1 | Aplikace bude poskytovat vstupní test |
| FP2 | Aplikace bude disponovat uživatelskými účty |
| FP3 | Aplikace bude poskytovat odborné materiály |
| FP4 | Aplikace bude nabízet denní dotazník |
| FP5 | Aplikace bude disponovat komunikačním rozhraním |
| FP6 | Aplikace bude disponovat uživatelskou nástěnkou |
| FP7 | Aplikace bude disponovat upozorněním uživatelů o aktivitě |

Níže budou více popsány funkční základní požadavky z tabulky č. 1.

Ad FP1) Aplikace novému uživateli nabídne možnost vyplnit vstupní test. Po vykonání vstupního testu budou uživateli zobrazeny výsledky a informace o vhodném postupu včetně nabídky registrace do aplikace.

Ad FP2) Aplikace bude disponovat uživatelskými účty třech různých rolí. Aplikace umožní uživateli registraci, přihlášení, odhlášení a správu účtu.

Ad FP3) Aplikace bude poskytovat odborné materiály ve formě článků, a to i nepřihlášeným uživatelům. Díky přístupu k materiálům bez nutnosti registrace se zvýší dosah aplikace v rámci SEO.

Ad FP4) Aplikace bude nabízet denní dotazník ve formě deníku. Uživatel jej bude moct vyplnit a zobrazit historii zaznamenaných odpovědí. Odpovědi dále budou odesílány zvolenému terapeutovi.

Ad FP5) Aplikace bude disponovat komunikačním rozhraním mezi základním uživatelem, tedy uživatelem, který v aplikace hledá pomoc, a terapeutem či dobrovolníkem.

Ad FP6) Aplikace bude disponovat uživatelskou nástěnkou, která přihlášenému uživateli zobrazí relevantní obsah jako jsou rozhovory z komunikačního rozhraní, nové materiály atp.

Ad FP7) Aplikace bude disponovat upozorněním uživatelů o aktivitě ve formě notifikační lišty.

### 4.2.2 Funkční požadavky na vstupní test

Tabulka 2 Funkční požadavky na vstupní test

| ID požadavku | Požadavek |
|---|---|
| FP1a | Otázky vstupního testu budou rozděleny do kategorií |
| FP1b | Otázky vstupního testu budou zobrazovány v jednotlivých krocích |
| FP1c | Aplikace bude umožňovat vrátit se k předchozímu kroku |
| FP1d | Aplikace bude po dokončení testu poskytovat výsledky |
| FP1e | Aplikace bude po dokončení testu poskytovat doporučení |
| FP1f | Aplikace bude po dokončení testu nabízet registraci |
| FP1g | Aplikace bude po dokončení testu nabízet přímý kontakt terapeuta |
| FP1h | V případě registrace bude aplikace ukládat výsledek testu |

Ad FP1a) Otázky vstupního testu budou rozděleny do jednotlivých kategorií tak, aby byly otázky stejného typu pohromadě a nedocházelo ke kombinaci spolu nesouvisejících otázek.

Ad FP1b) Otázky vstupního testu budou zobrazovány v jednotlivých krocích, kde každý krok bude vždy obsahovat otázky dané kategorie.

Ad FP1c) Aplikace bude uživateli při vykonávání testu umožňovat vrátit se k předchozímu kroku a upravit tak odpovědi na dané otázky.

Ad FP1d) Aplikace bude po dokončení testu poskytovat uživateli jeho výsledky ve formě bodů a popisu psychického stavu.

Ad FP1e) Aplikace bude po dokončení testu poskytovat uživateli doporučení v návaznosti na výsledek testu.

Ad FP1f) Aplikace bude po dokončení testu nabízet možnost vytvoření uživatelského účtu. Uživateli budou zároveň zobrazeny informace o aplikaci a jejích výhodách.

Ad FP1g) Aplikace bude po dokončení testu zobrazovat dostupné terapeuty a nabídne možnost přímého kontaktování.

Ad FP1h) V případě, že se uživatel do aplikace zaregistruje, výsledky testu budou uloženy k jeho uživatelskému účtu.

### 4.2.3 Funkční požadavky na uživatelské účty

Tabulka 3 Funkční požadavky na uživatelské účty

| ID požadavku | Požadavek |
|---|---|
| FP2a | Aplikace bude obsahovat tři uživatelské role |
| FP2b | Aplikace bude poskytovat přihlášení uživatele |
| FP2c | Aplikace bude poskytovat registraci uživatele |
| FP2d | Aplikace bude poskytovat odhlášení uživatele |
| FP2e | Aplikace bude uživateli umožňovat úpravu účtu |
| FP2f | Aplikace bude disponovat uživatelskými profily terapeutů a dobrovolníků |
| FP2g | Aplikace bude umožňovat terapeutům a dobrovolníkům přidávat popis a fotografii k jejich uživatelským profilům |
| FP2h | Profil terapeuta bude obsahovat výpis jím vytvořených materiálů. |

Ad FP2a) Aplikace bude obsahovat tři uživatelské role, a to uživatele, terapeuta a dobrovolníka.

Ad FP2b) Aplikace bude poskytovat přihlášení uživatele skrze přihlašovací formulář, kam uživatel zadá svoje údaje a následně bude autorizován a přihlášen do aplikace.

Ad FP2c) Aplikace bude poskytovat registraci uživatele skrze registrační formulář na který bude uživatel odkázán po vykonání vstupního testu. Do registračního formuláře uživatel zadá požadované údaje a bude mu vytvořen uživatelský účet.

Ad FP2d) Aplikace bude poskytovat možnost odhlášení uživatele.

Ad FP2e) Aplikace bude uživateli umožňovat upravovat účet. Uživateli tedy bude umožněna úprava jeho údajů.

Ad FP2f) Aplikace bude disponovat uživatelskými profily terapeutů a dobrovolníků. Uživatelé, kteří v aplikaci vyhledávají pomoc, si následně jejich profily budou moci zobrazovat.

Ad FP2g) Aplikace bude uživatelům role terapeut a dobrovolník umožňovat přidávat popis k jejich uživatelským profilům, který ostatním uživatelům poskytne užitečné informace. Bude také možné přidat profilovou fotografii.

Ad FP2h) Uživatelský profil terapeuta bude disponovat výpisem materiálů jejichž je daný terapeut autorem.

### 4.2.4 Funkční požadavky na odborné materiály

Tabulka 4 Funkční požadavky na odborné materiály

| ID požadavku | Požadavek |
|---|---|
| FP3a | Aplikace bude umožňovat zobrazovat materiály, a to i nepřihlášenému uživateli |
| FP3b | Materiály budou obsahovat náhledový obrázek a titulek |
| FP3c | Materiály budou nést informaci o bodovém rozsahu |
| FP3d | Materiály budou rozřazeny do jednotlivých kategorií |
| FP3e | Aplikace bude umožňovat vyhledávat materiály pomocí zadání klíčových slov |
| FP3f | Aplikace bude umožňovat filtrovat materiály dle kategorie |
| FP3g | Aplikace bude umožňovat vyfiltrovat relevantní materiály dle bodového rozsahu. |
| FP3h | Aplikace bude poskytovat možnost přidání nových materiálů |
| FP3ch | Aplikace bude poskytovat možnost úpravy přidaných materiálů |
| FP3i | Aplikace bude umožňovat přidané materiály odstranit |

Ad FP3a) Aplikace bude umožňovat zobrazovat materiály ve formě článků jako jednotlivé podstránky i nepřihlášeným uživatelům.

Ad FP3b) Všechny materiály budou mít vlastní náhledový obrázek a titulek, který se bude zobrazovat ve všech výpisech materiálů.

Ad FP3c) Veškeré materiály budou také obsahovat informaci o bodovém rozsahu, tedy informaci o tom, pro jaký typ uživatelů je daný materiál vhodný.

Ad FP3d) Jednotlivé materiály budou rozřazeny do kategorií. Kategorie u článků bude obsahovat odkaz na stránku s výpisem článků dané kategorie.

Ad FP3e) Aplikace bude uživateli umožňovat vyhledávat materiály podle klíčových slov zadaných do pole pro vyhledávání.

Ad FP3f) Aplikace bude uživateli umožňovat filtrovat materiály dle vybrané kategorie.

Ad FP3g) Aplikace bude uživateli role vyhořelý poskytovat možnost vyfiltrovat materiály, které bodovým rozsahem odpovídají jeho výsledku testu.

Ad FP3h) Aplikace bude poskytovat uživatelům role terapeut možnost vytvoření nových materiálů.

Ad FP3ch) Aplikace bude uživatelům role terapeut umožňovat upravovat vlastní materiály.

Ad FP3i) Aplikace bude uživatelům role terapeut umožňovat odstranit vlastní materiály.

### 4.2.5 Funkční požadavky na denní dotazník

Tabulka 5 Funkční požadavky na denní dotazník

| **ID požadavku** | **Požadavek** |
|---|---|
| FP4a | Aplikace bude uživateli umožňovat vyplnit dotazník |
| FP4b | Aplikace bude ukládat historii odpovědí a umožňovat uživateli jej zobrazit |
| FP4c | Aplikace bude odpovědi týdně zasílat vybranému terapeutovi |

Ad FP4a) Aplikace bude základnímu uživateli umožňovat vyplnit denní dotazník ve formě deníku. Deník bude vždy obsahovat krátkou otázku s popisem a vstupní pole pro odpovědi.

Ad FP4b) Aplikace bude ukládat historii odpovědí uživatele, které si daný uživatel poté bude moci kdykoliv zobrazit.

Ad FP4c) Aplikace bude odpovědi uživatele týdně zasílat zvolenému terapeutovi a předá mu tak informaci o vývoji zdravotního stavu uživatele.

### 4.2.6 Funkční požadavky na komunikační rozhraní

Tabulka 6 Funkční požadavky na komunikační rozhraní

| **ID požadavku** | **Požadavek** |
|---|---|
| FP5a | Aplikace bude umožňovat soukromou komunikaci mezi základními uživateli a dobrovolníky či terapeuty |
| FP5b | Aplikace bude umožňovat základnímu uživateli požádat terapeuta či dobrovolníka o zahájení konverzace |
| FP5c | Aplikace bude posouvat komunikační okno při každé nové zprávě |

Ad FP5a) Aplikace bude umožňovat soukromou komunikaci mezi základním uživatelem, který hledá v aplikaci pomoc, a osloveným terapeutem či dobrovolníkem.

Ad FP5b) Aplikace bude umožňovat základnímu uživateli požádat terapeuta či dobrovolníka o zahájení konverzace pomocí tlačítka na stránce s komunikačním rozhraním či na profilu daného terapeuta nebo dobrovolníka.

Ad FP5c) Aplikace po vytvoření nové zprávy, tedy jejím odeslání či přijetí, bude sama komunikační okno posouvat k nejaktuálnější zprávě směrem dolů, tak aby uživatel nemusel okno posouvat manuálně.

### 4.2.7 Funkční požadavky na uživatelskou nástěnku

Tabulka 7 Funkční požadavky na uživatelskou nástěnku

| ID požadavku | Požadavek |
|---|---|
| FP6a | Uživatelská nástěnka bude obsahovat výpis posledních rozhovorů a umožňovat se skrze ně přesměrovat do dané konverzace |
| FP6b | Uživatelská nástěnka bude obsahovat výpis nových materiálů |
| FP6c | Uživatelská nástěnka bude obsahovat okno s denním dotazníkem a jeho historií |

Ad FP6a) Uživatelská nástěnka bude obsahovat výpis posledních konverzací přihlášeného uživatele s terapeuty či dobrovolníky a umožní se skrze ně přesměrovat do dané konverzace.

Ad FP6b) Uživatelská nástěnka bude obsahovat výpis nových matriálů přes který bude umožněno uživateli na tyto matriály přistoupit.

Ad FP6c) Uživatelská nástěnka bude obsahovat okno s denním dotazníkem, který bude každých 24 hodin obsahovat novou otázku. Uživateli zde bude také umožněno dotazník vyplnit a zobrazit historii odpovědí daného uživatele.

### 4.2.8 Funkční požadavky na upozornění

Tabulka 8 Funkční požadavky na upozornění

| ID požadavku | Požadavek |
|---|---|
| FP7a | Aplikace bude uživateli zasílat upozornění o příchozí zprávě z komunikačního rozhraní |

| | |
|---|---|
| FP7b | Aplikace bude uživateli zasílat upozornění o aktualizovaném denním dotazníku |
| FP7c | Aplikace bude uživateli umožňovat zobrazit veškeré notifikace v notifikační liště |

Ad FP7a) Aplikace bude uživatelům všech rolí zasílat upozornění o nové příchozí zprávě z komunikačního rozhraní.

Ad FP7b) Aplikace bude zasílat základnímu uživateli zasílat upozornění o aktualizovaném denním dotazníku jako připomínku k jeho vyplnění.

Ad FP7c) Aplikace bude umožňovat uživatelům jakékoliv role zobrazit v notifikační liště všechna upozornění, která uživatele po kliknutí odkážou na danou stránku (např. okno komunikačního rozhraní atp.).

## 4.3 NEFUNKČNÍ POŽADAVKY

Nefunkční požadavky jsou doplňkem požadavků funkčních, který nesouvisí s funkčním hlediskem vyvíjeného softwaru. Neříkají, co by měl systém umět a jaké funkce by měl poskytovat, ale definují očekávané vlastnosti systému. Může se jednat například o požadavky na zabezpečenost či na dobu odezvy systému.

Tabulka 9 Nefunkční požadavky

| **ID požadavku** | **Požadavek** |
|---|---|
| NFP1 | Klientská část aplikace bude multiplatformní a zobrazována skrze webový prohlížeč |
| NFP2 | Klientská část aplikace bude podporovat responsivní design |
| NFP3 | Aplikace bude do jisté míry umožňovat interakci bez připojení k internetu |
| NFP4 | Klientská část aplikace bude schopna provozu ve všech moderních prohlížečích |
| NFP5 | Serverová část aplikace bude implementována pomocí technologie Node.js |
| NFP6 | Ke komunikaci klientské části se serverovou bude využito technologie GraphQL |
| NFP7 | Aplikace bude pracovat s nerelační dokumentovou databází MongoDB |

# 5 NÁVRH APLIKACE

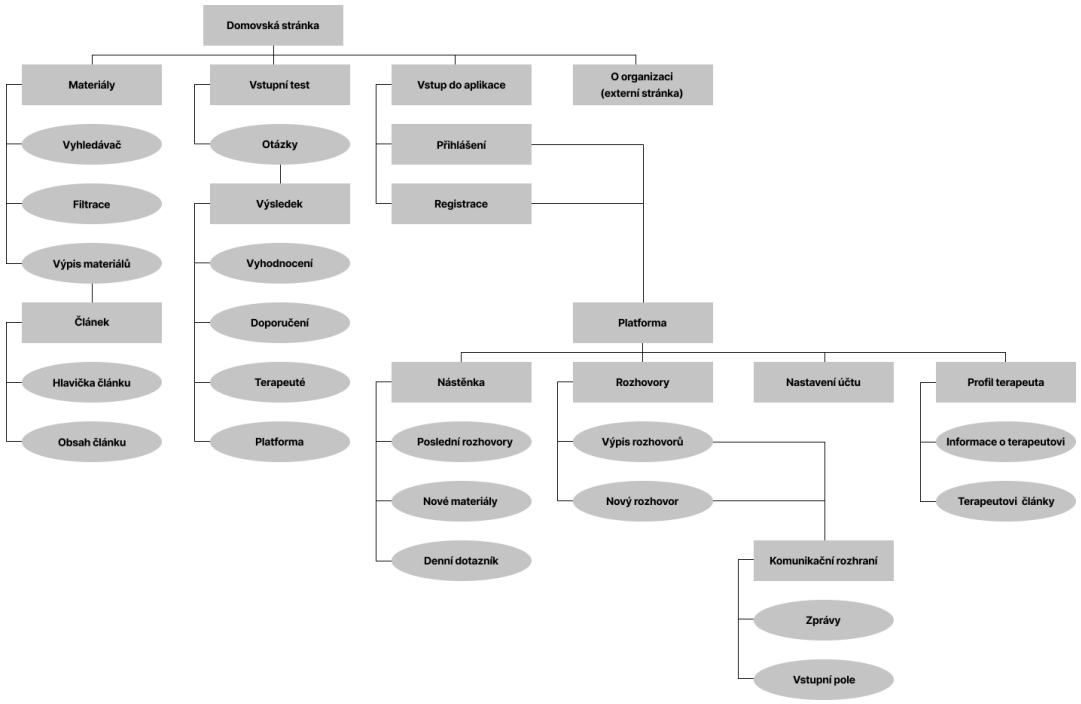
Návrh je fáze, ve které se na základě sesbíraných požadavků tvoří návrh budoucího systému. Je to fáze, která předchází od a, od které se následně odvíjí fáze implementace. Je tedy velmi důležité, aby byl systém navržen správně a navazoval na definované požadavky. V případě nesprávně navrženého systému je pravděpodobné, že v pozdějších fázích budou vznikat kritické chyby, které mohou značně ovlivnit pracnost a časovou náročnost projektu.

## 5.1 Návrh informační architektury

Důležitou součástí návrhové fáze je návrh informační architektury. Informační architektura se zaměřuje na organizaci informací v rámci vyvíjeného digitálního produktu tak, aby bylo pro uživatele nalezení požadovaných informací co nejintuitivnější a nejjednodušší. Správné rozmístění a spojení informací je důležité, protože jakmile bude uživatel k nalezení informace muset vynaložit větší úsilí a vyhledávání bude příliš komplikované, hrozí, že web či aplikaci opustí. Informační architekturu lze vizualizovat pomocí různých diagramů jako jsou například myšlenkové mapy či stromové diagramy.

### 5.1.1 Informační architektura aplikace Nevyhasni

Na obrázku níže lze vidět navržená informační architektura v podobě diagramu, kde tvar obdélníku zastává roli stránky a tvar elipsy zastupuje komponentu dané stránky.



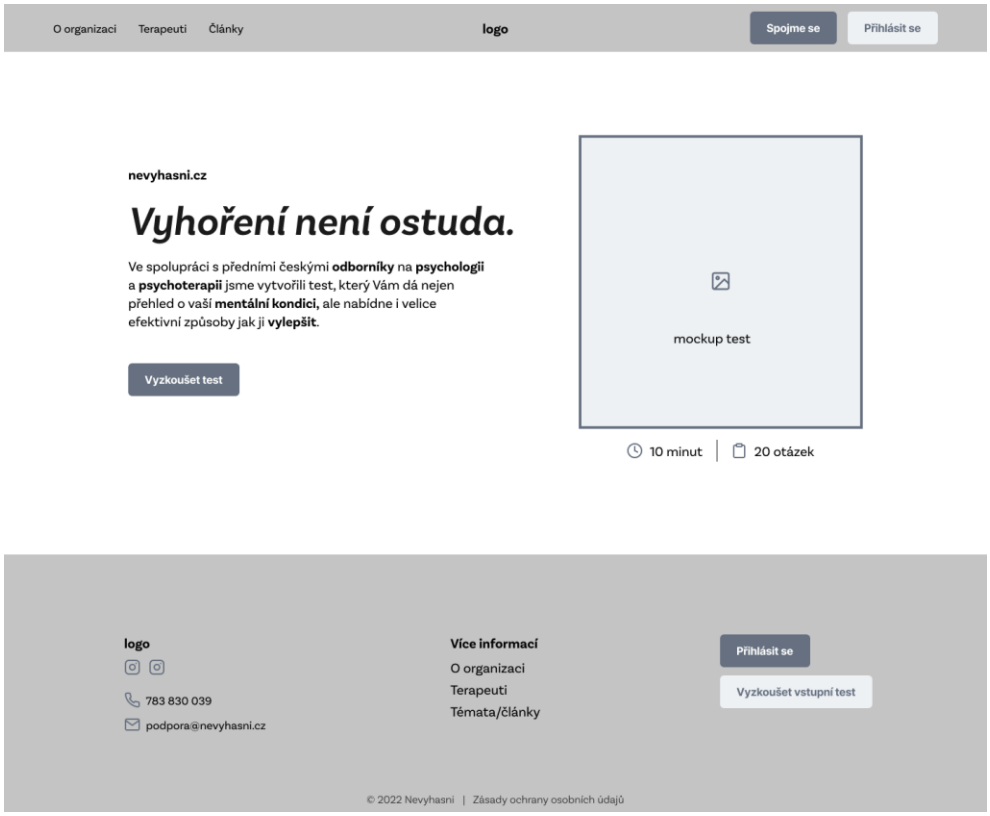

Obrázek 9 Informační architektura aplikace Nevyhasni (vlastní zpracování)

### 5.1.2 Drátěný model aplikace Nevyhasni

Po dokončení návrhu informační architektury je vhodné navázat návrhem tzv. wireframe, také označovaného jako drátěný model aplikace. Jedná se o první vizualizaci vzhledu vyvíjené aplikace, kde jsou znázorněny především její hlavní prvky a jejich rozložení na jednotlivých podstránkách. Výhodou je, že je snazší provádět úpravy v drátěném modelu než upravovat kompletní design uživatelského rozhraní, proto je vhodné navrhnout základní vizuální podobu aplikace právě pomocí drátěného modelu a až poté se věnovat celkovému vzhledu vyvíjeného produktu.

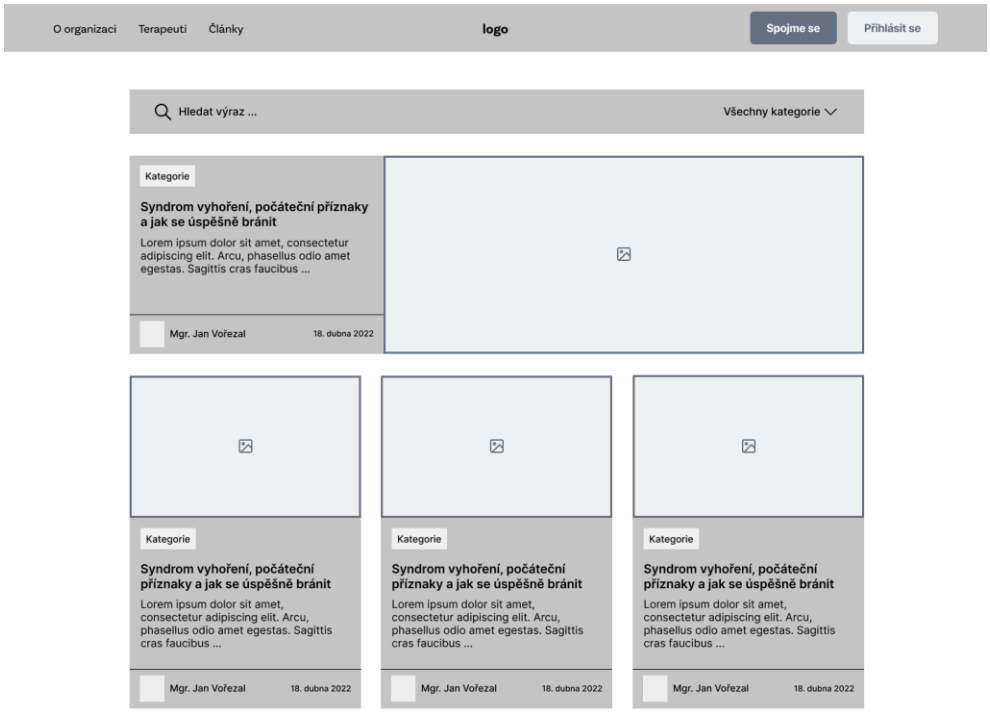
Následující obrázky budou zobrazovat základní drátěný model aplikace Nevyhasni, který byl navrhován v čele s profesionálním designérem pomocí online nástroje Figma. Jedná se o první vizualizaci aplikace a je tedy možné, že implementovaná aplikace se bude lišit v detailech.

**Domovská stránka** zobrazena na obrázku 10 obsahuje navigaci, patičku a popis vstupního testu a tlačítko pomocí, kterého se na něj uživatel může přesměrovat.




Obrázek 10 Wireframe domovské stránky

**Stránka s materiály** zobrazena na obrázku 11 také disponuje navigací a patičkou, která však není v návrhu viditelná. Dalším prvkem je lišta s vyhledávačem, kam uživatel může zadat jakýkoliv hledaný výraz a rozbalovacím menu ve kterém si může zvolit libovolnou kategorii. Pod touto lištou se nachází samotný výpis článků, který bude odpovídat hledaným výrazům či zvolené kategorii. Pokud uživatel nezvolí žádnou kategorii a hledaný výraz, budou se zobrazovat veškeré dostupné články.

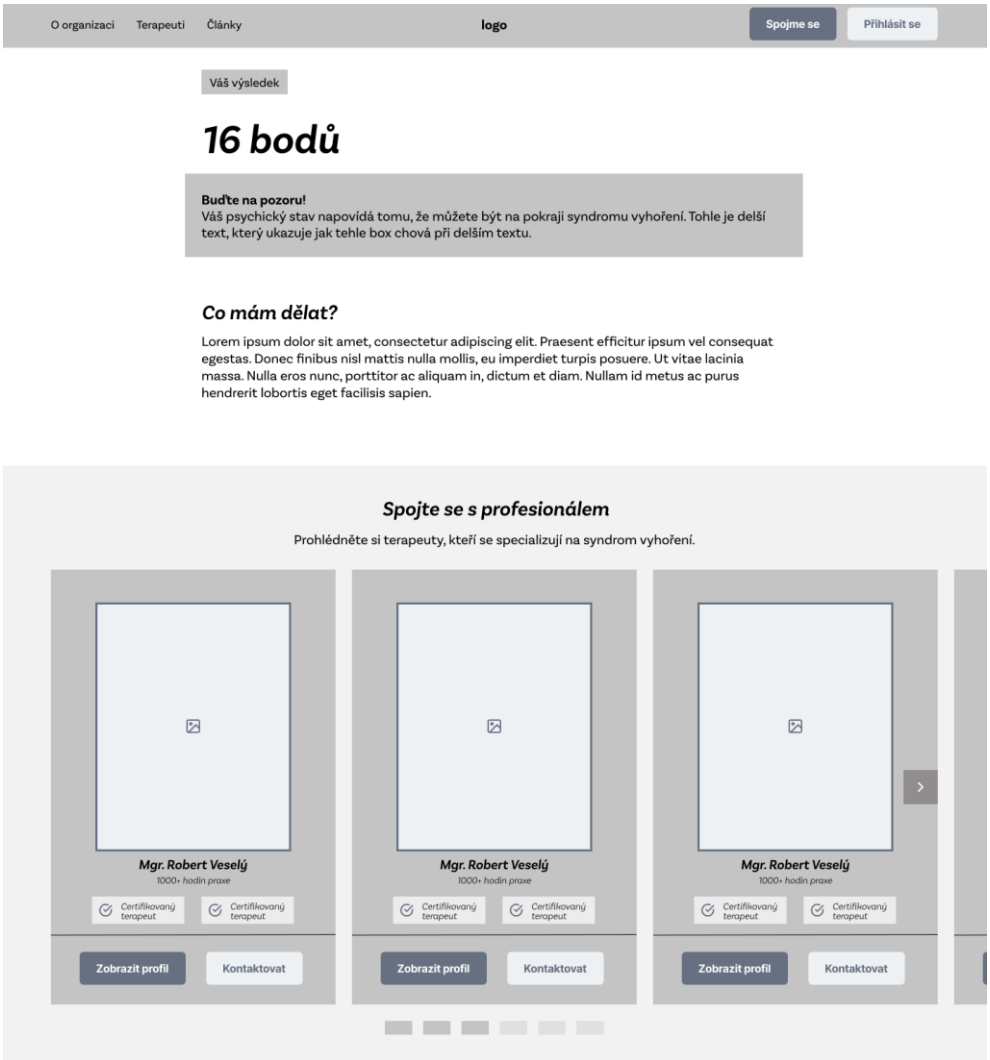



Obrázek 11 Wireframe stránky s články

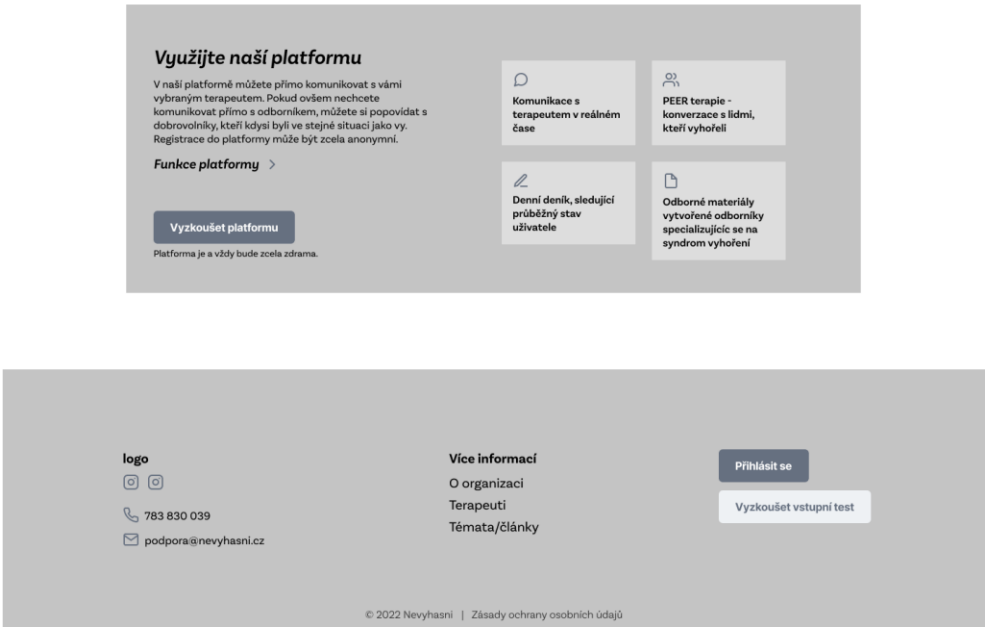
**Vstupní test** zobrazen v návrhu na obrázku 12 obsahuje navigaci a jednotlivé otázky a na ně možné odpovědi. Jak již bylo definováno ve funkčních požadavcích, otázky jsou vizuálně rozděleny do kategorií tak, aby v každém kroku byly jen otázky dané kategorie a v následujícím kroku otázky kategorie jiné. Dále je zde tlačítko „Další“, které uživatele přesune na další krok. Pokud by uživatel byl na dalším kroku, bude zde i tlačítko „Předchozí“ a uživatel se tak bude moci k předchozím otázkám vrátit.


Obrázek 12 Wireframe vstupního testu

**Výsledek vstupního testu** zobrazen na obrázku 13 a 14 (obrázek byl kvůli velkému rozsahu rozdělen) obsahuje navigaci, patičku, výsledek ve formě bodů a slovního popisu, doporučení pro uživatele, výpis terapeutů v podobě slideru s možností zobrazení profilu terapeuta pomocí tlačítka „Zobrazit profil“ či přímého kontaktování terapeuta pomocí tlačítka „Kontaktovat“. Níže, jak lze vidět na obrázku 14, se nachází popis aplikace Nevyhasni a jejích hlavních funkcí a tlačítko „Vyzkoušet platformu“, které uživatele přesměruje na registrační stránku. Pokud se zde uživatel zaregistruje, bude uložen jeho výsledek vstupního testu.

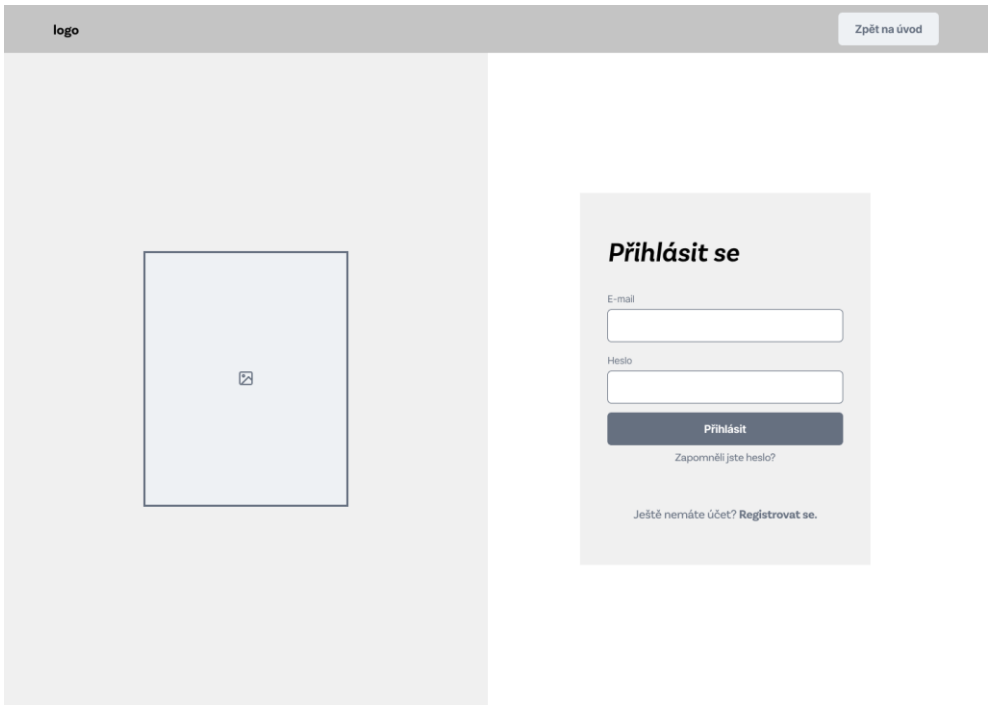



Obrázek 13 Wireframe výsledku vstupního testu (první část)



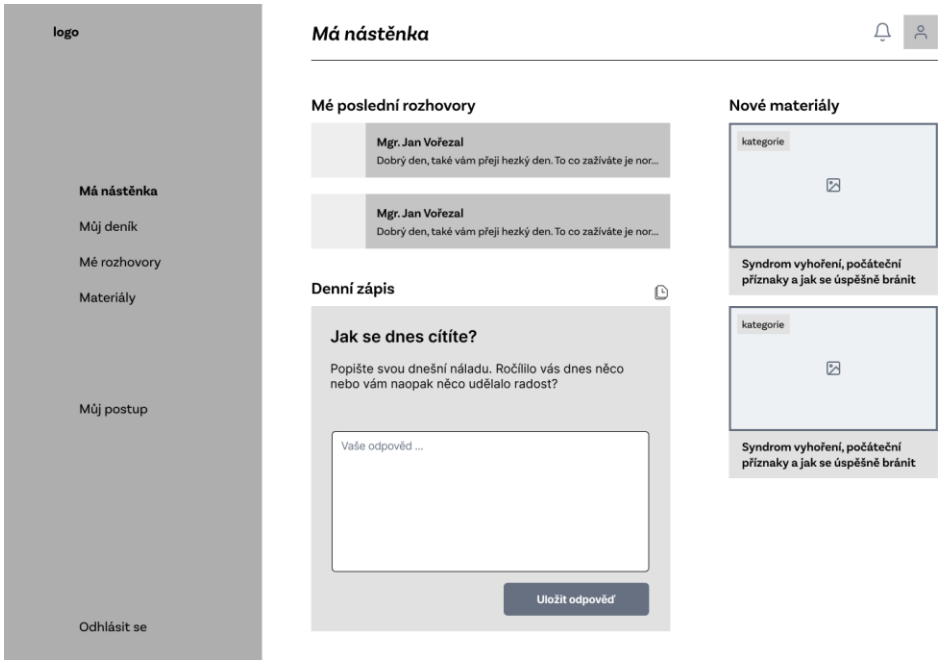

Obrázek 14 Wireframe vstupního testu (druhá část)

**Přihlašovací stránka** zobrazená na obrázku 15 na levé straně obrazovky obsahuje zástupný obrázek, který ve výsledném designu bude nahrazen bannerem s logem Nevyhasni a užitečnými informacemi. Na pravé straně obrazovky se nachází box s přihlašovacím formulářem a odkazem na registraci, pro případ, že uživatel ještě nemá vytvořený účet. Dále přihlašovací stránka disponuje navigací, která se ale od předchozích liší tak, že nezahrnuje všechny odkazy a tlačítka jsou nahrazena tlačítkem „Zpět na úvod“, které uživatele přesměruje zpět na domovskou stránku.



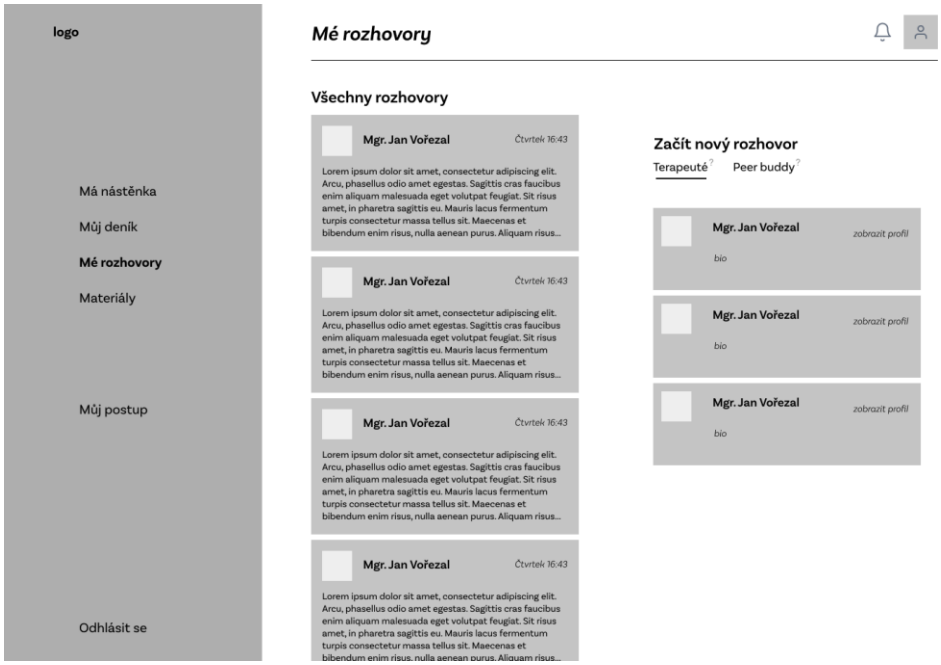

Obrázek 15 Wireframe přihlašovací stránky

**Uživatelská nástěnka** zobrazená na obrázku 16 disponuje postranní navigací, horní navigací s názvem aktuální podstránky, tlačítkem pro oznámení, které po rozkliknutí rozbalí seznam notifikací a ikonou účtu, která uživatele přesměruje na stránku pro úpravu uživatelského účtu. Dále nástěnka zahrnuje výpis nedávných rozhovorů, výpis nových relevantních materiálů a denní dotazník, který zde uživatel může vyplnit či se pomocí ikony přesměrovat na historii jeho odpovědí.



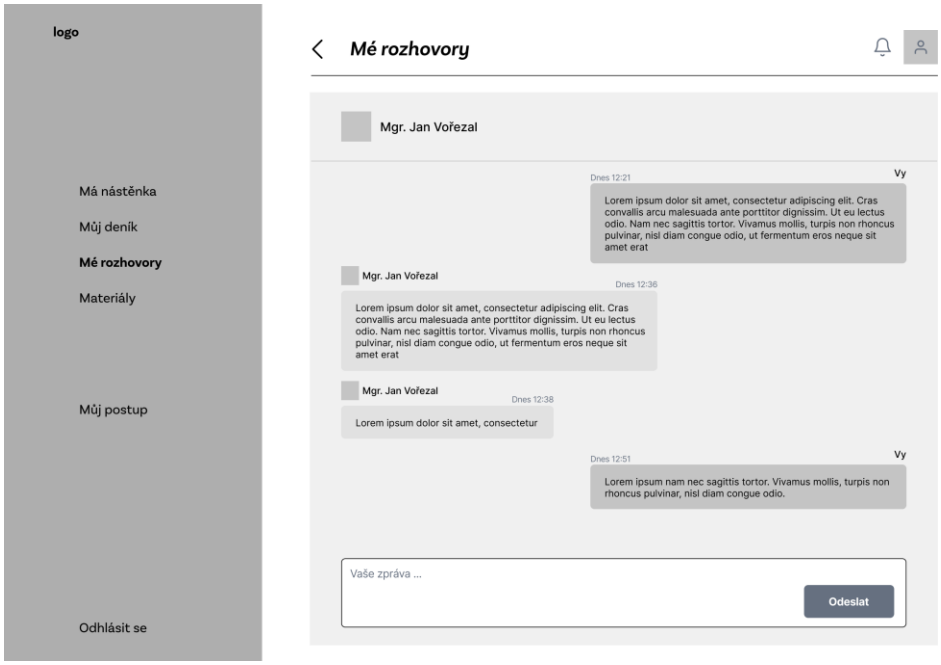

Obrázek 16 Wireframe uživatelské nástěnky

**Stránka s rozhovory uživatele** viditelná na obrázku 17, opět disponuje postranní a horní navigací. Dalším prvkem stránky je seznam všech rozhovorů uživatele a výpis dostupných terapeutů či dobrovolníků, zde označených jako „Peer buddy“, s možností zahájení nové konverzace. Uživatel si zde také může vybrat, zda chce začít nový rozhovor s terapeutem či s dobrovolníkem.




Obrázek 17 Wireframe stránky s rozhovory uživatele

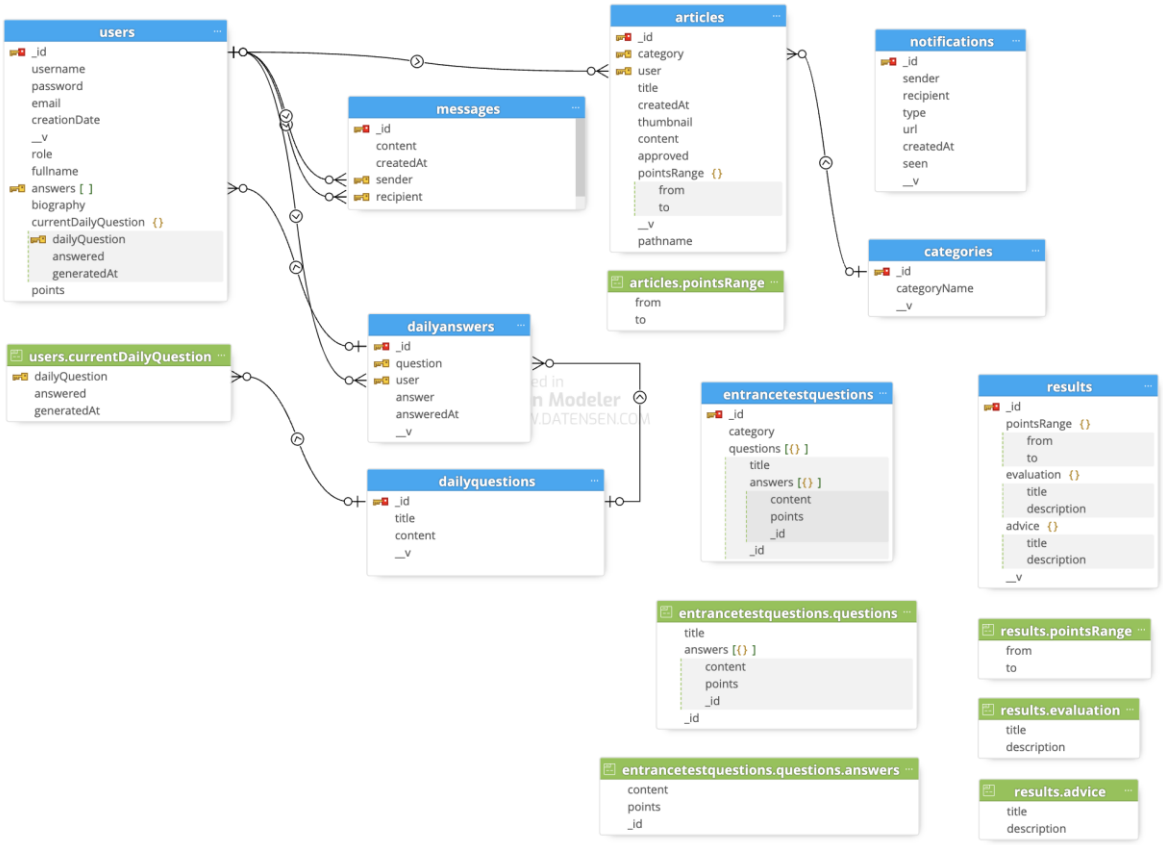
**Komunikační rozhraní** na obrázku 18 obsahuje, stejně jako v předchozích případech, postranní a horní navigaci. Horní navigace zde ale navíc disponuje tlačítkem pomocí, kterého se uživatel může vrátit zpět na stránku se všemi rozhovory. Samotné komunikační okno obsahuje lištu s profilovým obrázkem a jménem terapeuta, přijaté a odeslané zprávy a také pole pro novou zprávu s tlačítkem pro odeslání.




Obrázek 18 Wireframe komunikačního rozhraní

## 5.2 Popis datového modelu

V této části bude popsán datový model aplikace. Při návrhu datového modelu je vhodné zohlednit typ databázového systému ve kterém se následně bude datový model realizovat, protože každý typ funguje na odlišných principech (viz kapitola 2). Aplikace Nevyhasni bude využívat nerelační dokumentové databáze MongoDB. Schéma bylo vytvořeno pomocí aplikace Moon Modeler. Ve schématu jsou vidět jednotlivé kolekce, označené modrou hlavičkou a jejich zanořené objekty, které jsou zde označeny zelenou hlavičkou.




Obrázek 19 Datový model aplikace

Následuje popis jednotlivých kolekcí:

- **users** – kolekce, která slouží pro ukládání všech uživatelů
- **messages** – kolekce pro ukládání zpráv z komunikačního rozhraní
- **articles** – kolekce, která uchovává terapeuty vytvořené materiály
- **dailyanswers** – kolekce pro uchovávání odpovědí uživatele z denního dotazníku
- **dailyquestions** – kolekce otázek ze které se v aplikaci náhodně vybírají dokumenty pro denní dotazník
- **entrancetestquestions** – kolekce kategorií s otázkami vstupního testu
- **results** – kolekce s výsledky pro vstupní test
- **notifications** – kolekce pro ukládání notifikací

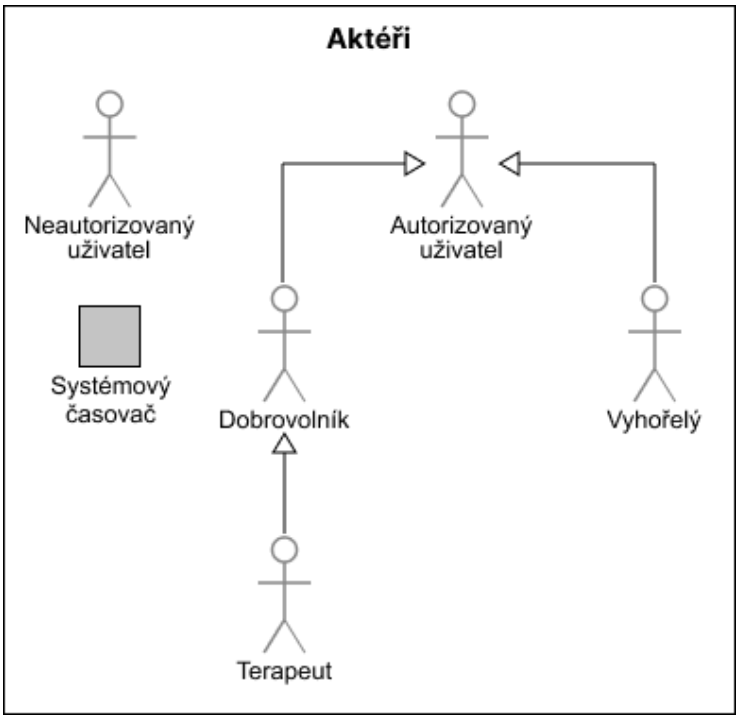
Struktura databáze bude více rozebrána v kapitole 6.2.

## 5.3 Aktéři a případy užití

V této podkapitole budou popsáni jednotliví aktéři a také chování aplikace z pohledu uživatele, pomocí diagramu případů užití.

### 5.3.1 Aktéři

Následující obrázek zobrazuje diagram s primárními aktéry sytému, tedy všemi typy uživatelů aplikace Nevyhasni. Diagram je navržený pomocí online nástroje GenMyModel.




Obrázek 20 Aktéři aplikace Nevyhasni

**Neautorizovaný uživatel** je návštěvník aplikace, který nedisponuje uživatelským účtem, či není v aplikaci přihlášený. Tento typ uživatele může vykonat vstupní test, zobrazovat a vyhledávat materiály a zobrazovat seznam terapeutů dostupných v aplikaci.

**Autorizovaný uživatel** je uživatel aplikace, který disponuje uživatelským účtem a je právě v aplikaci přihlášený. Autorizovaný uživatel může spravovat svůj uživatelský účet, komunikovat pomocí komunikačního rozhraní a zobrazovat profil terapeutů či dobrovolníků. Od autorizovaného uživatele se dále odvíjejí tři uživatelské role, tedy následující typy uživatelů:

1. Uživatel, který trpí syndromem vyhoření nebo má podezření, že by se u něho mohl syndrom vyhoření v blízké době objevit. Jedná se tedy o uživatele, který

v aplikaci vyhledává pomoc. Dále bude tento typ uživatele nazývaný jen jako ***vyhořelý***.

2. Uživatel, který chce vyhořelému prostřednictvím aplikace pomoct. Takový uživatel se dělí do dvou rolí:

    a. Odborný dobrovolník, který má s psychologií a terapií už nějakou zkušenost. Často se může jednat o vysokoškolského studenta psychologie či jiného podobného oboru. Níže bude uživatel této role nazývaný pouze jako ***dobrovolník***.

    b. Odborník na psychologii a psychoterapii (dále jen ***terapeut***)

**Vyhořelý** je uživatel, který dědí všechny vlastnosti a funkce autorizovaného uživatele. Kromě toho může vyplňovat denní dotazník a zobrazovat si historii minulých odpovědí. V rámci komunikačního rozhraní může komunikovat s rolemi dobrovolník a terapeut.

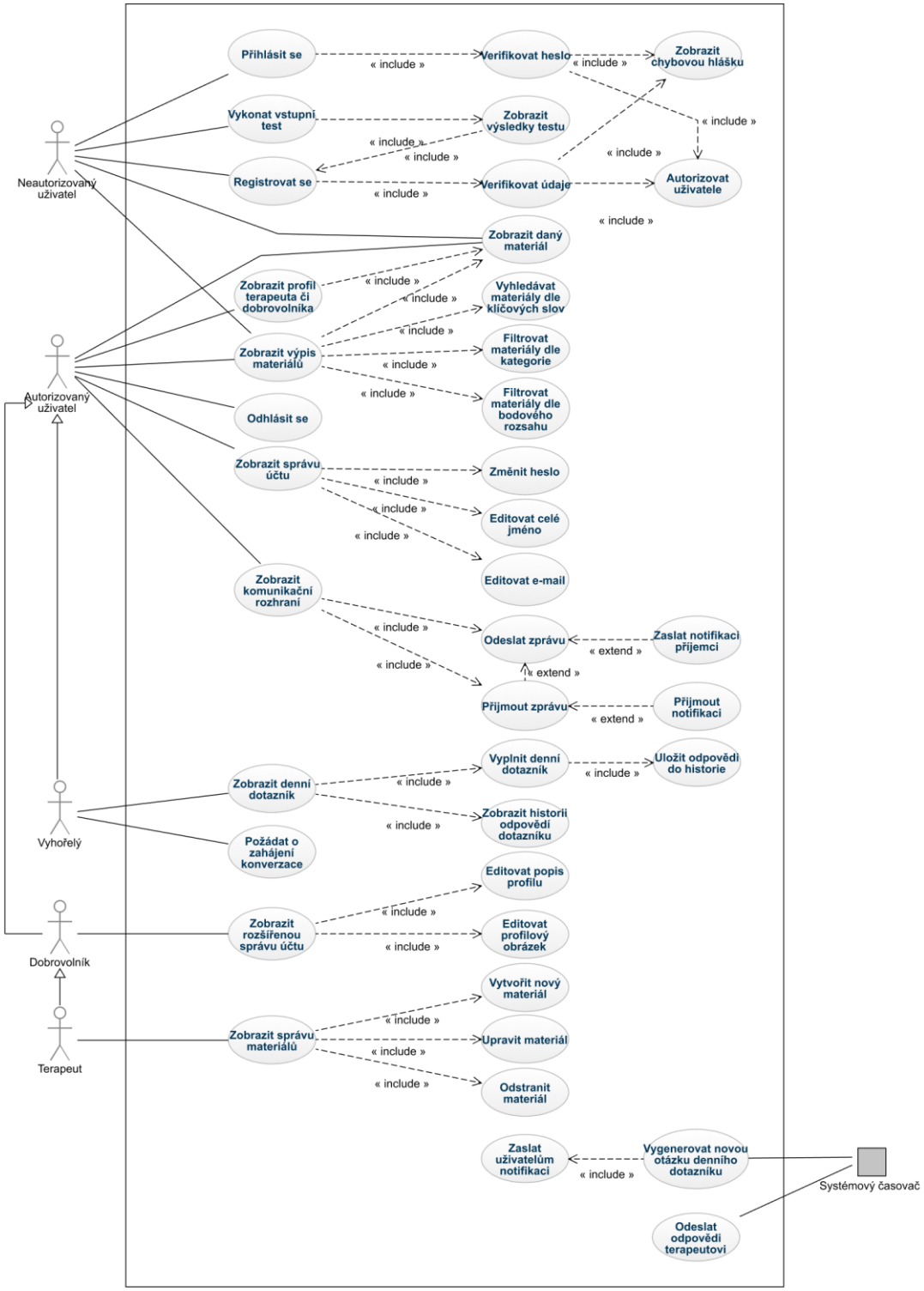
**Dobrovolník** také dědí veškeré vlastnosti a funkce autorizovaného uživatele. Navíc jeho účet disponuje uživatelským profilem v rámci kterého si může přidat popis a fotografii profilu. Dobrovolník může komunikovat s uživateli role vyhořelý.

**Terapeut** dědí veškeré vlastnosti a funkce autorizovaného uživatele a dobrovolníka. To znamená, že stejně jako dobrovolník, má svůj uživatelský profil a může komunikovat s uživateli role vyhořelý. Nadto je terapeutovi umožněno publikovat, upravovat a mazat vlastní materiály, které se poté také zobrazují na jeho uživatelském profilu.

**Systémový časovač** je sekundární aktér, který vykonává dané případy užití na základě uběhnuté doby. Konkrétně odesílá sesbírané odpovědi uživatele na denní dotazník terapeutovi a generuje nové otázky denního dotazníku.

### 5.3.2 Digram případů užití

Diagram případů užití je součástí UML, grafického jazyka pro specifikaci, navrhování a vizualizaci softwarových systémů. Diagram se skládá z jednotlivých případů užití, aktérů a jejich vztahů. Pomocí případů užití popisuje chování systémů z pohledu uživatelů, kteří jsou zde tedy označování jako aktéři. Diagram případů užití pro aplikaci Nevyhasni je zobrazen níže na obrázku 21.




Obrázek 21 Diagram případů užití

## 5.4 Uživatelské scénáře

V této podkapitole budou popsány uživatelské scénáře navazující na diagram případů užití z předchozí podkapitoly, které popisují tok událostí během provádění jednotlivých případů užití.

### 5.4.1 Registrace uživatele

Tabulka 10 [UC01] Registrace uživatele

| **[UC001] Registrace uživatele** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu registrace a následné autorizace uživatele | | |
| **Aktér:**<br>Neautorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s registračním formulářem | | |
| **Výstupní podmínky:**<br>Je vytvořen nový uživatelská účet<br>Uživatel je autorizován | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní veškeré údaje do příslušných polí |
| 2 | Aktér | Aktér klikne na tlačítko „Vytvořit účet“ |
| 3 | Systém | Systém provede validaci zadaných dat |
| 4 | Systém | Systém autorizuje uživatele do aplikace |
| 5 | Systém | Systém zobrazí hlášku o úspěšné registraci |
| **Alternativní scénáře:**<br>UC001a – Alternativní scénář: Neúspěšná validace registračních dat | | |

Tabulka 11 [UC001a] Neúspěšná validace registračních dat

| **Alternativní scénář: [UC001a] Neúspěšná validace registračních dat** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy byla do registračního formuláře zadána neplatná data. | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 4 | Systém | Systém zachytí nevalidní data |
| 5 | Systém | Systém zobrazí hlášku s chybou |

### 5.4.2 Přihlášení uživatele

Tabulka 12 [UC002] Přihlášení uživatele

| **[UC002] Přihlášení uživatele** |
|---|
| **Charakteristika:**<br>Zachycení procesu přihlášení uživatele |
| **Aktér:**<br>Neautorizovaný uživatel |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s přihlašovacím formulářem |
| **Výstupní podmínky:**<br>Aktér je autorizován |

| Krok | Aktér/Systém | Popis |
|---|---|---|
| 1 | Aktér | Aktér vyplní veškeré údaje do příslušných polí |
| 2 | Aktér | Aktér klikne na tlačítko „Přihlásit se“ |
| 3 | Systém | Systém ověří existenci uživatele |
| 4 | Systém | Systém provede verifikaci hesla |
| 5 | Systém | Systém autorizuje uživatele do aplikace |
| 6 | Systém | Systém zobrazí hlášku o úspěšném přihlášení |
| **Alternativní scénáře:**<br>UC002a – Alternativní scénář: Uživatel nebyl nalezen<br>UC002b – Alternativní scénář: Zadané heslo je neplatné | | |

Tabulka 13 [UC002a] Uživatel nebyl nalezen

| **Alternativní scénář: [UC002a] Uživatel nebyl nalezen** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy přihlašovaná uživatel nebyl nalezen. | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 4 | Systém | Systém v databázi nenalezne shodu uživatele |
| 5 | Systém | Systém zobrazí hlášku s chybou |

Tabulka 14 [UC002b] Zadané heslo je neplatné

| **Alternativní scénář: [UC002b] Zadané heslo je neplatné** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy do přihlašovacího formuláře bylo zadáno neplatné heslo | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 5 | Systém | Systém zachytí neplatné heslo |
| 6 | Systém | Systém zobrazí hlášku s chybou |

### 5.4.3 Vstupní test

Tabulka 15 [UC003] Vykonání vstupního testu

| **[UC003] Vykonání vstupního testu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu vykonání vstupního testu | | |
| **Aktér:**<br>Uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena první kategorie otázek na stránce vstupního testu | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován na stránku s výsledkem | | |
| **Krok** | **Aktér/Systém** | **Popis** |

| | | |
|---|---|---|
| 1 | Aktér | Aktér u jednotlivých otázek vybere danou odpověď kliknutím |
| 2 | Aktér | Aktér klikne na tlačítko „Další“ |
| 3 | Systém | Systém zobrazí další kategorii otázek |
| 4 | Aktér | Aktér klikne na tlačítko „Odeslat“ |
| 5 | Systém | Systém vyhodnotí výsledek |
| 6 | Systém | Systém aktéra přesměruje na stránku s výsledkem |

### 5.4.4 Zobrazení profilu terapeuta

Tabulka 16 [UC004] Zobrazení profilu terapeuta

| **[UC004] Zobrazení profilu terapeuta** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu k zobrazení profilu terapeuta | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Uživatel je autorizován | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován profil daného terapeuta | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na odkaz na profil terapeuta |
| 2 | Systém | Systém aktéra přesměruje na stránku s profilem terapeuta |

### 5.4.5 Materiály

Tabulka 17 [UC005] Zobrazení výpisu materiálů

| **[UC005] Zobrazení výpisu materiálů** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu k zobrazení výpisu materiálů | | |
| **Aktér:**<br>Autorizovaný nebo neautorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován stránku s výpisem materiálů | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na odkaz na odkaz „Materiály“ |
| 2 | Systém | Systém aktéra přesměruje na stránku s výpisem materiálů |

Tabulka 18 [UC006] Vyhledávání materiálů podle klíčových slov

| **[UC006] Vyhledávání materiálů podle klíčových slov** |
|---|
| **Charakteristika:** |

| Zachycení procesu vyhledávání materiálů podle klíčových slov | | |
|---|---|---|
| **Aktér:**<br>Autorizovaný nebo neautorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s výpisem materiálů (UC005) | | |
| **Výstupní podmínky:**<br>Aktérovi se zobrazí materiály shodující se se zadaným výrazem | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér zadá hledaný výraz do vstupního pole v liště |
| 2 | Systém | Systém vyfiltruje materiály shodující se se zadaným výrazem a zobrazí výsledky |
| **Alternativní scénáře:**<br>UC006a – Alternativní scénář: Nebyly nalezeny žádné materiály | | |

Tabulka 19 [UC006a] Nebyly nalezeny žádné materiály

| **Alternativní scénář: [UC006a] Nebyly nalezeny žádné materiály** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy nejsou nalezeny žádné výsledky | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 2 | Systém | Systém nenalezne žádné shody a zobrazí prázdnou stránku |

Tabulka 20 [UC007] Filtrování materiálů dle kategorie

| **[UC007] Filtrování materiálů dle kategorie** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu filtrování materiálů dle kategorie | | |
| **Aktér:**<br>Autorizovaný nebo neautorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s výpisem materiálů (UC005) | | |
| **Výstupní podmínky:**<br>Aktérovi se zobrazí pouze materiály z vybrané kategorie | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na rozbalovací menu v liště |
| 2 | Systém | Systém zobrazí dostupné kategorie |
| 3 | Aktér | Aktér klikne na zvolenou kategorii |
| 4 | Systém | Systém vyfiltruje materiály dle zvolené kategorie a zobrazí výsledky |

Tabulka 21 [UC008] Filtrování materiálů dle bodového rozsahu

| **[UC008] Filtrování materiálů dle bodového rozsahu** |
|---|
| **Charakteristika:** |

| Zachycení procesu filtrování relevantních materiálů dle bodového rozsahu | | |
|---|---|---|
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s výpisem materiálů (UC005)<br>Aktér má dokončený vstupní test, a tedy i uložený bodový výsledek | | |
| **Výstupní podmínky:**<br>Aktérovi se zobrazí pouze materiály odpovídající jeho bodovému výsledku | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér zaškrtne pole „Filtrovat dle relevance“ v liště |
| 2 | Systém | Systém vyfiltruje materiály, které svým bodovým rozsahem odpovídají bodovému výsledku aktéra a zobrazí výsledky |

Tabulka 22 [UC009] Zobrazení stránky s materiálem

| **[UC009] Zobrazení stránky s materiálem** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu k zobrazení stránky s materiálem | | |
| **Aktér:**<br>Autorizovaný nebo neautorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován na stránku s materiálem | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na odkaz na daný materiál |
| 2 | Systém | Systém aktéra přesměruje na stránku s materiálem |

### 5.4.6 Odhlášení uživatele

Tabulka 23 [UC010] Odhlášení uživatele

| **[UC010] Odhlášení uživatele** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu odhlášení uživatele | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktér je odhlášen z aplikace | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na tlačítko „Odhlásit se“ |
| 2 | Systém | Systém odhlásí aktéra z aplikace |

### 5.4.7 Správa účtů

Tabulka 24 [UC011] Zobrazení správy účtů

| **[UC011] Zobrazení správy účtů** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu k zobrazení správy účtu | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazeno rozhraní aplikace s odkazem na správu účtu | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován na stránku pro správu účtu | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na ikonu s odkazem na správu účtu |
| 2 | Systém | Systém aktéra přesměruje na stránku pro správu účtu |

Tabulka 25 [UC012] Změna hesla

| **[UC012] Změna hesla** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu změny hesla uživatele | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je změněno heslo | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní všechna pole pro změnu hesla |
| 2 | Aktér | Aktér klikne na tlačítko „Změnit heslo“ |
| 3 | Systém | Systém verifikuje současné heslo |
| 4 | Systém | Systém porovná nové heslo a potvrzení nového hesla |
| 5 | Systém | Systém uloží nové heslo |
| **Alternativní scénáře:**<br>UC012a – Alternativní scénář: Bylo zadáno špatné heslo<br>UC012b – Alternativní scénář: Zadaná nová hesla se neshodují | | |

Tabulka 26 [UC012a] Bylo zadáno špatné heslo

| **Alternativní scénář: [UC012a] Bylo zadáno špatné heslo** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy bylo zadáno špatné heslo | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 4 | Systém | Systém zachytí neplatné heslo |

| | | |
|---|---|---|
| 5 | Systém | Systém zobrazí hlášku s chybou |

Tabulka 27 [UC012b] Zadaná nová hesla se neshodují

| **Alternativní scénář: [UC012b] Zadaná nová hesla se neshodují** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy se zadaná nová hesla neshodují | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 5 | Systém | Systém zjistí, že zadaná hesla neshodují |
| 6 | Systém | Systém zobrazí hlášku s chybou |

Tabulka 28 [UC013] Změna e-mailu

| **[UC013] Změna e-mailu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu změny e-mailové adresy uživatele | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je změněna e-mailová adresa | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní vstupní pole pro e-mailovou adresu |
| 2 | Aktér | Aktér klikne na tlačítko „Aktualizovat účet“ |
| 3 | Systém | Systém provede validaci zadané e-mailové adresy |
| 4 | Systém | Systém uloží novou e-mail adresu uživatele |
| **Alternativní scénáře:**<br>UC013a – Alternativní scénář: Byla zadána neplatná e-mailová adresa | | |

Tabulka 29 [UC013a] Byla zadána neplatná e-mailová adresa

| **Alternativní scénář: [UC013a] Byla zadána neplatná e-mailová adresa** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy se aktér zadal nevalidní e-mailovou adresu | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 4 | Systém | Systém zjistí, že zadaná adresa není validní |
| 5 | Systém | Systém zobrazí hlášku s chybou |

Tabulka 30 [UC013] Změna jména

| **[UC013] Změna jména** |
|---|
| **Charakteristika:** |

| Zachycení procesu změny jména | | |
|---|---|---|
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je změněno jméno | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní vstupní pole jméno |
| 2 | Aktér | Aktér klikne na tlačítko „Aktualizovat účet“ |
| 3 | Systém | Systém uloží nové jméno uživatele |

Tabulka 31 [UC014] Editace popisu profilu

| **[UC014] Editace popisu profilu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu editace popisu profilu | | |
| **Aktér:**<br>Terapeut nebo dobrovolník | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je uložen nový popis profilu | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní vstupní pole popis profilu |
| 2 | Aktér | Aktér klikne na tlačítko „Aktualizovat účet“ |
| 3 | Systém | Systém uloží nové popis profilu |

Tabulka 32 [UC015] Editace profilového obrázku

| **[UC015] Editace profilového obrázku** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu editace profilového obrázku | | |
| **Aktér:**<br>Terapeut nebo dobrovolník | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je uložen nový popis profilu | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér vyplní přidá URL adresu obrázku |
| 2 | Aktér | Aktér klikne na tlačítko „Aktualizovat účet“ |
| 3 | Systém | Systém uloží nové popis profilu |

### 5.4.8 Komunikační rozhraní

Tabulka 33 [UC016] Zobrazení komunikačního rozhraní

| **[UC016] Zobrazení komunikačního rozhraní** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu k zobrazení komunikačního rozhraní | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazeno rozhraní aplikace s odkazem na komunikační rozhraní | | |
| **Výstupní podmínky:**<br>Aktér je přesměrován na stránku s komunikačním rozhraním | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na zprávu s odkazem na komunikační rozhraní s daným uživatelem |
| 2 | Systém | Systém aktéra přesměruje na stránku s komunikačním rozhraním |

Tabulka 34 [UC017] Odeslání zprávy

| **[UC017] Odeslání zprávy** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu odeslání nové zprávy | | |
| **Aktér:**<br>Autorizovaný uživatel | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazeno komunikační rozhraní s daným uživatelem (UC016) | | |
| **Výstupní podmínky:**<br>Nová zpráva je odeslána a zobrazena účastníkům konverzace | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér zadá zprávu do vstupního pole |
| 2 | Aktér | Aktér klikne na tlačítko pro odeslání |
| 3 | Systém | Systém uloží novou zprávu |
| 4 | Systém | Systém naservíruje novou zprávu oběma účastníkům konverzace |
| 5 | Systém | Systém vytvoří notifikaci o zprávě pro příjemce |

### 5.4.9 Denní dotazník

Tabulka 35 [UC018] Zobrazení denního dotazníku

| **[UC018] Zobrazení denního dotazníku** |
|---|
| **Charakteristika:**<br>Zachycení procesu zobrazení denního dotazníku |
| **Aktér:**<br>Vyhořelý |

| **Vstupní podmínky:**<br>Nejsou | | |
|---|---|---|
| **Výstupní podmínky:**<br>Aktér je přesměrován na stránku s denním dotazníkem | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na na prvek s odkazem na nástěnku |
| 2 | Systém | Systém aktéra přesměruje na stránku s denním dotazníkem |

Tabulka 36 [UC019] Vyplnění denního dotazníku

| **[UC019] Vyplnění denního dotazníku** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu vyplnění denního dotazníku | | |
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s denním dotazníkem (UC018) | | |
| **Výstupní podmínky:**<br>Systém uloží novou odpověď na otázku denního dotazníku | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér zadá odpověď do vstupního pole |
| 2 | Aktér | Aktér klikne na tlačítko „Uložit“ |
| 3 | Systém | Systém uloží novou odpověď do historie a označí současnou otázku jako zodpovězenou |

Tabulka 37 [UC020] Zobrazení historie odpovědí denního dotazníku

| **[UC020] Zobrazení historie odpovědí denního dotazníku** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu zobrazení historie odpovědí denního dotazníku | | |
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka s denním dotazníkem (UC018) | | |
| **Výstupní podmínky:**<br>Aktérovi je zobrazeno okno s historií odpovědí | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na ikonu historie |
| 2 | Systém | Sytém zobrazí okno s historií odpovědí denního dotazníku |

### 5.4.10 Správa materiálů

Tabulka 38 [UC021] Zobrazení správy materiálů

| **[UC021] Zobrazení správy materiálů** |
|---|
| **Charakteristika:** |

| Zachycení procesu zobrazení správy materiálů | | |
|---|---|---|
| **Aktér:**<br>Terapeut | | |
| **Vstupní podmínky:**<br>Nejsou | | |
| **Výstupní podmínky:**<br>Aktérovi je zobrazena stránka se správou materiálů | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na odkaz „Správa materiálů“ |
| 2 | Systém | Sytém aktéra přesměruje na stránku se správou materiálů |

Tabulka 39 [UC022] Vytvoření nového materiálu

| **[UC022] Vytvoření nového materiálu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu vytvoření nového materiálu | | |
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka se správou materiálů (UC021) | | |
| **Výstupní podmínky:**<br>Vytvořen nový materiál | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne na tlačítko „Nový materiál“ |
| 2 | Systém | Sytém zobrazí okno s formulářem pro vytvoření nového materiálu |
| 3 | Aktér | Aktér vyplní všechna vstupní pole |
| 4 | Aktér | Aktér klikne na tlačítko „Přidat materiál“ |
| 5 | Systém | Systém provede validaci zadaných dat |
| 6 | Systém | Systém uloží nový materiál a zobrazí jej uživateli |
| **Alternativní scénáře:**<br>UC022a – Alternativní scénář: Nebyla vyplněna všechna pole | | |

Tabulka 40 [UC022a] Nebyla vyplněna všechna pole

| **Alternativní scénář: [UC022a] Nebyla vyplněna všechna pole** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy nebyla vyplněna všechna povinná pole | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 6 | Systém | Systém zachytí prázdný vstup |
| 7 | Systém | Systém zobrazí hlášku s chybou |

Tabulka 41 [UC023] Úprava materiálu

| **[UC023] Úprava materiálu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu upravení již existujícího materiálu | | |
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka se správou materiálů (UC021) | | |
| **Výstupní podmínky:**<br>Upravený materiál je uložen a zobrazen uživateli | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne ikonu pro úpravu u daného materiálu ve výpisu |
| 2 | Systém | Sytém zobrazí okno s formulářem pro úpravu materiálu s předvyplněnými daty |
| 3 | Aktér | Aktér změní hodnoty požadovaných polí |
| 4 | Aktér | Aktér klikne na tlačítko „Upravit materiál“ |
| 5 | Systém | Systém provede validaci zadaných dat |
| 6 | Systém | Systém uloží upravený materiál a úpravy zobrazí uživateli |
| **Alternativní scénáře:**<br>UC022a – Alternativní scénář: Nebyla vyplněna všechna pole | | |

Tabulka 42 [UC024] Odstranění materiálu

| **[UC024] Odstranění materiálu** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení procesu odstranění materiálu | | |
| **Aktér:**<br>Vyhořelý | | |
| **Vstupní podmínky:**<br>Aktérovi je zobrazena stránka se správou materiálů (UC021) | | |
| **Výstupní podmínky:**<br>Materiál je odstraněn | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 1 | Aktér | Aktér klikne ikonu pro odstranění u daného materiálu ve výpisu |
| 2 | Systém | Sytém zobrazí dotazovací okno, které se ptá, zdali chce aktér opravdu odstranit materiál |
| 3 | Aktér | Aktér klikne na tlačítko „Ano, odstranit“ |
| 4 | Systém | Systém daný materiál odstraní a změny zobrazí uživateli |
| **Alternativní scénáře:**<br>UC024a – Alternativní scénář: Uživatel nepotvrdil odstranění | | |

Tabulka 43 [UC024a] Uživatel nepotvrdil odstranění

| **Alternativní scénář: [UC024a] Uživatel nepotvrdil odstranění** | | |
|---|---|---|
| **Charakteristika:**<br>Zachycení situace, kdy uživatel v dotazovacím okně nepotvrdil odstranění materiálu | | |
| **Krok** | **Aktér/Systém** | **Popis** |
| 3 | Aktér | Aktér klikne na tlačítko „Ne, zrušit“ |
| 4 | Systém | Systém zavře dotazovací okno |

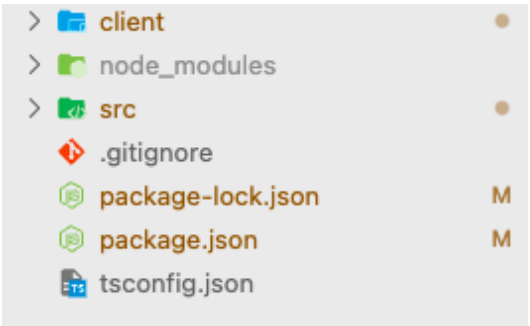
# 6 STRUKTURA VÝSLEDNÉ APLIKACE

Tato kapitola se bude věnovat struktuře výsledné aplikace, konkrétně zde bude popsána adresářová struktura projektu, struktura databáze a způsob jakým komunikuje serverová část s částí klientskou.

## 6.1 Adresářová struktura projektu

Návrh vhodné adresářové struktury je jedním z prvních kroků tvorby softwarového projektu. Navržená struktura by měla být přehledná a přizpůsobená pro budoucí rozšiřování aplikace. Také by měla dodržovat doporučené postupy přímo od vývojářů použitých technologií. Avšak některé technologie, včetně například použité knihovny React, nemají jasně definovaný a doporučený postup pro tvorbu adresářové struktury, a tak je tvorba adresářové struktury čistě na vývojáři či vývojového týmu projektu.

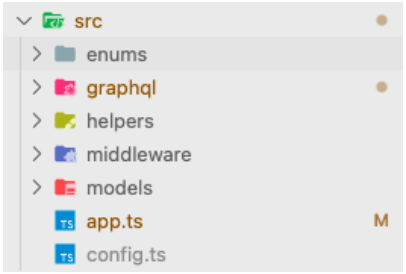
Aplikace je rozdělena do dvou hlavních částí, a to na klienta a server (viz obrázek 22). Tedy veškerý kód pro klientskou část aplikace se nachází ve složce *client*, kód serverové části je pak uchováván ve složce *src*. V kořenové adresáři se dále nachází konfigurační soubor pro TypeScript *tsconfig.json*, soubory pro balíkovací systém *package.json* a *package-lock.json*. Součástí hlavního adresáře je také složka *node_modules*, ve které jsou veškeré balíky využívány serverovou částí aplikace a soubor *.gitignore* ve kterém jsou uvedeny soubory, které se nemají odesílat do vzdáleného repozitáře.




Obrázek 22 Kořenový adresář aplikace

### 6.1.1 Serverová část

Serverová část aplikace je implementována pomocí technologií popsaných v teoretické části práce, tedy pomocí runtime prostředí Node.js, middlewaru Express a knihovny pro implementaci GraphQL serveru Apollo Server.



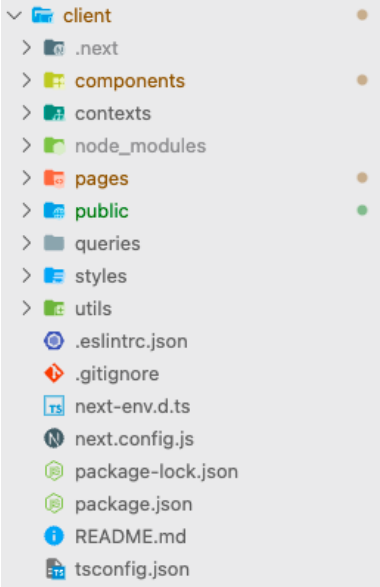

Obrázek 23 Adresář src (serverová část aplikace)

Níže bude popsána adresářová struktura této části viditelné na obrázku 23 výše.

- **enums** – obsahuje soubory s výčtovými typy (enumerators)
- **graphql** – obsahuje definované GraphQL schéma a veškeré obslužné funkce pro GraphQL mutace a dotazy
- **helpers** – zde se nachází soubory s pomocnými funkcemi, které mohou být užitečné v různých místech kódu, jedná se například o funkci pro validaci emailu či funkci pro podepsání tokenu JWT
- **middleware** – drží soubory s funkcemi, které se volají před vykonáním určitých operací, například před vytvořením nového článku je nutné ověřit token uživatele v hlavičce požadavku a na základě toho požadavek provést či vrátit výjimku
- **models** – obsahuje soubory s Mongoose schématy a modely
- **app.ts** – v tomto souboru dochází k inicializaci a následnému spuštění serveru
- **config.ts** – soubor s nastavením, který obsahuje tajná data (JWT klíč pro podpis a verifikaci, připojovací řetězec pro databázi atp.), tento soubor je zahrnut v souboru *.gitignore*, aby se neodesílal do vzdáleného repozitáře

### 6.1.2 Klientská část

Klientská část je implementována pomocí React frameworku Next.js a knihovny pro komunikaci s GraphQL serverem Apollo Client. Tato část se v projektu nachází v adresáři *client* (viz obrázek 24).




Obrázek 24 Adresář client (klientská část aplikace)

Bude následovat popis jednotlivých podadresářů a souborů vyjma souborů, které již byly popsány v předchozí části (tyto soubory zde mají stejný účel):

- **components** – Obsahuje jednotlivé React komponenty rozděleny do podadresářů tak, aby souvisely se stránkou, ke které patří. Sdílené komponenty, které jsou využity na vícero různých stránkách, jsou ve speciálním podadresáři nazvaném *shared*. Komponenty mohou v případě potřeby obsahovat *scss* modul pro zápis přidružených kaskádových stylů.
- **contexts** – Zde se nachází poskytovatelé kontextu, kteří slouží ke sdílení dat mezi komponenty aplikace a eliminují nutnost propisovat data skrze vlastnosti komponent.
- **pages** – Zahrnuje veškeré podstránky webové aplikace, kde název souboru odpovídá URL cestě. Next.js se všemi soubory v tomto adresáři zachází jako se stránkami a automaticky je předkresluje na a straně serveru. Z tohoto důvodu zde není vhodné mít uložené jednotlivé komponenty a je lepší je mít v odděleném adresáři.
- **public** – Slouží k ukládání statických souborů jako jsou například obrázky, *manifest.json* pro PWA atd. V kódu je možné k těmto souborům přistoupit pomocí základní adresy, tedy například */logo.svg*.
- **queries** – Uchovává veškeré GraphQL dotazy a mutace, pomocí kterých klient komunikuje se serverem.

- **styles** – Ukládá hlavní kaskádové styly ve formátu *scss*. Tento adresář se dále dělí na složku *libs*, která obsahuje soubory vlastní vytvořené *scss* knihovny (jedná se například o třídy pro kontejnery, odsazování, zarovnávání apod.), složku *modules*, která obsahuje styly pro dané stejnojmenné podstránky. Dále obsahuje soubory *globals.scss*, *components.scss*, *animations.scss*, které obsahují globální styly, které je možné využít v rámci celé Next.js aplikace a soubor *variables.scss*, který ostatní souborům se styly poskytuje proměnné (např. barvy, velikosti fontů atd.).
- **utils** – Obsahuje soubory s pomocnými funkcemi včetně inicializace Apollo klienta či getter a setter JWT přístupového tokenu, který je z bezpečnostních důvodů po celou dobu chodu aplikace uložen právě pouze v paměti aplikace.
- **.eslintrc.json** – ESLint konfigurační soubor pro statickou analýzu kódu.
- **next-env.d.ts** – Soubor, který zajišťuje, že Next.js typy budou brány v potaz TypeScriptovým kompilátorem.
- **next.config.js** – Soubor pro vlastní konfiguraci Next.js. V tomto případě je využit primárně k implementaci PWA pomocí knihovny *next-pwa*.

## 6.2 Struktura databáze

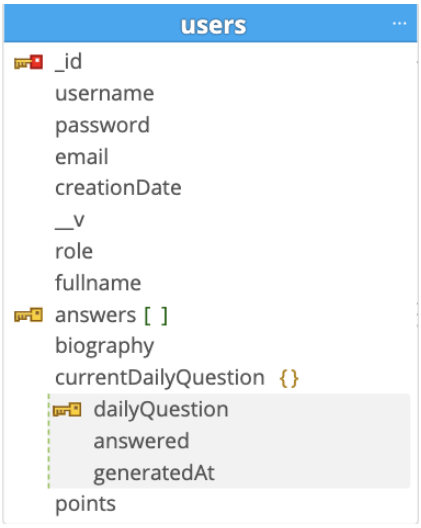
K uchovávání dat aplikace je použita nerelační dokumentová databáze MongoDB. V případě rostoucí uživatelské základny bude jednodušší databázi rozšířit díky možnosti horizontálního škálování a v případě budoucích úprav aplikace a datového modelu nebude potřeba provádět složité migrace (viz kapitola 3.4).

Vzhledem k faktu, že aplikace využívá objektově dokumentového mapování pomocí knihovny Mongoose, struktura databáze odpovídá datovému modelu. V této části bude více popsána struktura jednotlivých kolekcí.

### 6.2.1 Kolekce users

Kolekce *users* slouží k ukládání všech typů uživatelů. Tato kolekce uchovává přístupové údaje ve formě uživatelského jména a hashovaného hesla, email, datum vytvoření účtu, uživatelskou roli, celé jméno, pole identifikátorů odpovědí na otázky denního dotazníku, biografii a body ze vstupního testu. Dále obsahuje současnou otázku denního dotazníku, která je zde uložena jako objekt s identifikátorem dané otázky, polem pro identifikaci, jestli byla otázka již zodpovězena a datem vygenerování otázky.

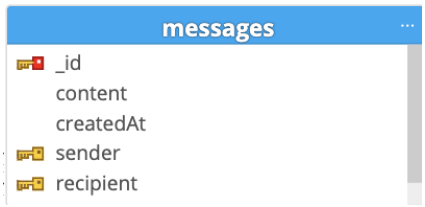
U této kolekce je vidět uplatněná výhoda flexibility dokumentových databází. Kolekce ukládá různé typy uživatelů i přes to, že mohou obsahovat jiná pole, a tedy se mohou strukturou lišit. Například uživatel typu vyhořelý oproti uživateli typu terapeut bude navíc obsahovat pole *points*, *answers* a *currentDailyQuestion*. Terapeut pak bude mít navíc pole *biography*, kterým naopak vyhořelý nedisponuje.




Obrázek 25 Kolekce users

### 6.2.2 Kolekce messages

Kolekce *messages* je určena pro ukládání zpráv z komunikačního rozhraní. Skládá se z obsahu zprávy, data vytvoření, identifikátoru odesílatele a příjemce.

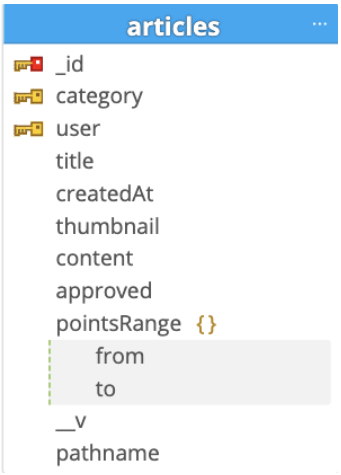



Obrázek 26 Kolekce messages

### 6.2.3 Kolekce articles

Kolekce *articles* zde slouží k uchovávání materiálů. Tato kolekce je složená z identifikátoru kategorie, identifikátoru uživatele, titulku, data vytvoření materiálu, náhledového obrázku, obsahu, informace o stavu materiálu (schváleno/neschváleno). Zajímavým polem této kolekce je bodový rozsah, zde nazvaný jako *pointsRange*. Bodový rozsah je realizovaný jako objekt s dvěma poli, a to *from* a *to*, tedy dolní a horní hranicí intervalu. Toto pole posléze umožňuje uživatelům v aplikaci filtrovat relevantní materiály podle výsledku vstupního

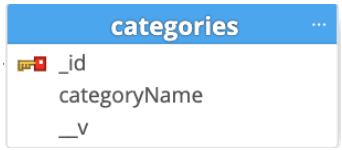
testu. Dále je zde pole *pathname*, které uchovává URL cestu vytvořenou z titulku odstraněním diakritiky, speciálních znaků a nahrazením mezer za pomlčky.




Obrázek 27 Kolekce articles

### 6.2.4 Kolekce categories

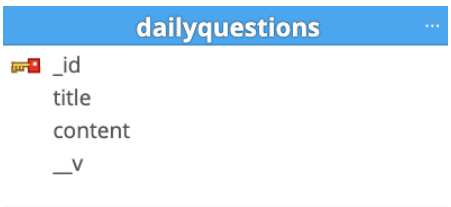
Kolekce *categories* je určená k ukládání kategorií materiálů. Kategorie obsahují pouze vlastní identifikátor a název.




Obrázek 28 Kolekce categories

### 6.2.5 Kolekce dailyquestions

Tato kolekce slouží k ukládání otázek denního dotazníku. Struktura denní otázky se skládá ze samotné otázky, zde označované jako *title* a doprovodného popisu této otázky, nazvaného jako *content*.




Obrázek 29 Kolekce dailyquestions

### 6.2.6 Kolekce dailyanswers

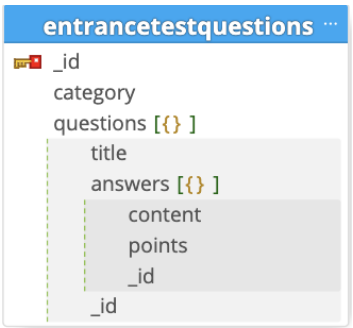
Odpovědi na dané denní otázky se pak ukládají v rámci kolekce *dailyanswers*. Odpovědi jsou pak s otázkou propojeny jejím identifikátorem. Odpověď dále také obsahuje

identifikátor uživatele, který tuto odpověď vytvořil, samotnou odpověď a datum jejího vytvoření.


Obrázek 30 Kolekce dailyanswers

### 6.2.7 Kolekce entrancetestquestions

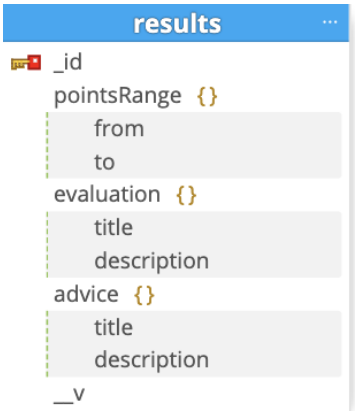
Otázky denního dotazníku jsou v databázi uloženy v kolekci *entrancetestquestions*, která je strukturovaná tak, aby co nejlépe vyhovovala potřebám interaktivní aplikace. Pro jednoduchou navigaci (předchozí / další) mezi kategoriemi denního dotazníku se tato kolekce skládá z názvu kategorie a pole daných otázek ve formě objektů. Vnořený objekt odpovědi se skládá z titulku otázky a možných odpovědí opět ve formě objektů. Objekt odpovědi pak obsahuje text odpovědi a bodovou hodnotu.




Obrázek 31 Kolekce entranacetestquestions

### 6.2.8 Kolekce results

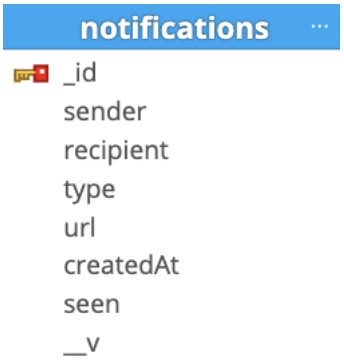
Různé výsledky pro vstupní test jsou ukládány v kolekci *results*. Podobně jako materiály z kolekce *articles* obsahuje i tato kolekce bodový rozsah podle kterého se po dokončení testu uživateli vybere příslušný dokument. Výsledek je dále složen z vyhodnocení a doporučení, kde jsou obě tato pole objekty, které obsahují titulek a popis.




Obrázek 32 Kolekce results

### 6.2.9 Kolekce notifications

Kolekce *notifications* slouží k uložení notifikací pro uživatele. Oproti například kolekci *messages* neobsahuje identifikátory uživatelů. Zde se totiž příjemce a odesílatel identifikuje pouze podle uživatelských jmen, protože není nutné získávat ostatní data. Dalšími poli této kolekce je typ notifikace (zpráva z komunikačního rozhraní, nový denní dotazník apod.), URL adresa, datum vytvoření a informace o tom, zda už uživatel notifikaci viděl.




Obrázek 33 Kolekce notifications

## 6.3 Komunikace klienta se serverem

Komunikace klientské částí s částí serverovou je realizována pomocí knihoven Apollo, konkrétně Apollo Server a Apollo Client. Na straně serveru běží vytvořené GraphQL API, které je dostupné na koncovém bodě */graphql*. Klientská část aplikace poté na tento koncový bod posílá jednotlivé požadavky ve formě mutací a dotazů, kdy dotaz slouží k získání dat a mutace k uložení, smazání či úpravě dat.

# 7 DŮLEŽITÉ ČÁSTI APLIKACE

Tato kapitola se bude zabývat popisem implementace důležitých částí aplikace. Vždy bude popsána jejich funkcionalita dané a zajímavá implementační část.

## 7.1 Úvodní test

První důležitou částí je úvodní test, který slouží k získání informace o současném psychickém stavu uživatele. Když uživatel dokončí test, budou mu sečteny body a vyhodnocen výsledek spolu s doporučením a nabídkou profesionálních terapeutů a registrace do aplikace.

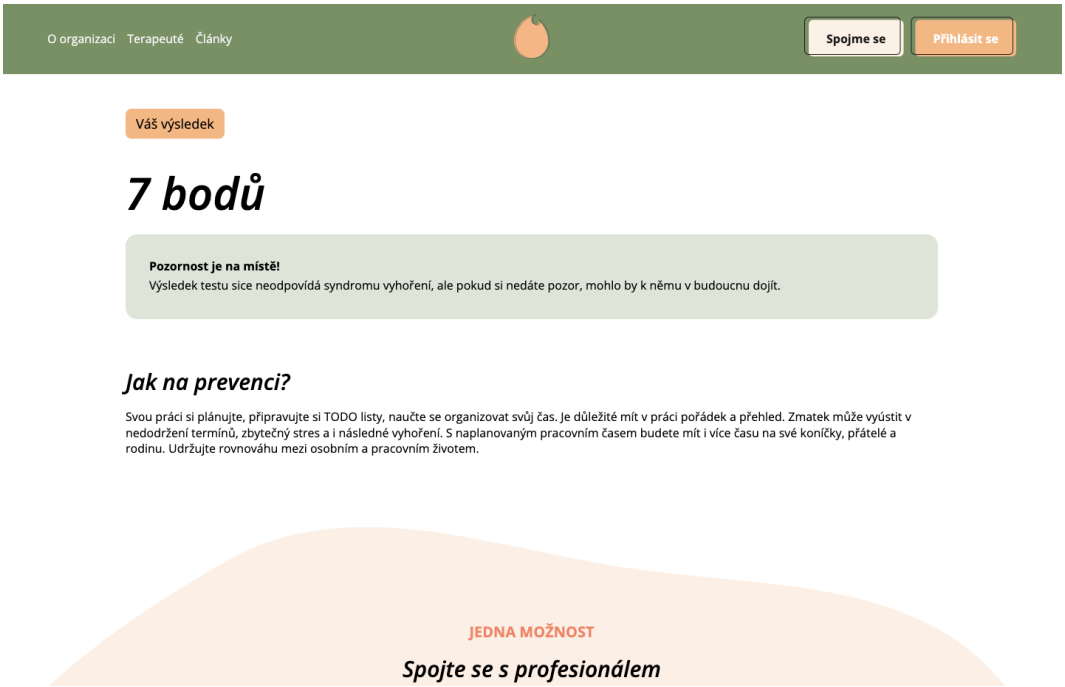
### 7.1.1 Popis funkcionality

Otázky vstupního testu jsou zobrazovány v jednotlivých kategoriích, jedna kategorie vždy znamená jeden krok. Až jsou všechny otázky v daném kroku vyplněny, je uživateli dovoleno přejít na další pomocí tlačítka „Další“, pokud se však chce vrátit k předchozímu kroku, může tak učinit pomocí tlačítka „Předchozí“.


Obrázek 34 Ukázka úvodního testu z aplikace

Na posledním kroku je tlačítko „Další“ vyměněno za tlačítko „Odeslat“. V momentě, kdy uživatel toto tlačítko stiskne, výsledek je uložen do lokální paměti prohlížeče a uživatel je přesměrován na stránku s výsledkem testu s URL parametrem obsahujícím získané body.

Při načítání výsledné stránky je zaslán GraphQL dotaz, který dle bodů z URL parametru na stranu klienta pošle data s výsledkem a doporučením. Na této stránce je dále zobrazena sekce s dostupnými terapeuty a sekce s popisem aplikace a nabídkou registrace. V případě, že si uživatel zvolí možnost registrace kliknutím na tlačítko „Vyzkoušet platformu“, je přesměrován na registrační stránku, kde je do databázové kolekce *users* po vyplnění údajů uložen spolu s přístupovými údaji i výsledek testu.



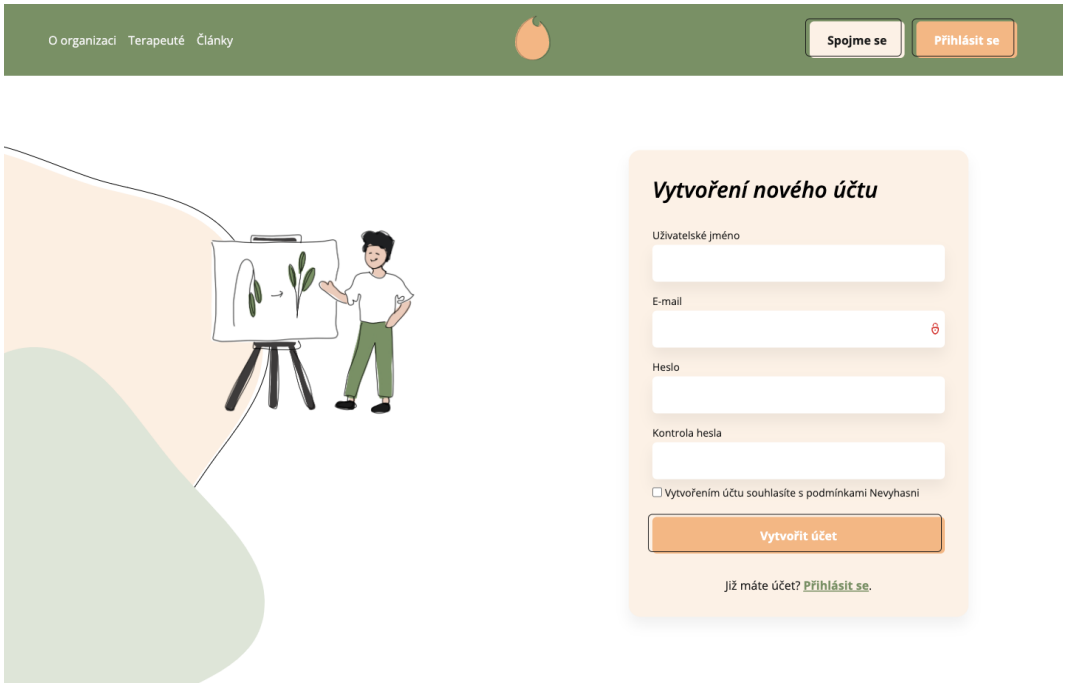

Obrázek 35 Ukázka výsledku testu z aplikace

## 7.2 Uživatelské účty

Aplikace disponuje uživatelskými účty s třemi možnými rolemi, a to rolí *VYHOŘELÝ*, *TERAPEUT* a *DOBROVOLNÍK*. Pokud se uživatel zaregistruje, je mu automaticky přiřazena role *VYHOŘELÝ*, ostatní role není možné zaregistrovat přímo, ale musí jej manuálně přiřadit administrátor aplikace.

### 7.2.1 Popis funkcionality

Pro přihlášení uživatele do aplikace je dostupný přihlašovací formulář. Pokud nepřihlášený uživatel ještě nemá účet, pod formulářem je odkaz s vybídkou pro registraci účtu, který po kliknutí vykreslí registrační formulář.




Obrázek 36 Ukázka registračního formuláře z aplikace

Když uživatel vyplní přihlašovací či registrační formulář, je přesměrován na hlavní stránku aplikace pro přihlášené uživatele, která je realizovaná ve formě nástěnky. Nástěnka a také ostatní stránky aplikace jsou přizpůsobeny přihlášenému uživateli dle jeho role. Všechny stránky pro přihlášené uživatele jsou vykreslovány spolu s postranním a horním panelem. Uživatelé role *VYHOŘELÝ* mají v postranním panelu k dispozici tři položky, a to nástěnku, rozhovory a materiály. Uživatelům spadajícím pod roli *DOBROVOLNÍK* či *TERAPEUT* je zde navíc zobrazována položka profil. Položka pro správu materiálů je dostupná pouze uživatelům role *TERAPEUT*.

## 7.3 Materiály

Další částí aplikace jsou odborné materiály ve formě článků, které si může zobrazit i nepřihlášený uživatel, avšak takový uživatel nemá možnost články filtrovat dle relevance. Poskytnutím možnosti zobrazení materiálů komukoliv se zvyšuje je dosah a SEO výkon aplikace, protože články budou zaindexovány vyhledávači jako je Google, Seznam atp.

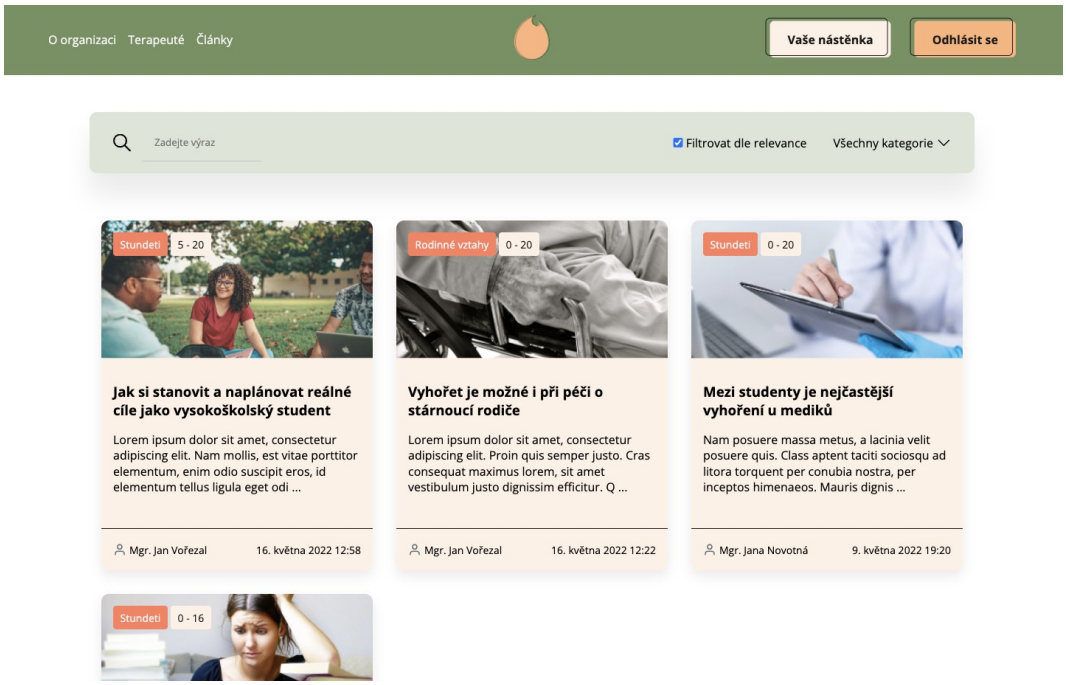
### 7.3.1 Popis funkcionality

Aplikace disponuje stránkou s výpisem materiálů, která mimo jiné nabízí i filtrační lištu, kde je možné filtrovat dle kategorie, filtrovat pouze relevantní materiály a vyhledávat dle

klíčových slov. Nejnovější materiály jsou také uživateli zobrazovány na uživatelské nástěnce.

Ve filtrační liště je defaultně zvolena možnost „Všechny kategorie“, která uživateli zobrazuje veškeré existující materiály. Pokud uživatel klikne na současně zvolenou kategorii v liště, otevře se rozbalovací menu s dalšími kategoriemi, které následně uživatel může vybrat a vyfiltrovat si tak výpis materiálů.

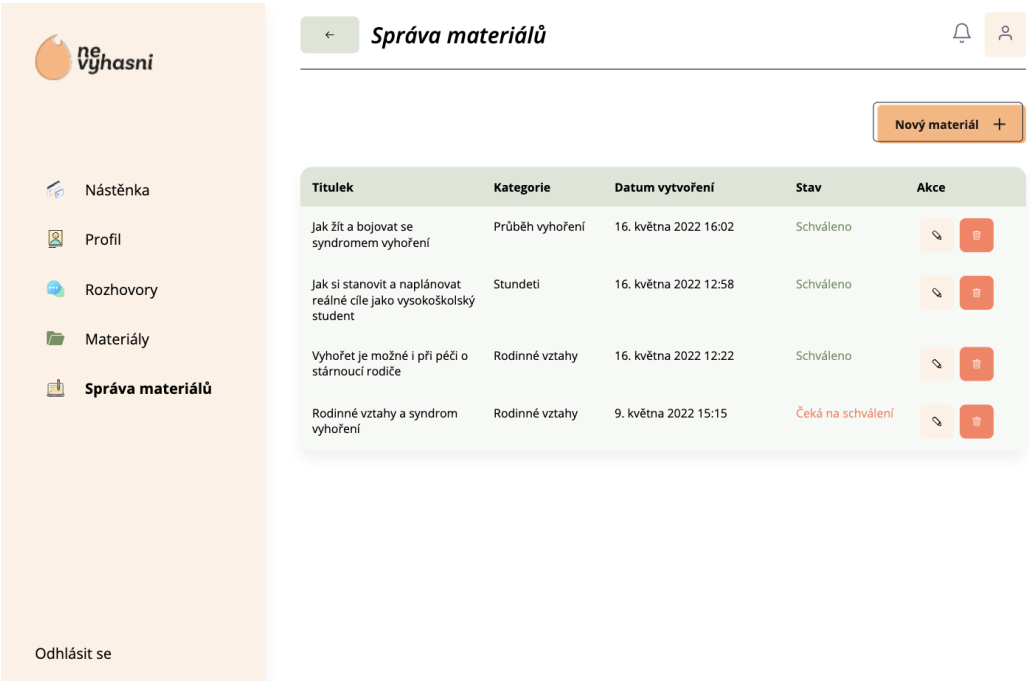
Vyhledávač umožní uživateli zadat hledaný výraz, který se poté hledá v titulku a obsahu článků. Při vyhledávání je ignorována diakritika a velká písmena.




Obrázek 37 Ukázka výpisu materiálů z aplikace

Každý materiál obsahuje rozsah bodů, který slouží pro filtraci dle relevance pro přihlášené uživatele. Pokud je uživatel přihlášen jako *VYHOŘELÝ*, má vyplněný vstupní test, a tedy i uložený bodový výsledek, je mu nabídnuta také možnost filtrování pouze pro něj relevantních materiálů pomocí zaškrtávacího pole v liště. Například pokud má uživatel 12 bodů, článek, který má rozsah 10–20 bodů, se mu bude zobrazovat i po filtraci, článek s rozsahem 0–10 bodů pak nikoliv. Pokud uživatel klikne na daný článek, bude přesměrován na stránku s článkem, která při požadavku pošle GraphQL dotaz na Apollo Server a získá tak požadovaná data článku, předkreslí je na straně Next.js serveru a zobrazí uživateli v prohlížeči.

Aplikace uživateli role *TERAPEUT* poskytuje stránku pro správu materiálů, kde mu je umožněno materiály vytvářet, upravovat a mazat. Ovšem pokud uživatel vytvoří nový materiál, bude nutné ho před publikováním manuálně schválit administrátorem aplikace. Na této stránce se nachází tabulka s výpisem jím vytvořených materiálů, která kromě dat z databáze obsahuje i funkční tlačítka pro úpravu a odstranění daného materiálu. Dále je zde tlačítko pro přidání nového materiálu s popisem „Přidat nový materiál“, které po kliknutí otevře pop-up okno s formulářem. Pokud uživatel klikne na tlačítko „Přidat nový materiál“, odešle se GraphQL mutace, která ověří totožnost uživatele, provede validaci zadaných dat, uloží nový záznam do databáze a vrátí zpět uživateli výsledek. Pokud ověření uživatele či validace selže, je ze serveru uživateli vrácena a následně zobrazena chybová hláška s výzva ke správnému vyplnění formuláře. V případě kliknutí na editační tlačítko daného materiálu je opět zobrazeno stejné pop-up okno, které ale obsahuje předvyplněné údaje již existujícího materiálu. Pokud uživatel článek upraví, bude nutné jej opět schválit. Stisknutí tlačítka pro smazání materiálu pak otevře pop-up okno s dotazem, zdali chce daný materiál uživatel opravdu smazat.




Obrázek 38 Ukázka stránky správy materiálů z aplikace

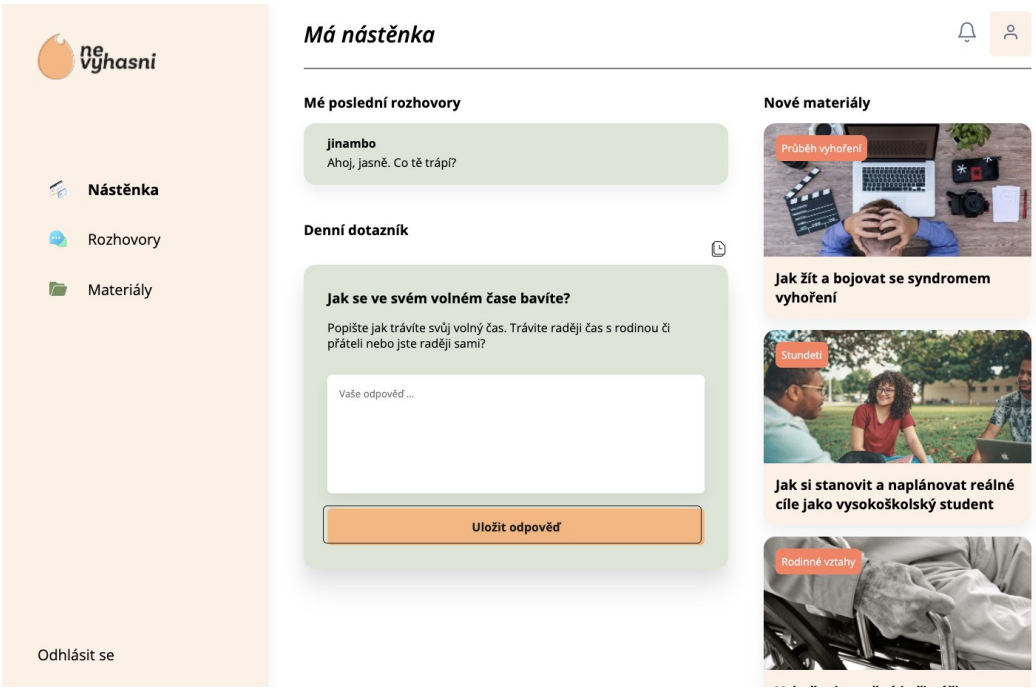
## 7.4 Denní dotazník

Další částí aplikace je denní dotazník ve formě deníku. Ten je dostupný pouze uživatelům role *VYHOŘELÝ*. Dotazník může uživatel každých 24 hodin vyplnit a také si zobrazit historii minulých odpovědí.

### 7.4.1 Popis funkcionality

Po prvním přihlášení uživatele do aplikace se odešle GraphQL mutace, která vybere náhodnou otázku z databáze, kterou spolu s datem vygenerování uloží k danému uživateli a následně ji zobrazí na hlavní stránce aplikace, tedy na uživatelovi nástěnce. Při každé další návštěvě se tato mutace opět odešle a zkontroluje, jestli od vygenerování otázky uplynulo 24 hodin. Pokud byla minulá otázka vygenerována před více než 24 hodinami, vygeneruje se otázka nová.

Denní dotazník je zobrazován na již zmiňované nástěnce ve formě okna s danou otázkou, jejím doplňkovým textem, vstupním polem a tlačítkem „Uložit odpověď“. Když uživatel dotazník vyplní a odešle, dotazník se zamkne na straně klienta i serveru a umožní uživateli odpovědět až na novou otázku. Nad oknem denního dotazníku je dostupná ikona, která po kliknutí otevře pop-up okno s historií odpovědí.




Obrázek 39 Ukázka nástěnky s denním dotazníkem z aplikace

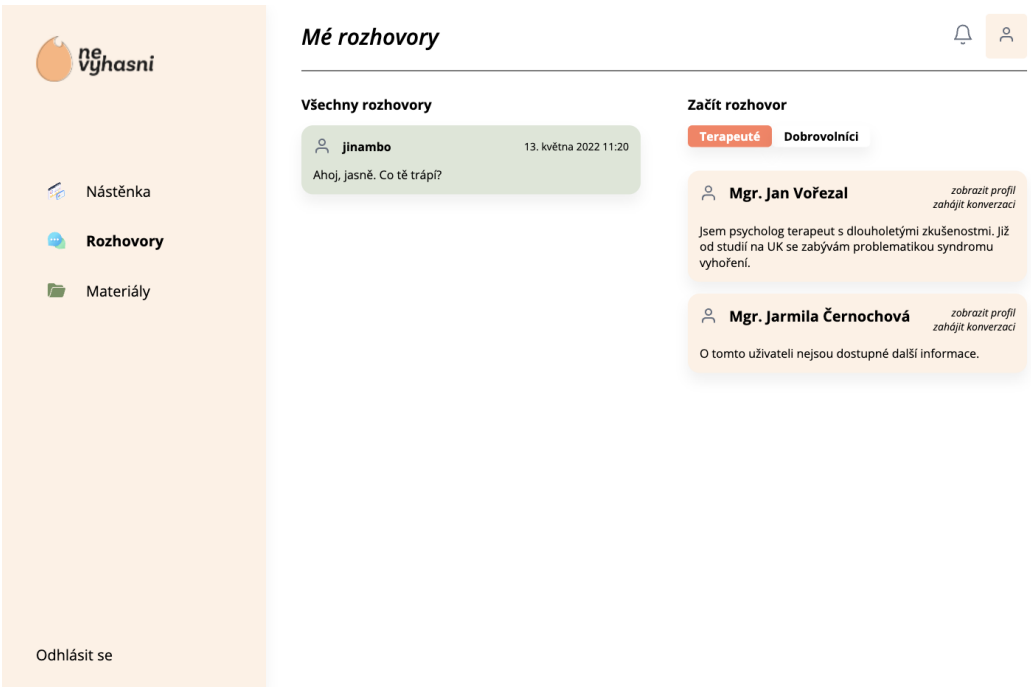
## 7.5 Komunikační rozhraní

Funkcí, která zastupuje základní myšlenku této aplikace, a to propojení lidí, kteří trpí syndromem vyhoření nebo z něj mají obavy, se zkušenými lidmi, kteří si vyhořením v minulosti již prošli nebo s profesionálními terapeuty, je komunikační rozhraní. Komunikační rozhraní je dostupné všem uživatelským rolím s tím, že uživatel role

*VYHOŘELÝ* může komunikovat pouze s uživateli role *TERAPEUT* a *DOBROVOLNÍK* a naopak.

### 7.5.1 Popis funkcionality

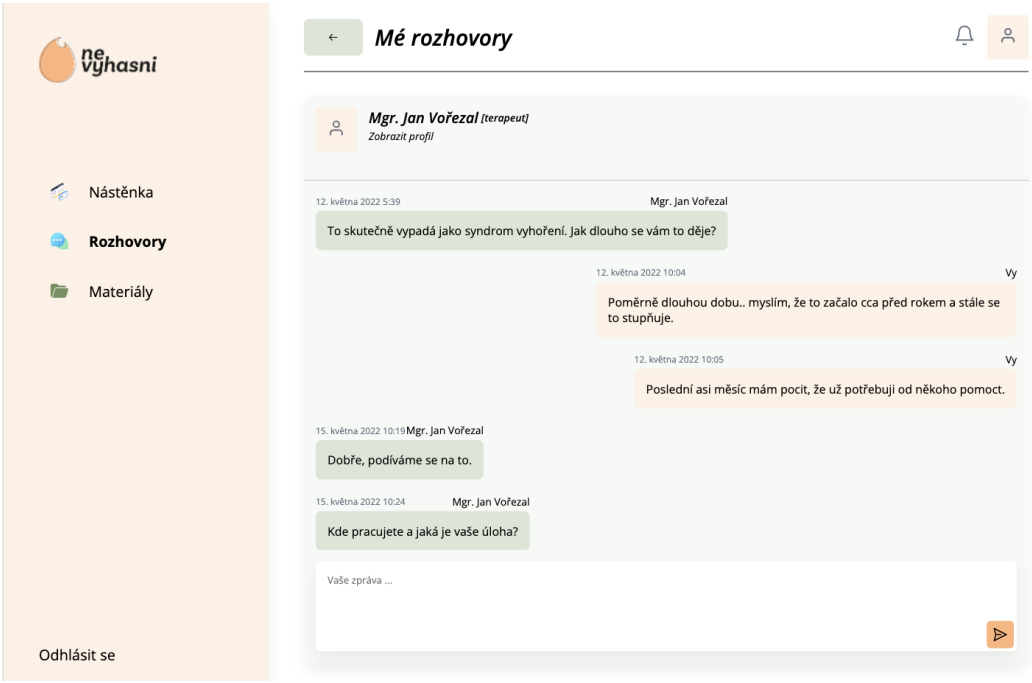
Aplikace poskytuje všem přihlášeným uživatelům stránku s rozhovory, kde se nachází výpis všech posledních zpráv od odesílatelů a nabídka uživatelů ke konverzaci, kteří jsou zde rozděleny podle uživatelské role. U každého uživatele v nabídce jsou k dispozici dva odkazy. Odkaz „zobrazit profil“ uživatele po kliknutí přesměruje na profil daného terapeuta či dobrovolníka a odkaz „zahájit konverzaci“ otevře okno komunikačního rozhraní s tímto uživatelem. V případě, že uživatel klikne na danou zprávu z výpisu, opět bude otevřeno okno komunikačního rozhraní. Výpis posledních rozhovorů je také zobrazován na nástěnce uživatele.




Obrázek 40 Ukázka stránky s rozhovory z aplikace

Okno komunikačního rozhraní se skládá z lišty, která obsahuje základní údaje o daném uživateli s možností přesměrování na jeho profil, těla okna, ve kterém jsou zobrazeny jednotlivé zprávy a vstupního pole s tlačítkem pro odeslání nové zprávy. Když uživatel odešle zprávu, na server se ze strany klienta pošle GraphQL mutace, která novou zprávu uloží do databáze. Oba účastníci konverzace pak také v krátkém čase obdrží aktualizované zprávy zobrazené v komunikačním rozhraní. Pokud daná konverzace již obsahuje více

zpráv, komunikační okno se samo automaticky při každé nové zprávě posouvá dolů k nejnovější zprávě.




Obrázek 41 Ukázka komunikačního okna z aplikace

# 8 NASAZENÍ STANDARDU PWA

Vytvořená aplikace dle Lighthouse, nástroje od společnosti Google, splňuje kritéria progresivních webových aplikací. V této kapitole bude popsán způsob, jakým toho bylo dosaženo.

## 8.1 Responsivní rozvržení aplikace

Důležitým aspektem PWA je responsivita. Rozhraní aplikace by se tedy mělo přizpůsobovat velikosti daného zařízení.

Pro práci s rozložením prvků uživatelského rozhraní byla v rámci vývoje vytvořena jednoduchá SCSS knihovna inspirována populární knihovnou Bootstrap. Knihovna poskytuje třídy pro kontejnery, sloupce, řádky, odsazení atd. Příklad použití bude demonstrován na následujícím kusu kódu z aplikace.

```
<main className="full overflow-scroll col-sm-9 col-12 p-x-3-sm p-x-1 p-b-2">
   <Navigation title={ title } canGoBack={ canGoBack } />
   <div className="m-t-2">
      { children }
   </div>
</main>
```

Kód 25 Ukázka využití vytvořené knihovny

Element *main* obsahuje hned několik různých tříd. Důležitými pro demonstraci jsou však třídy *col-sm-9* a *col-12*, které zastupují již zmiňované sloupce. Knihovna je nastavena tak, aby šířku řádku rozdělila na 12 sloupců, pokud má tedy element třídu *col-12*, je široký jako celý jeho rodičovský element. Navíc jsou zde dostupné třídy s tzv. breakpointy, tedy s body, které definují CSS vlastnosti pro různé velikosti zařízení. Třída *col-sm-9* říká, že pokud bude šířka zařízení větší než definovaný breakpoint, v názvu třídy označený *sm*, element bude široký jako 9 pomyslných sloupců, jinak se aplikuje třída *col-12* a element tedy bude široká jako 12 sloupců.

Na některé větší úpravy uživatelského rozhraní ale tyto funkce knihovny nestačily a bylo je nutné definovat v SCSS souboru dané komponenty či stránky.

## 8.2 Implementace souboru manifest.json

Dalším důležitým aspektem pro vytvoření PWA je implementace souboru manifest.json, který obsahuje metadata aplikace jako je její název, popis, ikony atd. V klientské části aplikace je soubor uložen v adresáři *public*, který slouží k ukládání statických souborů.

Soubor je poté do aplikace nalinkován v hlavičce speciálního souboru *_document.tsx* poskytovaného frameworkem Next.js. Implementací souboru manifest.json bylo dosaženo možnosti nainstalovat aplikaci přímo do zařízení.

## 8.3 Implementace service workeru

Vytvořená aplikace ukládá svůj obsah do cache paměti a v případě, že se uživatel pokusí danou stránku navštívit bez připojení k internetu, je mu z cache paměti naservírována. Pokud uživatel danou stránku uloženou nemá, je mu zobrazena stránka s informací, že není připojen k internetu. Implementace service workeru je zajištěna pomocí pluginu next-pwa, který byl popsán v rámci teoretické části.

Nainstalovaný balík next-pwa je pro implementaci nejprve nutné importovat v konfiguračním souboru next.config.js a nastavit dle vlastních požadavků. Způsob implementace v aplikaci je blíže vidět v kódu níže.

```
const withPWA = require('next-pwa')
const runtimeCaching = require("next-pwa/cache")
module.exports = withPWA({
  reactStrictMode: true,
  pwa: {
    dest: "public",
    runtimeCaching,
    register: true,
    disable: process.env.NODE_ENV == 'development'
  },
  env: {
    API_URL: process.env.API_URL,
  }
})
```

Kód 26 Ukázka implementace balíčku next-pwa v souboru next.config.js

Nastavení se poté exportuje jako javascriptový modul, který je zabalený do funkce withPWA. Jedná se o React vzor HOC (High-Order Component), který v podstatě vezme danou komponentu a vrátí ji rozšířenou o dané funkce. V tomto případě o funkce balíku next-pwa. Po konfiguraci je nutné sestavit aplikaci pomocí příkazu *npm run build*, kdy se vygeneruje příslušný service worker.

Stránka, která je uživateli servírována service workerem, pokud nemá přístup k internetu a nemá dotazovanou stránku uloženou v cache paměti, je vytvořena v souboru *_offline.tsx*.

# 9 NASAZENÍ NA SERVER A ZÁKLADNÍ ZABEZPEČENÍ

Tato kapitola se bude zabývat nasazením aplikace na server a také základním zabezpečením aplikace.

## 9.1 Nasazení aplikace na server

Vzhledem k tomu, že projekt není obsahově plně připraven a stále je v procesu přípravy dat, není možné aplikaci nasadit na produkční server. Projekt je však nasazen na testovacím serveru, který produkční server simuluje.

Vytvořená aplikace byla nasazena na cloud platformu Heroku, která umožnila rychlé spuštění aplikace bez nutnosti složitých konfigurací serveru.

Nasazení na Heroku předcházelo nastavení npm skriptů pro sestavování a spouštění produkčních verzí klientské a serverové části aplikace. Toto nastavení se realizuje v souboru *pacakage.json*.

```
"scripts": {
  "start": "node dist/app.js",
  "dev": "nodemon src/app.ts",
  "build": "tsc -p ."
},
```

Kód 27 Package.json soubor serverové části aplikace

V serverové části bylo také ještě potřeba upravit pevně definovaný port na lokální proměnou, kterou Heroku automaticky vytvoří.

Heroku při nasazení aplikaci nejprve sestaví pomocí příkazu *npm run build*, který spouští skript definovaný právě v souboru *package.json* pod položkou *build*. Poté aplikaci spouští v produkční verzi pomocí příkazu *npm run start*.

Pro účely nasazení byly všechny soubory a složky mimo složku *client* přesunuty do nové složky pojmenované *server*.

## 9.2 Základní zabezpečení aplikace

Vytvořená aplikace se skládá ze dvou částí, první část poskytuje GraphQL API, které zajišťuje manipulaci s daty a jejich přenos mezi klientem a databází. V aplikaci disponují uživatelé různých rolí, a proto je velmi důležité mít vytvořené API řádně zabezpečené a uživatele autorizovat.

Realizace autorizace je řešena pomocí dvou JSON Web tokenů, a to přístupovým tokenem a obnovovacím tokenem. Jak již bylo více popsáno v teoretické části (kapitola 3.6), tyto tokeny se skládají z hlavičky, dat a podpisu.

Je velmi důležité ukládat přístupové tokeny na bezpečné místo, tak aby nebyly přístupné útočníkům. Lokální paměť prohlížeče či klientské cookies nejsou vhodnou volbou pro uložení tokenu, protože k nim má přístup jakýkoliv skript vložený útočníkem do prohlížeče oběti. Nejlepším řešením se v současnosti zdá být uložení tokenu do proměnné v klientské části aplikace. Zde by ale mohl nastat problém v momentě, kdy uživatel aplikaci načte znovu a přijde tak o svůj token a nezbude mu nic jiného, než se znovu přihlásit a vygenerovat tak token nový. K řešení tohoto problému byl v aplikaci implementován již zmiňovaný obnovovací token. Tento token je uložený v serverové cookie a v případě obnovení stránky vygeneruje nový přístupový token, který je opět uložen v proměnné. V ideálním případě by měl mít přístupový token velmi malou trvanlivost, aby jej v případě odcizení neměl útočník k dispozici dlouhou dobu. V aplikaci je doba vypršení přístupového tokenu nastavena na 30 vteřin. [57]

Při přihlášení či registraci se na straně serveru vytvoří nový podepsaný obnovovací token. V rámci implementace Apollo klienta, který zajišťuje komunikaci s GraphQL serverem je vytvořen tzv. TokenRefreshLink, který posílá požadavek s obnovovacím tokenem v cookie na získání přístupového tokenu. Pokud přístupový token vyprší či je stránka obnovena, pošle požadavek znovu. Každý požadavek přihlášeného uživatele na API má pak v hlavičce uložený přístupový token a před obsluhou dané mutace se nejprve ověří.

```
export const createToken = user => {
   return jwt.sign({
      id: user.id,
      email: user.email,
      username: user.username,
      role: user.role
   }, JWT_KEY, { expiresIn: '30s' })
}
```

Kód 28 Ukázka vytvoření přístupového tokenu

Samozřejmostí je pak také fakt, že aplikace při registraci uživatele jeho heslo před uložením do databáze hashuje. Pokud dojde k úniku dat z databáze, hesla uživatelů jsou stále v relativním bezpečí.

# 10 BUDOUCÍ VÝVOJ APLIKACE

Projekt je teprve ve svých začátcích a do budoucna se zcela jistě budou přidávat nové funkce, či upravovat funkce stávající. V této krátké kapitole budou popsány funkce, které již byly konzultovány a do budoucna se chystá jejich implementace. Také zde budou zmíněny funkce, které jsou již v rámci této bakalářské práce navrženy, ale kvůli obsáhlosti nebo nedokončené domluvě o přesné funkcionalitě zatím nebyly implementovány.

Jednou z funkcí, která je zde již zmíněna v rámci návrhu, ale není implementována v kódu aplikace je zasílání notifikací. Nicméně je v kódu připraven model notifikace a také funkční vzhled notifikačního okna. Notifikace v budoucnu určitě implementovány, protože jsou důležitou součástí aplikace pro udržení si stálých uživatelů. Notifikace, které se budou zobrazovat přímo v aplikaci budou asociovány s push notifikacemi zasílanými service workerem. Problémem může být fakt, že push notifikace dosud nebyly implementovány pro mobilní operační sysém iOS a tak aplikace může přijít o pozornost uživatelů, kteří používají mobilní zařízení Apple.

Další funkcí, která byla zmíněna, ale nebyla implementována je odesílání mailů se souhrnem odpovědí z denního dotazníku daného uživatele terapeutovi. Je ještě potřeba toto konzultovat se zainteresovanými terapeuty a najít způsob, který bude vyhovovat oběma účastníkům této interakce.

Dále není implementována možnost zaslat žádost o zprávu terapeutovi či dobrovolníkovi. V budoucnu dobré tuto funkci mít, protože může uživateli ulehčit pochyby, zda terapeuta kontaktovat či nikoliv.

Plánovanou funkcí, která v návrhu není, ale v současnosti je konzultována, je vytvoření chatbota. Po konzultaci s lidmi, kteří si prošli těžkým syndromem vyhoření bylo zjištěno, že chatbot může být ideálním pomocníkem pro lidi v této situaci. Lidé totiž potřebují, aby někdo reagoval rychle, když jim je nejhůře.

Také je plánována funkce skupinové konverzace, která by všem uživatelům umožnila komunikovat mezi sebou. Zatím ale není jasné, jestli by tato funkce přinesla uživatelům výhody.

V budoucnu se také bude vylepšovat design aplikace a přizpůsobovat uživatelská zkušenost potřebám uživatelů, tak aby pro ně aplikace byla co nejpříjemnější.

# ZÁVĚR

Tato práce se primárně zabývala návrhem a vývojem progresivní webové aplikace pro boj se syndromem vyhoření. Kde je hlavní myšlenkou propojení lidí, kteří trpí syndromem vyhoření anebo na něj mají podezření, s profesionálními terapeuty a dobrovolníky, kteří již se syndromem vyhoření mají vlastní zkušenost.

Klíčovým výstupem této práce je funkční prototyp progresivní webové aplikace pojmenované Nevyhasni, na který se v budoucnu bude navazovat vylepšováním stávajících a přidáváním nových funkcí.

Teoretická část zpočátku uvádí do problematiky progresivních webových aplikací, popisuje jejich vlastnosti a technologie využívané k jejich implementaci a předvádí úspěšné progresivní webové aplikace velkých firem. Dále jsou v rámci teoretické části popsány technologie, které jsou využité k praktické implementaci aplikace.

Praktická část začíná popisem obecného zadání projektu, které poté slouží jako základní podnět pro definování funkčních a nefunkčních požadavků. Po definici požadavků následuje návrh aplikace, kde je nejprve vizuálně zobrazena informační architektura, která je v další části rozšířená o ukázky navrženého drátěného module s popisem jednotlivých prvků. Praktická část dále pokračuje popisem datového modelu. Poté jsou popsáni jednotliví aktéři aplikace, diagram případů užití a navazující uživatelské scénáře.

Další část je zaměřená na popis výsledné aplikace, kde je nejprve pro přehled rozebrána adresářová struktura klientské a serverové části aplikace a struktura databáze. Následně praktická část popisuje důležité části aplikace a příklad jejich implementace a také způsob nasazení standardu PWA.

Konec praktické části je věnován nasazení aplikace na testovací server, který simuluje chod produkčního serveru a základnímu zabezpečení aplikace, kde je popsán způsob zajištění bezpečné autorizace.

Práce je také doplněna o kapitolu, která pojednává o budoucích plánech rozšíření vytvořené aplikace.

# SEZNAM POUŽITÉ LITERATURY

[1] Introduction to progressive web apps. *Mozilla* [online]. 2022 [cit. 2022-02-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#discoverability

[2] Why should you care about Progressive Web Apps (PWAs)?. *Codeburst* [online]. 2018 [cit. 2022-02-10]. Dostupné z: https://codeburst.io/why-should-you-care-about-progressive-web-apps-pwas-3c3f73cb8c92

[3] What are Progressive Web Apps?. *Web.dev* [online]. 2020 [cit. 2022-02-10]. Dostupné z: https://web.dev/what-are-pwas/

[4] Progresivní webové aplikace: Co to je? A jak webu zařídit plné hodnocení PWA v Lighthouse. *Vzhurudolu* [online]. 2019 [cit. 2022-02-10]. Dostupné z: https://www.vzhurudolu.cz/prirucka/pwa

[5] The history of PWA development. *Divante* [online]. [cit. 2022-02-11]. Dostupné z: https://www.divante.com/pwabook/chapter/02-the-history-of-pwas#:~:text=The%20idea%20of%20a%20PWA,the%20iPhone%20introduction%20in%202007

[6] Responsive design. *Mozilla* [online]. 2022 [cit. 2022-02-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

[7] Best PWA examples. Success stories of Progressive Web Apps. *Asperbrothers* [online]. 2019 [cit. 2022-02-11]. Dostupné z: https://asperbrothers.com/blog/best-pwa-examples/

[8] Twitter. In: *Wikipedia: the free encyclopedia* [online]. 2022 [cit. 2022-02-11]. Dostupné z: https://en.wikipedia.org/wiki/Twitter

[9] Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage. *Google* [online]. 2017 [cit. 2022-02-11]. Dostupné z: https://developers.google.com/web/showcase/2017/twitter

[10] Progressive Web Apps: Escaping Tabs Without Losing Our Soul. *Infrequently* [online]. 2015 [cit. 2022-02-12]. Dostupné z: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

[11] SERVICE WORKER: OFFLINE STRÁNKY A PUSH NOTIFIKACE. *Nothrem* [online]. 2019 [cit. 2022-02-13]. Dostupné z: http://js.nothrem.cz/service-worker/

[12] Proč a jak psát progresivní webové aplikace. *Ackee* [online]. 2017 [cit. 2022-02-13]. Dostupné z: https://www.ackee.cz/blog/proc-a-jak-psat-progresivni-webove-aplikace/

[13] What is NoSQL?. *Mongodb* [online]. 2022 [cit. 2022-02-13]. Dostupné z: https://www.mongodb.com/nosql-explained

[14] Relační databáze vs. nerelační databáze: jaké jsou mezi nimi rozdíly?. *Master* [online]. 2022 [cit. 2022-02-16]. Dostupné z: https://www.master.cz/blog/relacni-databaze-nerelacni-databaze-jake-jsou-rozdily/

[15] Understanding the Different Types of NoSQL Databases. *Mongodb* [online]. 2022 [cit. 2022-02-16]. Dostupné z: https://www.mongodb.com/scale/types-of-nosql-databases

[16] What is a Document Database?. *Mongodb* [online]. 2022 [cit. 2022-02-16]. Dostupné z: https://www.mongodb.com/document-databases

[17] Graph database. In: *Wikipedia* [online]. 2022 [cit. 2022-02-19]. Dostupné z: https://en.wikipedia.org/wiki/Graph_database

[18] About JavaScript. *Mozilla* [online]. 2022 [cit. 2022-02-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[19] JavaScript: Prototypové (prototype-based). In: *Wikipedia* [online]. 2022 [cit. 2022-02-19]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript#Prototypov%C3%A9_(prototype-based)

[20] PEHLIVANIAN, Ara a Don NGUYEN. JavaScript okamžitě. Brno: Computer Press, 2014. ISBN 978-80-251-4163-2.

[21] First-class Function. *Mozilla* [online]. 2022 [cit. 2022-02-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function

[22] Source-to-source compiler. In: *Wikipedia* [online]. 2022 [cit. 2022-02-22]. Dostupné z: https://en.wikipedia.org/wiki/Source-to-source_compiler

[23] WIEGERS, Karl a Joy BEATTY. Software Requirements (Developer Best Practices). 3rd edition. Microsoft Press, 2013. ISBN 978-0735679665.

[24] What Is the Cost of a Requirement Error?. *Stickyminds* [online]. 2007 [cit. 2022-02-24]. Dostupné z: https://www.stickyminds.com/article/what-cost-requirement-error

[25] Introduction to Node.js. *Nodejs* [online]. [cit. 2022-02-25]. Dostupné z: https://nodejs.dev/learn

[26] A brief history of Node.js. *Nodejs* [online]. [cit. 2022-02-25]. Dostupné z: https://nodejs.dev/learn/a-brief-history-of-nodejs

[27] Node.js – s JavaScriptem na server. *Zdrojak* [online]. 2010 [cit. 2022-02-26]. Dostupné z: https://zdrojak.cz/clanky/node-js-s-javascriptem-na-server/

[28] PROČ K VÝVOJI WEBOVÝCH APLIKACÍ POUŽÍT TECHNOLOGII NODEJS?. *Rascasone* [online]. 2021 [cit. 2022-02-26]. Dostupné z: https://www.rascasone.com/cs/blog/node-js-architektura-moduly-npm

[29] Lekce 1 - Úvod do Node.js *Itnetwork* [online]. 2018 [cit. 2022-02-27]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs

[30] What is Next.js?. *Nextjs* [online]. 2022 [cit. 2022-03-02]. Dostupné z: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs

[31] How Next.js Works. *Nextjs* [online]. 2022 [cit. 2022-03-02]. Dostupné z: https://nextjs.org/learn/foundations/how-nextjs-works/rendering

[32] Dealing with SEO in Next.js. *Datocms* [online]. 2021 [cit. 2022-03-02]. Dostupné z: https://www.datocms.com/blog/dealing-with-nextjs-seo

[33] How Next.js Works. *Nextjs* [online]. 2022 [cit. 2022-03-05]. Dostupné z: https://nextjs.org/learn/foundations/how-nextjs-works/cdns-and-edge

[34] GraphQL is the better REST. *Howtographql* [online]. [cit. 2022-03-05]. Dostupné z: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

[35] Schemas and Types. *Howtographql* [online]. [cit. 2022-03-09]. Dostupné z: https://graphql.org/learn/schema/

[36] GraphQL schema basics. *Apollographql* [online]. [cit. 2022-03-10]. Dostupné z: https://www.apollographql.com/docs/apollo-server/schema/schema/

[37] How a GraphQL query fetches data. *Apollographql* [online]. [cit. 2022-03-12]. Dostupné z: https://www.apollographql.com/tutorials/fullstack-quickstart/writing-query-resolvers

[38] The Apollo Graph Platform: Bring your graph from prototype to production. *Apollographql* [online]. [cit. 2022-03-13]. Dostupné z: https://www.apollographql.com/docs/intro/platform/

[39] Introduction to JSON Web Tokens. *Jwt* [online]. [cit. 2022-03-14]. Dostupné z: https://jwt.io/introduction

[40] About npm. *Npmjs* [online]. [cit. 2022-03-16]. Dostupné z: https://docs.npmjs.com/about-npm

[41] NPM: Průvodce začátky a základními příkazy. *Vzhurudolu* [online]. 2018 [cit. 2022-03-18]. Dostupné z: https://www.vzhurudolu.cz/prirucka/npm

[42] GetServerSideProps. *Nextjs* [online]. [cit. 2022-03-19]. Dostupné z: https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props

[43] Introduction to Apollo Server. *Apollographql* [online]. [cit. 2022-03-19]. Dostupné z: https://www.apollographql.com/docs/apollo-server/

[44] CSS preprocessor. *Mozilla* [online]. [cit. 2022-03-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor

[45] MongoDB & Mongoose: Compatibility and Comparison. *Mongodb* [online]. 2022 [cit. 2022-03-23]. Dostupné z: https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/

[46] Objektově relační mapování. In: *Wikipedia* [online]. 2022 [cit. 2022-03-25]. Dostupné z: https://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD

[47] Metodiky pro organizaci CSS kódu: Pište styly bez bolehlavů. *Vzhurudolu* [online]. 2020 [cit. 2022-03-26]. Dostupné z: https://www.vzhurudolu.cz/prirucka/css-metodiky

[48] BEM: Pojmenovávací konvence pro třídy v CSS. *Vzhurudolu* [online]. 2017 [cit. 2022-03-28]. Dostupné z: https://www.vzhurudolu.cz/prirucka/bem

[49] Why Use MongoDB and When to Use It?. *Mongodb* [online]. 2022 [cit. 2022-03-28]. Dostupné z: https://www.mongodb.com/why-use-mongodb

[50] FIRESHIP. MongoDB in 100 Seconds. In: *YouTube* [online]. 14. 07. 2021 [cit. 2022-03-29]. Dostupné z: https://www.youtube.com/watch?v=-bt_y4Loofg&ab_channel=Fireship

[51] BE A BETTER DEV. System Design: What is Horizontal vs Vertical Scaling?. In: *YouTube* [online]. 03. 06. 2020 [cit. 2022-04-02]. Dostupné z: https://www.youtube.com/watch?v=p1YQU5sEz4g&ab_channel=BeABetterDev

[52] How to Scale MongoDB. *Mongodb* [online]. 2022 [cit. 2022-04-04]. Dostupné z: https://www.mongodb.com/basics/scaling

[53] MongoDB Replication. *Mongodb* [online]. 2022 [cit. 2022-04-04]. Dostupné z: https://www.mongodb.com/basics/replication

[54] What is Express.js?. *Codecademy* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.codecademy.com/article/what-is-express-js

[55] Frameworks built on Express. *Expressjs* [online]. 2017 [cit. 2022-04-08]. Dostupné z: https://expressjs.com/en/resources/frameworks.html

[56] Zero Config PWA Plugin for Next.js. In: Github [online]. 2022 [cit. 2022-04-11]. Dostupné z: https://github.com/shadowwalker/next-pwa#readme

[57] The Ultimate Guide to handling JWTs on frontend clients (GraphQL). Hasura [online]. 2022 [cit. 2022-04-11]. Dostupné z: https://hasura.io/blog/best-practices-of-using-jwt-with-graphql

[58] PĚTINA ČECHŮ SE POTÝKÁ SE SYNDROMEM VYHOŘENÍ. 1. lékařská fakulta Univerzity Karlovy [online]. 2018 [cit. 2022-04-20]. Dostupné z: https://www.lf1.cuni.cz/petina-cechu-se-potyka-se-syndromem-vyhoreni

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| SMS | Short Message Service |
| PWA | Progressive Web Apps |
| AJAX | Asynchronous JavaScript and XML |
| SDK | Software development kit |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| W3C | World Wide Web Consortium |
| HTML | HyperText Markup Language |
| JSON | JavaScript Object Notation |
| BSON | Binary Javascript Object Notation |
| XML | Extensible Markup Language |
| SQL | Structured Query Language |
| API | Application Programming Interface |
| OOP | Object-oriented programming |
| PHP | PHP: Hypertext Preprocessor |
| JSX | JavaScript Syntax Extension |
| SEO | Search engine optimization |
| CSR | Client-Side Rendering |
| SSR | Server-Side Rendering |
| SSG | Static Site Generation |
| CDN | Content delivery network |
| CSS | Cascading Style Sheets |
| ORM | Object–Relational Mapping |
| ODM | Object Document Mapping |
| SASS | Syntactically Awesome Style Sheets |

| | |
|---|---|
| LESS | Leaner Style Sheets |
| OOCSS | Object-oriented CSS |
| BEM | Block Element Modifier |
| JWT | JSON Web Token |
| SHA256 | Secure Hash Algorithm 256 |
| UML | Unified Modeling Language |
| HOC | Higher-Order Components |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: PŘILOŽENÉ CD

Obsah CD přiloženého k bakalářské práci:

- bakalářskou práci
- zdrojové kódy vytvořené aplikace
- návrh aplikace: drátěný model, diagram případů užití a datový model