Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Webová aplikace pro stimulaci akustickými podněty k terapeutickému využití

Plzeň 2023 Bc. Daniel Wébr

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Daniel WÉBR**
Osobní číslo:	**A21N0078P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Webová aplikace pro stimulaci akustickými podněty k terapeutickému využití**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte problematiku terapeutického využití stimulace akustickým tónovým podnětem.
2. Seznamte se s vhodnými technologiemi pro vývoj a nasazení webové aplikace.
3. Dle požadavků terapeuta navrhněte webovou aplikaci, která umožní spravovat, upravovat, definovat, přehrávat akustické podněty vytvořené terapeutem.
4. S pomocí vhodně vybraných technologií aplikaci implementujte.
5. Výsledné řešení důkladně otestujte a zhodnoťte dosažené výsledky práce.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce.

Vedoucí diplomové práce: **Ing. Petr Brůha**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2023

Bc. Daniel Wébr

Abstract

This master thesis focuses on the development of a web application for the therapeutic use of acoustic stimulation, which allows the creation of sound therapy sessions according to the individual needs of clients. The theoretical part of the thesis deals with the concept of frequency therapy, analysis of the web environment and an overview of available technologies and methods for creating an effective web application. The practical part includes a description of the technological decisions and the design of the solution, which was developed in collaboration with an expert therapist. Furthermore, the thesis discusses the implementation and deployment of the application, including established DevOps practices. Finally, the thesis discusses the validation of the application through automated tests and user testing.

Key words: web application, frequency therapy, music therapy, therapeutic forks, Next.js, React.js, cloud services

Abstrakt

Tato diplomová práce se zaměřuje na vývoj webové aplikace pro terapeutické využití stimulace akustických podnětů, která umožňuje vytváření zvukových terapií dle individuálních potřeb jednotlivých klientů. V teoretické části se práce věnuje konceptu frekvenční terapie, analýze webového prostředí a přehledu dostupných technologií a metod pro vytvoření efektivní webové aplikace. Praktická část zahrnuje popis technologických rozhodnutí a návrh řešení, který byl vypracován ve spolupráci s odborným terapeutem. Dále se práce zabývá implementací a nasazením aplikace, včetně zavedených DevOps praktik. Závěr práce se věnuje validaci aplikace prostřednictvím automatizovaných testů a uživatelského testování.

Klíčová slova: webová aplikace, frekvenční terapie, muzikoterapie, terapeutické ladičky, Next.js, React.js, cloudové služby

Poděkování

Chtěl bych vyjádřit své upřímné poděkování všem, kteří mi poskytli podporu a pomoc během přípravy mé diplomové práce. Především bych rád poděkoval mému vedoucímu diplomové práce, Ing. Petrovi Brůhovi, za jeho zkušené vedení, trpělivost a odborné rady, které mi poskytl v průběhu celého procesu psaní práce. Dále bych chtěl poděkovat odbornému terapeutovi Bc. Richardu Frouzovi za jeho spolupráci na návrhu a testování aplikace a za sdílení svých znalostí a zkušeností v oblasti frekvenční terapie. Svou vděčnost bych rád vyjádřil i všem lidem, kteří se podíleli na testování aplikace.

Obsah

1 Úvod

Hudba byla již od svého vzniku využívána k mnoha různým účelům a představuje tak neodmyslitelnou součást lidské kultury a historie. V současnosti se stále více ukazuje, že zvukové frekvence mohou být účinným nástrojem i v mnoha oblastech medicíny [11].

Efektivní využití terapeutických zvuků vyžaduje pravidelný poslech, a to často i několikrát denně. Nicméně, to není snadno realizovatelné, jelikož jsou zapotřebí speciální nástroje, jako například terapeutické ladičky. Tyto pomůcky bývají často nákladné a vyžadují odbornou manipulaci. Pravidelná kontaktní terapie s terapeutem nedokáže efektivně pokrýt potřebný počet užívání pro klienty, a to především z časových důvodů obou stran a finančních důvodů ze strany klienta.

Tato diplomová práce se zaměřuje na vývoj webové aplikace ve spolupráci s odborným terapeutem, která má umožnit vhodné rozšíření stávající kontaktní terapie. Cílem je poskytnou terapeutům způsob, jak vytvářet individuální frekvenční stopy, které budou moci klienti poslouchat sami bez nutnosti častých návštěv. Terapeut bude mít také možnost sledovat, jak klienti aplikaci používají, a adekvátně terapii přizpůsobovat.

Teoretická část práce se zaměřuje na zvuk a jeho terapeutické užití v muzikoterapii, s důrazem na využití terapeutických ladiček. Dále se zabývá webovým prostředím, architekturou a technologiemi (výběr webové platformy vychází z aplikačních požadavků a zadání diplomové práce). Následuje kapitola o cloudových technologiích se zaměřením na jejich roli při vývoji moderních webových aplikací. Dále je přiblížena webová bezpečnost a popsány vhodné praktiky při návrhu uživatelského rozhraní. V neposlední řadě je představena problematika nasazení webové aplikace, včetně základních konceptů DevOps. V závěru této části jsou analyzována existující řešení.

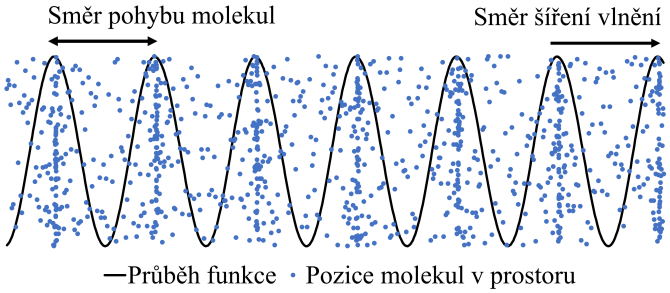
Praktická část práce začíná kapitolou podrobně popisující návrh aplikace na základě stanoveného problému, provedené analýzy a požadavků terapeuta. Následuje kapitola věnující se implementaci navržené aplikace. Zde je popsána zvolená architektura, konkrétní využité technologie, struktura projektu, postupy zvolené pro řešení specifických problémů a implementované praktiky pro zabezpečení aplikace. Další kapitola pojednává o nasazení vytvořené aplikace, automatizaci tohoto procesu využitím CI/CD pipeline a o hostování a monitorování aplikace v produkčním prostředí. V poslední kapitole je přiblížena validace aplikace zahrnující automatizované testy, uživatelské testovaní a testování výkonu.

2 Zvuk

Kapitola se věnuje zvuku, jeho definici a vlastnostem. Slouží především jako úvod nadcházející kapitole o muzikoterapii, pro kterou jsou zvukové frekvence klíčovým prvkem.

2.1 Definice zvuku

Zvuk je mechanické vlnění šířící se v látkovém prostředí. Člověk dokáže slyšet frekvence pouze v přibližném rozmezí od 20 Hz do 20 kHz. V médiu, které přenáší zvukové vlny, dochází k vytváření oblastí s vyšším a nižším tlakem (viz obrázek 2.1). Rychlost šíření je závislá na prostředí [27, 35].

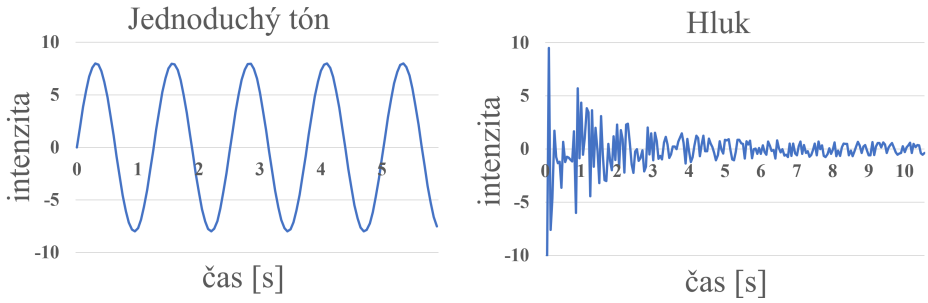


Obrázek 2.1: Šíření zvuku prostorem [40]

2.2 Základní charakteristiky zvuku

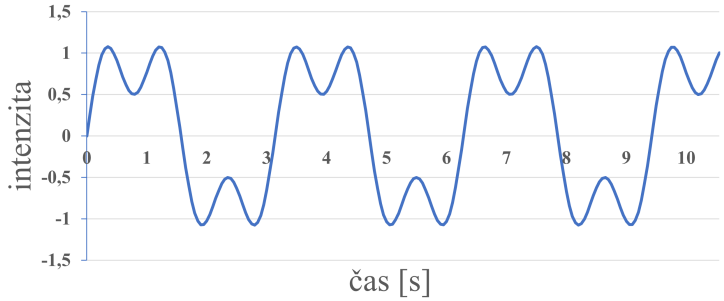
Tóny a hluky (viz obrázek 2.2) jsou dvě základní kategorie zvuků. Tóny se vyznačují pravidelným vlnovým průběhem, což vytváří příjemné zvuky pro lidský sluch. Jsou především součástí hudby a mluveného jazyka ve formě samohlásek. Na druhé straně, vlnový průběh hluků a šumů je neuspořádaný. Ačkoliv jsou hluky méně harmonické, vyskytují se běžně v hudbě i lidském jazyce. Většina hudebních nástrojů je navržena tak, aby produkovala tóny, nicméně existují výjimky, jako například bicí nástroje. Celkově lze říci, že tóny jsou vnímány jako harmonické a příjemné zvuky, zatímco hluky jsou považovány za chaotické a mohou být vnímány jako rušivé [27, 40].

Zvuky s periodickým průběhem, označované jako tóny, lze na základě vlastností vlnového průběhu dále klasifikovat na jednoduché a složené. Jednoduché tóny mají sinusový vlnový průběh, a jedná se tak o nejjednodušší



Obrázek 2.2: Tón a hluk [40]

formu vlnění obsahující pouze jednu frekvenci. Složené tóny (viz obrázek 2.3) se skládají ze z vícero frekvencí, přičemž nejnižší z nich je základní frekvence, která určuje periodický průběh celého tónu. Vyšší harmonické frekvence se také nazývají alikvotní tóny, jejich zastoupení mění charakteristiku zvuku a je označováno jako zbarvení tónu [35, 40].



Obrázek 2.3: Složený tón [40]

Mezi další základní vlastnosti tónu patří výška a hlasitost. Výška tónu je určena frekvencí, a často se používá výška relativní, která udává poměr frekvence daného tónu k frekvenci referenční. Běžně se jako referenční tón používá komorní A s frekvencí 440 Hz. Hlasitost zvuku je určena množstvím energie, kterou vlnění přenáší, neboli intenzitou. Hlasitost zvuku je úzce spojena s lidskou fyziologickou a psychickou percepcí zvuku, a proto nelze popsat pouze prostřednictvím fyzikálních vlastností prostředí, ve kterém se zvuk šíří. Jelikož lidský sluch nevnímá změny intenzity zvuku lineárně, ale spíše podle logaritmické křivky (což popisuje Weberův-Fechnerův zákon), a vzhledem k širokému rozsahu hodnot, se pro popis hlasitosti využívá jednotka decibel (dB) s logaritmickou stupnicí. Jako referenční hodnota jednotky je stanovena minimální hlasitost, při které zdravý lidský sluch dokáže slyšet tón s frekvencí 1 kHz [27, 35].

3 Muzikoterapie

Kapitola se zaměřuje na muzikoterapii a její aplikaci v praxi. Přibližuje vliv hudby a zvuku na člověka, pozici muzikoterapie v moderní medicíně, terapeutické ladičky a proces, jakým jsou používány.

3.1 Definice muzikoterapie

Muzikoterapie je terapeutická disciplína, která využívá hudbu a hudební prvky k dosažení cílů mimo hudební oblast. Tyto cíle se obvykle týkají léčby, ale mohou zahrnovat také osobní růst, zlepšení kvality života nebo mezilidských vztahů (viz obrázek 3.1) [11, 41].

Celostní medicína je v dnešní době stále více uznávána, což vede k rostoucímu zájmu o muzikoterapii ze strany zdravotnických profesionálů. Řada lékařů a odborníků potvrzuje, že muzikoterapie přináší pacientům znatelné zlepšení, a to buď přímo v rámci léčby určitých potíží, nebo ve zlepšení jejich celkového stavu [11].

3.2 Výzkum muzikoterapie

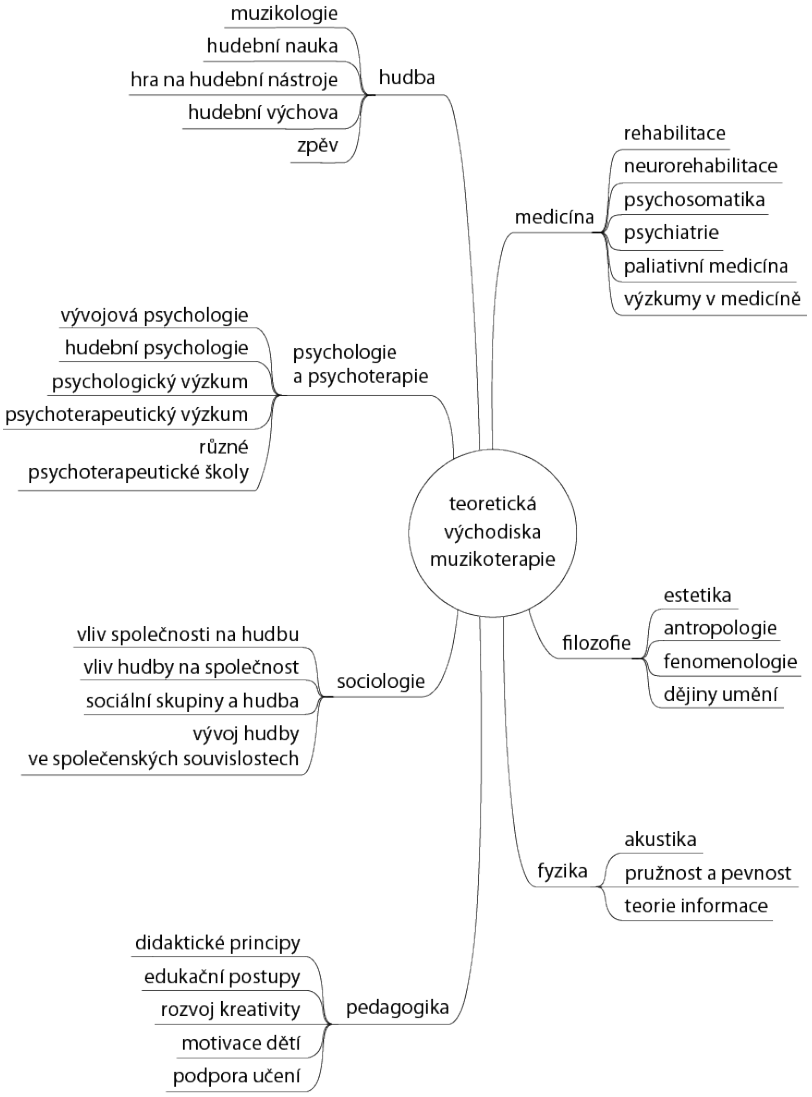
Muzikoterapie má dlouhou historii, která sahá tisíce let do minulosti. Léčení hudbou se nezávisle vyvíjelo po celém světě a v současnosti se rozvíjí i v českém prostředí. Výzkumy ukazují, že muzikoterapie má příznivý dopad na zdraví jednotlivce a že se účinně kombinuje s jinými terapeutickými přístupy, zejména u osob s tělesným a mentálním postižením. Výsledky ukazují, že hudba a zvukové frekvence mohou být účinným prostředkem pro podporu relaxace, snížení stresu, zlepšení spánku a další aspekty, a to i u pacientů s psychiatrickými poruchami, jako jsou autismus a ADHD[1] [1, 11, 30, 41].

3.3 Nástroje v muzikoterapii

V terapeutickém kontextu se často využívá klasická hudba, ale i hudba pocházející z východních kultur. V rámci muzikoterapie se také využívají různé

[1] ADHD (Attention Deficit Hyperactivity Disorder) je neurovývojová porucha charakterizovaná problémy s pozorností, hyperaktivitou a impulzivitou.

nástroje, jako například terapeutické ladičky. Často se lze také setkat s tradičními hudebními nástroji z různých kultur, jako jsou tibetské mísy, didgeridoo, dřevěné píšťalky, gongy, bubny či činely [41].



Obrázek 3.1: Teoretická východiska muzikoterapie [11]

3.3.1 Využití terapeutických ladiček

Tato část se zaměřuje na terapeutické ladičky, což jsou specializované nástroje navržené pro muzikoterapii, umožňující reprodukci konkrétních frekvencí.

Druhy terapeutických ladiček

Existují dva základní druhy terapeutických ladiček: se závažím a bez závaží (viz obrázek 3.2). Ladičky se závažím pracují především na fyzické úrovni a mohou být umístěny přímo na těle pacienta. Jejich těžší konstrukce umožňuje přenos vibrací do hloubky těla. Na druhou stranu, ladičky bez závaží pracují více na zvukové úrovni, poskytují intenzivnější zvuk a jsou lehčí [28, 41].

Obrázek 3.2: Terapeutické ladičky se závažím a bez závaží (Zdroj: `https://www.anthonyproducts.com/store/tuning-fork-set`)

Frekvence terapeutických ladiček

Terapeutické ladičky se vyrábí a kalibrují tak, aby produkovaly jednu konkrétní frekvenci. Produkovaná frekvence se může mírně lišit v závislosti na vlastnostech prostředí jako jsou teplota a nadmořská výška. Terapeutické ladičky běžně využívají tóny vycházející z diatonické stupnice s referenčním tónem C o frekvenci 256 Hz. Tento ladící systém se liší od běžně používaného tónu komorního A s kmitočtem 440 Hz. Mezi nejčastěji využívané frekvence terapeutických ladiček se řadí následující [28]:

- C – 256 Hz
- D – 288 Hz
- E – 320 Hz
- F – 341,3 Hz
- G – 384 Hz

- A – 426,7 Hz
- B – 480 Hz
- C – 512 Hz

Procedura využitím terapeutických ladiček

Rozeznění ladičky probíhá úderem o aktivátor, kterým může být dřevěná palička s gumovým nebo dřevěným koncem, gumový předmět, nebo i hokejový puk. Po rozeznění ladičky terapeut manipuluje s nástrojem v prostoru kolem pacienta nebo jej umisťuje přímo na tělo pacienta, v závislosti na druhu ladičky a terapeutických potřeb. Během terapie mohou být ladičky použity samostatně nebo i v kombinaci. Terapeut může rovněž pracovat s různými tóny a frekvencemi, aby stimuloval různé části těla nebo mozkové frekvence [28, 41].

4 WWW

WWW (World Wide Web, často nazývaný pouze web) je klíčovým prvkem našeho každodenního života a základním konceptem při vývoji webových aplikací. Je tedy vhodné porozumět jeho principům a součástem. Následující text se zaměřuje na přiblížení webového prostředí, jeho základních komponent a fungování.

4.1 Definice WWW

WWW je systém vzájemně propojených veřejných webových stránek přístupných prostřednictvím internetu. Web není totéž co internet, ale web je jednou z mnoha aplikací fungujících v rámci internetu. Webovou architekturu navrhl Tim Berners-Lee a zároveň pomohl vytvořit její základní prvky publikované v CERNu v roce 1990 [18, 24].

Pro identifikaci a lokalizaci webových stránek a dalších zdrojů na internetu se využívají adresy URL (Uniform Resource Locator). Struktura URL zahrnuje protokol (např. HTTP nebo HTTPS), doménové jméno, cestu ke zdroji a někdy i další parametry. Díky URL mohou uživatelé snadno navigovat mezi různými webovými stránkami a zdroji.

4.2 Základní prvky WWW

Mezi klíčové prvky WWW patří webový server, webový prohlížeč a webová stránka.

Webová stránka

Webová stránka je digitální dokument zobrazitelný v prohlížeči, vytvořený pomocí HTML (viz kapitola 6.1). Klíčové prvky webových stránek zahrnují:

1. **styl a struktura:** Pro vizuální úpravy se využívá CSS (viz kapitola 6.2),
2. **interaktivita:** Skripty (nejčastěji JavaScript, viz kapitola 6.3) přidávají dynamiku a interakci,
3. **média:** Obrázky, zvuky a videa zobrazované na stránce.

Webový prohlížeč

Webový prohlížeč nebo pouze prohlížeč je program, který načítá a zobrazuje stránky z webu. Dokáže upravovat dokumenty získané od webového serveru prostřednictvím JavaScriptu, který dynamicky manipuluje s obsahem stránky pomocí objektového modelu dokumentu. K provádění JavaScriptu jsou v prohlížečích používány speciální JavaScriptové „enginy“ (např. V8 nebo SpiderMonkey) [24].

Mezi nejznámější webové prohlížeče patří Google Chrome, Safari, Mozilla Firefox, Microsoft Edge a Opera.

Webový server

Webové servery zprostředkovávají přístup k hostovaným souborům uživatelům. K tomu potřebují rozumět URL adresám a protokolu HTTP (viz kapitola 4.3). Základních typy webových serverů jsou statický a dynamický.

Statický webový server odesílá hostované soubory do prohlížeče v nezměněné podobě. Tyto servery jsou vhodné pro jednoduché webové stránky používané k prezentaci informací.

Dynamický webový server kombinuje statický server s dalšími komponentami, jako je aplikační server a databáze. Tyto servery implementují aplikační logiku a pracují s persistovanými daty. Také zpravidla umožňují dodatečné funkce skrze webové API (viz kapitola 5.3). Umožňují tak vytvářet dynamický a personalizovaný obsah [24].

4.3 HTTP

HTTP (HyperText Transfer Protocol) je základní síťový protokol, který umožňuje komunikaci mezi webovými prohlížeči a servery a hraje klíčovou roli v přenosu dat na webu. Jeho význam spočívá v jeho schopnosti zjednodušit a standardizovat komunikaci mezi webovými klienty a servery [18, 24].

HTTP je založen na klient-server architektuře, což znamená, že existuje webový klient (např. webový prohlížeč), který požaduje informace a webový server, který tyto informace poskytuje. Klient posílá požadavek na server, který na základě tohoto požadavku vytváří odpověď a posílá ji zpět klientovi. V rámci HTTP se požadavky a odpovědi definují pomocí zpráv, které jsou tvořeny hlavičkou a tělem [18, 24].

HTTP používá standardizované metody požadavků, které umožňují klientům získat, aktualizovat a mazat informace na serveru. Mezi nejčastěji používané metody patří GET, POST, PUT a DELETE. GET metoda se

používá pro získání informací ze serveru, POST pro odeslání dat na server, PUT pro aktualizaci dat na serveru a DELETE pro smazání dat na serveru [24].

CORS

CORS (Cross-Origin Resource Sharing) je bezpečnostní mechanismus založený na HTTP hlavičkách, který umožňuje řízení přístupu frontendového JavaScript kódu k odpovědím na požadavky z jiných domén, než je doména provozující server. CORS řeší omezení zavedená politikou stejného původu (same-origin policy), která v základním nastavení zakazuje přístup ke zdrojům z jiných domén. Webovým serverům poskytuje možnost povolit přístup ke svým zdrojům i z jiných domén, čímž podporuje interoperabilitu webových aplikací [24].

Cookies

Soubory cookies jsou malé bloky dat, které webová stránka může využitím protokolu HTTP uložit do zařízení uživatele. Hraje klíčovou roli při identifikaci uživatelů navštěvujících stránku opakovaně a personalizaci zobrazeného obsahu. Uživatelé mohou své prohlížeče nastavit tak, aby soubory cookie přijímaly, odmítaly, nebo je automaticky odstraňovaly. Webové stránky, které využívají soubory cookie, o tom musí uživatele informovat a nesmějí bez souhlasu takto sbírat osobní údaje [18, 24].

4.3.1 HTTPS

HTTPS (Hypertext Transfer Protocol Secure) je varianta protokolu HTTP, která využívá zabezpečený kanál vytvořený protokolem TLS (Transport Layer Security), který dříve býval znám jako SSL (Secure Sockets Layer). Používání protokolu HTTPS je dnes základem každé kvalitní webové aplikace, jelikož zajišťuje, že komunikace mezi klientem a serverem je šifrovaná a bezpečná. Také pomáhá chránit proti phishingovým útokům, kdy útočník vydávající se za legitimní webovou stránku získá citlivé informace od uživatelů [14, 24].

Pro navázání zabezpečeného připojení používá protokol TLS platný digitální certifikát. Digitální certifikát je soubor, který váže veřejně známý kryptografický klíč k organizaci. Tento klíč slouží k ověření identity serveru. Všechny moderní prohlížeče vyžadují platný digitální certifikát pro navázání zabezpečeného připojení k webové stránce [24].

5 Architektura webové aplikace

Kapitola se zaměřuje na klíčové aspekty typické pro aplikační architekturu ve webového prostředí, včetně základních komponent a standardních metod pro implementaci webového API.

5.1 Úvod do webových architektur

Softwarová architektura je základní struktura umožňující efektivní a bezpečné fungování aplikací. Typicky se skládá ze vzájemně propojených a izolovaných komponent zaměřených na určité aspekty. Správně navržená architektura je klíčová pro udržení rychlosti, škálovatelnosti, bezpečnosti a funkčnosti aplikace.

Webová architektura se zaměřuje na komunikaci komponent prostřednictvím internetu. Základem této komunikace je protokol HTTP (viz kapitola 4.3), který funguje na principu server-klient. Klient, obvykle webový prohlížeč uživatele, vytváří požadavky skrze webové API na server, který následně zpracovává tyto požadavky a poskytuje odpovídající informace nebo služby [38].

5.2 Třívrstvá architektura

Třívrstvá architektura je základním konceptem, který popisuje tři klíčové vrstvy: prezentační, aplikační a datovou. Je možné ho využít při návrhu mnoha různých aplikací a jedná se o standardní princip fungování webových aplikací [38].

Prezentační vrstva

Prezentační vrstva představuje uživatelské rozhraní webové aplikace, které běží na straně uživatele v internetovém prohlížeči. Tato část je zodpovědná za zobrazení dat uživateli a za interakci s nimi. Využívá zejména kód napsaný v jazycích HTML, CSS a JavaScript, který je zpracován prohlížečem. Pro usnadnění vývoje se často využívají frameworky a knihovny, jako jsou Angular, React nebo Vue. Tyto nástroje poskytují šablony a komponenty

pro tvorbu uživatelského rozhraní, což výrazně zrychluje vývoj a zlepšuje udržitelnost kódu [38]. Technologie jsou blíže popsány v kapitole 6.

Aplikační vrstva

Aplikační vrstva běží na straně serveru a reaguje na HTTP zprávy od prezenční vrstvy. Obsahuje aplikační logiku a zajišťuje propojení mezi dalšími dvěma vrstvami. Serverová strana může být napsána v různých programovacích jazycích, jako jsou např. Java, JavaScript, PHP, Ruby nebo Python. Pro usnadnění vývoje backendu se často využívají frameworky, jako jsou Spring (Java), Django (Python) nebo Ruby on Rails (Ruby). Tyto frameworky poskytují nástroje a komponenty pro efektivní vývoj, testování a nasazení webových aplikací [38]. Blíže jsou popsány v kapitole 7.

Datová vrstva

Datová vrstva je zodpovědná za ukládání a správu dat používaných aplikací. Jejím hlavním prvkem je databázový server, který může být provozován na stejném stroji jako aplikační server nebo odděleně. V rámci datové vrstvy se mohou používat různé typy databází, jako jsou relační, NoSQL, nebo grafové databáze, v závislosti na potřebách aplikace a struktuře dat. Volba databázové technologie ovlivňuje výkon, škálovatelnost a flexibilitu, což je klíčové pro celkovou funkčnost aplikace [38].

5.3 Webové API

API (Application Programming Interface) je pojem, označující softwarové rozhraní, které umožňuje různým aplikacím vzájemnou komunikaci a integraci. API poskytuje definici metod, funkcí a datových struktur, které slouží jako předpis toho, co všechno je možné od aplikace požadovat a jak si s ní vyměňovat data [33].

V kontextu webového prostředí se termín webové API používá pro označení sady standardů a protokolů, které umožňují webovým aplikacím výměnu dat a funkcí přes internet. V dnešní době se webové API stává stále důležitějším prvkem webového vývoje. To především díky rozmáhajícím se servisově orientovaným architekturám a cloudovým technologiím (viz kapitola 8) [33].

Existuje mnoho různých způsobů, protokolů a nástrojů, které lze využít při tvorbě webového API, řadí se mezi ně např. REST, SOAP, GraphQL, RPC a další.

REST

REST (Representational State Transfer) je architektonický styl, pro komunikaci s webovými aplikacemi prostřednictvím standardních HTTP metod, jako jsou GET, POST, PUT a DELETE. RESTful služby používají jednotné rozhraní, které se skládá z identifikátorů zdrojů a reprezentací zdrojů v různých formátech (např. JSON, XML) [33]. Služba by měla splňovat následující požadavky pro vyhovění paradigmatu REST [10]:

1. **klient-server architektura:** Klient a server jsou od sebe izolováni,
2. **bezstavovost:** Server nepřechází mezi více stavy,
3. **ukládání do mezipaměti:** Data mohou být uložena do mezipaměti pro menší a rychlejší přenosy dat,
4. **jednotné rozhraní:** Server má jednotné a předvídatelné rozhraní,
5. **vícevrstvý systém:** Architektura je rozdělena do vrstev, které si předávají data,
6. **kód na vyžádání:** Umožňuje zaslání kódu, který se následně spustí na straně serveru, to ale přináší bezpečnostní rizika a vlastnost tedy není striktně vyžadována.

SOAP

SOAP (Simple Object Access Protocol) je protokol používající XML formát pro zpracování zpráv a definující standardy pro různé aspekty webových služeb, jako jsou popis rozhraní, zabezpečení a transakce. SOAP služby mají několik výhod, jako je široká podpora ze strany nástrojů a knihoven pro vývoj a testování, ale jsou často kritizovány pro svou složitost a zbytečnou režii při přenosu dat [33].

GraphQL

GraphQL je dotazovací jazyk pro API, který umožňuje klientům specifikovat, jaká data potřebují, a získat je v odpovědi ve strukturovaném formátu. GraphQL nabízí mnoho výhod, jako je možnost získání všech požadovaných dat v jednom dotazu, efektivní využití mezipaměti a rychlé vytváření rozhraní. Avšak, jeho použití může být komplexnější při vytváření požadavků a vyžaduje speciální nástroje, což může být překážkou [33].

RPC

RPC (Remote Procedure Call) je způsob tvorby API, který umožňuje provádění funkcí ve vzdáleném kontextu (klient spouští funkce serveru). Z pohledu programátora se volání a definice vzdálené funkce neliší příliš od funkcí lokálních. Díky tomu je tvorba API velmi rychlá a efektivní. Za tímto pohodlím je skrytá nemalá režie, která je často implementovaná ve využité knihovně. RPC používá GET pro získání informací a POST pro vše ostatní. Nevýhodou je závislost na konkrétních technologiích a ztráta obecnosti API způsobená často velkou svázaností s klientskou aplikací, pro kterou je API tvořeno. Proto se nejedná o ideální způsob definování webového API, pokud má být využíváno mnoha různými klientskými aplikacemi [33].

WebSocket

WebSocket je komunikační protokol umožňující dynamickou komunikaci mezi klientem a serverem přes jedno TCP spojení. Vytváří tenkou vrstvou nad TCP/IP technologií. S použitím WebSocket API může klient poslat zprávu na server a obdržet odezvu bez nutnosti periodického dotazování serveru na odpověď [33].

6 Technologie na straně klienta

Kapitola se zaměřuje na klíčové technologie využívané při tvorbě uživatelského rozhraní webových stránek a aplikací. Je zde přiblížena základní trojice technologií, která zahrnuje HTML, CSS a JavaScript. Dále je popsán programovací jazyk TypeScript a JavaScriptové frameworky.

6.1 HTML

HTML (HyperText Markup Language) je značkovací jazyk, který definuje strukturu webových stránek a je zásadním stavebním prvkem na webu. Jazyk byl poprvé představen v roce 1993. Od té doby se rychle vyvíjel a stal se standardním značkovacím jazykem na celém světě. Poslední a nejvíce využívanou verzí je HTML5 [24, 38].

HTML dokument se skládá z řady elementů, které se používají k obalení různých částí obsahu tak, aby vypadal určitým způsobem nebo se určitým způsobem choval. Elementy se do sebe mohou zanořovat a vytváří tak stromovou strukturu. Jsou odděleny od ostatního textu v dokumentu pomocí „tagů“, kterým se mohou předávat data pomocí atributů nastavující jejich vzhled a chování [24].

6.2 CSS

CSS (Cascading Style Sheets) je jazyk, který umožňuje oddělit obsah stránky od jejího vizuálního zobrazení, což usnadňuje správu a úpravu vzhledu stránek. CSS poskytuje například nástroje pro úpravu stylů písma, barev, rozložení stránky, mezery mezi prvky a přidání animací a dalších vizuálních prvků. Výhodnou vlastností CSS je, že jednou definovaný styl lze opakovaně používat pro více prvků. Stejně tak lze použít jeden styl na více webů [24, 38].

V CSS jsou základními stavebními bloky selektory a deklarace. Selektory určují, na které HTML prvky se mají styly aplikovat, zatímco deklarace obsahují vlastnosti a jejich hodnoty, které určují vzhled webové stránky [24].

Kaskádování je klíčová vlastnost CSS, která řídí, jaký styl má být upřednostněn, pokud existuje více pravidel s různými styly pro stejný prvek. Pro

zajištění správného zobrazení stránky je styl upřednostňován podle následujícího řádu: důležitost a původ, specifičnost selektorů a pořadí v kódu [24].

6.3 JavaScript

JavaScript je interpretovaný, just-in-time kompilovaný programovací jazyk. Vykonává se na straně klienta a umožňuje přidávat interaktivitu na webové stránky prostřednictvím manipulace s objektovým modelem dokumentu. Tím dává možnost vytváření bohatých uživatelských rozhraní a vylepšení uživatelského zážitku [24].

Jazyk navíc umožňuje využitím technologie Node.js (viz kapitola 7.6) i vývoj serverové části aplikace. Díky této široké uplatnitelnosti se JavaScript stal nejpoužívanějším programovacím jazykem [24, 38].

6.3.1 TypeScript

TypeScript je programovací jazyk založený na JavaScriptu, který přináší silné typování a další syntaktická rozšíření. Výsledkem je lepší integrace s vývojovým prostředím, přehlednější kód a včasnější odhalování chyb díky statické analýze. TypeScript se překládá do čistého JavaScriptu, což znamená, že je kompatibilní s existujícím JavaScriptovým kódem a prostředím [23].

Jeho hlavní výhodou je zavedení typů, které umožňují vývojářům psát bezpečnější a strukturovanější kód s nižším rizikem chyb za běhu programu. Statické typování také usnadňuje vývoj skrze vyšší míru „našeptávání“ kódu vývojovým prostředím. TypeScript lze využít jak pro psaní klientského JavaScriptu běžícího v prohlížeči, tak pro JavaScript běžící na serveru (např. Node.js).

Pro udržení kvality kódu a dodržování dobrých praktik se často TypeScript kombinuje s lint nástroji, jako je ESLint, které vynucují konkrétní pravidla a standardy [23, 24].

6.4 JavaScriptové frameworky

JavaScriptové frameworky jsou klíčovou součástí moderního frontendového vývoje webových aplikací. Poskytují vývojářům osvědčené nástroje, strukturu a metodiky pro efektivní vytváření škálovatelných a interaktivních webových aplikací. Díky tomu je výsledný kód předvídatelný, udržitelný

a snadno škálovatelný. To také usnadňuje spolupráci ve větších týmech [24]. Mezi nejpopulárnější JavaScriptové frameworky patří React, Angular a Vue, které jsou podrobněji popsány v následujících kapitolách.

6.4.1 React

React je open-source JavaScriptový framework pro vytváření interaktivních uživatelských rozhraní. Byl vyvinut společností Facebook v roce 2013 a od té doby se stal jedním z nejpopulárnějších nástrojů pro vývoj webových aplikací. React je vhodný pro vývoj různých typů aplikací, včetně webových a mobilních, a to ve spolupráci s dalšími knihovnami, jako jsou React Native a ReactDOM [24, 34].

Jedním z klíčových prvků Reactu je jeho využití virtuálního DOM. Tento přístup umožňuje efektivně aktualizovat uživatelské rozhraní tak, že místo aktualizace celého DOM se aktualizují pouze relevantní části. Díky tomu je dosaženo lepšího výkonu a efektivity, což přispívá uživatelské zkušenosti [24, 34].

React staví na komponentovém přístupu, což znamená, že kód je modulární a snadno znovu použitelný v různých částech aplikace. Komponenty lze chápat jako samostatné bloky, které mohou být opakovaně použity a snadno aktualizovány. Tento přístup usnadňuje údržbu a škálování aplikací [38].

Díky rozsáhlé dokumentaci a široké škále nástrojů je React silnýou technologií pro tvorbu uživatelských rozhraní. Na druhou stranu vlastnosti jazyka, vhodné a populární knihovny se často mění a přibývají, to vyžaduje, aby vývojáři tyto změny sledovali a rychle se jim přizpůsobovali. Tento fakt může být nepříjemný pro začínající vývojáře. Navíc se React zaměřuje pouze na část uživatelského rozhraní, což znamená, že pro další funkce na straně klienta je nutné využívat další knihovny. To přináší možnost volby vhodných knihoven, nejedná se ale nutně o výhodu, jelikož není standardní struktura aplikace a projekty se mohou velmi lišit [34, 36].

6.4.2 Angular

Angular je open-source framework pro vývoj webových aplikací, který byl vytvořen společností Google. Díky své komponentové architektuře a použití obousměrné datové vazby poskytuje Angular rychlý a efektivní způsob práce s DOM, což zlepšuje výkon aplikace [24, 38].

Jedna z výhod Angularu spočívá v tom, že nabízí další užitečné funkce, jako jsou NGModules, směrování, dependency injection, testování a řadu

pluginů a nástrojů. Tyto funkce umožňují rychle vytvářet prototypy a aplikace [38].

V porovnání s jinými frameworky, jako je React, nabízí Angular vlastnost obousměrné datové vazby. Díky tomu se změny v modelu okamžitě promítají do zobrazení a naopak, což usnadňuje práci s aplikací a interaktivitu pro uživatele [38].

6.4.3 Vue

Vue je JavaScriptový framework, který se zaměřuje na snadnou použitelnost a výkonnost při vývoji uživatelských rozhraní pro webové, desktopové a mobilní aplikace. Jeho snadná instalace a možnost postupného vylepšování stávajícího HTML kódu činí Vue vhodnou volbou pro modernizaci existujících projektů, například jako náhradu knihovny jQuery [24, 38].

Díky použití virtuálního DOM nabízí Vue rychlý výkon a efektivní aktualizace komponent na stránce. Obousměrná datová vazba zajišťuje efektivní synchronizaci mezi datovými modely a souvisejícími komponentami, což usnadňuje správu kódu a udržitelnost aplikací. Komponenty jsou modulární, opakovaně použitelné a snadno integrovatelné do stávajících aplikací [38].

Přestože Vue disponuje silnou komunitní podporou, je třeba poznamenat, že je z velké části adoptován čínskými společnostmi, což může pro anglicky mluvící vývojáře způsobit jazykovou bariéru. Pro rozsáhlejší projekty může být sada nástrojů omezená a přílišná flexibilita, kterou Vue nabízí, může vést k problémům s kvalitou a udržitelností kódu v rozsáhlých projektech [38].

7 Serverové technologie

Klíčovým aspektem webových aplikací je serverová strana, která tvoří jádro architektury a zajišťuje správné fungování, bezpečnost a výkon. Tato kapitola se zaměřuje na sadu nejpoužívanějších serverových technologií, které zahrnují různé frameworky, programovací jazyky a běhová prostředí.

7.1 ASP.NET

ASP.NET je ope-source framework, který byl vyvinut společností Microsoft pro vývoj webových aplikací a služeb pomocí rozhraní .NET. Framework je primárně navržen pro použití s programovacím jazykem C# a poskytuje komplexní sadu nástrojů a funkcionalit, které usnadňují rychlé vytváření, nasazování a udržování webových aplikací [22].

Podporuje architekturu MVC (model-view-controller). Nabízí Razor syntaxi, která umožňuje snadno vytvářet dynamické webové stránky pomocí kombinace HTML a C# kódu. Navíc poskytuje mnoho funkcí pro efektivní správu dat, zabezpečení a optimalizaci výkonu aplikací. K těmto funkcím patří například Entity Framework, který umožňuje snadnou práci s databázemi a mapování objektů na databázové tabulky, zabezpečení aplikací pomocí autentizace a autorizace, a podpora pro cachování a kompresi dat pro zlepšení výkonu aplikace [22].

ASP.NET je těsně integrován s cloudovým řešením společnosti Microsoft, Microsoft Azure. Tato integrace umožňuje snadné nasazení a škálování aplikací v cloudovém prostředí. Azure také nabízí řadu služeb, které mohou být využity pro zlepšení funkcionalit, zabezpečení a výkonu svých webových aplikací.

7.2 Spring

Spring je open-source framework pro vývoj enterprise aplikací a služeb v jazyce Java. Framework se zaměřuje na zjednodušení vývoje a nasazení webových aplikací a poskytování široké škály funkcí a modulů pro různé účely [5].

Používá MVC architekturu, což usnadňuje oddělení logiky aplikace od prezentace dat. Je integrován s dalšími technologiemi Java, jako je JPA (Java Persistence API) pro správu persistovaných dat. JPA umožňuje snadno pra-

covat s databázemi a ukládat data do nich pomocí objektově-relačního mapování. Poskytuje také podporu pro tvorbu dynamických webových stránek pomocí JSP (JavaServer Pages). Navíc Spring framework obsahuje modul Spring Security, který poskytuje rozsáhlé možnosti pro zabezpečení aplikací [5].

7.3 Django

Django je open-ource framework pro vývoj webových aplikací v jazyce Python. Framework nabízí soubor nástrojů a funkcionalit, které usnadňují rychlé vytváření, nasazení a údržbu webových aplikací [9].

Používá architekturu založenou na MVT model-view-template návrhovém vzoru, což je variací na klasický MVC vzor. Poskytuje bohatou podporu pro správu relačních databází pomocí svého vlastního ORM systému. Obsahuje systém šablon, který umožňuje snadno vytvářet dynamické webové stránky s minimálním opakováním kódu. Poskytuje také podporu pro vytváření webových formulářů, což zahrnuje validaci dat, manipulaci s formuláři a zpracování dat z formulářů. Klade velký důraz na bezpečnost a obsahuje mnoho integrovaných bezpečnostních funkcí, které chrání aplikace před nejčastějšími webovými útoky [9].

7.4 Ruby on Rails

Ruby on Rails je open-source framework pro vývoj webových aplikací, který je napsán v programovacím jazyce Ruby. Framework klade důraz na konvence, což vede k čistějšímu, snadněji čitelnému a udržovatelnému kódu. Navíc je navržen tak, aby usnadnil a zrychlil vývoj webových aplikací tím, že poskytuje předem definovanou architekturu, sadu nástrojů a konvence pro návrh aplikací [29].

Používá koncept MVC architektury, což umožňuje oddělit logiku aplikace od prezentační vrstvy. Poskytuje mnoho modulů a knihoven, které umožňují rychle a snadno rozšiřovat své aplikace. Mezi tyto moduly patří například ActiveRecord pro práci s databázemi, ActionMailer pro zasílání e-mailů, ActionPack pro tvorbu webových stránek a ActionController pro zpracování HTTP požadavků [29].

7.5 PHP

PHP (rekurzivní zkratka PHP: Hypertext Preprocessor) je open-source skriptovací jazyk, který se široce používá pro vývoj webových aplikací. PHP je jedním z nejstarších a nejrozšířenějších webových jazyků, který se začal používat v roce 1995. Díky jeho jednoduchosti a přístupnosti získal velkou popularitu. Jedná se o server-side jazyk, což znamená, že se kód PHP vykonává na serveru a výsledný HTML kód se odesílá do webového prohlížeče klienta [26].

7.5.1 Laravel

Laravel je open-source framework jazyka PHP, který poskytuje různé nástroje a komponenty pro snadný vývoj a správu webových aplikací. Používá MVC architekturu a také obsahuje vlastní systém šablon nazvaný Blade, který umožňuje snadno vytvářet dynamické HTML šablony s PHP kódem. Framework Laravel poskytuje podporu pro práci s databázemi prostřednictvím Eloquent ORM. Také zahrnuje integrovaný systém pro správu autentizace a autorizace uživatelů, což usnadňuje zabezpečení aplikace [19].

7.6 Node.js

Node.js je open-source JavaScriptový runtime, který umožňuje vývojářům psát serverovou stranu aplikací v JavaScriptu. Tato technologie rozšiřuje možnosti JavaScriptu, který byl původně navržen jako jazyk pro vývoj webových aplikací na straně klienta. Node.js je postaven na enginu V8 od Google, což znamená, že může rychle a efektivně zpracovávat JavaScriptový kód. Vyznačuje asynchronním zpracováním vstupu/výstupu, což umožňuje rychle a efektivně obsluhovat velké množství požadavků [25].

Node.js má velkou komunitu vývojářů a rozsáhlou knihovnu modulů, známou jako NPM (Node Package Manager). Díky NPM lze rychle a snadno rozšiřovat aplikace o nové funkce a nástroje. To umožňuje zkrátit čas potřebný k vývoji webových aplikací a zároveň zvyšovat jejich kvalitu a spolehlivost [25].

7.6.1 Next.js

Next.js je open-source framework pro vývoj webových aplikací, který integruje vícero technologií. Využívá běhové prostředí Node.js pro serverovou část a React pro část klientskou. Je navržen tak, aby usnadnil a zrychlil vývoj

webových aplikací tím, že poskytuje širokou sestavu funkcí, nástrojů a optimalizací. Next.js dodává projektu strukturu a doporučené postupy, které využité technologie neposkytují. Umožňuje tak i u velkých a komplexních projektů udržet škálovatelnost, přehlednost, udržitelnost, strukturu a rychlost [36].

Jedna z hlavních výhod Next.js je integrace Node.js a React v rámci jednoho projektu, což umožňuje snadno vytvářet full-stack aplikace s vysokým výkonem a rychlostí využitím jednoho programovacího jazyka. Next.js navíc nabízí SSR (server-side rendering) a SSG (static site generation) [36].

SSR je technika, kdy se kód React aplikace provádí na serveru a výsledná HTML stránka se pošle na klienta. To umožňuje lepší výkon a SEO (search engine optimization), protože vyhledávače mohou indexovat kompletní HTML obsah stránky. SSR také umožňuje aplikaci rychle načíst a poskytuje uživatelským prohlížečům obsah, který mohou zobrazit okamžitě, zatímco se načítají další dynamické prvky aplikace [36].

SSG je technika, kdy se stránka vygeneruje v době kompilace a výsledný soubor se pošle na server. To znamená, že při každé návštěvě stránky není nutné vykonávat kód na serveru, což snižuje nároky na server a zvyšuje rychlost načítání stránky [36].

Next.js také nabízí automatickou optimalizaci výkonu, například také tím, že se zajišťuje, že pouze nezbytné JavaScriptové knihovny se načtou na stránku. To zvyšuje rychlost načítání stránky a zlepšuje uživatelskou zkušenost [36].

7.7 Porovnání technologií

V následující části je formou tabulek 7.1 a 7.2 prezentováno srovnání výše představených a popsaných technologií pro vývoj webových aplikací. Z důvodu omezeného prostoru jsou tabulky rozděleny do dvou částí, které porovnávají různé technologie. Tabulky zobrazují srovnání těchto technologií podle různých aspektů, jako jsou jazyk, licence, architektura, výkon a další.

Je třeba zdůraznit, že některé informace v tabulkách jsou zjednodušené a shrnuté, což je dáno omezeným prostorem a formátem tabulek. Tyto zjednodušené informace vycházejí z dřívějšího popisu technologií a podrobnější analýzy.

Aspekt	ASP.NET	Spring	Django
Jazyk	C#, VB.NET	Java	Python
Licence	Apache Licence	Apache Licence	BSD Licence
Vestavěný ORM	Entity Framework	Hibernate	Ano
Šablonovací jazyk	Razor	Thymeleaf, JSP	Django Template
MVC architektura	Ano	Ano	MVT
Výkon	Vysoký	Vysoký	Střední
Náročnost na naučení	Střední	Vysoká	Střední
Vestavěné bezpečnostní funkce	Ano	Ano	Ano
Škálovatelnost	Vysoká	Vysoká	Střední
Podpora API	Ano	Ano	Ano
Testovací nástroje	MSTest, NUnit, xUnit	JUnit, Mockito, TestNG	unittest, pytest, nose
Dependency Injection	Ano	Ano	Ne (možnost integrace s knihovnami)
Komunita	Velká	Velká	Velká

Tabulka 7.1: Porovnání technologií pro vývoj webových aplikací (1. část)

Aspekt	Ruby on Rails	Laravel	Next.js
Jazyk	Ruby	PHP	JavaScript, TypeScript
Licence	MIT Licence	MIT Licence	MIT Licence
Vestavěný ORM	ActiveRecord	Eloquent	Ne (možnost integrace s libovolným ORM)
Šablonovací jazyk	ERB, Haml	Blade	React
MVC architektura	Ano	Ano	Není vynucena, ale lze implementovat
Výkon	Střední	Střední	Vysoký (SSR, SSG, ISR)
Náročnost na naučení	Střední	Střední	Střední
Vestavěné bezpečnostní funkce	Ano	Ano	Základní (možnost rozšíření pomocí knihoven)
Škálovatelnost	Střední	Střední	Vysoká
Podpora API	Ano	Ano	Ano
Testovací nástroje	RSpec, MiniTest, Test::Unit	PHPUnit, Laravel Dusk	Jest, React Testing Library
Dependency Injection	Ne (možnost integrace s knihovnami)	Ano	Ne (možnost integrace s knihovnami)
Komunita	Velká	Velká	Velká

Tabulka 7.2: Porovnání technologií pro vývoj webových aplikací (2. část)

8 Cloud computing

Tato kapitola představuje pojem cloud computing a jeho roli v moderním světe informačních technologií. Popsány jsou jeho kategorie a příbuzné pojmy jako jsou serverless computing a edge computing.

8.1 Definice cloud computingu

Cloud computing představuje koncept zajišťování IT zdrojů a služeb skrze internetové prostředí, který umožňuje flexibilní, škálovatelné a často cenově efektivní řešení. Tento model zpravidla funguje na principu „pay-as-you-go“, což znamená, že zákazníkům jsou naúčtovány pouze využité prostředky a služby [4].

Namísto nákupu a správy potřebných prostředků, je možné využít cloudové služby nabízející výpočetní a úložné kapacity, síťovou infrastrukturu a běhová prostředí jež zprostředkovávají externí dodavatelé cloudových řešení. Mezi nejvýznamnější dodavatele cloudových služeb se řadí Amazon Web Services (AWS), Microsoft Azure a Google Cloud Platform [3].

Při vhodném zvolení a využití cloudových služeb lze dosáhnout řady výhod, jako například snížení časových a finančních nákladů potřebných na provoz a údržbu softwaru a hardwaru, urychlení vývoje a nasazení, dynamické škálování podle aktuálního provozu a zlepšení spolehlivosti a dostupnosti [4].

8.2 Typy cloud computingu

Cloud computing se dělí do tří hlavních kategorií: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) a SaaS (Software as a Service). Tyto třídy nabízejí různé úrovně kontroly, flexibility a možnosti správy, což umožňuje jejich efektivní využití pro širokou škálu potřeb [3].

IaaS

IaaS nabízí základní stavební kameny pro cloudové řešení, jako jsou síťové prvky, počítače (virtuální nebo na vyhrazeném hardwaru) a úložiště dat. Tento model nabízí nejvyšší úroveň kontroly, volnosti a flexibility. Přístup k nabízeným prostředkům je totiž podobný tradičnímu využívání vlastních výpočetních prostředků [3, 4].

PaaS

PaaS zjednodušuje správu fyzické infrastruktury a běhového prostředí. Je efektivnější pro většinu projektů bez speciálních požadavků na infrastrukturu, jelikož eliminuje nutnost provozovat a spravovat hardware, softwarové prostředí a další aspekty spojené s provozem aplikace [3, 4].

SaaS

SaaS nabízí kompletní produkt, který je provozován a spravován dodavatelem cloudového řešení. Klienti přistupují k aplikaci prostřednictvím webového API, desktopové aplikace nebo webového prohlížeče a mohou ji konfigurovat dle svých potřeb. SaaS přináší vysokou míru abstrakce a eliminuje potřebu zasahovat do vnitřní architektury a běhového prostředí aplikace. Často poskytuje výhody, jako je ochrana proti ztrátě dat, ale vše vychází z podmínek užití, které určuje poskytovatel [3, 4].

XaaS

XaaS (Anything as a Service) je obecný termín, který označuje širokou škálu služeb souvisejících s cloud computingem a vzdáleným přístupem. Zahrnuje různé produkty, nástroje a technologie, které jsou poskytovány jako služba přes internet, a to včetně IaaS, PaaS a SaaS modelů. XaaS může zahrnovat také další služby, jako jsou bezpečnost jako služba (Security as a Service, SECaaS), komunikace jako služba (Communication as a Service, CaaS) či monitorování jako služba (Monitoring as a Service, MaaS) [3, 4].

8.3 Serverless computing

Serverless computing (nazývaný také jednoduše serverless) je model cloud computingu, který se zaměřuje na eliminaci nutnosti správy backendové infrastruktury ze strany vývojářů. V serverless prostředí poskytovatel cloudu zajišťuje provozování, škálování, plánování a opravy infrastruktury [15].

Tento model spouští kód aplikace pouze na základě jednotlivých požadavků, což umožňuje automatické škálování infrastruktury nahoru a dolů v závislosti na počtu požadavků. Díky tomu zákazníci platí pouze za zdroje využívané v době, kdy aplikace běží, což minimalizuje náklady na nevyužitou kapacitu. Často se využívá pro stavově bezkontextové (stateless) aplikace, kde není potřeba udržovat stav mezi jednotlivými požadavky [15].

Serverless model přináší několik klíčových výhod, jako jsou snížené náklady na provoz, zjednodušená správa a možnost rychlého škálování. Je třeba

však zvážit i jeho omezení, jako jsou například potenciálně delší doba odezvy při prvním spuštění (tzv. cold start) nebo omezení v délce běhu funkcí [15].

FaaS

FaaS (Function as a Service) je podmnožina serverless computingu, která se zaměřuje na spouštění jednotlivých funkcí aplikace v reakci na konkrétní události. Umožňuje vývojářům psát a nasazovat malé, nezávislé kusy kódu, které reagují na události, jako jsou HTTP požadavky, změny v databázi nebo zprávy v systémech front. Vše kromě samotného kódu – fyzický hardware, operační systém virtuálního počítače a správa softwaru webového serveru – je automaticky zajišťováno poskytovatelem cloudových služeb v reálném čase v průběhu provádění kódu a po dokončení provádění je opět odpojeno. Účtování začíná v okamžiku zahájení provádění a končí v okamžiku ukončení provádění [15].

8.4 Edge computing

Edge computing je koncept, který se zaměřuje na přiblížení výpočetních prostředků ke zdroji dat s cílem snížit latenci a zvýšit efektivitu využití šířky pásma. Zjednodušeně řečeno, edge computing přesouvá část výpočetních procesů z centrálního cloudu na lokální místa, jako jsou uživatelská zařízení, IoT (Internet of Things) zařízení nebo geograficky blízké servery. Tím se minimalizuje množství komunikace na dlouhé vzdálenosti mezi klientem a serverem [7].

9 Webová bezpečnost

Kapitola se zaměřuje na hlavní témata a koncepty, které jsou nezbytné pro vytvoření bezpečné webové aplikace. Bezpečnost je klíčovým aspektem, který by měl být brán v úvahu při vývoji jakékoli aplikace. U webových aplikací je to obzvláště důležité, jelikož jejich snadná dostupnost prostřednictvím internetu zvyšuje riziko zneužití.

9.1 Úvod do webové bezpečnosti

Jedním ze základních pravidel webové bezpečnosti je využívání protokolu HTTPS, který poskytuje šifrovanou komunikaci mezi klientem a serverem. To minimalizuje riziko odposlechu citlivých dat a phishingových útoků [20]. Blíže byl protokol popsán v kapitole 4.3.1.

Další nezbytnou součástí bezpečnosti webové aplikace jsou principy autentizace a autorizace, včetně požadavků na uživatelská hesla a jejich uchovávání v bezpečné formě v databázi. Kromě toho je důležité implementovat ochranné praktiky proti konkrétním útokům, jako jsou XSS a CSRF. Je také nutné monitorovat vzniklé události a chod aplikace k identifikaci podezřelých aktivit. Tyto koncepty budou postupně přiblíženy v následujících kapitolách.

9.2 Autentizace

Autentizace je proces ověření totožnosti uživatele. K tomu se typicky využívá přihlašovací formulář vyžadující uživatelské jméno a heslo. Pro udržení platnosti přihlášení po dobu používání aplikace jsou v kontextu webových aplikací zpravidla využity autentizační tokeny, které jsou posílány v hlavičce HTTP požadavku, nebo prostřednictvím cookies [4, 14].

9.2.1 Metody autentizace

Nejpoužívanější metody autentizace pro webové aplikace využívají JWT (JSON Web Token) nebo uživatelské relace.

Uživatelská relace

Uživatelská relace zahrnuje server, který vytváří a udržuje záznam o relaci. Relační záznam je obvykle uložen v databázi a klient posílá identifikátor relace při každém požadavku. Server po obdržení požadavku vyhledá relaci v databázi a ověří její existenci a platnost [12].

Tento přístup je jednoduchý a přímočarý, ale může vyžadovat značnou režii, protože každý požadavek vyžaduje dotazování databáze pro nalezení relace. Navíc, relační autentizace je stavová, což znamená, že je v rozporu s principy RESTful architektury, která klade důraz na nezávislost jednotlivých požadavků [12].

JWT

JWT (JSON Web Token) ukládá veškeré potřebné informace (ID uživatele, role, atd.) v zašifrované formě přímo v tokenu, který není uložen v databázi, ale je poslán serverem zpět uživateli. Ten pro autentizaci musí token odeslat s každým požadavkem. Server token dekóduje a získá tak informace o uživateli [12].

Hlavní výhodou JWT je bezstavovost a eliminace potřeby ukládání nebo čtení informací z databáze, což zvyšuje rychlost aplikace a snižuje využití výpočetních zdrojů. Mezi nevýhody patří omezená možnost zneplatnění tokenu, proto je důležité přidělit tokenům krátkou platnost [12].

Ačkoli je možné vytvořit systém pro udržování seznamu neplatných tokenů v databázi a umožnit tak jejich okamžité zneplatnění, přináší to všechny nevýhody relačního řešení, jako je nutnost dotazování databáze při každém požadavku. Pokud je tedy nutné mít možnost okamžitého zneplatnění tokenů, je vhodnější použít relační řešení [12].

9.2.2 Politika a ukládání hesel

Silná hesla jsou jedním ze základních způsobů, jak ochránit uživatelské účty a zabránit neoprávněnému přístupu útočníka. Bezpečná aplikace by měla uživatele přinutit k vytváření silných hesel, které jsou těžko uhádnutelná a odolná proti různým typům útoků, jako jsou útok hrubou silou a slovníkové útoky. Toho lze dosáhnout pomocí minimální délky hesla, kombinace různých typů znaků, jako jsou malá a velká písmena, číslice a speciální znaky.

Ukládání citlivých údajů, jako jsou hesla, vyžaduje zvýšenou pozornost a obezřetnost. Hesla by nikdy neměla být uložena jako prostý text, ale místo toho by měla být uložena ve formě haše, což je výstup získaný pomocí speciálních hašovacích funkcí. Hesla tak nejsou snadno přístupná a viditelná pro

útočníky ani pro administrátory aplikace. Jedná se o jednu z nutných úrovní zabezpečení, zabraňující útočníkům zjistit hesla uživatelů, i když dokáží disponovat neoprávněným přístupem do databáze.

9.3 Autorizace

Autorizace je proces, který určuje, ke kterým prostředkům a akcím mají uživatelé přístup. Dobře navržená webová aplikace by měla mít centralizovanou autorizační logiku. Pokud jsou autorizační kontroly opakovaně implementovány v každém API rozhraní bez dodržení principu DRY (Don't Repeat Yourself), je aplikace náchylná na zranitelnosti [14, 20].

9.3.1 Metody autorizace

Mezi základní dvě metody pro určení práv uživatele se řadí RBAC a ABAC.

RBAC

RBAC (Role-Based Access Control) je jedním z nejběžnějších přístupů k autorizaci. Je založený na rolích, které mohou být přiřazeny uživatelům. Přístupová práva jsou pak udělována na základě těchto rolí. To umožňuje jednoduše spravovat přístupová práva pro skupiny uživatelů, místo toho, aby bylo nutné je spravovat individuálně pro každého uživatele.

ABAC

ABAC (Attribute-Based Access Control) je komplexnější a flexibilnější přístup k autorizaci, který umožňuje určovat přístupová práva na základě dynamických atributů uživatele, prostředků a dalších kontextových informací. ABAC umožňuje vytvářet podrobnější a více podmíněná přístupová pravidla, která se mohou lépe přizpůsobit různým scénářům a požadavkům na zabezpečení.

9.4 Webové bezpečnostní hrozby

V této kapitole budou popsány nejběžnější webové hrozby, které musí být řešeny v každé webové aplikaci.

9.4.1 XSS

XSS (Cross-Site Scripting) patří mezi nejrozšířenější zranitelnosti na internetu a postihuje webové aplikace, které provádějí skripty na straně uživatele v internetovém prohlížeči. XSS útoky se objevují již od počátku webu, ale s rostoucí uživatelskou interakcí v moderních webových aplikacích se stávají sofistikovanějšími a různorodějšími [14]. Pro ochranu webových aplikací proti XSS útokům je třeba zavést určitá opatření, mezi ty nejúčinější patří:

- **ošetření vstupních dat:** Filtrování a validace uživatelských vstupů za účelem odstranění potenciálně škodlivých částí kódu,
- **escapování speciálních znaků:** Použití bezpečných metod pro vkládání kódu do stránky, jako je například escapování speciálních znaků,
- **použití HTTP–only cookies:** Tímto způsobem se brání přístup ke cookies z klientského JavaScriptu a zamezuje tedy jejich zneužití,
- **implementace CSP (Content Security Policy):** CSP je hlavička HTTP, která omezuje možnost načítání kódu ze zdrojů mimo doménu a poskytuje tak další bezpečnostní opatření [14].

9.4.2 CSRF

CSRF (Cross-Site Request Forgery) je typ útoku na webové aplikace, při kterém útočník využívá přihlášení uživatele na cílovém webu a ze svého webu provede škodlivý požadavek. Útok využívá důvěryhodného vztahu mezi prohlížečem a webovou stránkou. Obvykle se provádí tak, že útočník vloží do své webové stránky odkazy nebo formuláře, které jsou vytvořeny tak, aby na pozadí odesílaly požadavky na API server webové stránky, kterou útočník chce napadnout [14, 20].

Tyto požadavky pak využívají ověřovací údaje uživatele, jako například relační token zasílaný v cookie, aby se zdálo, že požadavek pochází ze správné webové stránky a byl úmyslně zadán uživatelem. CSRF útoky jsou často velmi těžko odhalitelné uživatelem, protože se provádějí na pozadí bez jakýchkoli viditelných indikací [14]. Jako ochranu proti CSRF útokům lze použít následující opatření:

- **využití HTTP hlavičky pro autentizaci:** místo ukládání autentizačních informací v cookies je možné použít jiné mechanismy, jako například autentizační tokeny v hlavičce HTTP požadavků. Tyto tokeny nevkládá automaticky prohlížeč, a je nutné je udržovat a vkládat

do požadavků v klientské části aplikace. Na rozdíl od cookies tedy webová stránka útočníka nedokáže tyto tokeny odeslat v CSRF útoku,

- **CSRF tokeny:** CSRF token je náhodně generovaný řetězec, který je vložen do formuláře nebo odkazu na stránce. Při odeslání formuláře či kliknutí na odkaz se ověřuje CSRF token, čímž se zajišťuje, že požadavek pochází z oprávněného zdroje [14],
- **sameSite cookies:** SameSite cookie brání odesílání cookies z jiné domény, než ze které byl vytvořen. Tím zabraňuje CSRF útokům, neboť útočník nemůže odeslat požadavek s platným cookie na cílový server z podvodné stránky. Tato ochrana je ale závislá na správné implementaci funkcionality prohlížečem [14],
- **využití CORS:** CORS je bezpečnostní mechanismus, který umožňuje webovým aplikacím řídit, jaké zdroje z jiných domén mohou být přístupné (viz kapitola 4.3). CORS politiky na serveru mohou být nastaveny tak, aby se povolily pouze oprávněné domény, což zabraňuje CSRF útokům tím, že útočníkova stránka nebude moci odeslat požadavek na cílový server [14].

9.4.3 SQL Injection

SQL injection útoky jsou realizovány prostřednictvím vstupních polí a parametrů webových formulářů nebo URL adres, kde útočník vloží škodlivý SQL kód. Pokud aplikace neověřuje a nezpracovává vstupy správně, tento škodlivý kód může být začleněn do SQL dotazu, který se následně spustí na databázovém serveru. Útočník může tímto způsobem číst, upravovat nebo odstraňovat data, získávat přístup k citlivým informacím, změnit strukturu databáze nebo upravit očekávaný výstup dotazů [14, 20]. K zajištění ochrany proti SQL injection útokům je třeba dodržovat následující techniky:

- **parametrizované dotazy:** Používání parametrizovaných a předpřipravených dotazů, které omezují možnost vložení škodlivého kódu útočníkem. To často interně řeší ORM technologie a technologie pro vytváření dotazů bez nutnosti psaní SQL kódu tzv. „query builders“ [14],
- **escapování znaků:** Escapování speciálních znaků v uživatelských vstupech, aby nebylo možné zneužít těchto znaků k manipulaci s SQL dotazy [14].

9.4.4 DoS

DoS (Denial of Service) útok je typ útoku na počítačový systém nebo síť, jehož cílem je znemožnit nebo výrazně ztížit přístup k danému systému pro legitimní uživatele. Toho je dosaženo zahlcením systému nebo sítě žádostmi a vypětím zdrojů, jako jsou procesorový čas, paměť nebo síťová kapacita [14].

DoS útoky existují v mnoha podobách, od dobře známé distribuované verze (DDoS), která zahrnuje tisíce nebo více koordinovaných zařízení, až po DoS na úrovni kódu, který ovlivňuje jednoho uživatele. Útoky DoS se také liší v závažnosti, od pomalého načítání webové stránky až po úplné shození serverů a nedostupnosti aplikace [14].

Tomuto typu útoku se obtížně brání a není snadné otestovat, jak moc je aplikace a infrastruktura proti němu odolná. Kromě několika výjimek zpravidla DoS útoky nezpůsobují trvalé škody na aplikaci, ale ovlivňují použitelnost aplikace pro legitimní uživatele. Tím může snižovat důvěru a spokojenost uživatelů, v horším případě jim neplánovaný výpadek může učinit škody, a to i finanční [14, 20].

9.5 Monitorování aplikace

Monitorování je proces sledování a zaznamenávání informací o událostech, které se dějí při provozu aplikace, systému nebo sítě. Je nezbytnou součástí každé aplikace, která dbá na bezpečí dat a spokojenost svých uživatelů. Pro bezpečnost aplikace má klíčovou roli především v následujících aspektech:

- **detekce bezpečnostních incidentů:** Záznamy mohou odhalit pokusy o neoprávněný přístup, škodlivou činnost nebo chyby v systému, které by mohly vést k ohrožení bezpečnosti. Při včasné detekci a rychlé reakci lze minimalizovat dopad na uživatele a chránit reputaci aplikace,
- **zjišťování zdroje incidentu:** Záznamy mohou poskytnout důkazy o tom, kdo nebo co způsobilo bezpečnostní incident, což umožňuje přijmout adekvátní opatření,
- **monitorování výkonu aplikace:** Záznamy umožňují sledování výkonu aplikace a odhalení možných problémů, které by mohly vést ke zhoršení použitelnosti nebo k selhání aplikace.

10 Návrh uživatelského rozhraní webové aplikace

Kapitola popisuje základní koncepty a praktiky při návrhu uživatelského rozhraní tak, aby bylo intuitivní a efektivní pro dosažení uživatelských potřeb. Je zde přiblížen proces návrhu uživatelského rozhraní, termín uživatelská zkušenost, grafický a responzivní design a v závěru je popsána problematika webové přístupnosti.

10.1 Proces návrhu uživatelského rozhraní

Uživatelské rozhraní (User Interface, UI) je klíčovým prvkem umožňující interakci mezi uživatelem a aplikací. Návrh uživatelského rozhraní zahrnuje vytvoření vizuálních prvků, navigačních struktur a interakcí. Při návrhu je důležité brát v úvahu povahu, potřeby a očekávání uživatelů. To zahrnuje pochopení, jak a jací uživatelé k aplikaci přistupují, jaké úkony chtějí provést, jaké jsou jejich dovednosti a jaké informace potřebují ke splnění svých cílů. Proces návrhu uživatelského rozhraní tedy začíná identifikací potřeb uživatelů a navrhnutí obrazovek a vztahů mezi nimi, které tyto potřeby naplňují. Tyto praktiky mají zásadní vliv na uživatelský zážitek (UX). Následuje grafický design, který se zabývá konkrétním vzhledem a působením jednotlivých prvků a celé aplikace. Tyto koncepty jsou blíže popsány v následujících kapitolách.

10.2 Uživatelská zkušenost

Uživatelská zkušenost (User Experience, UX) zahrnuje celkový dojem uživatele z interakce s aplikací a jejím využitím k dosažení určitého cíle. UX je velmi důležitý faktor pro úspěch aplikace, protože uživatelé mají tendenci využívat aplikace, které jsou snadné a intuitivní [42].

Proces UX návrhu zahrnuje několik kroků. Prvním krokem je analýza uživatelů a jejich potřeb, cílů a očekávání. Druhým krokem je návrh vhodné struktury a pracovních postupů v aplikaci, vycházejících z informací získaných v předchozím kroku. Třetím krokem je tvorba wireframů[1] a prototypů,

[1] **Wireframe** je zjednodušený náčrt webové stránky nebo aplikace bez grafického designu a detailů, který slouží k vizualizaci a testování základních strukturálních prvků.

které umožňují testování aplikace v raných fázích vývoje a tím získání zpětné vazby. Čtvrtým krokem je implementace a testování v reálném prostředí. Posledním krokem je průběžné vylepšování a aktualizace aplikace na základě zpětné vazby uživatelů z produkčního prostředí [42].

10.3 Grafický design

Grafický design vychází z navržené struktury a snaží se o atraktivní a efektivní vzhled zapadající do vizuálního stylu a kontextu aplikace. Je důležitý pro působení a celkový první dojem aplikace na uživatele. Součástí grafického designu je výběr tvarů, barev, typografie, ikon a dalších grafických prvků. Vhodně zvolený grafický styl je také klíčový pro intuitivního a přehledného UI [42].

10.4 Responzivní design

Responzivní design je přístup k návrhu uživatelského rozhraní umožňující přizpůsobení aplikací různým velikostem a rozlišením obrazovek. V dnešní době se webové aplikace používají na široké škále zařízení, proto je korektní responzivita UI klíčová. Často se v tomto kontextu prosazuje koncept „mobile first“, který stanovuje, že návrh uživatelského rozhraní by měl vycházet především z mobilního rozvržení a poté se rozšiřovat na větší obrazovky [24].

10.5 Webová přístupnost

Webová přístupnost znamená, že webová stránka nebo aplikace jsou navrženy tak, aby byly přístupné pro všechny uživatele, včetně těch se zdravotním postižením, jako jsou například osoby se sluchovým či zrakovým postižením, s omezenou pohyblivostí nebo s kognitivními obtížemi. Existují tři úrovně přístupnosti: A, AA a AAA. Každá úroveň specifikuje určité požadavky, které musí být splněny, aby stránka mohla spadat do této kategorie. Při návrhu uživatelského rozhraní je dobré brát ohled na webovou přístupnost od samého začátku. Tento přístup se nazývá „accessibility by design“ [24, 39].

11 Nasazení webové aplikace

Kapitola se zaměřuje na klíčové pojmy a témata, které je nutné znát, pro úspěšné a efektivní uvedení webové aplikace do produkce. Nejprve je vysvětlen proces nasazení a popsány různé možnosti hostování webové aplikace. Dále je věnována pozornost technologiím, které se v tomto kontextu používají. Posledním tématem je zautomatizování procesu nasazení pomocí praktik a nástrojů DevOps.

11.1 Proces nasazení webové aplikace

Nasazení webové aplikace je proces, při kterém se aplikace přenáší z vývojového prostředí do prostředí, kde je k dispozici pro testery nebo samotné uživatele. Tento proces zahrnuje několik důležitých kroků, jako je sestavení aplikace, testování, přenos instalace, konfigurace, spuštění a monitorování aplikace. Nasazení může být realizováno ručně nebo automatizováno pomocí nástrojů a technik, které usnadňují a urychlují tento proces. Cílem nasazení je zajistit, že aplikace bude fungovat korektně a stabilně v produkčním prostředí a bude k dispozici uživatelům.

11.2 Prostředí pro hostování aplikace

Jedním z tradičních způsobů hostování webových aplikací je provozování na vlastním serveru. Tento přístup je vhodný v případě dostatku technických zdrojů a schopností provozovat a spravovat serverovou infrastrukturu. Výhodou je plná kontrola nad hardwarovými i softwarovými zdroji a zabezpečením. Nicméně provozování vlastního serveru může být nákladné, jelikož je potřeba pořídit a udržovat hardware, zajišťovat chlazení a energii pro provoz serveru, a zároveň zajistit správu a údržbu serveru. Navíc, škálování aplikace může být obtížné a nákladné [32].

Další možností pro nasazení webové aplikace je využití cloudu (viz kapitola 8). Při volbě této možnosti je důležité zohlednit několik faktorů, jako je volba poskytovatele, výběr typu cloudového řešení, náklady na provoz a bezpečnost. Nabízená cloudová řešení se liší v aspektech, jako jsou cenová dostupnost, údržba, konfigurovatelnost, škálovatelnost a dostupnost. IaaS poskytuje organizacím přístup k hardwarovým prostředkům a umožňuje tak odstínit starosti s jejich správou. PaaS na druhé straně poskytuje hotovou

platformu pro nasazení a umožňuje tak proces nasazení pro vývojáře téměř plně odstínit. Nasazení v cloudu přináší řadu výhod, jako je snadná škálovatelnost a nízké nároky na údržbu. Nabízí také flexibilní finanční náklady dle využitých zdrojů. Je ale potřeba dát pozor, z důvodu komplexity a počtu poskytovaných služeb nemusí být cloudové služby dlouhodobě finančně výhodné. Navíc zahrnují menší kontrolu nad hardwarovými zdroji, softwarovým prostředím, bezpečností a dostupností [32].

11.3 Kontejnerizace

Kontejnerizace zjednodušuje vývoj a nasazení aplikací prostřednictvím kontejnerů, které poskytují definovaná a izolovaná prostředí. Díky tomu se vývojáři mohou soustředit na samotnou aplikaci a nemusí řešit problémy, které by mohly být způsobeny různými konfiguracemi systému či nesprávně nainstalovanými závislostmi. Vývojový proces je tak efektivnější a rychlejší, což umožňuje dosáhnout výsledků v kratším čase [21].

Vývoj webových aplikací s využitím kontejnerů je v praxi často prováděn pomocí technologie Docker. Tato technologie poskytuje prostředky pro vytváření, nasazení a spouštění aplikací v kontejnerech, které obsahují vše potřebné pro jejich běh [21].

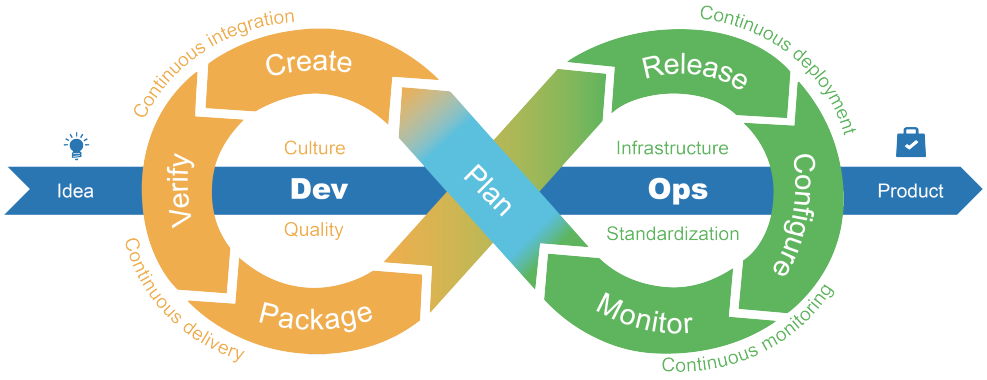
S kontejnerizací je také spojen pojem orchestrace, jehož cílem je automatizovat a zefektivnit správu životního cyklu kontejnerů, od jejich vytvoření, spuštění, škálování až po jejich ukončení. Orchestrace také řeší problémy, jako jsou zotavení z chyb, load balancing a sdílení zdrojů mezi více kontejnery. Mezi nejznámější a nejpoužívanější nástroje pro tyto účely patří Docker Compose a Kubernetes [16].

11.4 DevOps

DevOps (Development and Operations) je koncept, který se zaměřuje na sjednocení vývoje a provozu softwarových aplikací, což má za cíl zrychlit a zefektivnit dodávání softwaru. Tato filosofie vychází z agilního vývoje, který klade důraz na flexibilitu, rychlou adaptaci na změny a spolupráci mezi týmy. Ke splnění těchto cílů se často využívají automatizované nástroje. Kapitola představuje základní koncepty, kterými jsou continuous integration, continuous delivery a continuous deployment. V závěru bude přiblížena CI/CD pipeline jakožto nástroj pro automatizaci tohoto procesu [8, 21].

Smyčka zpětné vazby (viz obrázek 11.1) je klíčovým prvkem tohoto konceptu. Zahrnuje průběžné získávání zpětné vazby od uživatelů a její zapra-

cování do aplikace. Tím se zajišťuje neustálé zlepšování a přizpůsobování aplikace potřebám uživatelů [8, 21].



Obrázek 11.1: DevOps smyčka zpětné vazby (Zdroj: `https://almbok.com/devops/devops`)

11.4.1 Continuous Integration

CI (Continuous Integration) je klíčový proces v rámci DevOps, který má za cíl minimalizovat riziko chyb a zvýšit efektivitu vývoje softwaru. CI spočívá v průběžné integraci a testování změn v kódu, což zajišťuje, že aplikace je neustále ověřována a validována. Tento proces zahrnuje sestavení kódu, spuštění testů a ověření správného fungování aplikace [8, 21].

Automatizované procesy hrají klíčovou roli v CI, protože umožňují provádět integraci často a včas. Díky častému testování jsou chyby odhaleny brzy, což snižuje nároky na jejich opravu a zlepšuje celkovou kvalitu aplikace. To také usnadňuje identifikaci a řešení problémů, protože vývojáři vidí konkrétní změny, které zapříčinily negativní výsledky testování. Pro úspěšné fungování CI je nezbytné, aby vývojáři psali testy již během vývoje funkcí a přizpůsobovali je změnám aktuálních požadavků [8, 21].

11.4.2 Continuous Delivery

CD (Continuous Delivery) je proces, který má za cíl minimalizovat časovou prodlevu mezi vytvořením kódu a jeho nasazením. CD staví na základech CI, jehož výstupem je sada artefaktů, včetně sestavené aplikace. Cílem CD je zajistit efektivní a bezpečný způsob, jak rychle dodávat sestavenou aplikaci koncovým uživatelům ve funkčním stavu [8, 21].

Četnost nasazování v CD závisí na konkrétních požadavcích projektu a zákazníka, ale obecně se preferuje časté a pravidelné nasazování. Tento

přístup zajišťuje rychlé opravy chyb, umožňuje rychle reagovat na zpětnou vazbu uživatelů a zvyšuje spokojenost s aplikací [8, 21].

11.4.3 Continuous Deployment

Continuous Deployment je koncept, který staví na základech Continuous Delivery, ale rozšiřuje jej o plnou automatizaci procesu nasazení do produkčního prostředí. Na rozdíl od Continuous Delivery, kde je k nasazení do produkce stále nutná manuální akce, Continuous Deployment automaticky nasazuje nové verze aplikace bez jakékoliv lidské interakce. Tento přístup nabízí vyšší stupeň agilního vývoje a přináší rychlejší zpětnou vazbu od koncových uživatelů [8].

Při plné automatizaci nasazení je zásadní důraz na kvalitní testování, které musí důkladně ověřit funkčnost aplikace před procesem nasazení. Pokud je nalezena jakákoliv chyba, proces se musí přerušit a zabránit tak poskytnutí nefunkční verze uživatelům. To vyžaduje vysoce spolehlivé a komplexní testovací nástroje a postupy, které zahrnují jednotkové, integrační a systémové testy [8].

11.4.4 CI/CD Pipeline

CI/CD pipeline je soubor automatizovaných procesů, který spojuje Continuous Integration a Continuous Delivery či Continuous Deployment. Tyto procesy se skládají z několika fází, které zahrnují sestavení, testování a nasazení aplikace. Fáze pipeline jsou prováděny postupně a mohou obsahovat více jednotlivých kroků, které mohou probíhat paralelně. Pokud fáze skončí chybou, pipeline se zastaví a nepokračuje na další fáze [8, 21].

Existuje mnoho nástrojů a technologií, které se používají pro implementaci CI/CD pipeline, jako jsou Jenkins, GitLab CI, Circle CI a GitHub Actions. Pipeline bývají často propojeny s verzovacím systémem, například git repositářem. Ty následně nabízejí funkce, jako například automatické sloučení změn z vedlejší do hlavní větve, v případě že ve větvi úspěšně prošla celá pipeline [17].

Spouštění pipeline může být manuální nebo automatické. Automatické spouštění pipeline může být vyvoláno různými událostmi, jako je push příkaz, vytvoření pull/merge requestu, vytvoření nového tagu nebo dokončení jiné pipeline. V rámci projektu je obvykle více pipeline pro různé účely, které reagují na různé události [17].

12 Analýza existujících řešení

Tato kapitola se zaměřuje na analýzu existujících řešení, které poskytují frekvenční terapii nebo související terapeutické metody v digitálním prostředí. Cílem je identifikovat možné konkurenty a inspirovat se jejich funkcemi a nedostatky, aby bylo možné navrhnout co nejefektivnější a uživatelsky přívětivou aplikaci. Přestože během hledání nebylo nalezeno žádné řešení, které by přímo konkurovalo navrhované aplikaci, bylo identifikováno několik existujících kategorií konkurenčních řešení, které využívají formy frekvenční terapie nebo muzikoterapie.

V digitálním prostředí je muzikoterapie nejčastěji provozována formou nahrávek a zvukových stop dostupných prostřednictvím populárních streamovacích platforem, jako jsou YouTube a Spotify. Typicky se jedná o různé formy binaurálních rytmů, které mohou být kombinovány s hudebními prvky a dalšími frekvencemi.

Struktura kapitoly zahrnuje přehled a analýzu podobných řešení a identifikaci jejich nedostatků a možností zlepšení. V závěru kapitoly budou shrnuty výsledky analýzy a nastíněny potenciální možnosti pro vývoj navrhované webové aplikace, která by zohlednila identifikované silné a slabé stránky stávajících řešení.

12.1 Existující řešení

Při analýze existujících řešení v oblasti zvukové terapie bylo identifikováno několik kategorií hotových řešení s různými účely a cíli, které se zabývají odlišnými aspekty terapeutického využití zvuků a frekvencí a mohou být relevantní pro vývoj navrhované webové aplikace. Tyto kategorie zahrnují zvukové stopy s binaurálními rytmy, AVS a podobná zařízení a aplikace pro léčbu tinnitusu, podporu mindfulness a meditační účely. Každá z těchto kategorií představuje řešení.

12.1.1 Binaurální rytmy

Binaurální rytmy představují zvukový fenomén, který vzniká, když se do každého ucha přehrávají různé frekvence zvuku. Mysl zpracovává tyto frekvence a vnímá iluzi třetího rytmu, který se nazývá binaurální rytmus. Uživatelé využívají binaurální rytmy k dosažení hlubokého stavu relaxace, snížení

stresu a úzkosti, zlepšení schopnosti soustředění, povzbuzení kreativity nebo podpory spánku.

Binaurální rytmy se staly na internetu velmi populárními a jsou dostupné prostřednictvím široké škály platforem, které nabízejí připravené zvukové stopy zaměřené na dosažení určitých stavů a cílů.

I když existuje mnoho příznivců binaurálních rytmů, je třeba zmínit, že vědecké důkazy o jejich účinnosti nejsou jednoznačné. Některé studie naznačují, že binaurální rytmy mohou mít pozitivní účinky na některé aspekty duševního zdraví a výkonu, zatímco jiné studie tyto nálezy nepotvrzují. Navíc je z široké škály dostupných stop obtížné rozlišit mezi kvalitními a efektivními zvukovými stopami.

12.1.2 AVS a podobná zařízení

AVS zařízení (audiovizuální stimulace) představují přistroje, které se snaží stimulovat mozek prostřednictvím zvukových a vizuálních signálů. Cílem těchto zařízení je synchronizovat mozkové vlny s určitou frekvencí, aby se dosáhlo žádoucího stavu mysli. AVS zařízení obvykle zahrnují sluchátka a speciální brýle s LED diodami, které blikají v určitém rytmu a intenzitě. Kromě AVS zařízení existují také různé variace těchto technologií zaměřující se na jiné smysly, jako je například vibroakustická stimulace.

Jedním z hlavních nevýhod AVS zařízení je, že rozhodování o tom, které programy použít a jak je použít, je často přímo na uživateli, který v této oblasti nemusí mít dostatečné znalosti. To může vést k suboptimálním výsledkům a nedostavení potenciálních efektů.

Dalším problémem spojeným s AVS zařízeními je jejich cena. Tyto přístroje jsou často velmi drahé, což omezuje jejich dostupnost pro běžné uživatele. Navíc vyžadují nosit přístroj sebou, což může být pro některé uživatele nepraktické a nepohodlné.

12.1.3 Aplikace pro léčbu tinnitusu

Tinnitus je zdravotní stav, který se projevuje vnímáním zvuků, které nejsou přítomny v okolním prostředí. Často se jedná o vysoký pískot, hučení nebo zvonění. Tinnitus může být způsoben různými faktory, jako jsou hluková trauma, stres, zánět středního ucha nebo Meniérova choroba. Léčba tinnitusu je často složitá a zahrnuje farmakologické i nefarmakologické přístupy [13, 31, 37].

Aplikace pro léčbu tinnitusu představují jeden z nefarmakologických přístupů k řešení tohoto problému. Tyto aplikace, dostupné ve formě webových

nebo mobilních aplikací, poskytují uživatelům možnost nastavit a vybrat požadovanou frekvenci zvuku, která by měla pomoci tlumit nebo maskovat tinnitus. Princip léčby spočívá v identifikaci frekvence tinnitusu u pacienta a následném přehrávání tohoto zvuku nebo zvuků, které mohou pacientovi poskytnout úlevu.

Ačkoli aplikace pro léčbu tinnitusu představují zajímavé využití frekvenční terapie, jejich cíle se liší od aplikace vyvíjené v rámci této diplomové práce. Je tedy možné, se těmito aplikacemi inspirovat pouze částečně.

12.1.4 Mindfulness a meditační aplikace

Mindfulness a meditační aplikace se stávají čím dál populárnějšími možnostmi pro zlepšení psychického a emočního stavu. Tyto aplikace, dostupné na mobilních platformách jako iOS a Android, poskytují uživatelům různé techniky a nástroje pro procvičování mindfulness, meditace a relaxace, které mohou pomoci snížit stres, zlepšit soustředění a podpořit kvalitní spánek.

Meditační sezení v těchto aplikacích zpravidla zahrnuje průvodce, který uživatele instruuje a vede skrze různé techniky, jako jsou dechová cvičení, soustředění na uvolnění částí těla a vizualizace. Aplikace často nabízí různé zvukové pozadí, jako zvuky moře, lesa nebo deště, které se snaží uklidnit mysl a podpořit relaxaci. Ačkoli tyto aplikace poskytují řadu nástrojů pro zlepšení psychického a emočního zdraví, jejich zaměření je obecnější a nejsou specificky navrženy pro odbornou léčbu pomocí specifických frekvencí.

12.2 Identifikace nedostatků nalezených řešení

Při analýze existujících řešení bylo zjištěno několik převažujících nedostatků, které ovlivňují efektivitu a spolehlivost frekvenční terapie nebo souvisejících terapeutických metod.

Kvalita a vědecký základ

Většina dostupných řešení postrádá pevný vědecký základ, a tak nedokáže prokázat svou účinnost. Přesto se však tyto produkty často prezentují pro své pozitivní účinky na psychické zdraví. Amatérské projekty a nezávislé iniciativy často nedostávají adekvátní odbornou podporu a spolupráci s odborníky z oblasti zdravotnictví a výzkumu, což může vést ke snížení důvěryhodnosti a kvality těchto řešení. Tyto neodborné projekty mohou takovým způsobem přinášet více škod než užitku a dávat uživatelům falešné naděje.

Spolupráce s terapeutem

Současná řešení neposkytují možnost vytvoření a upravování terapie individuálním potřebám pacienta na základě spolupráce s odborným terapeutem. Řešení, která by umožňovala těsnější spolupráci mezi terapeuty a pacienty, by mohla zlepšit účinnost léčby a přispět k lepším výsledkům terapie.

Přístupnost a použivatelnost

Existující řešení mohou být obtížně dostupná, neintuitivní nebo nekompatibilní s různými platformami a zařízeními. Uživatelé mohou mít potíže s nalezením a použitím těchto řešení, což snižuje jejich motivaci k pravidelnému využívání. Vývoj webové aplikace s intuitivním a přehledným uživatelským rozhraním, které je kompatibilní s širokou škálou zařízení, by zlepšilo přístupnost a snadnost použití.

12.3 Závěr analýzy nalezených řešení

Během analýzy existujících řešení nebyla nalezena žádná aplikace zaměřená na frekvenční terapii, která by představovala přímou konkurenci té, navržené v této diplomové práci. Toto zjištění zdůrazňuje potřebu takového řešení pro zdravotníky, terapeuty a pacienty.

Terapie specifickými frekvencemi je poměrně novým a málo probádaným oborem v rámci muzikoterapie, což může vysvětlovat absenci takových aplikací na trhu. Nicméně, vzhledem k rostoucí popularitě a diskusím kolem této disciplíny, lze očekávat, že zájem odborníků v oblasti zdravotnictví a terapeutů o podobné aplikace naroste.

Mnoho existujících řešení pro zvukovou terapii skrze digitální prostředí není založeno na pevném vědeckém základě nebo odborném výzkumu, což omezuje jejich účinnost a důvěryhodnost. Vytvořená aplikace by měla být široce dostupná a založena na aktuálních výzkumných poznatcích v oblasti frekvenční terapie a muzikoterapie. Současná řešení často neposkytují možnost přizpůsobení frekvencí individuálním potřebám pacienta na základě spolupráce s odborným terapeutem. Navíc, AVS a podobná zařízení jsou často nákladná a nepraktická.

13 Návrh aplikace

Kapitola se zaměřuje na návrh řešení a další praktiky podporující úspěch projektu. Úvodní část se věnuje tomu, jak byl tento proces konzultován s odborným terapeutem a je představen konkrétní problém v rámci muzikoterapie a jeho navržené řešení prostřednictvím webové aplikace. Následně je popsána doména muzikoterapie s využitím terapeutických nástrojů se zaměřením na definice klíčových entit a jejich vzájemných vztahů. Další kapitola analyzuje jednotlivé aktéry a jejich cíle v kontextu aplikace. Poté následuje sekce, která specifikuje funkční a mimofunkční požadavky na aplikaci, jež byly identifikovány na základě analýzy potřeb uživatelů a domény muzikoterapie. Následně jsou popsána potenciální rizika a omezení související s vývojem aplikace, což pomáhá vytvořit plán pro jejich řešení nebo minimalizaci dopadu. V závěru kapitoly je popsán proces návrhu uživatelského rozhraní, jeho struktura a klíčové prvky.

13.1 Spolupráce s odborným terapeutem

Celý proces návrhu byl prováděn ve spolupráci se zkušeným terapeutem Bc. Richardem Frouzem MBA, který vystudoval fyzioterapii na Západočeské univerzitě v Plzni a dále MBA studium se zaměřením na psychologická témata. Má praxi v oblastech muzikoterapie a frekvenční terapie, s využitím terapeutických ladiček, a také celostní terapie (zahrnující fyzioterapii, výživu a duševní zdraví). Dále má zkušenosti s terapií dětí s poruchou autistického spektra a vedením párové psychoterapie s jejich rodiči.

Ve spolupráci s terapeutem byla diskutována většina témat, která jsou popsána v rámci této sekce. Jeho odborné znalosti a zkušenosti z domény terapie a využití zvukových frekvencí k terapeutickým účelům byly zásadní pro specifikaci požadavků a návrh aplikace. Tato spolupráce umožnila vytvořit kvalitní návrh a prototyp, což pomohlo omezit míru změnových požadavků během vývoje. Přesto byla aplikace už od raných fází vývoje pravidelně prezentována a validována terapeutem, což umožnilo včasné upřesňování směru vývoje.

13.1.1 Požadavky terapeuta

Mezi požadavky terapeuta se řadilo analyzovat existující řešení a dále implementovat webovou aplikaci s následujícími aspekty:

- pozvánky do aplikace využitím emailů,
- nahrávání terapeutických zvuků,
- organizace terapeutických zvuků využitím balíků,
- vytváření komplexní zvukové terapie využitím série opakujících se terapeutických zvuků,
- nastavení názvu terapie,
- omezení platnosti terapie,
- omezení počtu přehrání terapie klientem za jeden den,
- snadné přehrání terapie klientem,
- monitorování využívání aplikace klientem,
- předvytváření playlistů využitím šablon,
- deaktivace terapeutů.

Požadavky na aplikaci jsou využitím následné analýzy detailněji a strukturovaněji popsány v kapitole 13.4.

13.2 Popis problému a navrženého řešení

Tato část se zaměřuje na vysvětlení problematiky, kterou má aplikace řešit, a následně navrhuje vhodné řešení skrze webovou aplikaci.

13.2.1 Definice problém

Problém je spojen s muzikoterapií a využitím terapeutických nástrojů (především terapeutických ladiček). Efektivní léčba pomocí zvukových frekvencí vyžaduje pravidelný poslech, a to i několikrát denně. Avšak tak časté návštěvy u terapeuta jsou obtížně realizovatelné jak z pohledu terapeuta, tak i jeho klientů (především z časových a finančních důvodů).

Jedním z možných řešení by bylo individuální zakoupení potřebných nástrojů a následné provozování terapie samostatně. Toto řešení však není dostupné pro většinu zájemců, jelikož například terapeutické ladičky jsou často nákladné a jejich správné použití vyžaduje zkušenosti a praxi. Člověk se při terapii potřebuje uvolnit a současná manipulace s terapeutickým nástrojem může být rozptylující. Z těchto důvodů je nutné najít alternativní řešení, které by bylo dostupné a vhodné pro širší spektrum zájemců.

13.2.2 Návrh řešení

Navrženým řešením stanoveného problému je webová aplikace umožňující snadný přístup k personalisovaným zvukovým terapiím vytvořených odbornými terapeuty. Cílem aplikace není kompletně nahradit osobní terapii s terapeutem, ale spíše ji doplňovat a podporovat. Aplikace by měla sloužit jako platforma pro terapeuty, kteří zde budou moci vytvářet a spravovat individuální rozšiřující terapie pro své klienty, určené pro užití bez přítomnosti terapeuta ze zařízení klienta (chytrá mobilní zařízení, osobní počítače či notebooky). Spolupráce mezi terapeutem a uživatelem bude v aplikaci vyžadována a aplikace nebude obsahovat žádné předpřipravené zvukové stopy, které by mohli uživatelé využít bez konzultace s odborníkem v této doméně. Tyto opatření budou zavedeny pro zachování odbornosti a kvality poskytovaných služeb.

V rámci aplikace bude moci existovat více nezávislých terapeutů, každý s vlastní klientelou. Zvuková terapie bude především zahrnovat sérii zvukových frekvencí, převážně z nahraných terapeutických ladiček. Avšak aplikace nebude striktně omezena pouze na tyto zvuky; mohou být nahrány jakékoliv potřebné zvukové stopy, které považuje terapeut za vhodné pro své klienty.

Výsledkem navrhovaného řešení by mělo být usnadnění a zkvalitnění služeb frekvenční terapie, zejména s terapeutickými ladičkami a podobnými nástroji, zvýšení její dostupnosti pro širší spektrum uživatelů a podpora pravidelného používání, které je klíčové pro její účinnost. To by mělo přispět k vyšší spokojenosti s průběhem a výsledky léčby.

13.3 Popis domény

Tato kapitola se věnuje problematice domény muzikoterapie, zaměřené na využití různých nástrojů, jako jsou například terapeutické ladičky. Pojem muzikoterapie a proces využití terapeutických ladiček byly již popsány v kapitole 3. Zde se konkrétně zaměřujeme na modelování procesu vytváření specializovaných terapeutických zvukových stop terapeuty pro jednotlivé klienty z pohledu vývojáře.

13.3.1 Glosář

Tato sekce se věnuje zavedení pojmů a jejich vztahů specifických pro doménu, které jsou využívány v rámci vývoje a návrhu aplikace. Slouží jako základ pro datové modelování a následné vytvoření databáze ukládající strukturovaná data.

Frekvence

Frekvence označuje jednu konkrétní nahranou zvukovou stopu. V kontextu terapeutických ladiček se jedná o nahraný frekvenční průběh jedné ladičky od rozeznění až po její úplné doznění. Každá frekvence spadá do určitého balíku.

Balík

Balík shromažďuje libovolné množství frekvencí a slouží především k usnadnění vyhledávání a třídění jednotlivých frekvencí. Koncept je užitečný zejména pro terapeuty, kteří se potřebují zorientovat v dostupných frekvencích například při vytváření playlistu pro klienta.

Playlist

Playlist je základním prvkem terapie a skládá se ze série libovolného počtu frekvencí, které mají pevně definované pořadí. Jednotlivé frekvence se často přehrávají vícekrát, a poté se přechází na další frekvenci. Playlist je vytvořen terapeutem, který na základě potřeb klienta vybírá vhodné frekvence a určuje jejich pořadí. Po přihlášení do aplikace může klient poslouchat tento pro něj na míru vytvořený playlist jako ucelenou zvukovou stopu.

Terapie

Terapie se skládá z playlistu a dalších informací, jako jsou název terapie, počet poslechů za den a datumové rozmezí, po které je terapie platná. Každý klient má právě jednu terapii, kterou mu vytváří a upravuje odpovídající terapeut.

Šablona

Šablony usnadňují práci terapeutům, protože se jedná o uložené předvytvořené playlisty. Tyto šablony si může vytvářet každý terapeut a vhodně pojmenovat. Následně při vytváření terapie pro klienta může terapeut šablonu importovat, čímž urychlí a zefektivní svou práci. Šablony také slouží pro uchování konkrétních znalostí o tom, jak přistupovat k podobné kategorii klientských problémů.

13.3.2 Analýza aktérů a jejich cílů

Analýza aktérů je zásadním krokem při návrhu aplikace. Je nezbytné identifikovat a analyzovat jednotlivé aktéry, aby bylo možné vytvořit aplikaci,

která uspokojí potřeby všech uživatelů. V rámci analýzy byli identifikováni tři klíčoví aktéři, a to administrátor, terapeut a klient. Každý z nich má odlišnou motivaci pro používání aplikace a skrze ní usiluje o dosažení různých cílů. Na základě těchto rozdílů jsou jim přiřazeny specifické funkce a oprávnění v aplikaci. Je důležité vzít v úvahu rozmanité znalosti, zkušenosti a dovednosti jednotlivých skupin uživatelů a přizpůsobit jim rozhraní a funkce aplikace.

Administrátor

Administrátor je jednotlivec, který rozumí problematice, kterou aplikace řeší, a je obeznámen s jejím celkovým fungováním. Jeho hlavním úkolem je správa uživatelů a dat v aplikaci. Počet administrátorů v aplikaci je malý (1 – 3). Administrátoři komunikují s terapeuty, přidávají nové terapeuty do systému, a zodpovídají za přidávání, úpravy a mazání frekvencí v aplikaci. Jejich cílem je udržet přehled o dění v aplikaci a zajistit její bezpečnost a bezproblémový chod.

Terapeut

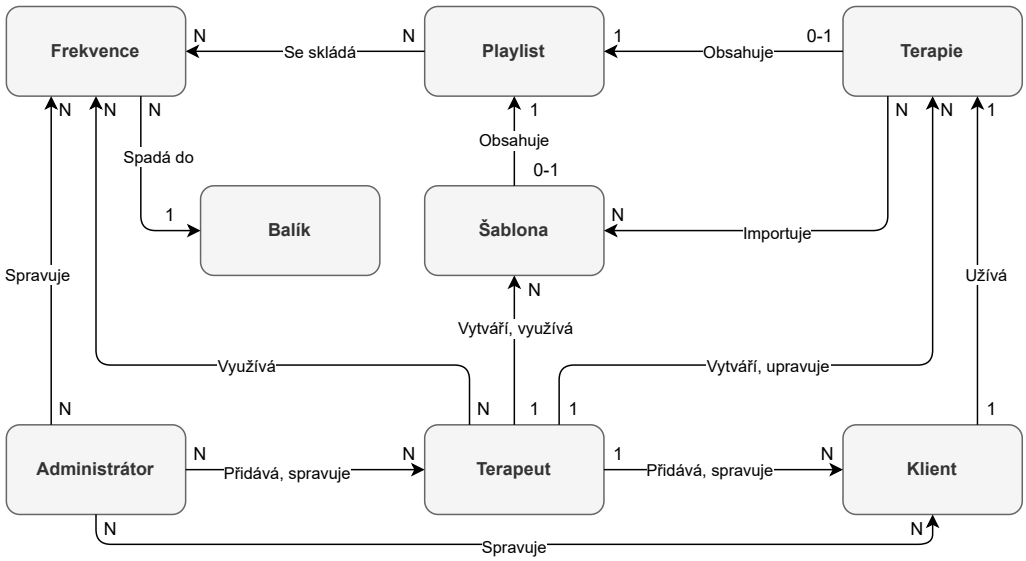
Terapeut je odborník v oblasti muzikoterapie, který využívá aplikaci pro vylepšení služeb vůči svým klientům. Nemusí však být zkušený v používání technologií a webových aplikací. Terapeut má typicky několik klientů, se kterými provozuje i kontaktní terapii, a může je do aplikace přidávat či odebírat. Na základě konzultací s klienty vytváří a upravuje zvukové terapie v aplikaci. Terapeut má přístup k frekvenčním stopám, ale nemůže je editovat. Může si vytvářet a používat vlastní šablony. Jeho hlavním cílem je zvýšit kvalitu svých služeb a tím spokojenost svých klientů. Předpokládá se možnost existence více terapeutů v rámci aplikace, každý se svojí odlišnou skupinou klientů.

Klient

Klient je osoba, která využívá služeb terapeuta za účelem zlepšení svého zdravotního stavu a kvality života. Aplikaci používá na základě pozvánky od svého terapeuta. Jeho motivací je snazší a pohodlnější způsob častějšího užívání terapie, čímž si přeje dosáhnout lepších výsledků. Hlavním cílem klienta v aplikaci je přehrávání své zvukové terapie. Klienti tvoří nejpočetnější skupinu uživatelů aplikace, která zahrnuje širokou demografickou škálu s různými zkušenostmi a dovednostmi. Pro tyto uživatele musí aplikace působit obzvláště jednoduše, intuitivně a přímočaře.

13.3.3 Doménový model

Doménový model (viz obrázek 13.1) představuje konceptuální reprezentaci domény. Vizualizuje vztahy a interakce mezi jednotlivými entitami a aktéry.



Obrázek 13.1: Doménový model

13.4 Specifikace požadavků

Tato kapitola se zaměřuje na specifikaci požadavků na webovou aplikaci. Jedná se klíčovou součást vývoje softwaru, protože stanovuje očekávání a cíle, které jsou základní podmínkou pro úspěch aplikace. Požadavky vychází z navrženého řešení a analýzy domény z dřívějších kapitol. Specifikace je rozdělena do dvou hlavních kategorií: funkční a mimofunkční požadavky.

13.4.1 Funkční požadavky

Funkční požadavky se týkají funkcí, které jsou přímo dostupné uživatelům v rámci aplikace. Každá funkce by měla sloužit určitému cíli a nabízet tak hodnotu uživatelům. Jedná se o ucelenou a detailní formu specifikace požadavků vycházející z požadavků terapeuta vypsaných v kapitole 13.1.1.

Uživatelské účty a přihlášení

Uživatelé se mohou po registraci přihlásit do aplikace pomocí svého emailu a hesla. Každý uživatel má svůj vlastní účet, který je spojen s určitou rolí.

Uživatelské rozhraní aplikace a dostupné funkce se odvíjejí od role daného uživatele. Aplikace poskytuje možnost změnit heslo, obnovit zapomenuté heslo prostřednictvím emailu a umožňuje uživatelům zůstat přihlášeni, což eliminuje potřebu opakovaného přihlašování.

Snadné přidání terapeutů a klientů

Pro jednoduché přidání terapeutů a klientů do aplikace byl navržen proces registrace založený na emailových pozvánkách. Administrátor může pozvat terapeuty do aplikace, zatímco terapeuti mohou přidávat své klienty. Pozvánka je odeslána z aplikace prostřednictvím emailu, který obsahuje odkaz pro dokončení registrace. Uživatel poté vyplní základní osobní informace, nastaví heslo a souhlasí s podmínkami používání aplikace a zásadami ochrany osobních údajů.

Správa uživatelů

Aplikace nabízí možnost prohlížení a správu uživatelských účtů. Uživatelé mohou být vyhledáváni pomocí relevantních kritérií a filtrů, zobrazovány jejich detaily a vztahy mezi terapeuty a klienty. Administrátoři mohou vidět, kdy byli uživatelé naposledy aktivní, a mají možnost odebírat uživatele nebo jejich účty dočasně deaktivovat. V případě deaktivace účtu terapeuta jsou automaticky deaktivovány účty všech jeho klientů.

Správa frekvencí

Administrátor má možnost spravovat frekvence v aplikaci. Frekvence mohou být přidávány s vybraným názvem a zařazeny do určitého balíku, přehrávány, upravovány nebo odebírány podle potřeby. Terapeut má přístup k seznamu frekvencí, ale nemá oprávnění je spravovat.

Správa šablon

Každý terapeut má k dispozici vlastní sadu šablon, které může vytvářet, upravovat, využívat pro tvorbu terapií pro klienty a odebírat. Šablony nejsou sdíleny mezi jednotlivými terapeuty. Šablony lze vytvořit buď ručně nebo odvozením z již existující terapie.

Vytvoření terapie

Terapeut má možnost vytvářet a upravovat specializovanou terapii pro své klienty. Hlavním prvkem terapie je individuálně vytvořený playlist nahra-

ných zvukových frekvencí. Terapeut také může terapii adekvátně pojmenovat a nastavit časové období, během kterého je terapie přístupná pro klienta. Navíc může nastavit počet povolených přehrání za den, kterého by klient měl dosáhnout. Aplikace mu nedovolí tento limit v rámci jednoho dne překročit.

Sledování používání terapie

Aplikace umožňuje terapeutům monitorovat, jak jejich klienti využívají předepsanou terapii. Terapeut může zjistit, kdy si klient terapii přehrával a jak často, což přispívá k lepší vzájemné komunikaci a spolupráci mezi terapeutem a klientem

Poslech frekvenční terapie klientem

Po přihlášení mají klienti přístup ke svým individuálně připraveným frekvenčním terapiím, které pro ně terapeut vytvořil. Terapie si mohou snadno přehrávat podle stanoveného počtu povolených přehrávání za den. Před zahájením poslechu je možnost si otestovat hlasitost zvuku a po zahájení poslechu ho nelze přerušit, pouze úplně zastavit. Toto zastavení se započítá jako jedno z přehrání, které má klient pro daný den přístupné. Tento přístup byl zvolen z důvodu, aby klienti k terapii přistupovali zodpovědně a opravdu si vyhradili potřebný čas.

13.4.2 Mimofunkční požadavky

Mimofunkční požadavky jsou nezbytné vlastnosti aplikace, které nejsou přímo spojeny s jejími funkcemi, ale jsou důležité pro splnění očekávaného fungování aplikace a uživatelského zážitku.

Bezpečnost

Aplikace by měla zajišťovat robustní proces pro autentizaci a autorizaci uživatelů. Kromě toho by měla být aplikace chráněna proti známým typům útoků na webové aplikace, jako jsou například SQL injection, XSS, CSRF a další. To zahrnuje vhodné způsoby výměny a ukládání dat. Přístup k nahraným terapeutickým zvukům by měl být omezen pouze na uživatele, kteří mají oprávnění pro poslech terapie, a frekvence by neměly být snadno dostupné pro stahování nebo sdílení mimo rámec aplikace. To vyžaduje řádné řízení přístupu, šifrování dat a sledování aktivit uživatelů, aby bylo možné identifikovat a zabránit potenciálnímu zneužití.

Dostupnost

Aplikace by měla být snadno dostupná pro uživatele s minimálními výpadky. Měla by být navržena tak, aby byla jednoduchá a srozumitelná i pro uživatele s minimálními zkušenostmi s používáním technologií a webových aplikací. To zahrnuje responzivní design a zajištění kompatibility s běžnými webovými prohlížeči a operačními systémy. Aplikace by měla také být navržena s ohledem na mezinárodní podporu, což zahrnuje možnost překladu uživatelského rozhraní a dalších textů do různých jazyků.

Nenáročná údržba a provoz

Pro snížení nákladů a zjednodušení správy aplikace by měla být co nejméně náročná na provoz. To zahrnuje automatizaci procesů, jako jsou nasazení a testování. Aplikace by měla vyžadovat minimální míru zásahů od administrátorů nutných pro řádný chod aplikace. Produkční prostředí by mělo být nenáročné na zdroje, jako jsou finance a čas věnovaný údržbě. Aplikace by měla být navržena tak, aby byla snadno škálovatelná a udržovatelná. To zahrnuje modularitu kódu, použití standardních návrhových vzorů a dokumentaci. Díky těmto vlastnostem by mělo být snadné opravit chyby, přidávat nové funkce a provádět úpravy podle potřeb uživatelů. Změny by měly být snadno aplikovatelné v produkčním prostředí bez negativního dopadu na stávající uživatele. Monitorování aplikace by mělo být prováděno průběžně a využíváno k analýze výkonu a identifikaci problémových oblastí.

Vytížení a objem dat

Aplikace by měla být schopna zvládat paralelní využívání až 50 terapeutů a 100 klientů současně, aniž by to negativně ovlivnilo výkon aplikace nebo uživatelskou zkušenost. Aplikace by měla být schopna zpracovávat a ukládat data pro až 1000 uživatelů a 100 různých frekvencí. Tento požadavek zahrnuje schopnost rychle vyhledávat a filtrovat relevantní informace. To vyžaduje zvolit vhodné produkční prostředí s možností škálovatelnosti, efektivní správu zdrojů a komunikace mezi klientskou a serverovou částí aplikace.

Rychlost odezvy

Aplikace by měla dosahovat rychlosti odezvy v řádu desítek až stovek milisekund, což lze dosáhnout pomocí vyrovnávací paměti, efektivních algoritmů a optimalizace přístupu k datům. Je třeba vzít v úvahu, že rychlost odezvy může být ovlivněna rychlostí internetového spojení, výkonem zařízení, množstvím uložených dat a aktuálním počtem přihlášených uživatelů,

kteří současně zatěžují aplikaci. Pro dosažení optimální rychlosti odezvy je důležité pravidelně sledovat a vyhodnocovat výkonnost aplikace.

13.5 Rizika a omezení projektu

V této kapitole jsou popsána rizika a omezení, která mohou ohrozit úspěšnost projektu. Identifikace rizik a stanovení vhodných strategií pro jejich řešení či minimalizaci jsou klíčové pro zajištění vyšší pravděpodobnosti úspěšnosti projektu.

Neznalost domény a sběr požadavků

Aby byla aplikace účinná v oblasti muzikoterapie, je nezbytné nastudovat tuto doménu a konzultovat návrh aplikace s odborníky v oboru. Tímto přístupem lze minimalizovat riziko vytvoření aplikace, která není kompatibilní s potřebami muzikoterapeutů a jejich klientů. Nejasné a měnící se požadavky mohou výrazně zpomalit vývoj projektu a vést k nesplnění očekávání jednotlivých stakeholderů. Je důležité provést pečlivou analýzu domény a požadavků včas, využít prototypů pro lepší pochopení potřeb a očekávání a průběžně stav validovat. Navíc by měl být projekt řízen agilně, s důrazem na časté a pravidelné dodávání funkčních částí se záměrem na sběr a zapracování zpětné vazby.

Potřeba vhodných sluchátek nebo reproduktorů

Efektivnost terapie závisí na kvalitní reprodukci terapeutických zvuků, což nemusí být dosaženo s běžnými zařízeními pro reprodukci zvuku. Někteří uživatelé mohou mít nepohodlná nebo nekvalitní sluchátka, což může ovlivnit jejich schopnost relaxovat a tím ovlivnit účinnost terapie. Tento problém vyplývá ze zvoleného řešení webové aplikace a může být minimalizován doporučením a otestování vhodného příslušenství.

Rizika spojená s využitím cloudových služeb

Při využití cloudových služeb je třeba zohlednit možná rizika, jako jsou například náklady na provoz, závislost na třetích stranách nebo potenciální problémy s ochranou dat. Je důležité provést analýzu dostupných řešení a vybrat takové, které nabízejí optimální rovnováhu mezi funkcemi, cenou a zabezpečením.

Právní zabezpečení a pravidla používání aplikace

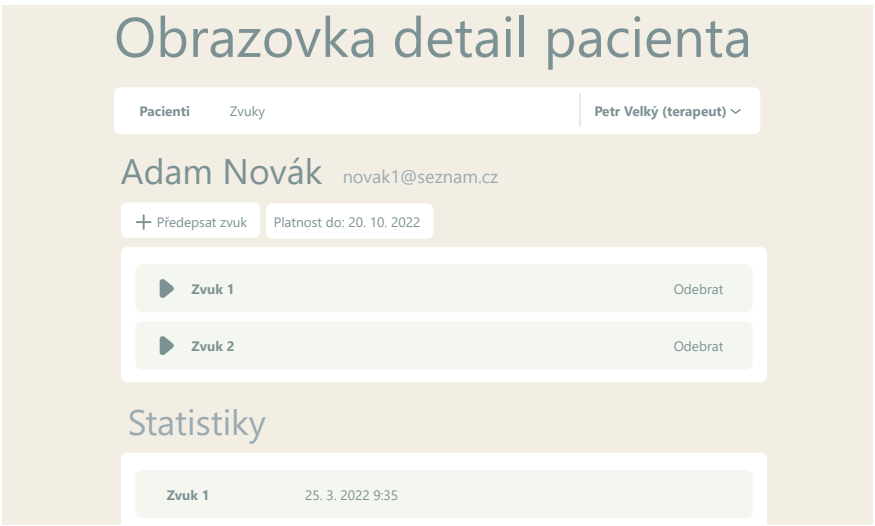
Aplikace musí splňovat právní požadavky týkající se ochrany osobních údajů a informovat uživatele o sběru a zpracování jejich dat. Je nutné zajistit, aby uživatelé souhlasili s podmínkami použití aplikace a ochranou osobních údajů již při registraci. Aplikace by měla také informovat o používání cookies a vyžádat si souhlas uživatele, pokud je to nutné v souladu s platnými právními předpisy.

13.6 Návrh uživatelského rozhraní

Tato kapitola se zaměřuje na návrh uživatelského rozhraní webové aplikace. Hlavním cílem tohoto procesu je poskytnout uživatelům intuitivní a efektivní způsob využití jednotlivých funkcí nabízených aplikací. Vychází z analýzy aktérů a funkčních požadavků (viz předchozí kapitoly). Návrh uživatelského rozhraní zahrnuje tvorbu wireframů a prototypů, iterativní zlepšování návrhu na základě zpětné vazby a také tvorbu grafického designu. Tyto aspekty budou popsány v následujících kapitolách.

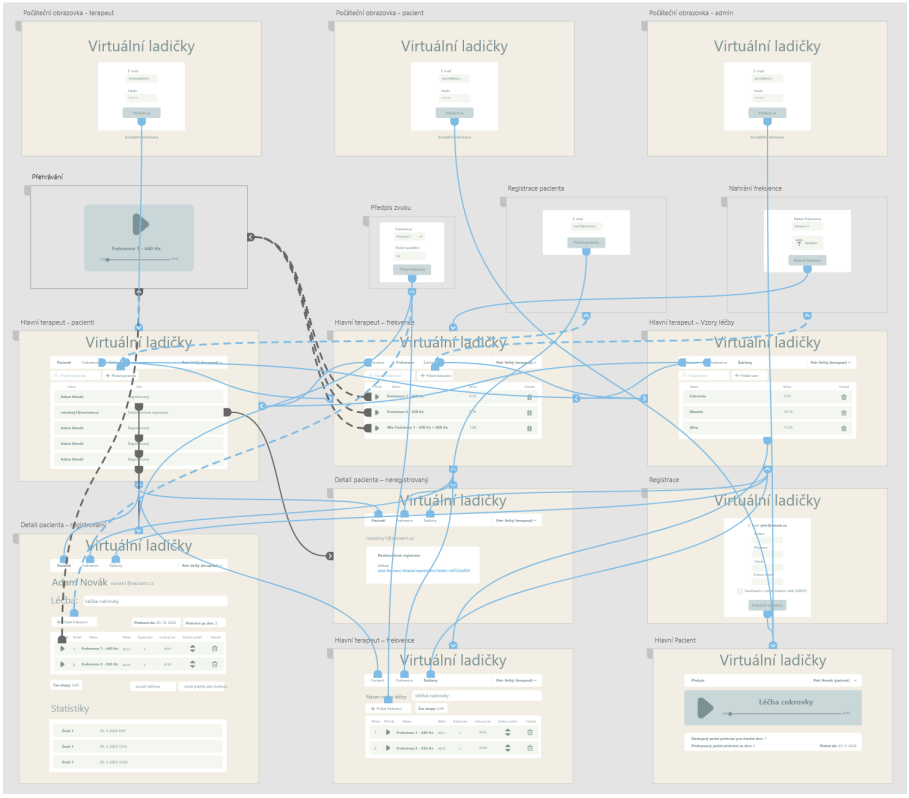
13.6.1 Postup při návrhu uživatelského rozhraní

Prvním krokem při návrhu uživatelského rozhraní bylo vytvoření tzv. wireframů. Tyto „drátěné modely“ sloužily jako základ pro identifikaci jednotlivých stránek, definování jejich účelu a struktury. Následně bylo prostřednictvím prototypů (viz obrázky 13.2 a 13.3) vytvořeno uživatelské rozhraní,



Obrázek 13.2: Obrazovka detailu uživatele v prototypu aplikace

které umožňovalo základní interakce a navigaci mezi jednotlivými obrazovkami. K tomu byl využit nástroj Adobe XD. V této fázi byla věnována malá pozornost grafickému designu, aby bylo možné se více soustředit na základní vlastnosti jako rozmístění klíčových prvků, účel jednotlivých obrazovek a jejich vzájemné vztahy. Cílem bylo dosáhnout efektivního a intuitivního využití aplikace. Tento postup umožnil získat lepší představu o tom, jak aplikace bude fungovat a jak ji uživatelé budou používat.



Obrázek 13.3: Provázanost obrazovek prototypu

Po dosažení uspokojivé kvality struktury uživatelského rozhraní byla věnována pozornost grafickému designu. Byl zvolen jednotný vizuální styl, který odpovídá charakteru aplikace a očekáváním cílové skupiny uživatelů. Dále bylo vytvořeno logo (viz obrázek 13.4), zvolena barevná paleta, typografie a ikony, které společně tvoří vizuální identitu aplikace a zajišťují srozumitelnost a přehlednost uživatelského rozhraní.

Před zahájení vývoje tedy proběhlo několik iterací, které zahrnovaly validaci návrhu a sběr zpětné vazby na základě vytvořených návrhů obrazovek a prototypů. Tento iterativní přístup umožnil minimalizovat zásahy do struktury uživatelského rozhraní a funkcí aplikace během vývoje, což přispělo k rychlejšímu a efektivnějšímu vývoji aplikace.

Obrázek 13.4: Logo aplikace znázorňující terapeutickou ladičku

13.6.2 Struktura uživatelského rozhraní

Navržené uživatelské rozhraní se skládá z přibližně 10 obrazovek, které jsou navrženy tak, aby plnily specifické cíle a vyhovovaly potřebám uživatelů. Většina obrazovek je přístupna pouze pro přihlášené uživatele a jejich obsah a nabízené funkce jsou dynamicky upravovány na základě role a pravomocí uživatele. Struktura uživatelského rozhraní byla navržena tak, aby byla jednoduchá, intuitivní a přehledná pro všechny typy uživatelů. Uživatelé mají možnost přecházet mezi jednotlivými obrazovkami využitím navigace v hlavičce stránky, která se zobrazuje všem přihlášeným uživatelům. Nabízené odkazy v navigaci se liší podle aktuální role uživatele. Obrazovky mají následující účel:

- **hlavní obrazovka** Tato obrazovka je první, kterou uživatelé uvidí po zadání domény aplikace do prohlížeče. Je jednoduchá a slouží jako úvod do aplikace. Umožňuje přechod na obrazovku pro přihlášení,
- **obrazovka správy uživatelů**: Tato obrazovka umožňuje prohlížení seznamu všech uživatelů s možností filtrování a přidávání nových uživatelů. Je přístupná pro administrátory a terapeuty. Terapeut má přístup pouze ke svým klientům a může jim také zaslat pozvánku. Administrátor má přístup ke všem uživatelům, více funkcím a může zaslat pozvánku terapeutovi,
- **obrazovka správy frekvencí**: Na této obrazovce je možné prohlížet seznam všech frekvencí nahraných v aplikaci s možností jejich filtrování a přehrávání. Je dostupná pouze pro administrátory a terapeuty. Administrátor má možnost odstraňování, úpravy a nahrávání nových frekvencí,
- **obrazovka správy šablon**: Tato obrazovka poskytuje možnost zobrazit seznam šablon s možností filtrování, přidávání nových šablon

a zobrazení detailu šablony. Je přístupná pouze administrátorovi a terapeutovi. Administrátor má přístup ke všem šablonám v aplikaci, terapeut má přístup pouze ke svým vlastním šablonám a může vytvářet nové šablony,

- **obrazovka detail šablony**: Tato obrazovka zobrazuje podrobnosti o konkrétní šabloně a umožňuje její úpravy. Je dostupná pro administrátory a terapeuty. Terapeut může vidět a upravovat pouze své vlastní šablony,
- **obrazovka detailu uživatele**: Tato obrazovka zobrazuje podrobné informace o jednotlivých uživatelích. Administrátor má přístup ke všem uživatelům, zatímco terapeut má možnost prohlížet pouze své vlastní klienty. Terapeut zde může vytvářet a upravovat vlastnosti a detaily terapie konkrétního klienta. Díky vnořené navigaci je možné prohlížet také záznamy o využívání aplikace klientem,
- **domovská obrazovka klienta**: Hlavním účelem této obrazovky je umožnit klientovi spustit a poslouchat frekvenční terapii. Je přístupná výhradně pro klienty a její design je zaměřen na jednoduchost a přehlednost,
- **obrazovka přihlášení**: Poskytuje možnost přihlášení uživatele prostřednictvím přihlašovacího formuláře. Po úspěšném přihlášení je uživatel přesměrován na domovskou obrazovku odpovídající jeho roli. V případě zapomenutí hesla umožňuje navigaci na obrazovku pro obnovu hesla,
- **obrazovka registrace**: Umožňuje dokončit registraci uživatele pomocí registračního formuláře. Pro zobrazení této obrazovky je nutné mít jedinečnou URL obsahující registrační token. Ta je přístupná pouze skrze odkaz zaslaný prostřednictvím emailové pozvánky. Tato obrazovka slouží pro registraci pacientů i terapeutů,
- **obrazovka zapomenutého hesla**: Tato obrazovka je užitečná pro případy, kdy uživatel zapomene své heslo. Na obrazovce uživatel zadá svůj email a pokud je platný, obdrží email s možností obnovit své heslo,
- **obrazovka pro obnovu hesla**: Tato obrazovka má za úkol umožnit uživatelům nastavit nové heslo v případě, že si své původní heslo již nepamatují. Pro zobrazení této obrazovky je vyžadována jedinečná

URL adresa, která obsahuje token pro obnovu hesla. Tento token je poskytnut pouze prostřednictvím odkazu v emailu, který uživatel obdrží poté, co požádá o obnovu hesla,

- **obrazovka s pravidly ochrany osobních údajů a podmínkami užití**: Tato obrazovka zahrnuje podrobný text týkající se pravidel ochrany osobních údajů a podmínek užití aplikace. Odkaz na tuto obrazovku je umístěn ve spodní části aplikace a je rovněž začleněn do registračního formuláře. Uživatel je v průběhu registrace povinen vyjádřit svůj souhlas s těmito pravidly a podmínkami.

Další funkce, jako například zobrazení vlastního profilu a změna hesla, jsou řešeny speciálními modálními okny přístupnými z kontextového menu aplikace.

14 Implementace navržené aplikace

Kapitola se zaměřuje na implementační aspekty navržené webové aplikace. Podrobně představuje zvolenou architekturu, technologie a nástroje použité v rámci vývoje. Následně je přiblížena struktura projektu a vysvětlena klíčová implementační rozhodnutí a řešení. Další kapitoly jsou věnovány využitým bezpečnostním praktikám, procesu autentizace a autorizace, uživatelskému rozhraní a základním komponentám aplikace.

14.1 Architektura a základní technologie

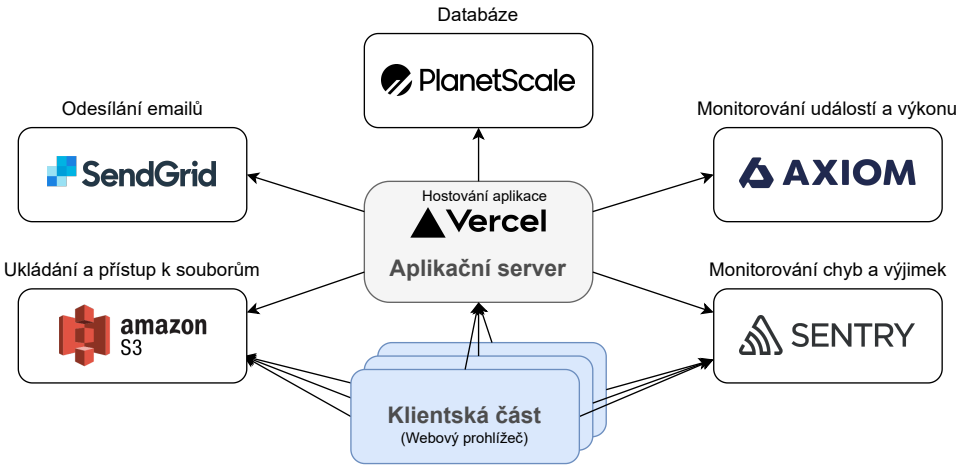
Tato část se zaměřuje na architekturu a základní technologie vybrané pro vývoj webové aplikace, která je předmětem této diplomové práce. Cílem je popsat základní charakteristiky a zdůvodnit, proč tyto technologie představují vhodný výběr oproti alternativním řešením.

Volba základních nástrojů vychází z analýzy popsané v teoretické části práce. Rozhodnutí bylo dále inspirováno sadou technologií známou jako „T3 Stack“, která se zaměřuje na vývoj moderních webových aplikací s důrazem na typovou bezpečnost kódu. Mezi základní vybrané technologie pro aplikaci se řadí Next.js, TypeScript, tRPC (komunikace mezi klientem a serverem) a Prisma (ORM). Tyto technologie byly zvoleny na základě jejich schopnosti společně tvořit ucelený a efektivní technologický základ pro vývoj moderní webové aplikace.

14.1.1 Architektura aplikace

Architektura webové aplikace je založena na třívrstvém modelu, který se skládá z prezentační, aplikační a datové vrstvy. Tento model je základním přístupem k vývoji webových aplikací a umožňuje oddělení logiky aplikace od uživatelského rozhraní a databáze. Třívrstvý model byl již podrobně popsán v kapitole 5.2. Jak serverová, tak i klientská strana navíc komunikují prostřednictvím webového rozhraní s dalšími integrovanými cloudovými službami, které se využívají pro realizaci různých funkcí, jako například odesílání emailů, monitorování, a bezpečné ukládání a přístup ke zvukovým souborům (viz obrázek 14.1).

Výběr technologie Next.js ovlivňuje architekturu aplikace tím, že zajišťuje optimalizovanou integraci mezi prezentační a aplikační vrstvou a generaci uživatelského rozhraní na serveru. Serverová a klientská strana jsou definovány v rámci jednoho projektu, což eliminuje nutnost udržovat dva samostatné repozitáře a poskytuje možnost snazší integrace a spolupráce mezi těmito stranami aplikace. Například tak umožňuje snadné sdílení kódu mezi klientskou a serverovou stranou, což zvyšuje efektivitu vývoje a zjednodušuje údržbu.



Obrázek 14.1: Integrace aplikace s webovými službami

14.1.2 T3 Stack

T3 Stack představuje integraci moderních technologií, která vznikla s cílem zefektivnit vývoj webových aplikací. Skládá se z navzájem se podporujících nástrojů, mezi které patří Next.js, TypeScript, tRPC, Prisma, Tailwind CSS a NextAuth.js [6]. Pro implementaci webové aplikace v rámci této diplomové práce byla zvolena pouze část těchto technologií. Konkrétně místo Tailwind CSS a NextAuth.js byla zvolena jiná řešení, která lépe odpovídala specifickým potřebám a požadavkům aplikace.

Hlavním důvodem, proč byla učiněna volba vycházet z technologií T3 Stacku, je jejich schopnost zajistit efektivní vývoj, robustní architekturu a přehledný, čitelný a udržitelný kód, a to všechno využitím jednoho programovacího jazyka (TypeScript). Výběr technologií je učiněn se záměrem dosáhnout vysoké typové bezpečnosti, kterou přináší jazyk TypeScript a jeho kvalitní integrace s ostatními zahrnutými technologiemi.

Technologie T3 Stack jsou moderní a aktuální, zaměřují se na efektivní psaní funkčních aspektů bez nutnosti velkého množství obklopujícího kódu.

Díky současné popularitě a široké komunitě vývojářů lze očekávat vysokou úroveň podpory a dostupnost kvalitních zdrojů pro řešení problémů. Tento přístup také zajišťuje, že aplikace bude snadno rozšiřitelná a udržitelná v budoucnu, což je důležité pro dlouhodobý úspěch projektu.

14.1.3 Next.js

Next.js je nástroj založený na frameworku React, který poskytuje optimalizované integrace a dodává projektu strukturu a doporučené postupy. Tato technologie byla již blíže představena v kapitole 7.6.1 a byla vybrána na základě porovnání v kapitole 7.7. Přináší výhody, jako je výkon, doporučená struktura, nabízené funkcionality pro efektivnější vývoj, sjednocení klientské a serverové strany v rámci jednoho projektu a jednoho programovacího jazyka, integrace s knihovnou React, podpora Node.js serveru a široká komunita vývojářů.

14.1.4 TypeScript

TypeScript je typovaná nadstavba JavaScriptu, která přináší statické typování a další vývojářské nástroje, které zlepšují čitelnost kódu a zvyšují jeho bezpečnost. Tato technologie byla podrobněji představena v kapitole 6.3.1.

Výběr TypeScriptu je vhodný zejména kvůli přidaným typům, které nabízejí typovou bezpečnost aplikace, což samotný JavaScript nedokáže poskytnout. Díky efektivnímu využití TypeScriptu je kód lépe čitelný, snadněji udržitelný a bezpečnější, jelikož typové chyby jsou odhaleny již ve vývojovém prostředí. Tím se zlepšuje kvalita kódu a snižují náklady na údržbu aplikace. Integrace TypeScriptu s ostatními použitými technologiemi, jako je Prisma a tRPC, zajišťuje širokou konzistentnost a typovou bezpečnost.

Pro vynucení používání vhodných praktik je využit nástroj ESLint (viz kapitola 14.2.4), který poskytuje sadu pravidel a doporučení pro psaní kódu.

14.1.5 tRPC

tRPC je knihovna pro tvorbu API, která umožňuje efektivní komunikaci mezi klientem a serverem. Tato technologie vychází z principů RPC (viz kapitola 5.3) a byla navržena tak, aby plně využila výhod TypeScriptu. Výběr tRPC pro vývoj webové aplikace je vhodný především díky zachování typové bezpečnosti při komunikaci mezi klientem a serverem. Je tak zajištěno, že na serveru jsou volány adekvátní funkce se správným typem dat, a správný typ dat je očekáván i v odpovědi. Díky tomu tRPC představuje další základní blok pro typovou bezpečnost kódu.

tRPC navíc usnadňuje vývoj aplikace tím, že vytváření koncových bodů API je velice přímočaré. Tím zefektivňuje vývoj a poskytuje přehlednější kód.

Nevýhodou technologie je nutnost korektní konfigurace tak, aby správně fungovala. Nicméně, díky rozsáhlé dokumentaci, podpoře komunity a integraci v rámci T3 stack, lze tento problém snadno vyřešit.

14.1.6 Prisma

Prisma je ORM (Object-Relational Mapping) technologie, která umožňuje pracovat s databází jako s objekty a používat objektově orientované koncepty pro práci s daty. Na rozdíl od psaní čistého SQL, kde programátor musí manuálně vytvářet SQL dotazy a mapovat výsledky zpět na objekty v programu, ORM toto mapování provádí automaticky. To zjednodušuje vývoj, zvyšuje produktivitu a snižuje chybovost. Mezi nevýhody se může řadit možný neefektivní způsob dotazování dat, který více zatěžuje databázi [2].

Prisma přináší silné typování, což umožňuje snadno definovat a manipulovat s datovými modely a zároveň garantuje bezpečnost a spolehlivost přístupu k datům. Pro definování struktury dat je využito speciální schéma, které slouží jako základ pro generování typů a dotazovacích funkcí. Navíc Prisma podporuje širokou škálu databázových technologií, což umožňuje flexibilitu při výběru vhodného řešení pro datové úložiště.

Vzhledem k těmto výhodám je Prisma vhodným nástrojem, neboť podporuje typovou bezpečnost, zjednodušuje komunikaci mezi aplikací a databází, a přispívá k rychlejšímu a efektivnějšímu vývoji aplikace.

14.2 Vývojové prostředí

Vývojové prostředí je klíčovým faktorem, který ovlivňuje produktivitu a efektivitu vývoje aplikace. V této kapitole je představena struktura vývojového prostředí a základní nástroje, které byly použity pro vývoj webové aplikace v rámci této diplomové práce.

14.2.1 Editor zdrojového kódu

Mezi základní použitý nástroj se řadí VSCode (Visual Studio Code), což je populární a výkonný open-source editor kódu vyvinutý společností Microsoft. Poskytuje integrované nástroje pro práci s TypeScriptem, což umožňuje pracovat s typově bezpečným kódem, automaticky opravovat chyby a rychle

navigovat mezi jednotlivými částmi projektu. Díky široké podpoře rozšíření bylo možné VSCode dále přizpůsobit potřebám projektu.

14.2.2 Správa verzí kódu

I přesto, že na projektu byla práce prováděna jedním vývojářem, použití systému správy verzí kódu je nezbytné pro efektivní sledování změn v kódu, zálohování a možnost snadného návratu k dřívějším verzím projektu. V tomto případě byl použit systém Git, což je populární distribuovaný systém správy verzí navržený pro rychlé a spolehlivé sledování změn v souborech.

Pro centralizované uložení repozitáře a další správu projektu byl použit GitLab. Ten nabízí přehledné a schopné webové rozhraní. Poskytuje tak pokročilé funkce pro správu projektů, jako je issue tracking, merge requests, code review nebo CI/CD.

14.2.3 Balíčkovací systém a závislosti

Balíčkovací systém je nástroj používaný k řízení závislostí a jejich verzí v softwarových projektech. V rámci tohoto projektu byl použit balíčkovací systém Pnpm, což je alternativa k tradičním nástrojům jako Npm (Node Package Manager) a Yarn. Pnpm nabízí efektivní ukládání balíčků prostřednictvím sdílení balíčků mezi projekty, rychlejší instalaci a aktualizaci závislostí, a zvýšenou bezpečnost.

Ke konfiguraci závislostí je využit soubor `package.json`. Obsahuje informace o názvu projektu, verzi, autorech a licenci. Zároveň definuje závislosti rozdělené do kategorií, jako jsou `dependencies` pro běhové závislosti, které jsou nezbytné pro běh aplikace, `devDependencies` pro vývojové závislosti, které jsou potřebné pouze pro vývoj a testování aplikace, a `peerDependencies` pro sdílené závislosti mezi balíčky, které vyžadují koordinaci verzí mezi více projekty. Soubor `package.json` také umožňuje definovat skripty, které usnadňují běžné úkoly, jako je spouštění testů, sestavování aplikace nebo spouštění lokálního vývojového serveru.

14.2.4 Nástroje pro kontrolu kvality kódu

V rámci vývoje je důležité zajistit, že kód je čitelný, udržovatelný a bez chyb. Pro kontrolu kvality kódu byly v projektu použity nástroje ESLint a Prettier, které byly integrovány v rámci rozšíření s prostředím VSCode.

ESLint

ESLint je populární nástroj pro analýzu kódu a identifikaci problémů v JavaScriptových a TypeScriptových projektech. Umožňuje vývojářům nastavit pravidla pro psaní kódu a automaticky kontroluje, zda kód splňuje tyto požadavky. ESLint pomáhá psát konzistentní a přehledný styl kódu a odhalit potenciální chyby, které by mohly vést k problémům v běhu aplikace. Pravidla lze vytvořit podle potřeb nebo také použít přednastavené sady pravidel.

Prettier

Prettier je nástroj pro formátování kódu. Při použití nástroje Prettier se vývojáři nemusí starat o ruční formátování kódu, což zajišťuje konzistentní styl kódu napříč celým projektem, bez ohledu na typ souboru a kdo a z jakého stroje ho upravoval. Prettier podporuje řadu jazyků, včetně JavaScriptu, TypeScriptu, HTML, CSS a dalších. Jeho použití zajišťuje, že kód je čistý, konzistentní a snadno čitelný pro všechny členy týmu.

14.3 Struktura projektu

Struktura a význam hlavních souborů a složek v projektu jsou následující:

- `prisma/schema.prisma` – soubor obsahující schéma databáze,
- `public` – složka obsahující veřejné soubory, jako jsou `robots.txt`, ikony, obrázky a lokalizace pro různé jazyky,
- `src` – složka obsahující zdrojový kód aplikace,
  - `components` – složka obsahující React komponenty použité v projektu, rozdělené do následujících kategorií,
    - `emails` – složka obsahující soubory definující komponenty emailů,
    - `helpers` – složka s pomocnými komponentami,
    - `inputs` – složka s komponentami pro vstupy a interakci,
    - `layouts` – složka obsahující obalující komponenty,
    - `modals` – složka s komponentami pro modální okna,
    - `sections` – složka obsahující komponenty pro jednotlivé sekce stránek,
    - `static` – složka se statickými komponentami,
  - `env` – definice a validace proměnných pro různá prostředí,

  - `pages` – složka obsahující definici jednotlivých stránek webové aplikace,
  - `schema` – složka obsahující soubory validačních schémat,
  - `server` – složka obsahující serverovou část aplikace, routery a další nástroje,
    * `router` – složka obsahující routery pro jednotlivé části serverové aplikace,
    * `utils` – složka obsahující pomocné funkce a nástroje používané v rámci backendové části aplikace. Ty slouží například pro autorizaci, autentizaci a definování a konfiguraci objektů,
  - `utils` – složka obsahující pomocné funkce, hooky a typy pro klientskou část aplikace,
    * `hooks` – složka obsahující vytvořené React hooky používané v aplikaci,
    * `types` – složka obsahující definice typů pro TypeScript,
- `__tests__` – složka obsahující testy pro klientskou a serverovou část projektu,
- `.env` – soubor obsahující konfigurační proměnné prostředí, jako například přístupy k databázi nebo API klíče,
- `.eslintrc.json` – konfigurace nástroje ESLint,
- `.prettierrc` – konfigurace nástroje Prettier,
- `tsconfig.json` – konfigurace TypeScriptu,
- `Dockerfile` – konfigurace nástroje Docker,
- `.gitlab-ci.yml` – konfigurace CI/CD pipeline pro GitLab,
- `package.json` – soubor obsahující informace o projektu, jeho závislostech a skriptech,
- `README.md` – stručná dokumentace projektu se seznamem využitých technologií a návodem na spuštění aplikace,
- `LICENSE` – MIT licence.

14.4 Komunikace mezi klientskou a serverovou částí

Tato kapitola se zaměřuje na komunikaci mezi klientskou stranou spuštěnou ve webovém prohlížeči a aplikačním serverem. Pro tuto komunikaci je využita knihovna tRPC (již zmíněna v kapitole 14.1.5) v kombinaci s knihovnou Superjson, která slouží pro efektivní serializaci a deserializaci přenášených dat.

14.4.1 Použití knihovny tRPC

Knihovna tRPC představuje efektivní způsob definice koncových bodů prostřednictvím dvou základních entit: `queries` a `mutations`. `Queries` využívají HTTP GET a slouží k získávání dat, zatímco `mutations` využívají HTTP POST a umožňují modifikaci, přidávání a odebírání záznamů. Tyto koncové body se sdružují do objektů zvaných `routery`, které lze dále spojovat a vytvářet tak komplexní webové API aplikace.

Hlavní aplikační router se skládá z následujících routerů, které se zaměřují na specifické oblasti:

- `auth` – router pro autentizaci a autorizaci,
- `frequencies` – router pro správu frekvencí,
- `playlists` – router pro správu playlistů,
- `templates` – router pro správu šablon,
- `treatments` – router pro správu terapie,
- `users` – router pro správu uživatelů.

Díky svému designu a integrací s TypeScriptem dokáže knihovna tRPC zaručit typovou bezpečnost při komunikaci mezi klientskou a serverovou částí aplikace. Funkce definované na serveru a volané z uživatelského kódu mají pevně definované vstupní a výstupní typy, což zvyšuje robustnost, čitelnost a snižuje pravděpodobnost chyb v kódu.

Výstupní typ funkce je automaticky generován na základě typu návratové hodnoty. Pro definici vstupního typu je použita knihovna Zod, která umožňuje vytvářet schémata objektů pro validaci vstupních dat (viz kapitola 14.6). Tímto způsobem lze specifikovat detailní požadavky na vstupní hodnoty a definovat chybové zprávy pro různé inkonzistence ve vstupních datech.

Výpis kódu 14.1 demonstruje definici `routeru` pro správu frekvencí a koncového bodu `get` typu `query`, který je přístupný pro autorizované uživatele. Ve vstupním schématu je vyžadováno ID frekvence. Tento parametr společně s automaticky vytvořeným kontextovým objektem tvoří vstupní parametry koncového bodu.

```
export const frequenciesRouter = createTRPCRouter({
  get: authProcedure()
    .input(getFrequencySchema)
    .query(
      async ({ input: { id }, ctx: { prisma, t } }) => {
        const frequency = await prisma.frequency.findUnique(
          {
            where: { id },
          }
        );
        if (!frequency) {
          throw new TRPCError({
            code: "NOT_FOUND",
            message: t("serverErrors.frequencyNotFound"),
          });
        }
        return s3GetDownloadURL(id + "." + frequency.type);
      }
    ),
```

Výpis kódu 14.1: Koncový bod pro získání frekvence

Výpis kódu 14.2 znázorňuje získání záznamů ze serveru o využívání terapie voláním koncového bodu `treatments.getLogs`. Jelikož se jedná o typ `query`, je využit React hook `useQuery`, kterému jsou předány potřebné parametry. Vrácený objekt obsahuje informace o stavu požadavku a po úspěšném načtení i samotná data.

```
const treatmentLogsQuery =
  api.treatments.getLogs.useQuery({
    id: treatmentId,
    page,
    orderBy,
    pageSize,
  });
```

Výpis kódu 14.2: Dotazování serveru z klientské části

14.4.2 Knihovna Superjson

Superjson je knihovna pro serializaci a deserializaci dat, která řeší omezení standardního JSON formátu a poskytuje rozšířenou podporu pro práci se slo-

žitějšími datovými strukturami. Je využita při přenosu dat mezi klientskou a serverovou částí aplikace, kde je třeba efektivně a bezpečně zpracovávat různé druhy dat.

Oproti běžnému JSON formátu, který může být příliš jednoduchý a omezený, Superjson nabízí širší možnosti. Mezi ně se například řadí serializace a deserializace datumů a vnořených objektů.

Správná konfigurace knihovny Superjson zajišťuje, že proces serializace a deserializace při volání vzdálených serverových funkcí probíhá automaticky. Tímto způsobem se zjednodušuje práce s daty a zvyšuje se efektivita při vývoji a údržbě aplikace.

14.5 Databázové řešení

V této kapitole se zaměříme na zvolené databázové řešení. K tomu byly využity technologie Prisma a PlanetScale. Nástroj Prisma byl již popsán dříve v kapitole 14.1.6, proto se zde zaměříme především na jeho implementační a technické aspekty. Dále je popsáno cloudové databázové řešení PlanetScale. V kapitole je také představen ER diagram, který znázorňuje strukturu databáze.

14.5.1 Použití knihovny Prisma

Prisma se skládá z několika částí, mezi které patří schéma, klient a migrace.

Databázové schéma je definováno pomocí PSL (Prisma Schema Language) (viz ukázka 14.3). Umožňuje deklarovat tabulky, sloupce, vztahy mezi tabulkami, indexy a omezení na úrovni databáze. Schéma se definuje v souboru s příponou `.prisma`.

```
model Treatment {
    id                  String      @id @default(cuid())
    name                String      @db.VarChar(32)
    playbacksPerDay     Int         @default(1)
    updatedAt           DateTime    @updatedAt
    validFrom           DateTime?
    validUntil          DateTime?
    patientId           String      @unique
    playlistId          String      @unique
    treatmentLogs       TreatmentLog[]
    @@index([patientId])
}
```

Výpis kódu 14.3: Základní definice modelu terapie

Prisma klient je automaticky generovaná knihovna, která se vytváří na základě Prisma schématu příkazem `generate`. Je navržena tak, aby poskytovala snadno použitelné, typově bezpečné a efektivní API pro práci s databází. Prisma klient umožňuje vytváření, čtení, aktualizaci a mazání záznamů v databázi pomocí jednoduchých a čitelných dotazů. Výpis kódu 14.4 znázorňuje dotazování na jméno, příjmení a ID všech registrovaných terapeutů.

```
prisma.user.findMany({
  where: {
    role: Role.Therapist,
    registeredAt: { not: null },
  },
  select: { name: true, surname: true, id: true },
})
```

Výpis kódu 14.4: Dotazování dat využitím klientu Prisma

Díky integraci s TypeScriptem jsou všechny funkce a objekty generované Prisma klientem typově bezpečné, což znamená, že vývojáři mají při práci s databází jistotu, že pracují s validními daty. Navíc, pokud dojde ke změně schématu databáze, Prisma klient automaticky aktualizuje typy a objekty v kódu, což minimalizuje riziko chyb při úpravách.

Prisma migrace je nástroj pro správu migrací, který umožňuje sledovat a provádět změny v databázovém schématu. Hlavní příkazy Prisma migrací zahrnují `migrate` a `push`. Příkaz `migrate` provádí databázové migrace a aktualizuje schéma, zatímco `push` aplikuje změněny schématu přímo na databázi bez vytváření migračního souboru.

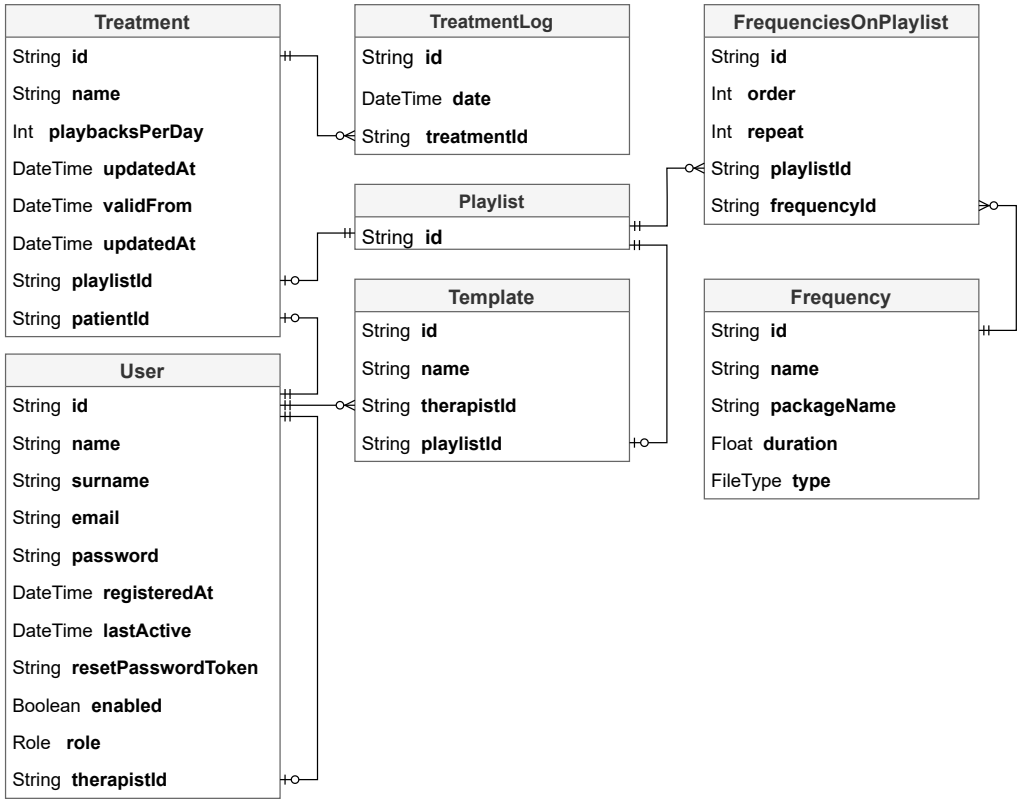
14.5.2 Databáze PlanetScale

PlanetScale je cloudová služba postavená na technologiích MySQL a Vitess, která poskytuje horizontálně škálovatelný a výkonný databázový systém. Jako cloudové řešení se jedná o PaaS (viz kapitola 8.2), které eliminuje nutnost správy hardwaru a softwarového prostředí, zajišťuje tak dostupnost, zálohy a škálování databáze.

PlanetScale byl zvolen pro tento projekt z důvodu jeho nenáročné údržby, škálovatelnosti a snadné integrace s ORM Prisma. Díky tomu se umožňuje soustředit na implementaci funkcí aplikace, bez nutnosti řešení problémů spojených se správou a efektivním chodem databáze. Navíc je služba do určitých omezení zdarma.

14.5.3 ER diagram

Entity-relationship (ER) diagram je slouží pro vizuální modelování struktury databáze, který pomáhá při návrhu a analýze databázových schémat. ER diagram zobrazuje jednotlivé entity (tabulky) a jejich vztahy v databázi. V rámci implementace aplikace byl ER diagram (viz obrázek 14.2) použit pro návrh databázového schématu, který byl následně definován pomocí Prisma Schema Language.



Obrázek 14.2: ER diagram

14.6 Validace vstupních dat

Validace vstupních dat je zásadním prvkem při vývoji interaktivních aplikací. Je důležité zajistit, že přijatá data jsou platná, splňují požadavky a jsou bezpečná. Pro validaci dat se v aplikaci využívají speciální validační schémata vytvořená pomocí knihovny Zod a provádí se na dvou úrovních: klientské a serverové.

14.6.1 Klientská validace dat

Klientská validace dat probíhá na straně uživatele v prohlížeči. Data se validují přímo v komponentě formuláře a odešlou se na server pouze v případě, že jsou platná. Klientská validace přináší rychlou zpětnou vazbu pro uživatele, kteří okamžitě zjistí důvod neplatnosti dat pomocí chybových hlášek. Díky klientské validaci je zátěž na serveru snížena, protože nevalidní data nejsou zbytečně odesílána na server.

14.6.2 Serverová validace dat

Serverová validace dat zajišťuje, že data přijatá prostřednictvím webového API jsou platná a bezpečná. Poskytuje další vrstvu zabezpečení a je nezbytná pro správné fungování aplikace. Ačkoliv validace na straně klienta zajišťuje platnost dat přijatých z vytvořeného uživatelského rozhraní, existují různé situace, kdy může být API voláno z jiných prostředí, která nezaručují korektnost dat. Mezi tyto případy patří útoky na aplikaci, které se snaží obejít klientskou validaci dat. Dalším příkladem mohou být integrace aplikace s jinými platformami nebo službami, které využívají webové API pro přístup k funkcím aplikačního serveru. V těchto případech je důležité zajistit, že i data zasílaná z těchto zdrojů splňují požadavky aplikace a neohrožují její bezpečnost, stabilitu a integritu.

14.6.3 Knihovna Zod

Knihovna Zod je nástroj pro validaci a parsování dat v TypeScriptu a JavaScriptu. Díky použití stejného programovacího jazyka, lze využít jednou definovaná schémata pro validaci dat jak na klientské straně, tak na serverové straně.

Zod nabízí jednoduché a přehledné API pro definování schémat a validaci dat. Schémata lze definovat pomocí konstruktorů Zod, které představují různé typy dat a omezení. Schémata lze poté použít pro validaci a parsování vstupních dat. Pokud data neodpovídají definovanému schématu, Zod vrátí určenou chybovou zprávu, která informuje o konkrétním problému. Pro korektní lokalizaci těchto chybových hlášek je potřeba generovat chybové hlášky pomocí lokalizační funkce.

Výpis kódu 14.5 demonstruje definici validačního schématu pro aktualizaci terapie. Je vyžadováno ID terapie, zatímco další parametry, jako jméno, počet přehrání za den, začátek a konec platnosti, jsou volitelné. Počet přehrání za den musí vyhovovat minimální a maximální hodnotě. Platnost může být zrušena předáním hodnoty `null`; v tomto případě musí mít jak začátek,

tak konec platnosti tuto hodnotu. Objekt `z` je součástí knihovny Zod a předávaný parametr `t` se využívá pro lokalizaci chybových hlášek.

```
export const updateTreatmentSchema = (t: T) =>
  z.object({
      id: z.string(),
      name: nameSchema(t).optional(),
      playbacksPerDay: z
        .number()
        .min(MIN_PLAYBACKS_PER_DAY)
        .max(MAX_PLAYBACKS_PER_DAY)
        .optional(),
      validUntil: z.date().optional().nullable(),
      validFrom: z.date().optional().nullable(),
    })
    .refine(
      (data) =>
        (data.validUntil === null) ===
        (data.validFrom === null)
    );
```

Výpis kódu 14.5: Definice schématu pro aktualizaci terapie

14.7 Zabezpečení aplikace

V této kapitole jsou představeny hlavní bezpečnostní praktiky použité při implementaci aplikace, které se týkají správy hesel, autentizace, autorizace, ochrany proti útokům a monitorování.

14.7.1 Správa hesel

Aby byla zajištěna dostatečná bezpečnost volených hesel, je požadována minimální délka 8 znaků, a heslo musí navíc obsahovat alespoň jedno malé písmeno, velké písmeno a číslici. Tato pravidla snižují pravděpodobnost úspěšného útoku hrubou silou na hesla uživatelů.

Pro ukládání hesel do databáze je používán hašovací algoritmus Argon2. Tento algoritmus je široce používán pro ukládání hesel a citlivých informací. Při přihlášení je tedy nutné vytvořit kontrolní součet hesla a porovnat ho s hodnotou uloženou v databázi.

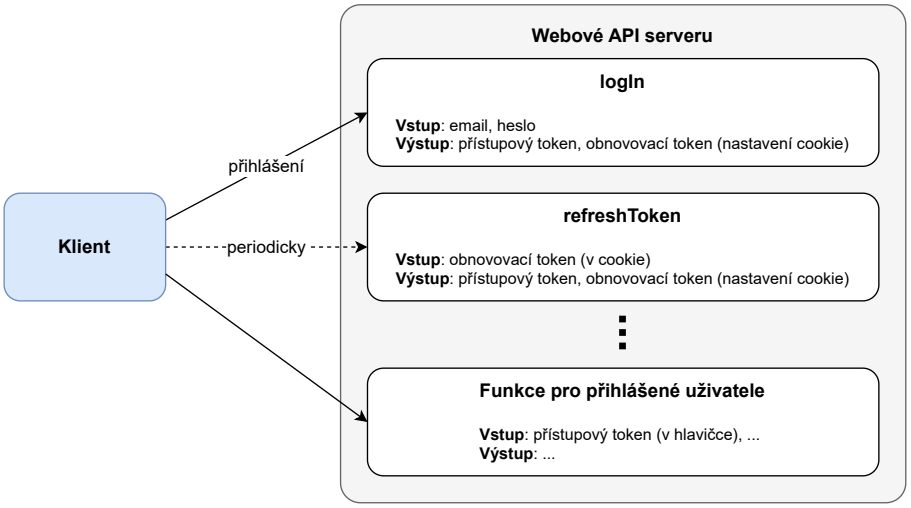
V případě, že uživatel heslo zapomene, může si vyžádat zaslání emailu s odkazem na formulář pro změnu hesla. Pro tento účel je vytvořen speciální token, který je následně uložen v databázi. Odkaz obsahující tento token je poslán uživateli emailem. Při otevření odkazu je token porovnán s tím uloženým v databázi. Pokud je platný, uživatel může pokračovat a změnit si

své heslo. Po změně hesla je token z databáze odstraněn, aby bylo zabráněno opětovné změně hesla. Token má krátkou platnost, což zvyšuje bezpečnost procesu.

Uživatelé mají také možnost si změnit své heslo po přihlášení do aplikace využitím modálního okna. Při změně je z důvodů bezpečnosti nutné zadat i stávající heslo.

14.7.2 Autentizace

Aplikace využívá JWT (viz kapitola 9.2.1) pro autentizaci a řízení přístupu. Token obsahuje zakódované informace o ID uživatele a jeho roli. V procesu autentizace (viz obrázek 14.3) jsou využity dva druhy tokenů: přístupový token (access token) a obnovovací token (refresh token).



Obrázek 14.3: Proces autentizace

Přístupový token

Přístupový token je vrácen jako odpověď na úspěšné přihlášení uživatele pomocí jména a hesla, a klient jej musí zaslat v hlavičce každého požadavku na server. Aby byla zajištěna ochrana proti CSRF útokům (viz kapitola 9.4.2) a XSS útokům (viz kapitola 9.4.1), token není posílán prostřednictvím cookies a je ukládán v paměti na straně klienta. Avšak, tato strategie může vést ke ztrátě tokenu, například při obnovení stránky. Přístupový token má navíc nastavenou krátkou dobu platnosti, což zvyšuje bezpečnost tím, že potenciálně zneužitelný token rychle expiruje. Proto pro zajištění plynulého používání aplikace a obnovení tokenu bez opakovaného zasílání jména a hesla je potřebný obnovovací token.

Obnovovací token

Obnovovací token se po přihlášení odesílá pomocí cookies a slouží k obnově přístupového tokenu. Jeho platnost je řádově delší než platnost přístupového tokenu. Pro jeho využití je vytvořen speciální koncový bod `refreshToken`, který zkontroluje validitu obnovovacího tokenu a v případě úspěchu vrátí nový přístupový token a aktualizuje i samotný obnovovací token.

Aby klient mohl aplikaci používat nepřetržitě déle než je platnost přístupového tokenu, klient na straně uživatele provádí tuto obnovu pravidelně na pozadí, aniž by si toho uživatel všiml. Obnovovací token se také využívá, když si uživatel přeje zůstat přihlášen. Tento token odesílaný přes cookies zůstane uložený v prohlížeči, a při příštím otevření aplikace se odešle na server, který v případě jeho platnosti vrátí přístupový token, čímž znovu přihlásí uživatele.

14.7.3 Autorizace

Autorizace v aplikaci je zajištěna pomocí modelu RBAC (viz kapitola 9.3.1). Aplikace rozlišuje mezi třemi hlavními uživatelskými rolemi: administrátor, terapeut a klient, které vycházejí z aktérů identifikovaných a popsaných v kapitole 13.3.2. Při každém požadavku na server se kontroluje přihlášení uživatele a jeho role. Oprávnění uživatelů jsou řešena na dvou úrovních: webového API a uživatelského rozhraní. Obě tyto úrovně jsou řešeny odděleně s centralizovaným přístupem přidělování přístupu, což umožňuje jednoduchým způsobem omezit přístup buď pouze pro přihlášené uživatele nebo pouze pro určité uživatelské role.

Kontrola oprávnění uvnitř koncových bodů

Uvnitř jednotlivých koncových bodů probíhá další kontrola, která ověřuje, zda je akce validní a oprávněná. Tato kontrola může zahrnovat ověření přístupu k datům jiných uživatelů, existenci požadovaných dat nebo další aspekty týkající se bezpečnosti a integrity dat. Pokud akce není vyhodnocena jako validní, je vrácena výjimka. Implementace koncového bodu s tímto chováním byla již znázorněna v ukázce 14.1. Výsledkem výjimek jsou specifické HTTP kódy a chybové zprávy v odpovědi, například `403 Forbidden` nebo `404 Not Found`, které signalizují klientovi problém s akcí či požadavkem.

Kontrola oprávnění v uživatelském rozhraní

Zobrazené komponenty a jejich aktivita se liší podle pravomocí uživatele, což zamezuje odesílání neoprávněných požadavků na server. Navíc, pokud se ne-

přihlášený uživatel pokusí přistupovat na stránky určené pro přihlášené uživatele, je přesměrován na přihlašovací stránku. Obdobně, pokud přihlášený uživatel přistupuje na stránky určené pro jiné role nebo pro nepřihlášené uživatele, je přesměrován na stránku odpovídající jeho roli. K implementaci této logiky je využita speciální funkce Next.js nazývaná `GetServerSideProps`, která probíhá vždy na serveru před odesláním odpovědi. Tu je nutné exportovat v souborech definující jednotlivé stránky. Výpis 14.6 demonstruje omezení přístupu na stránku pro správu frekvencí pouze pro terapeuty a administrátory.

```
export const getServerSideProps = withAuthPage(
  [Role.Therapist, Role.Admin],
  ["frequencies"]
);
```

Výpis kódu 14.6: Omezení přístupu na stránku

14.7.4 Monitorování aplikace

Monitorování aplikace je klíčové pro zajištění bezpečnosti (podrobněji popsáno v kapitole 9.5). Aplikace zaznamenává především události, které manipulují s daty, jako je přidávání, mazání a upravování záznamů. Každý záznam obsahuje informace o druhu akce, identifikaci manipulovaných dat a identifikaci uživatele, který akci provedl (viz výpis kódu 14.7). Pro efektivní správu záznamů je využit nástroj Axiom, který umožňuje sledovat chování aplikace a tak identifikovat nastalé i potencionální bezpečnostní problémy a rizika. Více je nástroj popsán v kapitole 15.3.1.

```
log.info("Updating treatment", {
  id,
  name,
  playbacksPerDay,
  validUntil,
  validFrom,
});
```

Výpis kódu 14.7: Vytvoření záznamu o aktualizaci terapie

14.7.5 Ochrana proti útokům

Aplikace je navržena tak, aby byla co nejvíce odolná proti různým typům útoků a zabezpečila uživatelská data. K ochraně aplikace byla zavedena následující opatření:

Ochrana proti SQL Injection

Aplikace využívá parametrizované dotazy skrze ORM knihovnu Prisma, která zajišťuje bezpečné zpracování dotazů do databáze. Tím se minimalizuje riziko útoku SQL Injection, který by mohl zneužít chybně zpracované vstupy pro manipulaci s databází.

Ochrana proti XSS útokům

Klientská část aplikace je navržena s použitím moderního frameworku React, který poskytuje vestavěnou ochranu proti XSS útokům tím, že automaticky escapuje výstupy zobrazované na stránce. Navíc jsou použity HTTP-only cookies, které zabraňují přístupu k obnovovacímu tokenu z klientského kódu. Aplikace se vyhýbá využití lokální paměti pro ukládání citlivých dat, což dále zvyšuje zabezpečení.

Ochrana proti CSRF útokům

Aplikace využívá přístupové tokeny, které nejsou posílány pomocí cookies, ale v hlavičce požadavku. Tím se snižuje riziko CSRF útoků. Navíc je použita krátká životnost relací a pro akce manipulující s daty je využita HTTP POST metoda. Pro dodatečnou ochranu je nastaven atribut `SameSite`, což zamezuje odesílání cookies z jiných domén a dále zvyšuje bezpečnost aplikace.

14.8 Zvukové soubory

Jednou z klíčových funkcí aplikace je nahrávání a následné přehrávání uložených zvukových souborů. Tato kapitola se zabývá technikami a nástroji, které byly použity pro realizaci této funkce. Cílem bylo také zajistit, že zvukové soubory jsou bezpečně uloženy, chráněny před neoprávněným přístupem a zároveň jsou dostupné pro autorizované uživatele. V rámci řešení byla zvolena technologie AWS S3.

14.8.1 AWS S3

AWS S3 (Amazon Web Services Simple Storage Service) je webová služba od společnosti Amazon pro ukládání objektů, která nabízí škálovatelné, dostupné a trvalé úložiště pro data. S3 je navržen tak, aby bylo jednoduché ukládat a načítat data s vysokou propustností a nízkou latencí.

S3 bucket (kbelík) je základní jednotka úložiště v S3, která slouží k organizaci a správě objektů (souborů) uložených v S3. Každý bucket má unikátní název a může obsahovat neomezený počet objektů.

Aplikace využívá AWS S3 pro ukládání zvukových souborů. Služba nabízí bezplatnou úroveň s omezenými prostředky a následně zpoplatněnou při překročení těchto limitů. Z těchto důvodů a pro zajištění rychlého chodu aplikace je kladen důraz na minimalizaci počtu požadavků na server AWS S3. Ve vytvořené aplikační databázi je implementována tabulka se záznamy o jednotlivých nahraných zvukových souborech. ID těchto záznamů slouží jako klíč pro přístup k souborům v S3 bucketu.

Bucket je nastaven jako privátní, což zajišťuje, že pouze oprávněné osoby nebo služby mají přístup k objektům. Přístup z aplikace je zajištěn pomocí speciálního zabezpečovacího klíče, který je uložen pouze na aplikačním serveru a z důvodů bezpečnosti se nikdy nepředává klientovi.

Předpodepsané URL

Předpodepsané URL (presigned URL) je mechanismus, který umožňuje omezený a dočasný přístup k objektům v S3 bucketu. Server generuje URL, která obsahuje autentizační informace a platnost, která je omezena na určitou dobu. Tímto způsobem je možné získat přístup k souborům uloženým v S3 bucketu z klientské strany, aniž by bylo nutné poskytnout přístupové klíče.

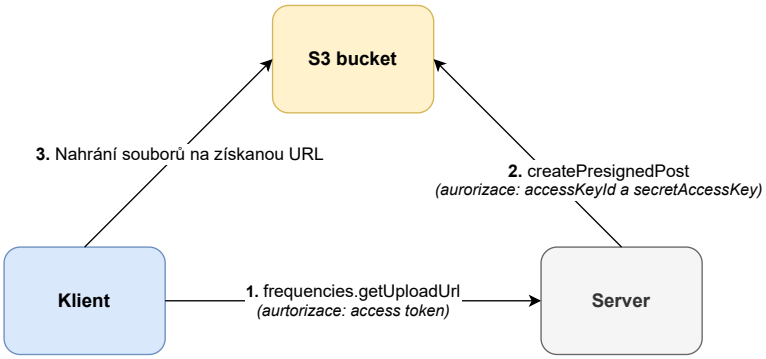
Aplikace využívá předpodepsané URL pro nahrávání a přehrávání souborů. Tyto URL adresy mají omezenou platnost, což zvyšuje bezpečnost a zamezuje neoprávněnému přístupu k souborům. Vytvoření URL pro nahrání souboru z klienta je znázorněno ve výpisu kódu 14.8.

```
export const s3GetUploadURL = async (fileName: string) =>
  createPresignedPost(s3, {
    Bucket: env.S3_BUCKET_NAME,
    Key: fileName,
    Expires: S3_URL_EXPIRATION_TIME,
    Conditions: [
      ["content-length-range", 0, MAX_AUDIO_FILE_SIZE],
    ],
  });
```

Výpis kódu 14.8: Vytvoření předpodepsané URL

14.8.2 Nahrávání souborů

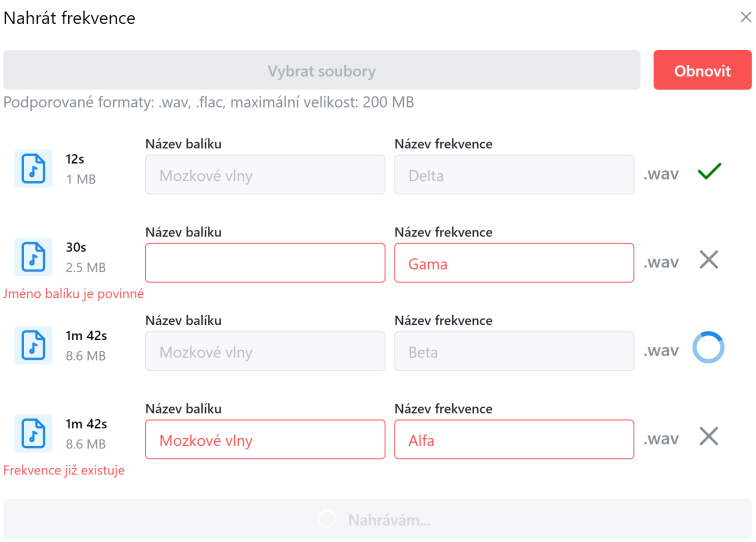
Proces nahrávání souborů je zabezpečen a zefektivněn pomocí předpodepsané URL (viz obrázek 14.4). Díky tomuto přístupu není nutné přesměrovávat data přes aplikační server, což zrychluje nahrávání souborů. Také zamezuje neautorizovanému nahrávání souborů do S3 mimo aplikaci. Maximální velikost souborů je omezena a jsou podporovány formáty `WAV` a `FLAC`.



Obrázek 14.4: Diagram nahrávání souboru na server S3

Modální okno pro nahrávání souborů

Nahrávání souborů probíhá prostřednictvím modálního okna v uživatelském rozhraní (viz obrázek 14.5). Modální okno je navrženo tak, aby bylo snadné

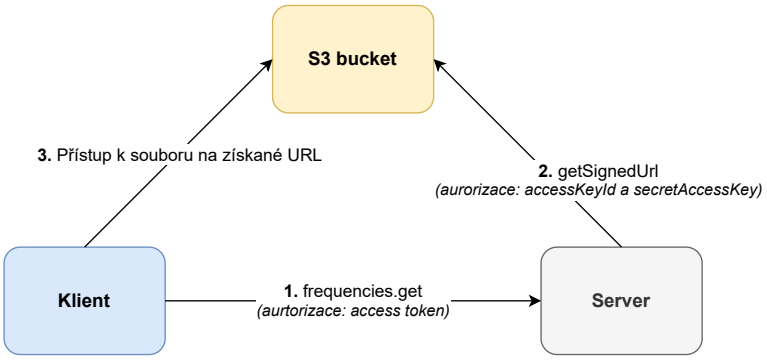


Obrázek 14.5: Modální okno pro nahrávání frekvencí

nahrávat více souborů paralelně a nastavit název a balík frekvence. Před nahráním souboru je důležité zkontrolovat, zda již v aplikaci neexistuje frekvence se stejným jménem, aby se předešlo nežádoucím duplikacím. Tato kontrola je provedena automaticky a upozorní uživatele na možný konflikt.

14.8.3 Přehrávání souborů

Přehrávání zvukových souborů je realizováno také využitím předpodepsaných URL (viz obrázek 14.6). Tímto způsobem je zajištěna ochrana zvukových souborů proti neoprávněnému přístupu mimo aplikaci. Při přehrávání souborů je využito progresivního stahování a prvku `HTMLAudioElement`.



Obrázek 14.6: Diagram přístupu k souboru na serveru S3

Progresivní stahování

Progresivní stahování znamená, že se začne soubor přehrávat paralelně s jeho stahováním. Díky tomu začne přehrávání zvuku ihned po stažení počáteční části souboru, což minimalizuje prodlevu mezi požadavkem a začátkem přehrávání. Navíc, není potřeba stahovat celý soubor, pokud uživatel přehrávání předčasně ukončí.

Oproti streamování, kde je soubor přenášen v reálném čase a přehráván ve formě kontinuálního toku, je progresivní stahování jednodušší na implementaci a méně náročné na server. Streamování má výhodu v lepším řízení šířky pásma a adaptivním přizpůsobením kvality dle rychlosti připojení, což však pro naše účely není nezbytné.

HTMLAudioElement

Pro přehrávání zvukových souborů je použit `HTMLAudioElement`, který poskytuje jednoduché a efektivní rozhraní pro manipulaci se zvukem na webo-

vých stránkách. `HTMLAudioElement` umožňuje nastavit zdroj zvukového souboru, ovládat hlasitost, spustit nebo zastavit přehrávání a sledovat události, jako je například ukončení přehrávání. Lze ho snadno vytvořit a použít přímo z JavaScriptu pomocí příkazu `new Audio()`. Tento přístup umožňuje rychlou a snadnou integraci do aplikace bez nutnosti použití HTML značek a dalších závislostí. Výpis kódu 14.9 demonstruje přehraní zvuku využitím `HTMLAudioElement`.

```
const playSound = async (
    soundPlayer: HTMLAudioElement,
    sound?: PlaylistItem
) => {
    if (!sound) return;
    soundPlayer.src = sound.url;
    soundPlayer.load();
    await soundPlayer.play();
    startInterval();
    setIsPlaying(true);
};
```

Výpis kódu 14.9: Přehrání zvuku v kódu

14.9 Odesílání emailů z aplikace

Aplikace využívá emaily pro odesílání pozvánek terapeutům a pacientům, a také v procesu obnovy zapomenutého hesla, kde slouží k ověření identity uživatele. Pro odesílání je použita služba SendGrid a pro stylování emailů je využita knihovna React Email.

14.9.1 Služba SendGrid

SendGrid je cloudová platforma pro odesílání emailů, která poskytuje škálovatelné a spolehlivé řešení pro správu a doručování emailů. SendGrid umožňuje snadnou integraci s aplikacemi prostřednictvím webového API, což zjednodušuje odesílání emailů bez nutnosti řešit komplikace spojené se správou vlastního emailového serveru. Služba SendGrid nabízí bezplatné možnosti s omezenými prostředky, jsou omezeny například počty odeslaných emailů za den. Výpis kódu 14.10 znázorňuje odeslání emailu z aplikace.

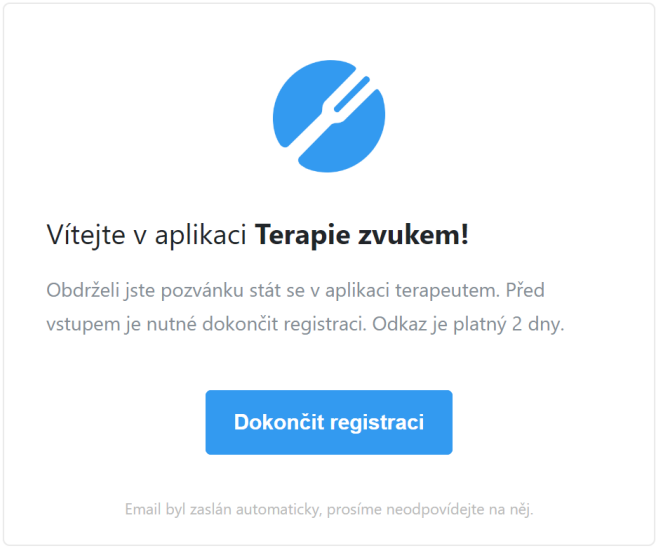
```
export const sendRegistrationEmail = async ({
  to,
  url,
  t,
  ...props
```

```
}: SendRegistrationEmailProps) => {
  const emailHtml = render(
    RegistrationEmail({ registrationUrl: url, t, ...props })
  );
  await mail.send({
    to,
    from: { email: env.EMAIL_FROM, name: t("appName") },
    subject: t("emails.registration.subject"),
    html: emailHtml,
  });
};
```

Výpis kódu 14.10: Odeslání emailu pro registraci

14.9.2 Knihovna React Email

React Email je knihovna, která zjednodušuje a zlepšuje proces stylování emailů. Vzhledem k tomu, že různí emailoví klienti a zařízení interpretují HTML a CSS kódy odlišně, může být vytváření emailových šablon s konzistentním a atraktivním vzhledem náročné. React Email řeší tuto problematiku tím, že umožňuje vytvářet strukturu a styl emailů pomocí React komponent. Ty jsou následně převedeny na univerzálně kompatibilní HTML a CSS kód, který bere v potaz zmíněná omezení. Vzhled registračního emailu vytvořeného pomocí této knihovny je znázorněn na obrázku 14.7.



Obrázek 14.7: Email pro dokončení registrace

14.10 Uživatelské rozhraní

Tato kapitola se zaměřuje na popis implementace uživatelského rozhraní webové aplikace, které bylo vyvinuto s využitím programovacího jazyka Typescript a frameworku React. V následujících částech jsou představeny základní prvky, které tvoří grafické rozhraní, a knihovny, které byly použity pro jeho vývoj. Dále jsou popsány vybrané komponenty a přiblíženy prvky, které vylepšují uživatelskou zkušenost (UX). V závěru kapitoly je popsána implementace lokalizace uživatelského rozhraní, umožňující podporu více jazyků.

14.10.1 Základní prvky

Tato část se zaměřuje na základní prvky, které jsou využívány pro tvorbu uceleného uživatelského rozhraní. Při implementaci byly vytvořeny jednotlivé obrazovky, které zpravidla využívají vytvořené komponenty, layouty a modální okna. Všechny tyto prvky spolu úzce souvisejí a dohromady tvoří srozumitelný a efektivní systém pro tvorbu grafického rozhraní webové aplikace.

Obrazovky

Obrazovky představují jednotlivé stránky aplikace a nacházejí se ve složce `Pages`. Struktura této složky odpovídá cestám URL, na kterých se obrazovky v aplikaci nacházejí. Toho je dosaženo pomocí frameworku Next.js, který také umožňuje snadné vytváření víceúrovňových cest (např. `/auth/login`) a parametrů uvnitř URL. Ty se definují využitím hranatých závorek v názvu souboru stránky. Také jsou implementovány některé stránky pro speciální případy, jako například když hledaná URL v aplikaci neexistuje, nebo když nastane chyba při načítání stránky.

Uvnitř souborů obrazovek jsou definovány a exportovány funkční React komponenty, které jsou typovány na typ `NextPage`. Stránkám jsou také definovány Next.js `ServerSideProps`, což jsou speciální funkce prováděné na straně serveru před zobrazením stránky. Ty se využívají pro inicializaci zdrojových souborů lokalizací a pro autorizaci.

Uvnitř kódu stránky jsou definovány základní vlastnosti specifické pro stránku. Pro minimalizaci opakování kódu, větší přehlednost a udržitelnost kódu jsou použity komponenty definované ve vlastních souborech. Pro opakující se vlastnosti stránek a komponent jsou využity React hooky. Jednotlivé stránky mají zpravidla jako kořenový element určitý `layout`, který zajišťuje stránce základní vlastnosti, jako je například komponenta navigace.

Komponenty

Komponenty tvoří základní stavební bloky aplikace a nacházejí se ve složce `src/components`. Tato složka je dále členěna podle druhu komponent. Jsou použity pro opakující se prvky uživatelského rozhraní nebo pro větší komponenty plnící určitou funkci. Díky tomu je dosaženo lepší přehlednosti a splněno oddělení zodpovědnosti (separation of concerns).

Komponenty jsou definovány jako React funkční komponenty s vstupními parametry určenými rozhraním. Mezi parametry často patří funkce pro zpětné volání (tzv. callbacky), které umožňují reakci na události, jako je kliknutí na tlačítko, nebo parametry pro úpravu vzhledu, funkcí a zobrazených dat komponenty. Výpis kódu 14.11 znázorňuje komponentu využívanou při filtrování seznamu dat dle zadaného textu.

```
interface SearchInputProps {
  filter: FilterType;
  updateFilter: (filter: FilterType) => void;
}
export const SearchInput = ({
  updateFilter,
  filter,
}: SearchInputProps) => {
  const { t } = useTranslation();
  return (
    <TextInput
      placeholder={t("searchPlaceholder")}
      onChange={(event) =>
        updateFilter({ search: event.currentTarget.value })
      }
      icon={<IconSearch size={18} stroke={1.5} />}
      value={filter.search ?? ""}
      rightSection={
        filter.search && (
          <ActionIcon
            onClick={() =>
                updateFilter({ search: undefined })}
          >
            <IconX size={18} />
          </ActionIcon>
        )
      }
    />
  );
};
```

Výpis kódu 14.11: Komponenta textového vyhledávání

Layouty

Layouty jsou speciální komponenty určené pro zobrazování stránek uvnitř sebe. Obsahují opakující se elementy a umožňují nastavit obsah obrazovky definující klíčovou funkcionalitu stránky. Pomocí layoutů se také nastavuje název stránky, který je následně viditelný jako název záložky v prohlížeči. V aplikaci jsou definovány dva druhy layoutů: veřejný a pro přihlášené uživatele.

Veřejný layout obsahuje primárně logo s názvem aplikace a je používán například pro stránku přihlášení, registrace a změny hesla. Layout pro přihlášené uživatele (viz obrázek 14.8). obsahuje navigaci, která se přizpůsobuje roli přihlášeného uživatele. Dále obsahuje jméno a roli uživatele a kontextové menu, umožňující například odhlášení a zobrazení profilu uživatele.

Obrázek 14.8: Layout pro terapeuta

Modální okna

Modální okna jsou grafické prvky, které částečně překrývají aktuální zobrazenou stránku. Využívají se pro dodatečné funkcionality aplikace a zobrazují se zpravidla v reakci na určitou událost. Modální okna mohou sloužit k potvrzení vážných a nevratných akcí, např. pro potvrzení odstranění uživatele, nebo k vytvoření menších funkcionalit, které nevyžadují vytváření nové stránky, ale souvisejí s určitou vlastností zobrazené stránky. V aplikaci je například vytvořené modální okno pro nahrání frekvencí do aplikace, které je dostupné ze stránky zobrazující seznam všech frekvencí (viz obrázek 14.5).

Vzhled modálních oken se definuje využitím React funkčních komponent a jejich zobrazení je řešeno pomocí funkce `openModal` definované v knihovně Mantine. Funkci `openModal` lze předat komponentu modálního okna, název modálu a další konfigurační parametry (viz výpis kódu 14.12). Tím je zajištěno korektní zobrazení a chování okna dle zásad webové přístupnosti.

```
1 interface CreateTemplateProps {
2   playlistId?: string;
3   redirect?: boolean;
4   t: TFunction;
5 }
6 export const openCreateTemplateModal = (
7   props: CreateTemplateProps
```

```
) => {
  openModal({
    title: props.t("createTemplate.title"),
    children: <CreateTemplate {...props} />,
  });
};
```

Výpis kódu 14.12: Vyvolání otevření modálního okna

14.10.2 Knihovna Mantine

Pro vývoj grafického rozhraní aplikace byla využita knihovna Mantine. Ta patří mezi grafické knihovny, což jsou sady nástrojů a komponent navržené pro usnadnění a urychlení vývoje webových aplikací díky předem navrženým a otestovaným vizuálním prvkům a funkcím. Mantine umožňuje snadno vytvářet konzistentní, responzivní a přístupné uživatelské rozhraní bez nutnosti implementace a stylování každé komponenty, tak aby byla funkční a vzhledově konzistentní. Mezi nabízené komponenty patří tlačítka, různé druhy vstupů, formulářové prvky, modální okna, navigační prvky a další vizuální prvky.

Knihovna Mantine se vyznačuje vysokou mírou přizpůsobitelnosti a snadným stylováním komponent různými způsoby. Tato vlastnost umožňuje přizpůsobit vzhled aplikace podle potřeb a preferencí. Navíc klade důraz na webovou přístupnost, což zajišťuje širokou dostupnost aplikací pro různé skupiny uživatelů, včetně osob se zdravotním postižením.

14.10.3 Sada ikon Tabler

V aplikaci byla využita sada ikon Tabler, která je navržená pro zlepšení vizuálního zážitku a přehlednosti uživatelského rozhraní. Ikonografie hraje klíčovou roli v návrhu uživatelských rozhraní, protože umožňuje rychlejší a intuitivnější identifikaci funkcí komponent, a to díky okamžité rozpoznatelnosti obrázků bez nutnosti čtení textu.

Sada ikon Tabler nabízí širokou škálu ikon, které pokrývají různé oblasti a kategorie, jako jsou navigace, interakce, nástroje, média a další. Tyto ikony jsou navrženy s ohledem na konzistentnost a čitelnost, což zajišťuje jednotný vizuální výraz aplikace.

Vložení ikony Tabler do aplikace je jednoduché a efektivní. Po nainstalování závislosti stačí importovat a použít React komponentu, která popisuje potřebnou ikonu. Tento přístup umožňuje snadné a rychlé začlenění ikon do aplikace, aniž by bylo nutné pracovat s externími zdroji nebo složitě upravo-

vat SVG soubory. Ukázka kódu 14.13 znázorňuje využití ikony odpadkového koše pro tlačítko smazání.

```
<ActionIcon
  ml={10}
  onClick={() =>
    openRemoveFrequencyModal({ id: frequency.id, t })
  }
>
  <IconTrash />
</ActionIcon>
```

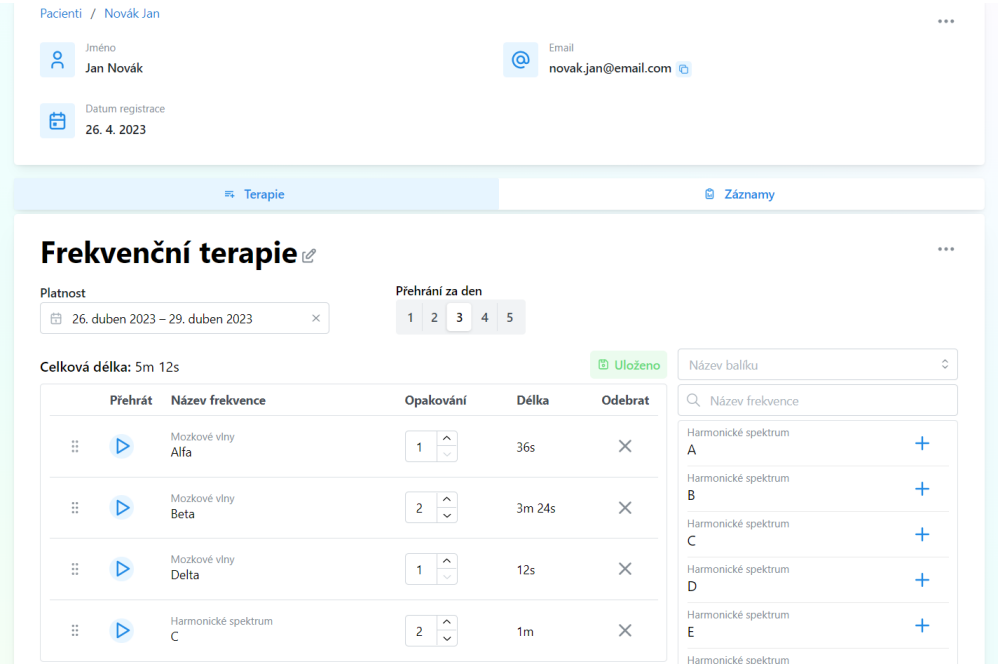
Výpis kódu 14.13: Využití ikony

14.10.4 Popis vybraných komponent

Tato kapitola se zaměřuje na detailní popsání vybraných komponent, které byly implementovány v rámci vyvinuté aplikace.

Playlist

Komponenta `Playlist` je navržena pro vytváření zvukových terapií. Je primárně určena pro terapeuty a používá se na stránce detailu klienta (viz obrázek 14.9) a stránce detailu šablony. Komponenta se skládá ze dvou hlavních



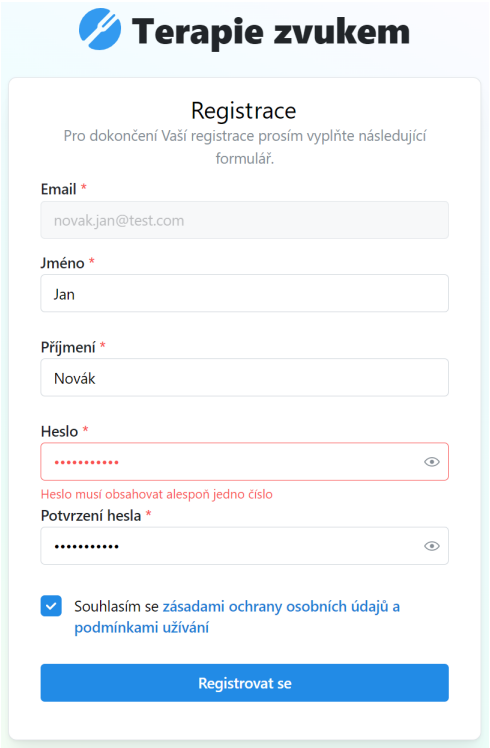
Obrázek 14.9: Stránka detailu klienta

částí: první část zobrazuje frekvence v playlistu a umožňuje nastavit jejich pořadí a počet opakování, zatímco druhá část slouží k vyhledávání dostupných frekvencí a jejich přidání do playlistu. Frekvence lze ve vyhledávání filtrovat podle názvu a názvu balíku.

Pro ukládání změn je použita `debounce` funkce, která snižuje zátěž serveru tím, že akumuluje více požadavků a posílá je společně. Komponenta využívá knihovnu React Beautiful Dnd pro přesun položek playlistu způsobem „drag and drop“, což umožňuje určení pořadí přehrávání. Seznam všech frekvencí pracuje na principu „infinite scroll“, kdy data jsou načítána dynamicky, jakmile se uživatel přiblíží ke konci seznamu.

Formulář

Pro vytváření a správu formulářů byla vytvořena pomocná komponenta `Form` (viz obrázek 14.10). Tato komponenta usnadňuje validaci dat, zobrazování chybových hlášek a odesílání dat na server po úspěšném vyplnění formuláře. Komponenta vyžaduje jako vstupní parametr Zod schéma, které definuje očekávané informace od uživatele. Podle tohoto schématu jsou data validována a na základě výsledků jsou vytvářeny chybové hlášky, které jsou často



Obrázek 14.10: Registrační formulář

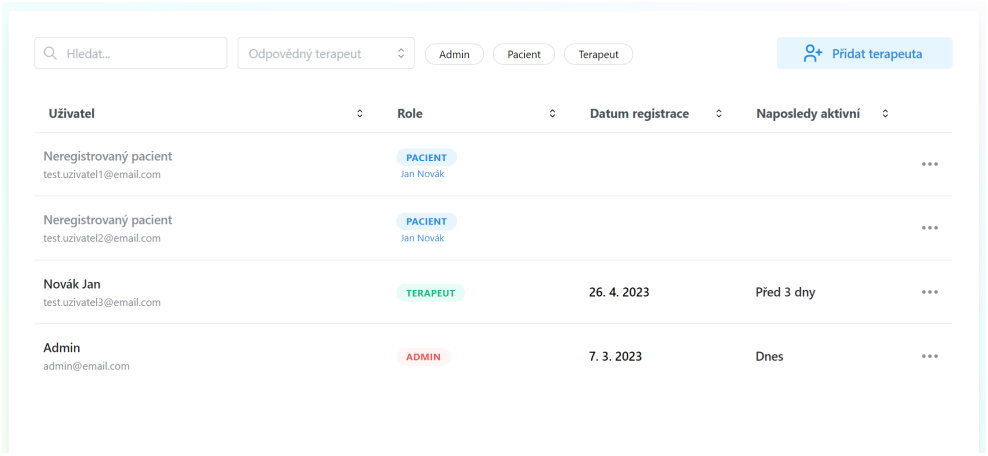
přiřazeny konkrétnímu prvku schématu. Pokud data nejsou validní, nelze je odeslat na server. Výpis kódu 14.14 znázorňuje využití komponenty `Form` pro vytvoření formuláře aktualizující jméno a balík frekvence.

```
<Form
    onSubmit={updateFrequencyMutation.mutateAsync}
    schema={updateFrequencySchema}
    initialValues={{ id, name, packageName }}
    submitButtonLabel={t("update.saveButton")}
>
  {({ getInputProps }) => (
    <>
      <TextInput
        required
        label={t("update.name")}
        {...getInputProps("name")}
      />
      <TextInput
        required
        label={t("update.packageName")}
        {...getInputProps("packageName")}
      />
    </>
  )}
</Form>
```

Výpis kódu 14.14: Formulář aktualizace frekvence

Tabulky

Pro zobrazení dat v tabulkách je použita komponenta `Table` z knihovny Mantine. Aplikace umožňuje funkce, jako jsou filtrování, řazení a stránkování dat, které jsou řešeny na straně serveru. Odpovědi na požadavky jsou navíc ukládány do vyrovnávací paměti, což snižuje objem dat pro přenos a zrychluje chod aplikace. Pro usnadnění vytváření požadavků na tabulková data na straně klienta, které zahrnují zmíněné funkcionality, byl vytvořen pomocný React Hook `useList`. Tento hook navíc umožňuje měnit počet požadovaných dat na stránku podle velikosti obrazovky. Přesun mezi stránkami je řešen komponentou `Pagination`. Pro tabulky byla navíc vytvořena komponenta `SortableTH`, která umožňuje řazení dat kliknutím na hlavičku sloupce. V tomto případě se střídají tři varianty řazení: výchozí, vzestupná a sestupná. Obrázek 14.11 znázorňuje seznam uživatelů s možností filtrování, stránkování, řazení dle jednotlivých sloupců a možností prokliknutí na detail uživatele.



Obrázek 14.11: Seznam uživatelů z pohledu administrátora

14.10.5 UX prvky

Při vytváření intuitivního a uživatelsky přívětivého rozhraní bylo využito různých UX praktik. Tyto prvky nepřidávají aplikaci přímou funkcionalitu, ale napomáhají uživatelům porozumět aplikaci a vytvářejí plynulejší a interaktivnější zážitek. Poskytují také další možnosti jak informovat uživatele o stavu aplikace. V aplikaci pro širokou škálu uživatelů to je velmi důležité, protože minimalizuje zmatení nových uživatelů a dodává pocit kvalitní aplikace.

Pro dosažení těchto cílů je v aplikaci mj. kladen důraz na vhodné využívání vizuálních prvků, jako jsou ikony, barvy a další prvky. Zpravidla je v aplikaci použita modrá barva jako neutrální, zelená jako pozitivní, šedá pro neaktivní prvky a červená barva pro signalizaci chyb a zásadních akcí.

Dále pro komunikaci s uživatelem jsou využity notifikace (viz výpis kódu 14.15). Ty se zobrazují v pravém dolním rohu aplikace, nepůsobí příliš rušivě a samy po uplynutí času zmizí. Využívají dříve popsaných barev pro jasnou signalizaci typu zobrazené zprávy. Informují a potvrzují vykonané akce, upozorňují na nevalidní data a vzniklé chyby.

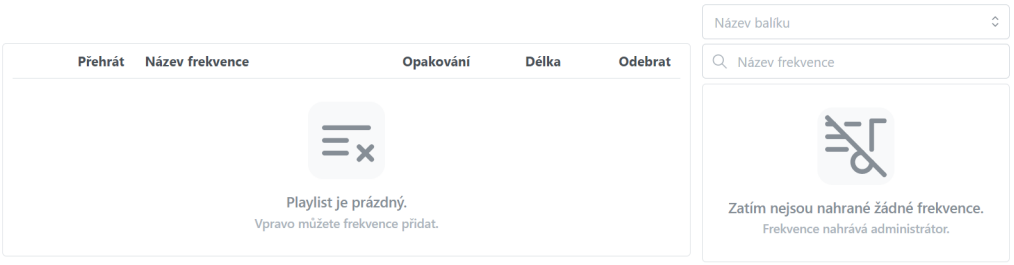
Placeholdery jsou dalším UX prvkem využitým v aplikaci (viz obrázek 14.12). Vyplňují místa, pro která ještě neexistují data, a informují tak vhodným způsobem, že aplikace funguje korektně a data pouze neexistují. Dále také seznamují uživatele s tím, jaká data mají na tomto místě očekávat a jaké kroky musí provést, nebo jaké události se musí stát, aby se data na tomto místě vyskytla.

Také jsou využity speciální prvky signalizující načítání dat. Kromě klasických komponent jsou využity celé kostry, které signalizují načítání jednotlivých elementů a umožňují tak dynamické vykreslování grafického rozhraní

s chybějícími daty. Tento přístup byl zvolen se snahou minimalizace CLS (Cumulative Layout Shift), tedy neočekávaným posunům uživatelského rozhraní, které narušují uživatelský zážitek.

```
const changePasswordMutation =
    api.auth.changePassword.useMutation({
      onSuccess: () => {
        closeAllModals();
        showNotification({
          title: t("changePassword.success.title"),
          message: t("changePassword.success.message"),
          color: "teal",
        });
      },
    });
```

Výpis kódu 14.15: Zobrazení notifikace po úspěšné změně hesla



Obrázek 14.12: Prázdné seznamy

14.10.6 Lokalizace

Jedním z klíčových faktorů pro zajištění široké dostupnosti aplikace je podpora více jazyků. Lokalizace textů v aplikaci umožňuje přizpůsobit obsah uživatelům mluvícím různými jazyky a zjednodušuje jejich interakci s aplikací. V současné době je aplikace lokalizována do češtiny a angličtiny, a to s využitím knihovny Next-i18next.

Knihovna Next-i18next

Knihovna Next-i18next je navržena speciálně pro Next.js aplikace a umožňuje snadnou integraci lokalizace. Pro definici lokalizovaných textů využívá soubory ve formátu JSON, které jsou rozděleny do jednotlivých jmenných prostorů. Lokalizační soubory jsou uloženy ve složce `public/locales` a jsou rozděleny do podadresářů pro každý podporovaný jazyk. V rámci těchto

adresářů jsou pak uloženy soubory s lokalizacemi pro jednotlivé jmenné prostory.

Hlavním jmenným prostorem je `common`, který obsahuje lokalizace používané napříč celou aplikací. Pro snížení datové náročnosti a zrychlení načítání stránek je nutné pro každou stránku explicitně definovat seznam použitých jmenných prostorů. Jmenný prostor `common` je zahrnut automaticky.

Díky využití jazyka TypeScript je možné z JSON struktury generovat odpovídající datové typy. Tím je zajištěno, že během vývoje budou klíče pro lokalizace kontrolovány a doplňovány, minimalizuje se tak možnost chyb. Výpis kódu 14.16 znázorňuje využití funkce `t` pro lokalizaci textů v hlavičce tabulky.

```
<thead>
  <tr>
    <th>{t("playlist.play")}</th>
    <th>{t("playlist.name")}</th>
    <th>{t("playlist.repeat")}</th>
    <th>{t("playlist.duration")}</th>
    <th>{t("playlist.remove")}</th>
 </tr>
</thead>
```

Výpis kódu 14.16: Využití lokalizační funkce

Nastavení jazyka v aplikaci

Pro přepínání jazyků v aplikaci byla vytvořena speciální komponenta, která umožňuje uživateli snadno změnit jazyk. Jazyk aplikace je nastaven automaticky podle výchozího jazyka prohlížeče. Uživatel má ale možnost jazyk změnit prostřednictvím dostupné komponenty.

Přepnutí jazyka konkrétně probíhá prostřednictvím úpravy URL aplikace. URL obsahuje klíč lokalizace, například `/en/auth/registration` pro anglickou verzi. V případě českého jazyka se označení `cs` nemusí uvádět, jelikož se jedná o výchozí jazyk aplikace.

15 Nasazení vytvořené aplikace

Nasazení aplikace je jedním z nejdůležitějších kroků ve vývojovém procesu, jelikož zde dochází k přenesení aplikace do testovacího či produkčního prostředí, kde ji budou využívat skuteční uživatelé. Pro zajištění správného fungování aplikace byla snaha efektivní implementace tohoto procesu, který zahrnuje automatizaci integrace a nasazení, hostování aplikace a následný monitoring a údržbu.

15.1 Automatizace integrace a nasazení

Pro zajištění efektivního průběhu nasazení a zrychlení vývojového procesu je tento proces automatizován. K tomu je využit nástroj GitLab CI.

15.1.1 GitLab CI

GitLab CI je nástroj pro automatizaci integrace a nasazení aplikace. Umožňuje průběžně testovat a nasazovat změny do vývojového nebo produkčního prostředí. GitLab CI poskytuje způsob, jak definovat různé úkoly, které mají být provedeny při nasazení, jako je sestavení, testování a nasazení aplikace. Jedná se o jeden z nejpopulárnějších nástrojů pro tyto účely a je také výhodný z důvodu integrace s GitLab repositářem která je využití pro vývoj aplikace.

15.1.2 CI/CD pipeline

V následujících odstavcích jsou popsány jednotlivé fáze CI/CD pipeline (viz obrázek 15.1). Jejich definice a konfigurace v YAML syntaxi se nachází v souboru `.gitlab-ci.yml`. Jsou rozděleny do několika úrovní: příprava, sestavení, testování a nasazení.

Příprava (`.pre`)

V této fázi se instalují všechny závislosti potřebné pro sestavení a testování aplikace. Příkazem `pnpm install` se nainstalují závislosti podle konfigurace v souboru `pnpm-lock.yaml`. Výsledné soubory `node_modules` a `.pnpm-store` jsou následně uloženy do cache pro rychlejší opětovný chod pipeline a předány dalším fázím formou artefaktů.

Sestavení (`build`)

Během fáze sestavení je nainstalován nástroj Vercel. Poté se pomocí příkazu `vercel pull` stáhnou potřebné konfigurace a `vercel build` sestaví aplikaci do produkční verze. Výsledek sestavení je uložen do složky `.vercel`, která se ukládá formou artefaktu a využívá při kroku nasazení.

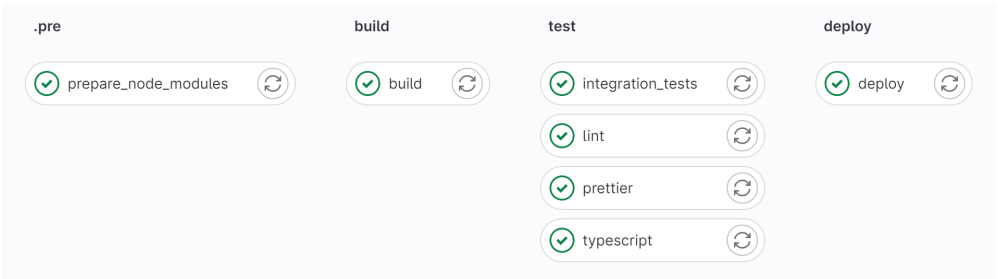
Testování (`test`)

Ve fázi testování jsou zařazeny následující paralelně vykonávané úkoly, které zajišťují kvalitu kódu a správnou funkčnost aplikace:

- **integrační testy**: Spouštějí se pomocí příkazu `pnpm test` a ověřují správnou funkcionalitu aplikace,
- **Lint**: Provádí kontrolu kvality kódu pomocí nástroje ESLint příkazem `pnpm lint`,
- **Prettier**: Kontroluje jednotné formátování kódu pomocí příkazu `pnpm prettier:check`,
- **TypeScript**: Provádí kontrolu správného použití typů příkazem `pnpm types`.

Nasazení (`deploy`)

V poslední fázi nasazení se využívá nástroj Vercel a výsledek sestavení z předchozí fáze. Pomocí příkazu `vercel deploy` se nasadí předem sestavená aplikace do produkčního prostředí.



Obrázek 15.1: Vizualizace úspěšné pipeline v nástroji GitLab

15.2 Hostování aplikace

Volba vhodného poskytovatele hostování je klíčová pro zajištění kvalitní dostupnosti a chodu aplikace. Během vývoje byly vyzkoušeny různé služby, které se liší v aspektech, jako jsou finanční náklady, omezení zdrojů nebo proces nasazení sestavené aplikace do produkčního prostředí. Některé služby vyžadují kontejnerizaci, a proto byl také vytvořen soubor `.Dockerfile`, který umožňuje nasadit aplikaci v Docker kontejneru. Bylo experimentováno se službami Railway, DigitalOcean a Vercel. Nakonec byla zvolena služba Vercel pro její výhody popsané v následující kapitole. Aplikace je dostupná na následující adrese:

`https://terapie-zvukem.vercel.app/`

15.2.1 Vercel

Vercel je platforma pro hostování webových aplikací, která nabízí širokou škálu funkcí pro vývojáře. Mezi tyto funkce patří:

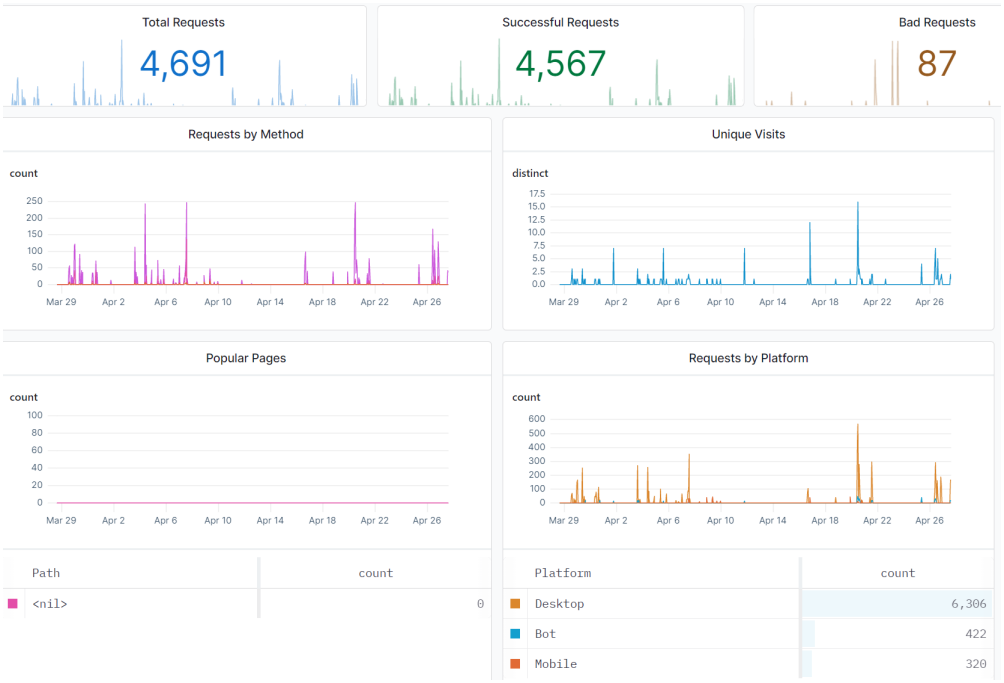
- **serverless prostředí:** Vercel umožňuje snadné vytváření a správu serverless funkcí, které zajišťují efektivní a škálovatelné zpracování požadavků,
- **edge prostředí:** Vercel poskytuje edge prostředí, což znamená, že statické části aplikace a serverless funkce jsou rozloženy na více serverech po celém světě, čímž se zajišťuje rychlé načítání a nízká latence bez ohledu na geografickou polohu uživatele,
- **integrace s Next.js:** Vercel je ideální volbou pro hostování aplikací postavených na Next.js, jelikož byl navržen s ohledem na tento framework a poskytuje optimální výkon a spolehlivost,
- **nízká náročnost na náklady:** Vercel nabízí zdarma hostování s poměrně nízkými omezeními, což umožňuje vývojářům snadno nasadit a udržovat aplikace bez značných finančních investic.

15.3 Monitorování a údržba

Po nasazení aplikace je nezbytné průběžně sledovat její výkon a zajistit, že aplikace zůstává funkční a dostupná pro uživatele. Monitorování aplikace také umožňuje sběr cenných informací, které mohou napomoci v jejím dalším vylepšování a optimalizaci. K tomuto účelu byly využity služby Axiom a Sentry.

15.3.1 Axiom

Axiom je nástroj pro monitorování běhu aplikací, který umožňuje sledovat výkon aplikace, zaznamenávat chyby a monitorovat metriky ve všech fázích vývoje a nasazení. Díky tomu, že byl navržen přímo pro technologii Next.js a Vercel, poskytuje velmi kvalitní a detailní záznamy. Axiom již byl zmíněn v aplikaci o zabezpečení aplikace, jelikož napomáhá předcházet a řešit bezpečnostní incidenty. Axiom navíc poskytuje přehledný a intuitivní webové rozhraní pro sledování a vizualizaci metrik, statistik a zobrazení a filtrování záznamů. Obrázek 15.2 znázorňuje grafické znázornění vybraných statistik ve webovém rozhraní aplikace.

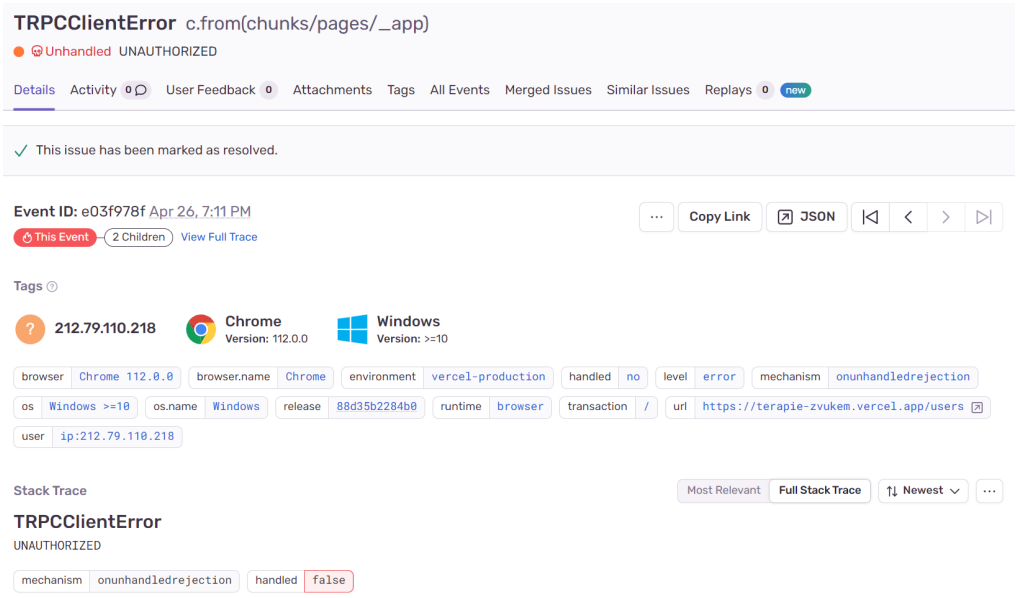


Obrázek 15.2: Vizualizace monitorovaných statistik v Axiom

15.3.2 Sentry

Sentry je nástroj pro sledování chyb v aplikaci, který umožňuje efektivně zaznamenávat a analyzovat chyby, které se vyskytují během provozu aplikace. Díky tomu lze zjistit, jaké výjimky vznikají během používání aplikace uživateli, které se vyskytují častěji a způsobují vážný problém pro fungování aplikace. Vývojáři se tak o chybách dozví dříve, než je uživatelé nahlásí, a mají tak možnost je rychleji řešit. Sentry poskytuje podrobné informace o každém incidentu, včetně kontextu, který mu předcházel, a prostředí, ve

kterém vznikl. To eliminuje nutnost získávat tyto klíčové informace od uživatelů a usnadňuje řešení problémů. Navíc službu je možné s určitými omezeními používat zdarma. Obrázek 15.3 znázorňuje některé informace dostupné o nahlášené chybě v nástroji Sentry.



Obrázek 15.3: Reportovaná chyba aplikace v nástroji Sentry

16 Validace aplikace

Validace aplikace je důležitým krokem v procesu vývoje, který zajišťuje, že aplikace splňuje požadavky a očekávání uživatelů, a zároveň funguje správně a efektivně. V rámci validace aplikace byly implementovány a zařazeny do CI/CD procesu automatické testy, dále bylo provedeno uživatelské testování a testování výkonu aplikace.

16.1 Automatizované testy

Pro ověření správného chování aplikace a zajištění vysoké kvality kódu byl implementován systém automatických testů. Tyto testy pokrývají různé aspekty aplikace, včetně integračních testů a testů uživatelského rozhraní. Testy jsou spouštěny automaticky v rámci CI/CD pipeline, je ale také možné je spustit manuálně pomocí příkazu `pnpm test`. Celkem bylo vytvořeno 127 testů s pokrytím uvedeným v tabulce 16.1 (analýza pokrytí proběhla využitím nástroje C8). Jsou umístěny ve složce `__tests__`, která rozděluje testy do dvou hlavních kategorií: testy klientské části a testy serverové části.

	Příkazy	Podmínky	Funkce	Řádky
Serverová část	85,03%	81,23%	70,7%	85,03%
Klientská část	79,71%	78,99%	52%	79,71%
Celkem	**83,73%**	**79,44%**	**67,98%**	**83,73%**

Tabulka 16.1: Pokrytí kódu testy

16.1.1 Využité technologie

Testy byly napsány v jazyce TypeScript a dále byly využity následující technologie.

Vitest

Jako základní technologie byla zvolena knihovna Vitest. Tento testovací nástroj, byl navržen s důrazem na rychlost a snadnou integraci s různými frameworky, včetně Reactu. Knihovna Vitest je postavena na základech technologie Vite, což je nástroj pro efektivní vytváření lokálních serverových

prostředí. Díky tomu knihovna nabízí vylepšený výkon a efektivitu při testování oproti tradičním testovacím nástrojům, jako je například Jest.

Vitest a podobné technologie využívají knihovny, jako jsou například JSDOM a Happy DOM, které dokáží simulovat prostředí webového prohlížeče v Node.js. Díky tomu je testování rychlejší a efektivnější. Knihovna Vitest ale podporuje i integraci s nástroji, které nabízejí testování v reálném prohlížeči. Technologie WebdriverIO nabízí prostředí prohlížečů Firefox, Chrome, Edge a Safari. Playwright, který poskytuje engine prohlížeče, nabízí Firefox, Webkit a Chromium. Tento způsob testování je však pomalejší, náročnější na zdroje, prostředí a konfiguraci.

React Testing Library

React Testing Library je robustní knihovna, která se zaměřuje na testování React komponent z uživatelského hlediska. Tato knihovna je klíčovým nástrojem pro testování uživatelského rozhraní aplikace.

React Testing Library poskytuje možnost snadno simulovat interakce s jednotlivými elementy a testovat pomocí asynchronních funkcí, což zahrnuje simulaci kliknutí, změny hodnot formulářových polí a dalších akcí, které mohou uživatelé provádět při použití aplikace. Tato schopnost zjednodušuje testování interaktivních komponent a dynamického chování aplikace. React Testing Library také nabízí pokročilé nástroje pro vyhledávání elementů na stránce a testování jejich vlastností. To umožňuje vývojářům snadno ověřovat, zda se komponenty zobrazují a chovají správně.

MSW

MSW (Mock Service Worker) je knihovna, která umožňuje efektivně simulovat odpovědi webového API v Node.js prostředí, aniž by bylo nutné spouštět skutečný server. V rámci diplomové práce byl MSW využit pro testování React komponent, které komunikují se serverovým API. Knihovna je schopna zachytávat požadavky generované tRPC funkcemi a dosazovat mockované odpovědi. Díky tomu lze izolovaně testovat komponenty závislé na datech ze serveru bez nutnosti zásahu do jejich kódu.

MSW umožňuje testovat komponenty v různých scénářích v závislosti na charakteristikách serverových dat. Například může být testováno, jak komponenty reagují na neúplná nebo neplatná data, chybové stavy či různé varianty úspěšných odpovědí. Dále MSW nabízí možnost testovat interakce s API prostřednictvím callbacků, což umožňuje ověřit, že komponenty správně volají serverové API s očekávanými parametry.

16.1.2 Testování uživatelského rozhraní

Testování uživatelského rozhraní zahrnuje ověřování správné funkčnosti a chování komponent, modálních oken a jednotlivých stránek aplikace. Testy se nacházejí ve složce `__tests__/frontend`, ta je organizována do souborů, které testují jednotlivé komponenty, modální okna a stránky aplikace.

V rámci testování uživatelského rozhraní je využita knihovna MSW, která umožňuje odposlouchávat požadavky na server a vrátit požadovaná mockovaná data. To vytváří izolaci od serverového kódu a zjednodušuje testování komponent vícero scénáři. V testech jsou často použity asynchronní testovací funkce. To zahrnuje čekání na vykreslení části komponenty, která je závislá na serverových datech, a testování dynamicky se měnících komponent v závislosti na uživatelské interakci.

Výpis kódu 16.1 demonstruje test přihlašovacího formuláře. Testuje se, zda jsou z komponenty odeslána zadaná data, proběhne přesměrování a je nastaven přístupový token.

```
const loginButton =
    getByText<"auth">("login.button").closest("button");
const emailInput =
    getByLabelText<"auth",HTMLInputElement>("email");
const passwordInput =
    getByLabelText<"auth", HTMLInputElement>("password");

const email = "test@email.com";
const password = "password";
const { callback, response } = loginResponseMock;

if (emailInput && passwordInput && loginButton) {
  await userEvent.type(emailInput, email);
  await userEvent.type(passwordInput, password);
  await userEvent.click(loginButton);
  await waitFor(() => {
    expect(callback).toBeCalledWith({
      email,
      password,
      stayLoggedIn: true,
    });
    expect(NextRouterMock.pathname)
        .toBe(response.redirectURL);
    expect(getAccessToken())
        .toStrictEqual(response.accessToken);
  });
}
```

Výpis kódu 16.1: Test formuláře přihlášení

Jednotlivé komponenty, stránky a modální okna jsou zpravidla testovány pomocí vícero testů, které zajišťují korektní chování v závislosti na mockovaných datech. V rámci testování je také nutné mockovat knihovny specifické pro běhové prostředí prohlížeče.

16.1.3 Testování serverové části

Testování serverové části zahrnuje ověřování správné funkčnosti a chování jednotlivých koncových bodů serverového API. Implementace testů se nachází ve složce `__tests__/backend`, která je organizována do souborů, které se zaměřují na jednotlivé routery vytvořené pro různé aspekty a doménové entity aplikace.

Pro testování serverové části je využita funkce `createCaller`, která umožňuje simulovat volání API s různými kontexty v rámci kódu bez nutnosti volat API externě. Díky tomu lze snadno určit kontext požadavku, což je užitečné zejména pro simulování požadavků od různých rolí přihlášených uživatelů a jejich interakce s API. Koncové body serverového API jsou zpravidla pokryty několika testy, které se liší v kontextu, vstupních datech a očekávaných výsledcích. To zahrnuje testování struktury a obsahu výstupních dat, vyhozených výjimek a chování aplikace v různých situacích.

Vzhledem k tomu, že serverová část aplikace pracuje s databází, je nezbytné zajistit správný obsah testovacích dat před spuštěním testů a následně odstranit tato data po dokončení testování. Tento postup zajišťuje izolaci jednotlivých testů a zabraňuje nechtěnému ovlivňování výsledků testů.

Test koncového bodu pro přihlášení se správnými přihlašovacími údaji, kontrolující úspěšnou odpověď, je znázorněn ve výpisu kódu 16.2.

```
1  const login = caller.auth.logIn({
2    email: registeredUserMail,
3    password: validPassword,
4  });
5
6  const output: Output = {
7    redirectURL: expect.any(String),
8    accessToken: {
9      refreshInterval: expect.any(Number),
10     token: expect.any(String),
11   },
12 };
13
14 await expect(login).resolves.toMatchObject(output);
```

Výpis kódu 16.2: Test koncového bodu přihlášení

16.2 Uživatelské testování

Uživatelské testování představuje zásadní složku ověřování kvality aplikace, protože poskytuje cennou zpětnou vazbu od reálných uživatelů. Tímto způsobem lze identifikovat potenciální problémy a nedostatky, které mohly být přehlédnuty během vývoje. Tyto problémy mohou zahrnovat nekorektní chování, neintuitivní nebo neefektivní uživatelské rozhraní či nesplnění skutečných potřeb a očekávání uživatelů. Uživatelské testování také hraje klíčovou roli v identifikaci oblastí pro vylepšení a rozšíření aplikace. Aplikace byla prezentována potenciálním uživatelům s cílem získat co nejvíce názorů a zkušeností. Konkrétně bylo provedeno testování ve spolupráci s 8 terapeuty a 8 klienty. Zpětná vazba byla shromažďována prostřednictvím speciálně navržených dotazníků.

16.2.1 Vytvořené dotazníky

Pro získání zpětné vazby byly vytvořeny dva typy dotazníků, které se zaměřují na klienty a terapeuty. To z důvodu, že způsob, jakým aplikaci používají, se značně liší. Dotazník určený pro klienty zahrnuje následující otázky:

1. Jaký je Váš věk?
2. Jak hodnotíte na stupnici 1-10 poslech frekvenční terapie z aplikace?
3. Setkali jste se při používání aplikace s nějakými technickými problémy? Pokud ano, popište je prosím.
4. Považujete používání uživatelského rozhraní aplikace za snadné a intuitivní? Pokud ne, jaká vylepšení byste navrhli?
5. Jsou nějaké funkce nebo vlastnosti, které byste rádi viděli v budoucí aktualizaci aplikace?
6. Dokážete si představit, že aplikaci budete pravidelně používat? Pokud ne, z jakého důvodu?
7. Posloucháte pravidelně jiné nahrávky se zvukovou terapií nebo používáte nějaké jiné aplikace pro zvukovou terapii/meditaci? Pokud ano, které?
8. Další zpětná vazba nebo návrhy týkající se naší aplikace.

Dotazník určený pro terapeuty zahrnuje následující otázky:

1. Jaký je Váš věk?

2. Setkali jste se při používání aplikace s nějakými technickými problémy? Pokud ano, popište je prosím.

3. Považujete používání uživatelského rozhraní aplikace za snadné a intuitivní? Pokud ne, jaká vylepšení byste navrhli?

4. Jsou nějaké funkce nebo vlastnosti, které byste rádi viděli v budoucí aktualizaci aplikace?

5. Dokážete si představit, že aplikaci budete využívat pro rozšíření kontaktní terapie svých pacientům? Pokud ne, z jakého důvodu?

6. Zde můžete sdělit další zpětnou vazbu nebo návrhy týkající se naší aplikace. Váš názor je pro nás důležitý, a bude využit pro další vývoj aplikace.

16.2.2 Získaná zpětná vazba

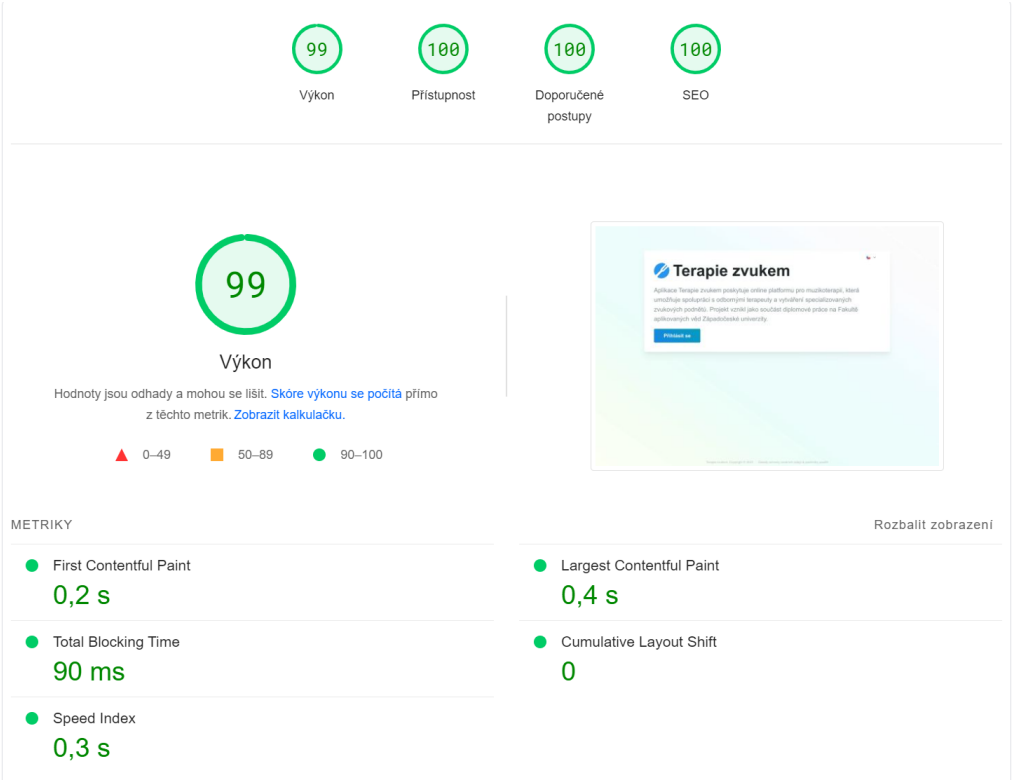
Na základě zpětné vazby od uživatelů bylo možné identifikovat oblasti, které vyžadovaly úpravy nebo vylepšení. Některé identifikované aspekty byly implementovány do aplikace, zatímco jiné vyžadovaly podrobnější analýzu případu užití a zatím nebyly zahrnuty. Příkladem implementovaných úprav je notifikace potvrzující úspěšnou úpravu terapie a možnost nastavit platnost terapie od konkrétního data.

Uživatelé hodnotili používání a návrh aplikace pozitivně a nebyly identifikovány žádné výraznější nedostatky. Terapeuti oceňovali aplikaci a uznávali smysl jejího používání v rámci svých služeb. Většina dotazovaných klientů měla zkušenosti s muzikoterapií v digitálním prostředí a hodnotila tuto aplikaci pozitivně ve srovnání s jinými dostupnými řešeními. Několikrát byla zmíněna potřeba kvalitních sluchátek nebo reproduktorů pro dosažení očekávaného zážitku.

16.3 Testování výkonu

Kromě uživatelského testování byl výkon aplikace také hodnocen pomocí specializovaných nástrojů, jako je například PageSpeed Insights od společnosti Google. Tento nástroj poskytuje užitečné statistiky, které zahrnují časy načítání stránek, přístupnost, optimalizaci pro vyhledávače (SEO) a dodržování doporučených postupů pro webové vývojáře.

Vytvořená webová aplikace si v těchto kategoriích vedla velmi dobře, což dokládá následující obrázek 16.1.



Obrázek 16.1: Analýza výkonu nástrojem PageSpeed Insights

17 Zhodnocení výsledků

V rámci této diplomové práce byla úspěšně navržena, implementována, otestována a nasazena do produkčního prostředí aplikace, splňující veškeré požadavky terapeuta, který ji postupně integruje do svých služeb. Aplikace byla také prezentována dalším terapeutům v oboru, kteří projevili zájem o její využití v rámci své praktické činnosti.

Vytvořený kód je krytý MIT licencí a mezi základní použité technologie patří TypeScript, Next.js (React.js + Node.js), ORM technologie Prisma a knihovna tRPC pro tvorbu webového API. Tyto nástroje byly pečlivě vybrány a představují moderní a populární způsob efektivního vývoje webových aplikací v rámci jednoho projektu s využitím jednoho programovacího jazyka. Společně navíc technologie nabízí vysokou typovou bezpečnost, kterou podporují v kombinaci s jazykem TypeScript. Typová bezpečnost je tak zajištěna i při dotazování a získávání dat z databáze, stejně jako při komunikaci mezi serverovou a klientskou částí aplikace. Pro vynucení využití dobrých praktik byl využit nástroj ESLint pro statickou kontrolu kódu.

Během implementace byl kladen důraz na bezpečnost aplikace. V rámci procesu POT (proof of technology) byla zkoumána technologie NextAuth a experimentováno s Next.js middlewarem za účelem zefektivnění procesu. Avšak tyto technologie nebyly nakonec využity kvůli jejich ranné fázi vývoje a nedostačujícím vlastnostem. Místo toho byl implementován vlastní proces autentizace.

Během vývoje bylo vyzkoušeno několik způsobů tvorby grafického rozhraní. Zpočátku byla využita knihovna Tailwind, avšak později byla upřednostněna grafická knihovna. Původně byla použita knihovna Chakra, ale nakonec byla zvolena knihovna Mantine, která nabízí širokou škálu komponent s kvalitní podporou konfigurace, stylování a zaměřením na webovou přístupnost. Pro další vývoj by bylo vhodné integrovat aplikaci s nástrojem jako Storybook, který umožňuje skrze poskytnuté rozhraním snadno a přehledně procházet a dokumentovat grafické komponenty aplikace.

Aplikace úspěšně využívá a integruje několik cloudových služeb, které poskytují vysokou dostupnost, nízké nároky na údržbu a vysokou škálovatelnost. Všechny použité služby jsou v rámci bezplatných limitů. Při vyšším zatížení aplikace však existuje riziko překročení těchto limitů a zpoplatnění služeb. Proto byla provedena analýza nejvhodnějších cloudových služeb a snaha vyhnout se těsné vazbě na konkrétní technologii, aby se předešlo proprietárnímu uzamčení (vendor lock-in).

Aplikace využívá DBaaS (Database as a Service) službu PlanetScale a pro ukládání zvukových souborů byla úspěšně využita cloudová služba AWS S3. Ta nabízí efektivní a bezpečné řešení skrze předpodepsané URL, které se generují na straně serveru na základě tajného klíče a umožňují omezený a dočasný přístup k datům a funkcím S3 serveru ze strany klienta.

Aplikace umožňuje terapeutům vytvářet zvukové terapie dle individuálních potřeb jednotlivých klientů, kteří je následně v aplikaci mohou poslouchat. Pro vytváření terapií je využit způsob „drag and drop“ a proces je optimalizován pro nízké vytížení serveru tím, že se požadavky shromažďují a odesílají současně. Aplikace umožňuje sledování používání aplikace uživateli a nastavení datumového rozmezí omezujícího přístup k terapii klientem, stejně jako maximální povolený počet přehrání za den, který aplikace uživateli dovolí.

Díky využití knihovny Next-i18next byla dosažena lokalizace serverového API i uživatelského rozhraní, která správně funguje i v procesu statické generace stránek (SSG) a serverového renderování obsahu (SSR). Ačkoliv jsou klíče jednotlivých textových lokalizací v aplikaci typované, generace těchto typů je komplexní, jelikož zahrnuje množství víceúrovňových JSON souborů. To způsobuje poměrně časově a výpočetně náročnou kontrolu integrity typů.

Aplikace také úspěšně implementuje proces odesílání emailů s využitím služby Sendgrid a jejich stylování prostřednictvím knihovny React Email. Aplikace tak umožňuje registraci uživatelů pomocí emailu v různých rolích, přihlášení uživatelů, včetně perzistentního přihlášení, změnu hesla a obnovu zapomenutého hesla prostřednictvím emailu. V aplikaci je efektivně vyřešena správa tabulkových dat s možností stránkování, vyhledávání, filtrování a řazení řešených na straně serveru, včetně využití vyrovnávací paměti pro větší rychlost a menší zatížení serveru. Aplikace klade důraz na intuitivitu, přístupnost a kvalitní uživatelský zážitek, což je dosaženo mimo jiné vědomým použitím barev, ikon, notifikací, modálních oken a načítacích koster, které potlačují kumulativní posuny uživatelského rozhraní (CLS).

Během realizace nasazení a hostování aplikace bylo experimentováno s několika službami. V rámci některých z nich bylo využito kontejnerů nástroje Docker. Nakonec byla zvolena služba Vercel, která nabízí optimalizované serverless prostředí pro Next.js. Aplikace zatím nemá zakoupenou vlastní doménu, ale využívá doménu třetího řádu zdarma. Nasazená produkční verze je monitorována pomocí MaaS (Monitoring as a Service) služeb Sentry a Axiom z důvodů bezpečnosti a pro analýzu výkonu, událostí a chyb, což umožňuje vylepšování a opravy aplikace.

Díky DevOps praktikám a zvoleným technologiím je kód přehledný, udržitelný a umožňuje snadno a efektivně vytvářet změny, integrovat je do

projektu a zpřístupnit uživatelům. Byla vytvořena CI/CD pipeline pomocí technologie GitLab CI a integrována s GitLab repozitářem. Pipeline využívá mezipaměť pro ukládání projektových závislostí a pomocí artefaktů předává vstupy jednotlivých kroků. V rámci pipeline je aplikace sestavena, otestována a nahrána do cílového prostředí. Během testování je provedena statická analýza pomocí nástrojů ESLint a Prettier, kontrola typové integrity a spuštěny automatizované testy.

Aplikace byla úspěšně otestována pomocí 127 automatizovaných testů, které dosáhly více než 80% pokrytí. Testy byly vytvořeny s využitím technologie Vitest a rozděleny na klientskou a serverovou část. Pro klientské testy byly navíc využity knihovny React Testing Library a Mocked Service Worker. Díky správné konfiguraci těchto knihoven bylo možné přerušovat požadavky na server během klientského testování a poskytovat mockovaná data jako odpovědi. To umožnilo efektivně testovat komponenty v různých situacích závislých na datech ze serveru. Kvalita aplikace byla navíc ověřena pomocí nástroje Google PageSpeed Insights.

V rámci uživatelského testování bylo vyplněno 16 formulářů, přičemž polovina byla zaměřena na spokojenost terapeutů a druhá polovina na klienty. Díky těmto formulářům byla získána zpětná vazba poukazující na aspekty pro další rozšíření a vylepšení. Některé vlastnosti byly do aplikace přidány, zatímco jiné aspekty vyžadují další analýzu.

S rostoucím využitím aplikace uživateli, zejména terapeuty, se očekává, že změnové požadavky a doporučení ze strany uživatelů budou přibývat. Před jejich implementací je nutné důkladně zhodnotit potřebu a případ užití, který funkci vyžaduje. Je důležité zavést takovéto standardní postupy pro udržení přehledné a intuitivní aplikace pro všechny uživatele, která efektivně splní cíle a vizi, se kterou byla vytvořena.

V rámci budoucích aktivit by bylo zajímavé zkoumat účinky poslechu terapeutických zvuků z aplikace prostřednictvím různých metod. Možností je například testování mozkové aktivity pomocí EEG nebo dotazování jednotlivců před a po pravidelném používání aplikace v určitém časovém období. Také by bylo vhodné přizpůsobit aplikaci tak, aby splňovala principy progresivní webové aplikace (PWA), což by umožnilo uživatelům stáhnout a používat aplikaci bez nutnosti připojení k internetu. Pro realizaci této funkce by bylo nutné déle vyřešit stahování a bezpečné ukládání zvukových souborů do uživatelských zařízení.

18 Závěr

V rámci diplomové práce vznikla webová aplikace, která umožňuje terapeutům vytvářet akustické podněty dle individuálních potřeb svých klientů. Uživatelé následně mohou tyto zvukové terapie v aplikaci poslouchat ze svého zařízení. Aplikace byla vyvinuta ve spolupráci s odborným terapeutem, což zajistilo její kvalitu a relevanci v oblasti frekvenční terapie.

Před zahájením vývoje byla provedena studie a konzultace problematiky terapeutické stimulace akustickými podněty, a to zejména využitím terapeutických ladiček. Také byla provedena analýza webového prostředí a dále technologií, nástrojů a praktik, které umožňují efektivní proces tvorby kvalitní a bezpečné webové aplikace od jejího návrhu až po nasazení a údržbu v produkčním prostředí. Následně byla analyzována nalezená existujících řešení.

Díky získaným poznatkům byla učiněna rozhodnutí výběru technologií a nástrojů pro vývoj. Dále byly navrženy funkce a uživatelské rozhraní aplikace. Následně byla realizována implementace, během které byla aplikace nadále iterativně prezentována a validována terapeutem. Mezi základní využité technologie patří TypeScript a Next.js. Pro ukládání a přístup ke zvukovým souborům byla využita služba AWS S3.

Aplikace byla nasazena v serverless prostředí využitím služby Vercel. Pro automatizaci procesu nasazení byla vytvořena CI/CD pipeline s fázemi sestavení, testování a přenesení do cílového prostředí. Fáze testování zahrnuje automatizované testy a statickou analýzu kódu včetně kontroly typové integrity. Aplikace byla navíc integrována se službami umožňující monitorování výkonu, událostí a vzniklých chyb v produkčním prostředí.

V rámci testování byly vytvořeny automatizované testy pokrývající velkou část aplikačního kódu. Dále bylo provedeno uživatelské testování. Dotazované subjekty skrze vytvořené formuláře vyjádřily převážnou spokojenost s vlastnostmi aplikace a pomohly identifikovat oblasti pro zlepšení.

Osobně hodnotím výsledky diplomové práce pozitivně. Byly pokryty jednotlivé body zadání a požadavky terapeuta, který na návrhu spolupracoval. Je tak s aplikací spokojený a postupně ji integruje do své praktické činnosti. Aplikace také zaujala řadu dalších terapeutů, což nasvědčuje jejímu potenciálu pro širší použití. Díky dosaženým výsledkům věřím, že aplikace bude i v budoucnu aktivně využívána a poskytne tak hodnotný přínos v oblasti muzikoterapie a frekvenční terapie.

Literatura

[1] Abedi Koupaei, M. et al. Sound therapy: an experimental study with autistic children. *Procedia-Social and Behavioral Sciences*. 2013, 84, s. 626–630.

[2] *Understanding Object-Relational Mapping: Pros, Cons, and Types* [online]. AltexSoft, 2021. [cit. 2023/02/28]. Dostupné z: `https://www.altexsoft.com/blog/object-relational-mapping/`.

[3] *What is cloud computing?* [online]. Amazon, 2023. [cit. 2023/02/25]. Dostupné z: `https://aws.amazon.com/what-is-cloud-computing/`.

[4] Antonín Pavlíček, J. S. *Základy moderní informatiky*. Professional publishing, 2022. ISBN 978-80-88260-59-2.

[5] *Learn Spring Boot* [online]. Baeldung, 2022. [cit. 2023/02/04]. Dostupné z: `https://www.baeldung.com/spring-boot`.

[6] Browne, T. *T3 Stack* [online]. 2023. [cit. 2023/01/30]. Dostupné z: `https://create.t3.gg/`.

[7] *What is edge computing?* [online]. Cloudfare, 2023. [cit. 2023/02/24]. Dostupné z: `https://www.cloudflare.com/learning/serverless/glossary/what-is-edge-computing/`.

[8] Coupland, M. *DevOps Adoption Strategies: Principles, Processes, Tools, and Trends*. Packt Publishing Ltd, 2021. ISBN 978-1-80107-632-6.

[9] *Django documentation* [online]. Django, 2023. [cit. 2023/02/05]. Dostupné z: `https://docs.djangoproject.com/en/4.1/`.

[10] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures* [online]. 2000. [cit. 2023/02/22]. Dostupné z: `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

[11] Gerlichová, M. *Muzikoterapie v praxi, Příběhy muzikoterapeutických cest - 2. přepracované a doplněné vydání*. Grada, 2021. ISBN 978-80-271-1791-8.

[12] Gorham, L. *JWTs vs. sessions: which authentication approach is right for you?* [online]. Stytch, 2023. [cit. 2023/03/05]. Dostupné z: `https://stytch.com/blog/jwts-vs-sessions-which-is-right-for-you/`.

[13] Hobson, J. – Chisholm, E. – El Refaie, A. Sound therapy (masking) in the management of tinnitus in adults. *Cochrane Database of Systematic Reviews*. 2012.

[14] Hoffman, A. *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. O'Reilly Media, 2020. ISBN 978-1492053118.

[15] *What is cloud computing?* [online]. IBM, 2023. [cit. 2023/02/27]. Dostupné z: `https://www.ibm.com/topics/cloud-computing`.

[16] *What is container orchestration?* [online]. IBM, 2023. [cit. 2023/03/25]. Dostupné z: `https://www.ibm.com/topics/container-orchestration`.

[17] *What Are CI/CD and the CI/CD Pipeline?* [online]. IBM Cloud Education, 2021. [cit. 2023/03/25]. Dostupné z: `https://www.ibm.com/cloud/blog/ci-cd-pipeline`.

[18] Kernighan, B. W. *Jak porozumět digitálnímu světu*. Princeton University Press, 2017. ISBN 978-80-7363-903-7.

[19] *Laravel Docs* [online]. Laravel, 2023. [cit. 2023/02/02]. Dostupné z: `https://laravel.com/docs/10.x`.

[20] McDonald, M. *Web Security for Developers: Real Threats, Practical Defense*. No Starch Press, 2020. ISBN 978-1593279943.

[21] Merced, A. *In-Depth Guide on Understanding Deploying Web Apps* [online]. DEV Community, 2021. [cit. 2023/03/20]. Dostupné z: `https://dev.to/alexmercedcoder/in-depth-guide-on-understanding-deploying-web-apps-56ap`.

[22] *ASP.NET documentation* [online]. Microsoft, 2023. [cit. 2023/02/04]. Dostupné z: `https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0`.

[23] *TypeScript Docs* [online]. Microsoft, 2023. [cit. 2023/01/31]. Dostupné z: `https://www.typescriptlang.org/docs/`.

[24] *MDN Web Docs* [online]. Mozilla, 2023. [cit. 2023/01/22]. Dostupné z: `https://developer.mozilla.org/`.

[25] *About Node.js* [online]. OpenJS Foundation, 2023. [cit. 2023/02/10]. Dostupné z: `https://nodejs.org/en/about/`.

[26] *PHP Manual* [online]. PHP Group, 2023. [cit. 2023/02/07]. Dostupné z: `https://www.php.net/manual/en/index.php#index`.

[27] Reichl, J. – Všetička, M. *Encyklopedie fyziky - mechanické kmitání a vlnění - zvukové vlnění* [online]. Multimedialní encyklopedie fyziky, 2006. [cit. 2023/02/01]. Dostupné z: http://fyzika.jreichl.com/main.article/view/152-zvukove-vlneni.

[28] Romanowska, B. *Muzikoterapie, Ladičky a léčení zvukem.* Alpress, 2005. ISBN 80-7362-067-7.

[29] *Getting Started with Rails* [online]. Rubyonrails, 2023. [cit. 2023/02/07]. Dostupné z: https://guides.rubyonrails.org/getting_started.html.

[30] Salamon, E. et al. Sound therapy induced relaxation: down regulating stress processes and pathologies. *Medical Science Monitor.* 2003, 9, 5, s. RA96–RA101.

[31] Sereda, M. et al. Sound therapy (using amplification devices and/or sound generators) for tinnitus. *Cochrane Database of Systematic Reviews.* 2018.

[32] Team, E. Y. *Best Practices for Application Deployment* [online]. Engine Yard, 2022. [cit. 2023/03/30]. Dostupné z: https://www.engineyard.com/blog/best-practices-for-application-deployment/.

[33] *A Guide to the Most Popular APIs: REST, SOAP, GraphQL, gRPC, and WebSockets* [online]. The Postman Team, 2022. [cit. 2023/02/20]. Dostupné z: https://blog.postman.com/guide-to-apis-rest-soap-graphql-grpc-websockets/.

[34] Thomas, M. T. *React in action.* Manning Publications Co., 2018. ISBN 978-1-61729-385-6.

[35] Vašina, M. *Environmentálná fyzika* [online]. Univerzita Tomáše Bati ve Zlíně, Fakulta technologická, 2015. [cit. 2023/01/20]. Ústav fyziky a materiálového inženýrství. Dostupné z: http://ufmi.ft.utb.cz/index.php?page=env_fyzika.

[36] *Next JS Docs* [online]. Vercel, 2023. [cit. 2023/02/02]. Dostupné z: https://nextjs.org/docs/.

[37] Wang, H. et al. The state of the art of sound therapy for subjective tinnitus in adults. *Therapeutic Advances in Chronic Disease.* 2020, 11.

[38] William. *Web Application Architecture* [online]. ClickIT, 2022. [cit. 2023/03/27]. Dostupné z: https://www.clickittech.com/devops/web-application-architecture/.

[39] *Making the Web Accessible* [online]. World Wide Web Consortium, 2023. [cit. 2023/03/10]. Dostupné z: https://www.w3.org/WAI/.

[40] Wébr, D. Hudební přehrávač s tvorbou psychoaktivní složky. bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2021. Dostupné z: `https://dspace5.zcu.cz/handle/11025/44235`.

[41] Zuzana, D. *Liečenie zvukom blahodárné účinky terapeutických ladiček.* Vlastní vydání, 2018. ISBN 978-80-973060-7-6.

[42] Řezáč, J. *Web ostrý jako břitva, Návrh fungujícího webu pro webdesignery a zadavatele projektů.* Baroque partners, 2014. ISBN 978-80-87923-01-6.

Seznam zkratek

ABAC	Attribute-Based Access Control
ADHD	Attention Deficit Hyperactivity Disorder
API	Application Programming Interface
ASP.NET	Active Server Pages Network Enabled Technologies
AVS	Audio-visual stimulation
AWS	Amazon Web Services
CaaS	Communication as a Service
CD	Continuous Deployment
CI	Continuous Integration
CERN	European Organization for Nuclear Research
CLS	Cumulative Layout Shift
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
Db	Decibel
DBaaS	Database as a Service
DDoS	Distributed Denial of Service
DOM	Document Object Model
DoS	Denial of Service
DRY	Don't Repeat Yourself
EEG	Electroencephalography
FaaS	Function as a Service
FLAC	Free Lossless Audio Codec
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HTML	Hypertext Markup Language
Hz	Hertz
HW	Hardware
IaaS	Infrastructure as a Service
ID	Identification
IoT	Internet of Things
IT	Information Technology

JPA	Java Persistence API
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JWT	JSON Web Token
LED	Light Emitting Diode
MaaS	Monitoring as a Service
MVC	Model-View-Controller
MSW	Mocked Service Worker
MVT	Model-View-Template
NPM	Node Package Manager
ORM	Object-Relational Mapping
PaaS	Platform as a Service
PHP	PHP: Hypertext Preprocessor
POT	Proof of technology
PWA	Progressive Web Apps
RBAC	Role-Based Access Control
REST	Representational State Transfer
RPC	Remote Procedure Call
S3	Simple Storage Service
SaaS	Software as a Service
SEO	Search Engine Optimization
SECaaS	Security as a Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
SW	Software
TLS	Transport Layer Security
TCP	Transmission Control Protocol
UI	User Interface
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
UX	User Experience
WAV	Waveform Audio File Format
XaaS	Anything as a Service
XML	Extensible Markup Language
XSS	Cross-Site Scripting

Seznam obrázků

Seznam výpisů kódu

Seznam tabulek

Přílohy

A Uživatelská příručka

Uživatelská příručka je rozdělena na obecné návody a následně návody určené pro konkrétní role.

Obecné

Registrace

Registrace do aplikace probíhá skrze odkaz obdržený v emailové pozvánce. Klienty zve do aplikace odpovídající terapeut, kterého do aplikace pozval administrátor. V emailové pozvánce se nachází odkaz na registrační formulář. V tom vyplňte potřebné informace a také je nutné souhlasit s podmínkami užití. Po registraci se skrze tlačítko můžete dostat na obrazovku přihlášení, kde se můžete přihlásit vaším emailem a nastaveným heslem.

Přihlášení

Tlačítko pro přechod na stránku přihlášení se nachází na hlavní obrazovce. Pro přihlášení vyplňte váš email a heslo. Po úspěšném přihlášení budete přesměrováni na stránku odpovídající vaší roli.

Změna hesla

Pro změnu hesla využijte modálního okna, které vyvoláte kliknutím na tlačítko „Změna hesla“ nacházející se v hlavním kontextovém menu. To naleznete po přihlášení kliknutím na tlačítko s vaším jménem v pravém horním rohu aplikace.

Zapomenuté heslo

V případě zapomenutého hesla klikněte na obrazovce přihlášení na tlačítko „Zapomenuté heslo“. Na následující obrazovce vyplňte svůj email a potvrďte. Na váš email přijde vyzvání k změně hesla obsahující tlačítko odkazující na potřebný formulář. V tom vyplňte nové heslo a potvrďte. Následně se do aplikace můžete přihlásit nově nastaveným heslem.

Změna jazyka

Aplikace umožňuje změnu jazyka rozhraní. V pravém horním rohu se nachází vlaječka signalizující aktuální jazyk. Pomocí kliknutí na toto tlačítko můžete jazyk upravit. Aktuálně je podporována angličtina a čeština.

Administrátor

Jako administrátor máte přístupné skrze uživatelské menu tři hlavní obrazovky: uživatelé, frekvence a šablony.

Správa uživatelů

Správa uživatelů je možná na obrazovce „Uživatelé“, kde je seznam všech uživatelů, ve kterém je možné vyhledávat, filtrovat a řadit.

Přidání terapeuta

Přidání terapeuta probíhá skrze tlačítko „Přidat terapeuta“ umístěné v pravém horním rohu na obrazovce uživatelů. Po kliknutí vyplňte email terapeuta. Následně mu přijde pozvánka pro dokončení registrace. Po jejím úspěšném vyplnění terapeutem je proces registrace hotový.

Detail uživatele

Detail uživatele je možné zobrazit kliknutím na řádek uživatele.

Smazání uživatele

Uživatele je možné smazat z obrazovky detailu uživatele nebo přímo v seznamu uživatelů využitím kontextového menu. Smazáním terapeuta se smažou i všichni svázaní klienti.

Deaktivace/aktivace uživatele

Uživatele je možné deaktivovat/aktivovat z obrazovky detailu uživatele nebo přímo v seznamu uživatelů využitím kontextového menu. Deaktivace terapeuta zahrnuje zabránění přístupu do aplikace i pro všechny jeho klienty.

Správa frekvencí

Správa frekvencí je možná na obrazovce „Frekvence“, kde je seznam všech frekvencí, ve kterém je možné vyhledávat, filtrovat, řadit a jednotlivé frekvence si přehrávat.

Nahrání frekvencí

Nahrání frekvencí probíhá skrze modální okno, které se zobrazí po kliknutí na tlačítko „Nahrát frekvence“ v pravé horní části obrazovky „Frekvence“. V modálním oknu lze zvolit více souborů. Je nutné určit unikátní název a balík frekvence.

Smazání frekvencí

Frekvence je možné smazat kliknutím na tlačítko odpadkového koše na odpovídající řádce.

Přehrání frekvence

Frekvence je možné přehrát kliknutím na tlačítko trojúhelníku na odpovídající řádce. Frekvenci lze smazat pouze v případě, že se nenachází na žádném playlistu.

Úprava frekvence

Frekvenci je možné upravit název a balík skrze modální okno zobrazené po kliknutí na tlačítko „Upravit“ na odpovídající řádce.

Správa šablon

Správa šablon je možná na obrazovce „Šablony“, kde je seznam všech šablon, ve kterém je možné vyhledávat, filtrovat a řadit.

Detail šablony

Detail šablony je možné zobrazit kliknutím na řádek šablony.

Smazání šablony

Šablonu je možné smazat z obrazovky detailu šablony nebo přímo v seznamu šablon využitím kontextového menu.

Terapeut

Jako terapeut máte přístupné skrze uživatelské menu tři hlavní obrazovky: pacienti, frekvence a šablony.

Správa pacientů

Správa Vašich pacientů je možná na obrazovce „Pacienti“, kde je seznam všech Vašich pacientů, ve kterém je možné vyhledávat, filtrovat a řadit.

Přidání pacienta

Přidání pacienta probíhá skrze tlačítko „Přidat pacienta“ umístěné v pravém horním rohu na obrazovce pacientů. Po kliknutí vyplňte email pacienta. Následně mu přijde pozvánka pro dokončení registrace. Po jejím úspěšném vyplnění pacientem je proces registrace hotový.

Detail pacienta

Detail pacienta je možné zobrazit kliknutím na řádek pacienta. Na této stránce lze upravovat terapii a sledovat využívání aplikace pacientem.

Smazání pacienta

Pacienta je možné smazat z obrazovky detailu pacienta nebo přímo v seznamu pacientů využitím kontextového menu.

Deaktivace/aktivace pacienta

Pacienta je možné deaktivovat/aktivovat z obrazovky detailu pacienta nebo přímo v seznamu pacientů využitím kontextového menu.

Seznam frekvencí

Seznam dostupných frekvencí je dostupný na obrazovce „Frekvence“, kde je seznam všech frekvencí, ve kterém je možné vyhledávat, filtrovat, řadit a jednotlivé frekvence si přehrávat.

Správa šablon

Správa šablon je možná na obrazovce „Šablony“, kde je seznam všech Vašich šablon, ve kterém je možné vyhledávat, filtrovat a řadit.

Vytvoření šablony

Vytvoření šablony probíhá skrze modální okno, které se zobrazí po kliknutí na tlačítko „Vytvořit šablonu“ v pravé horní části obrazovky „Šablony“. Následně je nutné určit název šablony. Po jejím vytvoření budete přesměrováni na obrazovku detailu šablony.

Vytvoření šablony z terapie

Vytvořit šablonu lze i z vytvořené terapie. Šablona tak bude obsahovat stejné frekvence a jejich pořadí. Toho lze dosáhnout na obrazovce detailu pacienta kliknutím na kontextové menu vedle názvu terapie.

Smazání šablony

Šablonu je možné smazat z obrazovky detailu šablony nebo přímo v seznamu šablon využitím kontextového menu.

Použití šablony

Šablonu je možné použít v terapii kliknutím na kontextové menu vedle názvu terapie. Do jedné terapie lze vložit více šablon. Tyto frekvence se vkládají postupně za sebou.

Správa terapie

Správa terapie pacientů probíhá na obrazovce detailu pacienta. Lze nastavit jméno terapie, počet přehrání za den a datumové rozmezí, po které je terapie platná. Pokud platnost není nastavena nebo aktuální datum nespadá do tohoto rozmezí, pacient terapii nemůže využívat. Vytváření terapie probíhá skrze vkládání frekvencí tlačítkem s ikonou plus ze seznamu v pravé části obrazovky. Lze vyhledávat podle názvu frekvence a podle názvu balíku. Frekvence v terapii lze vidět v komponentě vlevo. Přehrávají se postupně odshora dolu a lze určit, kolikrát se každá frekvence přehraje. Je také vidět celkový čas terapie.

Klient

Jako klient máte po přihlášení dostupnou pouze jednu obrazovku umožňující přehrání terapie.

Přehrání terapie

Přehrání terapie probíhá tlačítkem uprostřed obrazovky z domovské obrazovky, na kterou budete přesměrováni po přihlášení do aplikace. Abyste si terapii mohli přehrát, musí být platná (zařizuje terapeut) a musíte mít dostupná přehrání pro dnešní den. Ty určuje terapeut a po jejich vyčerpání se obnoví následující den. Před přehráním máte možnost vyzkoušet hlasitost zvuku. Přehrání nelze přerušit, pouze úplně zastavit. Zastavené přehrání se započítá jako platné přehrání pro daný den.

B Popis adresářové struktury přiloženého souboru

- adresář **Aplikace_a_knihovny** – zdrojové kódy vytvořené aplikace (struktura adresáře viz kapitola 14.3),
- adresář **Text_prace**
  - adresář **src** – zdrojové kódy dokumentu ve formátu LaTeX,
  - **A21N0078P_text-prace.pdf** – dokument diplomové práce ve formátu PDF,
- adresář **Vysledky**
  - **dotazniky_klient.pdf** – vyplněné dotazníky v rámci uživatelského testování zaměřeného na klienty,
  - **dotazniky_terapeut.pdf** – vyplněné dotazníky v rámci uživatelského testování zaměřeného na terapeuty,
- adresář **Poster**
  - **A21N0078P_poster.pdf** – poster diplomové práce ve formátu PDF,
  - **A21N0078P_poster.pub** – poster diplomové práce ve formátu Microsoft Publisher,
- **Readme.txt** – popis adresářové struktury přiloženého souboru.