FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Specializovaná webová aplikace pro výuku hry na ukulele
Student:	Dan Balarin
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Práce má za cíl vytvořit webovou aplikaci výuku hry na ukulele. Mezi hlavní požadované funkce patří výuka akordů, strumming patterns, progression (přechytávání akordů), metronom a správa databáze písniček s akordy včetně strumming patterns. Při vývoji postupujte v souladu s metodami softwarového inženýrství.

- Analyzujte klíčové pojmy a postupy pro výuku hry na ukulele z dostupných zdrojů.
- Proveďte stručnou rešerši existujících aplikací obdobného typu.
- Navrhněte vlastní webovou aplikaci s důrazem na udržitelnost a možnosti budoucího rozvoje. Při návrhu rozdělte aplikaci na backend a frontend.
- Implementujte webovou aplikaci dle návrhu, řádně ji otestujte a zdokumentujte. Volbu případných dalších technologií zdůvodněte.
- Zhodnoťte přínosy aplikace, porovnejte s konkurenčními řešeními a navrhněte další možnosti rozvoje do budoucna.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. listopadu 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Specializovaná webová aplikace pro výuku hry na ukulele

Dan Balarin

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

30. května 2020

Poděkování

Děkuji vedoucímu práce, Ing. Marku Suchánkovi, za odborné vedení a ochotu při tvorbě práce. Dále chci poděkovat přítelkyni a rodičům za podporu a schovívavost v průběhu studia.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. května 2020 ……

Odkaz na tuto práci

Balarin, Dan. *Specializovaná webová aplikace pro výuku hry na ukulele.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW: ⟨`https://github.com/danbalarin/bachelors-thesis`⟩.

Abstrakt

Bakalářská práce se zabývá vývojem webové aplikace podporující výuku hraní na ukulele. Aplikace je vyvíjena v souladu s metodami softwarového inženýrství. Práce zkoumá existující řešení, dostupné technologie a cílovou skupinu. Na základě těchto poznatků je vytvořena funkční specifikace a jsou vybrané vhodné technologie. Za pomoci těchto technologií je následně provedena implementace, testování a automatické nasazování. Výsledkem je funkční aplikace umožňující výuku hraní na ukulele, která je díky zvolené architektuře snadno rozšiřitelná.

Klíčová slova ukulele, metronom, akordy, interaktivní výuka, webová aplikace, React, GraphQL, kontinuální integrace, automatické nasazování

Abstract

The Bachelor thesis describes the development of web application for learning how to play on the ukulele. The application is developed according to the methods of software engineering. Thesis researches existing solutions, available technologies, and target audience. Based on this knowledge, a functional specification is created and suitable technologies are selected. With the help of these technologies, the application is implemented, tested, and automatically deployed. The result is a functional application for learning how to play on the ukulele, which is thanks to the good system architecture easily expandable.

Keywords ukulele, metronome, chords, interactive lessons, web application, React, GraphQL, continuous integration, automatic deployment

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Motivace

V posledních letech roste popularita hry na hudební nástroje a s rozvojem informačních technologií s tím souvisí i snaha uživatelů učit se sami za pomoci veřejně dostupných aplikací. Pro začátečníky se doporučují nástroje buď drnkací (ukulele, kytara) nebo dechové (harmonika, flétna) [1, 2].

Ukulele oproti kytaře má tu výhodu, že je menší, má o dvě struny méně, struny tolik neřežou do prstů a je tedy jednodušší pro začátečníka. Oproti nástrojům dechovým má hlavně tu výhodu, že se na něj dá zahrát více populárních písní.

Aktuálně existuje již několik aplikací, ať už webových nebo mobilních, které umožňují uživateli učit se hrát na ukulele sám doma. Nevýhody těchto aplikací jsou, že jsou buď placené, neobsahují všechny funkcionality, nebo nejsou uživatelsky přívětivé. Rozborem již existujících řešení se podrobněji zabývá kapitola Analýza.

K této problematice jsem se dostal, jelikož jsem se sám začal učit hrát na ukulele. Existující aplikace mi nevyhovují, a to převážně kvůli chybějícím funkcím nebo neintuitivnímu ovládání. Z tohoto důvodu mi přišlo ideální vytvořit vlastní aplikaci, která bude obsahovat veškeré funkce chybějící u existujících aplikací, bude snadno ovladatelná a bude fungovat na počítači i chytrém telefonu.

Cíle práce

Hlavním cílem práce je navrhnout, vytvořit a otestovat aplikaci, která umožní začátečníkovi naučit se hrát na ukulele, čehož lze dosáhnout několika způsoby, kterými se zabývá v kapitola Analýza. Dílčím cílem je umožnit již zkušenému hráči vyhledat jednotlivé akordy nebo akordy a text k písni.

Důležitou součástí aplikace bude jednoduchost použití, uživatelská přívětivost a zároveň možnost použití některých složitějších mechanismů, jako je například výuka přechytávání akordů s možností zapnutí metronomu, nebo vyhledávání v několika různých množinách (např. akordy, písně a autoři).

Členění práce

Práce je členěna do kapitol odpovídajících krokům vývoje software podle principů metodologie unifikovaného vývoje [3, s. 51–68].

První kapitola stanovuje a ujasňuje základní pojmy, které se týkají teorie hudby a hraní na ukulele.

Kapitola Analýza se zabývá průzkumem existujících aplikací, zjištění cílové skupiny, dostupných technologií a platforem a možnostmi, jak k danému problému přistoupit.

Třetí kapitola práce, se věnuje návrhu aplikace. Návrh aplikace je nedílnou součástí vývoje složitějších systémů a aplikací. Umožní programátorovi, nebo týmu programátorů, mít lepší představu o celém projektu, zjistit nedostatky ještě před implementací, nebo například stanovit místa v aplikaci kde může v budoucnu dojít k rozšíření a aplikaci na to řádně připravit. Tato kapitola obsahuje formální stanovení požadavků na aplikaci a vytvoření náhledů uživatelského rozhraní.

Kapitola Realizace rozebírá prostředí v jakém probíhal vývoj aplikace, konečný výběr technologií, určení logického členění a architektury aplikace a některé aspekty zdrojového kódu samotné aplikace. Správně zvolená architektura aplikace usnadní vývoj i případné rozšiřování, a může umožnit přepoužití části aplikace při práci na aplikaci jiné.

Pátá kapitola se věnuje testováním aplikace. Testování je naprosto nezbytné pro vývoj kvalitní aplikace. Umožňuje odhalit nedostatky, díky kterým může docházet k úniku citlivých informací a tím poškodit uživatele nebo provozovatele. Proto se každý systém a aplikace testuje a to na několika úrovních.

Poslední kapitola probírá automatické testování a nasazování. Automatizace testů a nasazování zrychluje vývoj a celkovou workflow týmu. Tato automatizace deleguje spouštění testů a buildů aplikace na dedikovaný server, takže se vývojář může zatím věnovat něčemu jinému.

Kapitola 1

Zavedení základních pojmů

Dříve, než se práce začne zabývat problematikou více do hloubky, je třeba si stanovit některé základní pojmy z teorie hudby a stručný popis ukulele. Po více informací bych doporučil odbornou literaturu, např. *Hudební slovník pro každého*[4].

1.1 Základ teorie hudby

Základ hudby je tón, zvuk s periodickým kmitáním o určité frekvenci, příkladem může být tón *C*. Následně hudební stupnice určuje počet tónů a půltónů a jejich frekvenční vzdálenost. Práce obsahuje pouze stupnice durové a mollové a to v souvislosti s akordy, příkladem může být stupnice *C dur*. Akord je souzvuk několika tónů, u ukulele to je souzvuk čtyř tónů, jeden pro každou strunu. Označení akordů vychází z tónů a vzdálenostmi mezi nimi, které se většinou řídí nějakou stupnicí, např. akord *Emi*, neboli *e moll* se řídí mollovou stupnicí.[4]

1.2 Popis ukulele

Ukulele je čtyřstrunný drnkací hudební nástroj, vzhledem podobný malé kytaře. Ukulele původně pochází z Havajských ostrovů kde vzniklo z nástroje zvaného Braguinha a o původu jeho jména se vedou spekulace [5].

Skládá se z dvou hlavních částí - těla a krku. Krk se následně skládá z hmatníku, který tvoří pražce a hlavice s ladící mechanikou. Struny jsou nataženy od kobylky až k ladící mechanice a zpravidla mají různé ladění. Běžné ladění udává ladění jednotlivých strun od shora dolů.

Obrázek 1.1: Příklad koncertního ukulele [6]

1.2.1 Hra na ukulele

Samotné hraní je velmi podobné hře na kytaru – ukazováček, prostředníček, prsteníček a malíček levé ruky přitiskávají struny mezi pražci, čímž určují akord a prsty pravé ruky přejíždějí po strunách někde na pomezí krku a těla ukulele. Způsobů, jak držet akordy, je několik a většinou záleží na dalším, resp. předchozím akordu, aby si hráč zjednodušil přechod. Stejně tak je i více způsobů jak přejíždět prsty po strunách. Prsty mohou jet dolů, nebo nahoru a to bříškem nebo nehtem prstu, či lehce klepnout do těla a tím zároveň utlumit struny[5]. Různým kombinacím těchto pohybů se říká strumming pattern.

1.2.2 Typy ukulele

Ukulele má několik variant, které se odlišují velikostí nástroje a barvou zvuku. Nejběžnější a zároveň doporučované pro začátečníky je ukulele sopránové. Sopránové ukulele je nejmenší a jeho běžné ladění je g,c,e,a nebo $a,d,f^{\#},h$.

Další ukulele jsou koncertní a tenorové, která jsou větší, ale ladí se stejně jako ukulele sopránové. Rozdíl je tedy v barvě zvuku a velikosti, tedy pohodlnosti držení.

Největší ukulele je barytonové, které je běžně laděné stejně jako vrchní čtyři struny kytary, tedy d,g,h,e, a je vhodné pro hráče kteří již umí hrát na kytaru, právě kvůli stejnému ladění.

Další, méně obvyklé, verze ukulele jsou šestistrunné, osmistrunné, desetistrunné a uke-bendžo (bendžolele), které vzniklo spojením ukulele a bendža.

Kapitola 2

Analýza

Proces vývoje aplikace začíná u analýzy. Tato kapitola zkoumá existující aplikace, dostupné technologie a cílovou skupinu.

2.1 Existující aplikace

Mezi alternativy k této práci lze řadit aplikace zabývající se výukou hry na ukulele nebo kytaru, zpěvníky a metronomy.

2.1.1 Výuka hry na ukulele

Většina materiálů pro výuku hry na ukulele je ve formě knih, nebo kurzů. Kurzy jsou buď osobní nebo online. Mezi největší a nejznámější online aplikace poskytující různé kurzy patří Udemy, který má i kurz hry na ukulele pro začátečníky [7]. Alternativou ke knihám a kurzům jsou ještě články nebo krátká nenavazující videa.

Články lze najít například na webu *kytary.cz* nebo *ukuguides.com*, které jsou zpracované formou tipů čemu věnovat pozornost a čemu se vyvarovat. Krátká videa zabývající se tipy pro hraní, nebo samotnými skladbami lze najít převážně na portálu *youtube.com*. Mezi tvůrce s nejvíce shlédnutými videi měsíčně patří např. Elise Ecklund[1] nebo John Atkins[2] vystupující pod přezdívkou The Ukulele Teacher. Oba autoři mají videa na veškerá témata hry na ukulele, od jeho výběru až po samotné písně s akordy a jak je hrát. John Atkins je navíc jedním z tvůrců aplikace The Ukulele App, která slouží hlavně pro výuku hry, ale neobsahuje žádné písně s akordy a většina funkcionalit je zpoplatněna.

[1] https://www.youtube.com/channel/UCUsCq3Hq5haa6tTph41hpJQ
[2] https://www.youtube.com/user/TheUkuleleTeacher

2.1.2 Zpěvníky

Zpěvníky jsou ve většině případů tištěné knihy, pdf soubory k tisku nebo v podobě webové či mobilní aplikace. Knižních zpěvníků ukulele je méně než kytarových a ukulele zpěvníků s českými písněmi je minimum. Soubory k tisku se dají sehnat i s českými písněmi, což částečně kompenzuje jejich absenci v podobě knižní.

Mezi významné webové aplikace se řadí *ukulele-tabs.com* nebo *ukutabs.com*. Mají možnost řazení a vyhledávání ve velkém repertoáru písní. Mezi jejich úskalí patří však absence responzivity (Ukutabs) a absence metronomu, či případné doporučení strumming patternu.

Nejstahovanější aplikace na mobilní telefony jsou např. Ukulele Tabs & Chords, nebo Ukulele Chords Pocket pro Android a The Ukulele App pro Android a iOS.

2.1.3 Metronomy

Metronomy existují ve formě fyzického zařízení, nebo aplikace, ať už webové či mobilní. Základní metronom poskytuje i webový vyhledávač *google.com*, který ovšem umožňuje pouze nastavení tempa.

Mezi jeden z nejlepších patří metronom webu *musicca.com* [8], který umožňuje nastavení tempa a dob, a metronom na webu *douglasniedt.com* [9], který sice umožňuje pouze $\frac{4}{4}$ takt, ale zase zobrazuje doby v podobě větších dlaždic, které je možné vnímat i periferním viděním, takže uživatel nemusí vnímat tempo jen pomocí zvuků. Mobilní aplikace zastupuje Metronome Beats a Tuner & Metronome pro Android a The Metronome by Soundbrenner pro iOS a Android.

2.2 Dostupné technologie

Aplikace by měla být co nejdostupnější pro běžného uživatele, ideálně by tedy neměla vyžadovat stahování nebo instalaci. Z podstaty této aplikace to ani není potřeba. Jediná výhoda kterou přináší desktopová aplikace je přímý přístup k hardware a lepší výkon, jelikož ale tento typ aplikace nepotřebuje vykreslovat pokročilou 3D grafiku nebo spouštět složité algoritmy, tak není nutné se tím zabývat.

Nabízí se tedy webová aplikace nebo mobilní aplikace. Oba typy aplikací se dělí na dvě části, část kterou vidí a se kterou interaguje uživatel nebo správce, která se nazývá frontend a část která obsluhuje práci s databází, zpracovává požadavky na změny, autorizuje uživatele atd, která se nazývá backend. Jeden backend přitom může obsluhovat frontend jak v podobě webové, tak i mobilní aplikace. Sekce se tedy dělí na Backend, Web a Mobil.

2.2.1 Backend

Backend tvoří páteř aplikací, zprostředkovává přístup k informacím, umožňuje úpravy a autorizuje uživatele. Z toho důvodu je třeba dbát na kvalitu a bezpečnost kódu. Špatně autorizovaný uživatel, nebo nezabezpečená část databáze je velká bezpečnostní trhlina, které se musí předejít. Nedá se spoléhat na to, že uživatele ověřil frontend, jelikož se s daty na frontendu dá manipulovat (jak na webu, tak i v mobilu), a proto veškeré ověřování probíhá na serveru. Ověřování na frontendu tedy slouží jako čistě estetická záležitost.

Dalším důležitým faktorem je rychlost. Rychlost může ovlivnit několik faktorů, jako je třeba využití složitých algoritmů bez paralelizace, pomalý databázový server, nebo neoptimalizované databázové požadavky.

Při výběru je třeba tedy dbát hlavně na bezpečnost a rychlost. Avšak je třeba zvážit i tzv. code reuse, tedy částečné přepoužití kódu, které muže zjednodušit a zpřehlednit kód aplikace.

JavaScript

JavaScript je *„otevřený multiplatformní skriptovací jazyk pro tvorbu a přizpůsobení aplikací v podnikových sítích a na internetu“* [10], jako první implementovaný prohlížečem Netscape Navigator 2.0 a rychle adaptován ostatními prohlížeči.

Výhodou je multiplatformnost a jednoduchost vývoje a nasazení. Hlavní nevýhodou je dynamické typování, které vede k horší udržitelnosti kódu a odhalení většiny chyb až při běhu aplikace (absence statické kontroly). Tyto neduhy se dají odstranit použitím nějaké syntaktické nadstavby, jako je třeba TypeScript nebo Flow, které přidávají statické typování proměnných a statickou kontrolu při překladu do JavaScriptu.

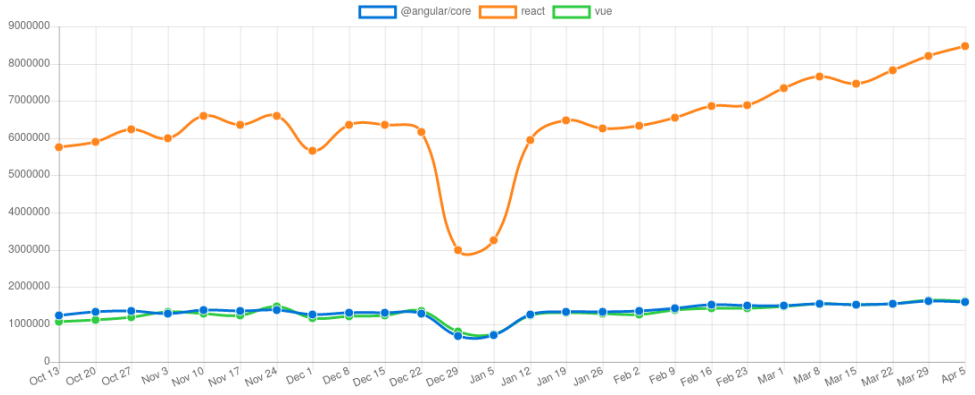
JavaScript se původně vyskytoval jen v prohlížeči, což znamená že nebyl použitelný pro vývoj desktopových a serverových aplikací. To se změnilo s příchodem Node.js. Node.js je prostředí, které umožňuje spouštět javascriptové skripty mimo prohlížeč.

Java

Staticky typovaný, multiplatformní programovací jazyk vycházející z jazyka C, zaštítěný firmou Oracle. Patří mezi nejpopulárnější a nejužívanější programovací jazyky [11]. Jeho hlavní užití je právě na straně serveru, a to v kombinaci s frameworkem Spring Boot [12].

C#

Jazyk velmi podobný Javě, vyvíjený firmou Microsoft. Jeho hlavní nevýhodou byla vysoká závislost na frameworku .NET, který až do příchodu alternativy .NET Core, byl spustitelný pouze na operačním systému Microsoft Windows.



Obrázek 2.1: Popularita javascriptových frameworků [17]

2.2.2 Web

Webová aplikace je pro uživatele nejpřístupnější formou, nemusí nic instalovat, stahovat, pouze otevře prohlížeč s webovou adresou a aplikaci má přístupnou jak na počítači, tak i na mobilu. Nevýhoda oproti mobilní aplikaci je, že vyžaduje přístup k internetu.

JavaScript

Pro vývoj webové aplikace lze v JavaScriptu zvolit z mnoha možností, ať už co se týče syntaktické nadstavby (TypeScript, Flow), nebo knihoven a frameworků. Jelikož jich je velké množství, tak si jen stručně probereme tři nejpopulárnější (viz obrázek 2.1) a to React [13], Angular [14] a Vue [15].

React je knihovna vyvíjená společností Facebook, která ho sama využívá pro tvorbu jejich aplikací, např Facebook nebo Messenger. Jeho hlavní výhodou je možnost si spoustu věcí přizpůsobit k vlastní potřebě (např. routing nebo globální uložiště dat), což mimo jiné vede k menší velikosti výsledné aplikace a vyšší rychlosti. React sám o sobě není ani závislý na prohlížeči a může být využit např. pro vývoj aplikace spouštěné z příkazové řádky.

Angular je framework od společnosti Google, taktéž využívaný v řadě aplikací. Hlavní výhodou je, že nabízí téměř vše co může programátor potřebovat, díky čemuž nemusí řešit občasné potíže s kompatibilitou knihoven jako u Reactu. Na druhou stranu je výsledná aplikace větší a pomalejší, jelikož obsahuje i nevyužívané funkce.

Vue je knihovna velmi podobná Reactu, sdílí spolu velkou část ideologií [16], ale Vue se zaměřuje na projekty malé až střední a React i na ty velké. Výhoda tedy je rychlost a jednoduchost, nevýhoda je, že s rostoucí velikostí se projekt stává méně přehledný a udržovatelný.

C#

V C# se dá vyvíjet i frontend a to za pomoci frameworku ASP.NET, který závisí na frameworku .NET. V dnešní době se na nové projekty již příliš nevyužívá. Lze jej využít v kombinaci s JavaScriptem a jeho knihovnami. Největší výhoda tohoto přístupu je striktní dodržení přístupu Model-View-Controler, nevýhodou jsou vyšší nároky na znalost obou technologií, což pro juniorního programátora může být problém.

2.2.3 Mobil

Mobilním technologiím se práce zabývá jen okrajově, protože cílem práce je webová aplikace. Přesto je jistá možnost jak z webové aplikace vytvořit aplikaci mobilní nebo z nějaké části sdílet kód mezi webovou a mobilní aplikací.

Responzivní webová aplikace

První a nejjednodušší možností jak vytvořit mobilní aplikaci je pouze optimalizovat aplikaci webovou tak, aby se dala prohlížet i na mobilu. Hlavní výhodou je nízká náročnost vývoje takové aplikace (v porovnání s tvorbou celé mobilní aplikace), nevýhodou však je, že aplikace nemá přístup k některým částem zařízení, třeba stavu baterie, gyroskopu, poloze a dalším.

Progressive Web App

Progressive Web App, nebo Progresivní Webová Aplikace, je způsob jak z webové aplikace udělat aplikaci mobilní bez nutnosti vytvářet nativní aplikaci. Tento způsob je kombinací responzivní webové aplikace, manifestu a service workeru. Manifest je soubor udávající informace pro mobilní prohlížeč, jak se aplikace má jmenovat, popis, ikony, barevné schéma, autor atd. Jedná se o obdobu manifestu, který mají mobilní aplikace. Service worker je speciální skript, který běží na pozadí a umožňuje načítat data, přistupovat k notifikacím nebo poloze (na mobilu i webu) atd. V případě, že webová aplikace splňuje všechny zmíněné náležitosti, a uživatel vstoupí na web pomocí jednoho z prohlížečů, které podporují PWA, pak má možnost si tu aplikaci přidat na plochu mobilu a spouštět stejně jako nainstalovanou aplikaci. Takto nainstalovaná aplikace funguje i bez přístupu k internetu, ale je to v podstatě stažený web spuštěný v prohlížeči. Hlavní nevýhodou je absence možnosti práce přímo s hardwarem telefonu, takže to není vhodné např. pro mobilní hry. Dalším problémem je podpora na zařízeních s operačním systémem iOS. Ačkoliv je společnost Apple původním tvůrcem této myšlenky [18], tak aplikace na iOS nemají přístup k tolika informacím a možnostem jako ty na Android a jsou tedy limitované.

Hybridní aplikace

Hybridní aplikace jsou posledním mezikrokem mezi mobilní a webovou aplikací. Jedná se o aplikaci webovou, která je později zkompilovaná i s jádrem prohlížeče a může být přidaná na Android Play Store, resp. iOS App Store. Takto vytvořená aplikace má již přístup ke všem částem telefonu, stejně jako aplikace nativní, ale nevýhoda je velikost balíčku potažmo aplikace. Ta v sobě obsahuje i část prohlížeče a aplikace má vyšší nároky na paměť a procesorový čas.

Nativní aplikace

Poslední kategorie jsou aplikace nativní, tedy přímo určené pouze na mobilní zařízení. Existuje mnoho programovacích jazyků, které lze při vývoji použít, ať už to je Java a Kotlin pro Android nebo Objective C a Swift pro iOS. Hlavní nevýhoda tedy je mnohem více práce a potenciálně duplicitní kód. Z toho důvodu existují různé multiplatformní frameworky, které tyto neduhy řeší.

V rámci této práce si zmíníme pouze React Native [19]. Jedná se o framework syntaxí velmi podobný frameworku ReactJS a důvod proč zmínit zrovna tento a ne ostatní, je možnost přepoužití kódu z webové aplikace. Jisté části aplikace napsané v ReactJS lze bez úpravy použít i v React Native a obráceně. Tedy aplikace psaná za použití React Native má stejné benefity jako aplikace nativní, ale zároveň v ní lze přepoužít části aplikace webové. Příklad takového přepoužití je v článku *Share Code between React and React Native Apps* [20].

Toto přepoužití vyplývá z architektury samotného Reactu, kde React je knihovna, která zpracovává vstupy, ovládá stav aplikace, kontroluje změny a generuje výstupy. Následně na řadu přichází tzv. „Renderer“, který funguje jako rozhraní mezi platformou (web, mobil nebo konzole) a Reactem. U webu je tímto rozhraním knihovna „React-DOM“ a u mobilu právě „React Native“.

2.3 Cílová skupina

Do cílové skupiny této práce se řadí kdokoliv, kdo má zájem se naučit hrát na ukulele, celosvětově. To znamená, že uživatelem může být muž či žena jakéhokoliv věku. Z toho se odvíjí požadavky na jazyk aplikace a uživatelskou přívětivost. Jelikož spektrum uživatelů je velmi rozsáhlé, tak uživatelské prostředí musí být jednoduché a přehledné. Webová aplikace umožní uživateli si aplikaci zobrazit jak na počítači, tak na mobilu či tabletu a zároveň nemá potřebu nic instalovat. Výchozím jazykem bude angličtina z důvodu celosvětové rozšířenosti.

Kapitola 3

Návrh

Dalším krokem vývoje software je návrh, který pomůže získat celkovou představu o aplikaci, jejích částech a případně problémových místech. Tato kapitola se zabývá požadavky na aplikaci a tvorbou funkční specifikace. Součástí specifikace jsou uživatelské role, případy užití a wireframy, které vývojáři pomůžou získat představu o grafické reprezentaci aplikace.

3.1 Funkční požadavky

FP01 Metronom
Aplikace zobrazí uživateli metronom. Uživatel má možnost zapnout či vypnout metronom a zvolit si rychlost.

FP02 Akordy
Aplikace umožní uživateli si zobrazit libovolnou podmnožinu akordů.

FP03 Strumming pattern
Aplikace zobrazí uživateli strumming pattern a metronom. Uživatel může ovládat metronom dle FP01 a zobrazení strumming patternu reaguje na nastavení rychlosti metronomu.

FP04 Vyhledávání
Aplikace umožní uživateli vyhledávat písně, akordy a strumming patterny.

FP05 Přechytávání akordů
Aplikace umožní uživateli si vytvořit vlastní seznam akordů s volbou nastavení metronomu dle FP01 a akordů dle FP02. Přihlášený uživatel má možnost si dané nastavení uložit pod libovolným jménem.

FP06 Přechytávání akordů dle písně
Aplikace zobrazí uživateli přechytávání akordů dle FP05 s přednastavenými parametry podle vybrané písně.

FP07 Zobrazení písně
Aplikace zobrazí uživateli text písně, akordy písně a strumming pattern dle FP03. Uživatel má možnost přechodu na přechytávání akordů dle FP06 a případně odkaz na přehrání písně na serveru třetí strany (např. Youtube, Spotify).

FP08 Registrace
Aplikace umožní zaregistrovat nového uživatele.

FP09 Přihlášení
Aplikace umožní uživateli se přihlásit.

FP22 Odhlášení
Aplikace umožní odhlásit přihlášeného uživatele.

FP23 Úprava profilu
Aplikace umožní přihlášenému uživateli měnit svoje údaje.

FP24 Označení písně jako oblíbená
Aplikace umožní přihlášenému uživateli označit píseň jako oblíbenou.

FP41 Vytvoření písně
Aplikace umožňuje moderátorovi vytvořit nový záznam o písni.

FP42 Úprava písně
Aplikace umožňuje moderátorovi upravit existující záznam o písni.

FP61 Úprava rolí uživatele
Aplikace umožňuje administrátorovi změnit role uživatele.

3.2 Nefunkční požadavky

NP01 Kompatibilita
Aplikace je dostupná přes webové rozhraní. Webové rozhraní poskytuje podporu pro prohlížeče Mozilla Firefox od verze 68, Google Chrome od verze 79 a Safari od verze 12.

NP02 Podpora menších obrazovek
Aplikace je responzivní, což znamená že se adaptuje na velikost obrazovky uživatele a podporuje dané prohlížeče na mobilních telefonech.

NP03 SEO
Aplikace je SEO-friendly, tedy splňuje všechny náležitosti pro internetové vyhledávače jako jsou Google nebo Seznam. Tyto náležitosti jsou testovány pomocí nástroje Google Lighthouse [21], kde aplikace musí dosáhnout celkového bodového ohodnocení alespoň 90 ze 100.

NP04 API
Aplikace vystavuje soukromé API a to ve formátu GraphQL.

NP05 Rozšiřitelnost
Aplikace je díky správnému návrhu a plánování možností rozšíření lehce rozšiřitelná.

NP06 Lokalizace
Aplikace je dostupná v anglickém jazyce.

3.3 Uživatelské role

Uživatel
Běžný uživatel, který se chce naučit hrát na ukulele nebo využít jinou funkcionalitu aplikace. Může použít metronom, učit se akordy, jejich přechytávání anebo celou píseň. Má možnost se registrovat.

Přihlášený uživatel
To samé jako uživatel, navíc s možností se přihlásit a odhlásit a ukládat oblíbené písně a vlastní přechytávání akordů.

Moderátor
To samé jako přihlášený uživatel s možností editovat písně (akordy, strumming pattern, odkaz na ukázku).

Administrátor
To samé jako moderátor s možností upravovat role uživatelů.

3.4 Případy užití

PU01 Metronom

Případ užití umožňuje uživateli nastavovat parametry metronomu a ovládat ho. Mezi parametry patří tempo a způsob ukazování tempa (zvuková signalizace, blikání, odpočet, nebo vibrace na telefonu).

HS: Zobrazení metronomu

1. Scénář začne, když uživatel zobrazí tuto komponentu.
2. Aplikace zobrazí metronom a k němu příslušející nastavení.

AS01: Zapnutí metronomu

1. Scénář začne, když uživatel klikne na tlačítko „Start“.
2. Aplikace spustí metronom s nastavenými parametry.
   - Aplikace reaguje na změny parametrů okamžitě.
3. Tlačítko „Start“ se změní na tlačítko „Stop“ a scénář končí.

AS02: Vypnutí metronomu

1. Scénář začne, když uživatel klikne na tlačítko „Stop“.
2. Aplikace vypne metronom.
3. Tlačítko „Stop“ se změní na tlačítko „Start“ a scénář končí.

PU02 Výuka akordů

Případ užití umožní uživateli vyhledávat akordy a vybírat zobrazenou podmnožinu.

HS: Zobrazení akordů

1. Scénář začne po přechodu na stránku s akordy.
2. Aplikace zobrazí uživateli vyhledávací pole, které se řídí dle AS01.

AS01: Vyhledání akordu

1. Scénář začne, když uživatel klikne do vyhledávacího pole nebo vyhledávací pole získá focus.
2. Aplikace reaguje na vstup uživatele tak, že po každé změně hodnoty vyfiltruje akordy podle Alg.1.

AS02: Přechytávání akordů

1. Scénář začne, když uživatel vybere více než jeden akord.
2. Aplikace umožní vybrané akordy řadit.
3. Aplikace zobrazí uživateli komponentu z PU01.

AS03: Přechytávání akordů podle písně

1. Scénář začne při přechodu ze stránky písně.
2. Aplikace nastaví hodnoty podle vstupu z odkazu.
3. Aplikace pokračuje AS02.

Alg.1

1. Filtruje akordy podle jejich jména obsahující zadaný podřetězec.
2. Vrátí vyfiltrované výsledky.

PU03 Výuka strumming patternů

Případ užití umožní uživateli zobrazit stránku s výukou strumming patternů.

HS: Zobrazení výuky strumming patternů

1. Scénář začne po přechodu na stránku s výukou strumming patternů.
2. Aplikace zobrazí uživateli stránku s výukou strumming patternů.
3. Aplikace zobrazí uživateli komponentu z PU01.
   - Pokud dojde k zapnutí/vypnutí metronomu, pak se spouští AS02.
4. Aplikace zobrazí uživateli komponentu z PU02.

AS01: Výběr strumming patternu

1. Scénář začne, když uživatel klikne na dropdown seznamu strumming patternů.
2. Po výběru z dropdownu aplikace aktualizuje aktivní strumming pattern.
3. Jestliže uživatel nepotvrdí výběr, pak aplikace nedělá nic, scénář končí.

AS02: Synchronizace a zvýrazňování

1. Pokud došlo k vypnutí metronomu, pak aplikace přestane zvýrazňovat a scénář končí.
2. Jinak aplikace začne zvýrazňovat směr hraní v synchronizaci s metronomem.

PU04 Výuka písní

Případ užití umožní uživateli zobrazovat písně s texty, akordy, metronomem a doporučeným strumming patternem.

HS: Zobrazení stránky písně

1. Scénář začne po přechodu na stránku písně.
2. Aplikace zobrazí komponentu z PU03.
   - Nastaví strumming pattern, akordy a tempo dle písně a znemožní editaci těchto parametrů.
3. Aplikace zobrazí text písně a ovládání automatického odsouvání.

AS01: Zapnutí automatického odsouvání

1. Scénář začne, když uživatel klikne na tlačítko „Start“.
2. Aplikace spustí automatické odsouvání s nastaveným parametrem.
   - Aplikace reaguje na změny parametru okamžitě.
3. Tlačítko „Start“ se změní na tlačítko „Stop“ a scénář končí.

AS02: Vypnutí automatického odsouvání

1. Scénář začne, když uživatel klikne na tlačítko „Stop“.
2. Aplikace vypne automatické odsouvání.
3. Tlačítko „Stop“ se změní na „Start“.

AS03: Označení písně jako oblíbená

1. Scénář začne při zobrazení komponenty, pokud je uživatel přihlášen.
2. Aplikace zobrazí tlačítko „Like“.
   - Pokud uživatel na tlačítko klikne a píseň nemá oblíbenou, pak aplikace nastaví píseň jako oblíbenou.
   - Pokud uživatel na tlačítko klikne a píseň má oblíbenou, pak aplikace odebere píseň jako oblíbenou.

PU05 Vyhledávání

Případ užití umožní uživateli vyhledávat akordy, strumming patterny a písně dle jejich jména nebo autora.

HS: Vyhledávání

1. Scénář začne, když uživatel klikne do vyhledávacího pole nebo vyhledávací pole získá focus.
2. Aplikace zobrazí uživateli našeptávač, který bude rozdělen na 3 části:
   a) Část s akordy
      - Zobrazuje výsledky podle Alg.1.
      - Bude zobrazovat maximálně 3 nejlepší výsledky.
   b) Část se strumming patterny.
      - Zobrazuje výsledky podle Alg.2.
      - Bude zobrazovat maximálně 3 nejlepší výsledky.
   c) Část s písněmi
      - Zobrazuje výsledky podle Alg.3.
      - Bude zobrazovat maximálně 5 nejlepších výsledků.
3. Pokud uživatel klikne na položku v našeptávači, tak bude přesměrován na příslušnou adresu a scénář končí.
4. Pokud uživatel klikne mimo našeptávač anebo vyhledávací pole ztratí focus, pak se našeptávač skryje a scénář končí.
5. Pokud uživatel potvrdí vyhledávání (např. zmáčknutím tlačítka „Enter“), pak je uživatel přesměrován na stránku s vyhledáváním a scénář končí.

Alg.1

1. Viz PU02 Alg.1.

Alg.2

1. Vrátí strumming patterny k písním vyhledaným pomocí Alg.3 ve stejném pořadí.

Alg.3

1. Vrátí nejlepší výsledky na základě shody textu se jmény písní a jejich autorů.

PU06 Správa písně

Případ užití umožňuje moderátorovi vytvořit novou píseň, případně upravit již existující.

HS: Zobrazení podrobností

1. Aplikace zobrazí formulář pro vyplnění vstupních dat s předvyplněnými hodnotami podle editované písně.
2. Jakmile moderátor potvrdí formulář, pak:
   a) Pokud jsou data nezměněna, scénář končí.
3. Aplikace validuje data podle Alg.1.
4. Pokud selže validace, pak je uživateli zobrazena chybová hláška a scénář končí.
5. Aplikace uloží data.

AS01: Vytvoření

1. Aplikace zobrazí formulář pro vyplnění vstupních dat podle HS: Zobrazení podrobností a nic nepředvyplňuje.

Alg.1

1. Píseň musí mít vyplněné jméno, text a akordy.
2. Píseň může mít vyplněný strumming pattern, autora a tempo.

PU07 Správa uživatelského profilu

Případ užití umožní uživateli si zobrazit vlastní uživatelský profil a upravovat některé hodnoty.

HS: Zobrazení profilu

1. Scénář začne, když uživatel zobrazí tuto komponentu.
2. Aplikace zobrazí uživatelský profil s předvyplněnými poli.

AS01: Úprava hodnot

1. Scénář začne, když uživatel změní hodnotu v některém z polí.
2. Aplikace umožní upravit hodnoty některých polí.
3. Aplikace zobrazí tlačítko sloužící k uložení informací „Save“.
4. Po kliknutí na tlačítko „Save“ aplikace provede validace polí podle Alg.1.

- Pokud validace proběhnou úspěšně, pak jsou informace odeslány na server a pokračuje se AS02.
- Jinak jsou zobrazeny chybové hlášky podle toho, jaké validace selhaly.

5. Pokud chce uživatel opustit stránku před uložením změněných hodnot, tak je o tom informován s možností uložit hodnoty.

AS02: Přenačtení hodnot

1. Scénář začne po kliknutí na tlačítko „Refresh“.
2. Aplikace přenačte hodnoty ze serveru a aktualizuje pole.
3. Scénář končí.

Alg.1

1. Pokud hodnota v poli „Email“ není platný email, pak algoritmus vrátí chybu.
2. Pokud je hodnota v poli „Heslo“ kratší než 5 znaků, pak algoritmus vrátí chybu.
3. Pokud hodnota v poli „Opakování hesla“ neodpovídá hodnotě v poli „heslo“, pak algoritmus vrátí chybu.

PU08 Správa uživatelských rolí

Případ užití umožní administrátorovi vyhledávat uživatele a měnit jejich role.

HS: Zobrazení vyhledávání

1. Scénář začne, když administrátor zobrazí tuto komponentu.
2. Aplikace zobrazí pole, které reaguje na vstup a vyhledává uživatele v tabulce výsledků.

AS01: Úprava rolí

1. Scénář začne, když administrátor změní nějakému uživateli roli.
2. Aplikace umožní upravit roli jednoho či více uživatelů najednou.
3. Aplikace zobrazí tlačítko sloužící k uložení informací „Save“.
4. Po kliknutí na tlačítko „Save“ aplikace provede uložení hodnot do databáze a pokračuje AS02.
5. Pokud uživatel chce opustit stránku před uložením změněných hodnot, tak je o tom informován s možností uložit hodnoty.

AS02: Přenačtení hodnot

1. Scénář začne po kliknutí na tlačítko „Refresh“.
2. Aplikace přenačte hodnoty ze serveru a aktualizuje pole.
3. Scénář končí.

PU09 Přihlašování

Případ užití umožní uživateli se přihlásit nebo odhlásit.

HS: Zobrazení tlačítka

1. Scénář začne po zobrazení komponenty.
2. Pokud uživatel není přihlášen, pak aplikace zobrazí tlačítko „Přihlásit“.
3. Jinak aplikace zobrazí tlačítko „Odhlásit“.

AS01: Přihlášení

1. Scénář začne po stisknutí tlačítka „Přihlásit“.
2. Aplikace zobrazí přihlašovací formulář a čeká na vyplnění a potvrzení.
3. Aplikace provede kontrolu vyplnění hodnot polí přihlašovací jméno a heslo. Pokud alespoň jedna není vyplněna, pak aplikace zobrazí chybovou hlášku a scénář končí.
4. Aplikace zjistí, jestli existuje přihlašovací jméno v databázi uživatelů a jestli heslo k němu přiřazené odpovídá tomu ze vstupu.
5. Pokud ano, pak je uživatel přihlášen a komponenta se přenačte.
6. Jinak aplikace zobrazí chybovou hlášku a scénář končí.

AS02: Odhlášení

1. Scénář začne po stisknutí tlačítka „Odhlásit“.
2. Aplikace odhlásí uživatele a přenačte komponentu.

PU10 Registrace

Případ užití umožní uživateli vytvořit nový účet.

HS: Zobrazení formuláře

1. Scénář začne po zobrazení komponenty.
2. Aplikace zobrazí uživateli formulář pro vyplnění následujících hodnot: „Přihlašovací jméno“, „email“, „heslo“, „opakování hesla“.
3. Aplikace zobrazí tlačítko „Registrovat“.

AS01: Registrace

1. Scénář začne po stisknutí tlačítka „Registrovat“.
2. Aplikace provede kontrolu dle Alg.1.
   - Pokud algoritmus vrátí chybu, pak je zobrazena a scénář končí.
   - Jinak jsou informace zaneseny do databáze a uživatel přihlášen.

Alg.1

1. Pokud je hodnota v poli „Přihlašovací jméno“ kratší než 6 znaků, pak algoritmus vrátí chybu.
2. Pokud hodnota v poli „Email“ není platný email, pak algoritmus vrátí chybu.
3. Pokud je hodnota v poli „Heslo“ kratší než 5 znaků, pak algoritmus vrátí chybu.
4. Pokud hodnota v poli „Opakování hesla“ neodpovídá hodnotě v poli „heslo“, pak algoritmus vrátí chybu.

3.4.1 Pokrytí případy užití

Z tabulky 3.1 je vidět, že všechny funkční požadavky na aplikaci jsou pokryty případy užití a tedy, že se na žádnou část nezapomnělo. V přiloženém obrázku 3.1 jsou znázorněni jednotliví aktéři, vztahy mezi nimi a vztahy k případům užití.

3.5 Návrh uživatelského rozhraní

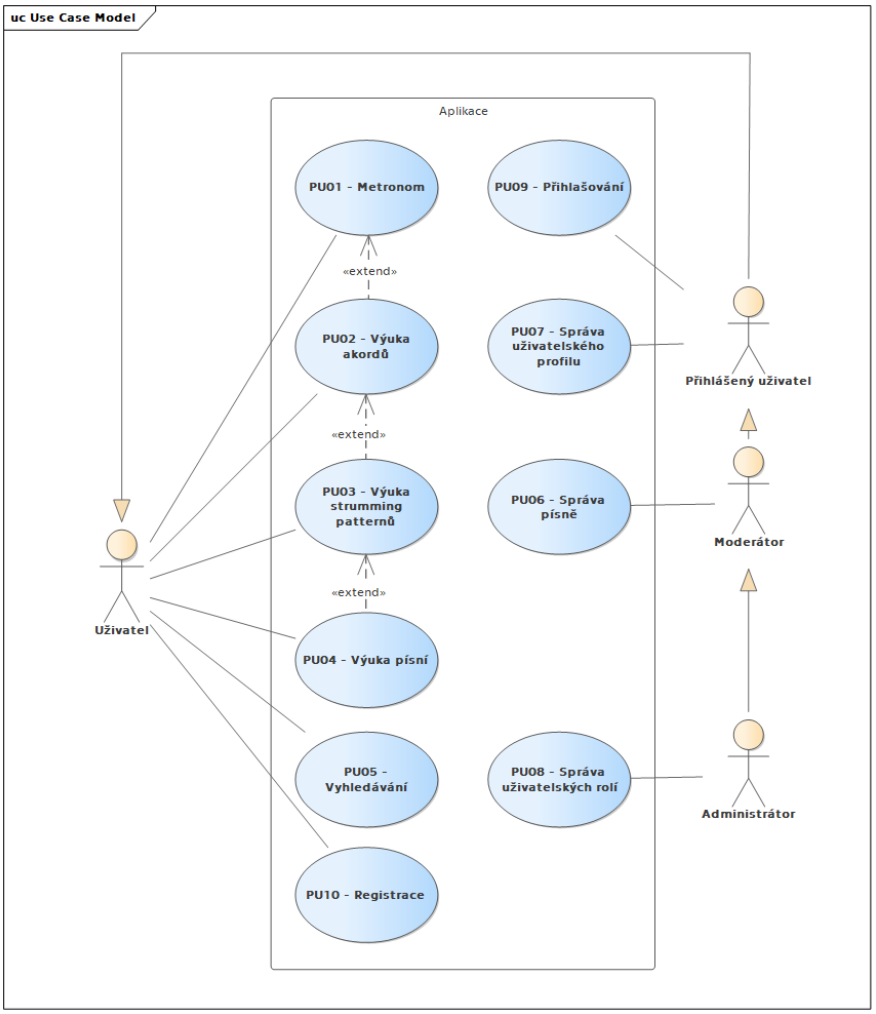
Návrh uživatelského rozhraní je důležitá součást vývoje software, protože to je ta část, kterou vidí uživatel. Sebelepší funkcionalita je aplikaci k ničemu, když bude nepřehledná a příliš komplikovaná. Je několik možností, jak přistoupit k takovému návrhu, dělené podle úrovně detailu, kterým se zabývají.

	PU01	PU02	PU03	PU04	PU05	PU06	PU07	PU08	PU09	PU10
FP01	✓		✓	✓						
FP02		✓	✓	✓						
FP03			✓	✓						
FP04					✓					
FP05		✓								
FP06		✓								
FP07				✓						
FP08										✓
FP09									✓	
FP22									✓	
FP23							✓			
FP24				✓						
FP41						✓				
FP42						✓				
FP61								✓		

Tabulka 3.1: Kontrola pokrytí funkčních požadavků případy užití

Nejvíce abstraktní, resp. nejméně se zabývající detaily je wireframe. Wireframe ukazuje část aplikace, její rozvržení a zabývá se pouze důležitými částmi. Další úrovní je mockup, který se na rozdíl od wireframu věnuje i méně podstatným částem aplikace a detailům jako jsou třeba barevná schémata či typografie. Poslední, nejpokročilejší a zároveň nejkomplikovanější možností je prototyp. Prototyp rozšiřuje mockup o interakce, popř. animace. Umožňuje simulovat interakce mezi částmi aplikace a reprezentuje podobu konečného produktu. Rozdíl mezi prototypem a konečným produktem je, že prototyp má většinou nasimulovaná data, která by aplikace jinak získávala z backendu. Prototypy jsou většinou vytvářeny před vývojem samotné aplikace pro zjištění, zdali má vůbec smysl takovou aplikaci vytvářet. Pokud má vývojář či tým zadaný požadavek na nějakou aplikaci, nemá smysl ztrácet čas s prototypem, jelikož aplikaci musí dodat.

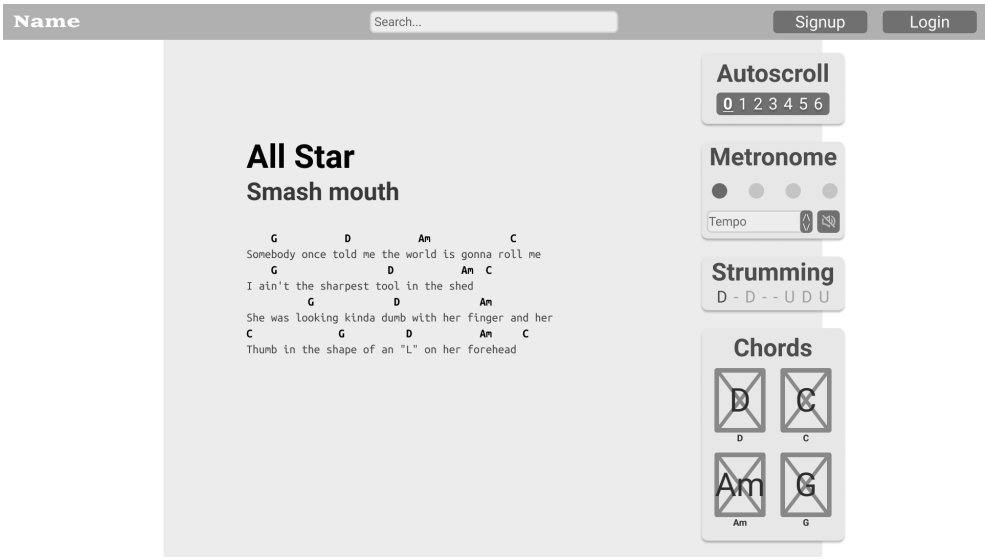
Práce uvažuje o návrhu uživatelského rozhraní jako o wireframu, jelikož v kontextu této práce nemá smysl dělat prototyp. Co se týče výběru wireframu oproti mockupu, pak to je čistě subjektivní. V praxi u větších firem vyvíjející jednu z mnoha aplikací je upřednostňován wireframe, jelikož detaily jako barevné schéma, či typografie se odvíjí od dalších aplikací a firemních politik. Wireframy jsou v rozlišení 1920×1080 pixelů, tedy FullHD a byly vytvořeny v nástroji *figma.com* [22], který umožňuje vytváření wireframů, mockupů i prototypů.



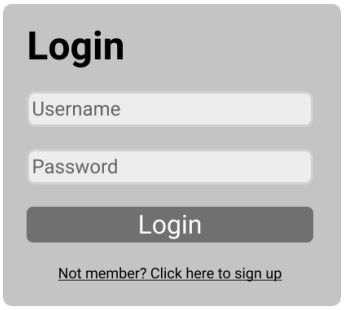
Obrázek 3.1: Diagram případu užití

3.5.1 Stránka s písní

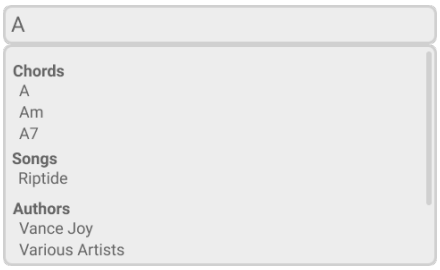
Nejdůležitější stránka celé aplikace. Jedná se o komponentu z PU04, tedy výuka písní a obsahuje komponenty Metronom, Akordy a Strumming pattern. Stránka obsahuje hlavní panel s možností vyhledávání a tlačítka pro registraci a přihlášení. V hlavní části je nejprve název písně, pak název autora a následně text a akordy písně. Zmíněné komponenty jsou po pravé straně, jelikož kdyby byly pozicovány nalevo, tak by se uživateli mohl číst špatně text, který by byl zarovnán vedle komponent. Celý obsah je na velké obrazovce odsazen na střed, tak aby se uživatel mohl soustředit pouze na část obrazovky.



(a) Stránka s písní



(b) Přilašovací okno



(c) Vyhledávací našeptávač

Obrázek 3.2: Wireframy

Kapitola 4

Realizace

Tato kapitola se bude zabývat konečným výběrem technologií a architekturou aplikace. Správným výběrem technologií a kvalitním návrhem aplikace je možné zajistit snadnou rozšiřitelnost aplikace a zároveň se zpřehlední kód.

4.1 Použité technologie

Pro vývoj samotné aplikace byl využit TypeScript, pro frontend framework React v kombinaci s Apollo Client [23]. Pro backend Node.JS, konkrétně knihovna Express.js [24] a Apollo Server [23] a pro komunikaci GraphQL. K vývoji však patří ještě další podpůrné nástroje, kterými se zabývají následující sekce, stejně jako podrobnějšímu popisu již zmíněných technologií.

4.1.1 GraphQL

GraphQL, kde *QL* je zkratka pro Query Language, je dotazovací jazyk, který je vyvíjený přímo pro použití v API, původně vyvíjený firmou *Facebook*, dnes již pod neziskovou organizací *GraphQL Foundation*. Na rozdíl od standardního REST API, které umožňuje získání, nebo úpravu jednoho objektu pomocí specifické cesty, tak GraphQL server naslouchá jedné cestě a jako parametr dostává stromovou strukturu zapsanou ve speciálním formátu. Tento přístup má několik výhod, umožňuje vývojáři vybrat si přímo jaká data požaduje (a tím snížit datový tok), slučovat více požadavků do jednoho a silné typování samotného API endpointu což umožňuje našeptávání a statickou kontrolu. Nevýhoda oproti REST API je využití jedné cesty a tím znemožnění cachování na straně prohlížeče. [25]

Ve spojitosti s GraphQL se práce zabývá pažadavky typu „query“ a „mutace“. Query je obdobou GET požadavku REST API, tedy pouze získá data. Mutace na druhou stranu je obdoba POST, UPDATE nebo DELETE požadavku, tedy jakákoliv operace, která nějak manipuluje s daty. Oba typy mohou mít proměnné a vracet nějaké hodnoty. Existuje ještě jeden specifický

typ požadavku, „subscription“, který umožňuje navázat trvalé spojení a dostávat okamžitá upozornění na nová data. Tento typ požadavku avšak není potřebný pro tento typ aplikace, hodí se však kdekoliv, kde je třeba upozornit na změny okamžitě, např. chat. [26]

4.1.2 MongoDB a Mongoose

MongoDB [27] je NoSQL databáze. Hlavní výhoda NoSQL databází je rychlejší vývoj a prototypování, jelikož se nemusí žádným způsobem vytvářet tabulky. Výhodou MongoDB je přístup k ukládání dat, a to ve formě dokumentů, kde každý dokument je tvořen z dvojic klíče a hodnoty. Takovýto dokument je až na typy shodný s JSON, tedy propojení MongoDB a javascriptové aplikace je triviální.

Mongoose [28] je tzv. Object Relation Mapping (ORM) nástroj vytvořený firmou *Automattic* přímo pro databázi MongoDB. To umožňuje vytvořit jednotlivé datové modely ze schémat. Schéma je objekt definující vlastnosti modelu, jako jsou pole, indexy, validace, virtuální pole a další [29]. Takto vytvořený model poskytuje možnost vytvářet, vyhledávat, upravovat či mazat záznamy v databázi.

Další a největší výhodou tohoto přístupu vytváření modelů je možnost z takových modelů automaticky vytvořit GraphQL mutace a query a to za použití knihoven *graphql-compose* a *graphql-compose-mongoose*. Tyto knihovny vytvoří základní query a mutace pro vyhledávání či úpravu modelů, které jsou snadno rozšiřitelné o další, vlastní operace nad modely.

4.1.3 Yarn v2

Yarn [30] je alternativa k npm, což je balíčkovací systém pro JavaScript, obdoba NuGetu pro C# nebo Mavenu pro Javu.

Od verze 2.0, taky zvané *berry*, integruje podporu tzv. monorepozitáře, tedy přístupu kdy jeden repozitář obsahuje několik podprojektů. Tuto možnost poskytuje i nástroj Lerna [31], kde se vývojáři Yarn v2 inspirovali. To vede k možnosti modularizace aplikace a zavedení Single responsibility principu.

Další podstatná funkcionalita Yarn v2 je vynucená podpora Plug'n Play (PnP) režimu. PnP je přístup zavedený týmem hlavních vývojářů Yarnu. Při standartní instalaci balíčků pomocí npm nebo yarn bez pnp, jsou balíčky staženy a rozbaleny do adresáře *node_modules* ze kterého potom čerpá Node Resolution Algorithm (NRA) [32]. Tento algoritmus při sestavování výsledné aplikace pro každé volání funkce `require()` vždy projde složku *node_modules* a pokud daný balíček nenajde, pak rekurzivně pokračuje v nadřazeném adresáři (za předpokladu, že nadřazený adresář obsahuje složku *node_modules*). Tento přístup je i s nějakými optimalizacemi velmi pomalý. Yarn s PnP provádí to, že při prvotní instalaci závislostí místo vytvoření *node_modules* složky, kam by se následně rozbalovaly balíčky, tak vytvoří vlastní adresář

.yarn/cache kam stáhne balíčky, nerozbaluje je a následně vytvoří soubor *pnp.js* který uchovává odkazy na všechny balíčky a závislosti. Výhodou je až o 70 % vyšší rychlost a menší zatížení disku a procesoru při instalaci. Nevýhodou je, že všechny instalované balíčky musí mít explicitní výpis všech svých závislostí, což velká část do dnešní doby nemá a spoléhá na to, že jejich závislosti prostě budou k dispozici a NRA je najde.

Jelikož Yarn v2 ještě nevyšel jako stabilní verze, nýbrž pouze jeho release kandidáti, tak se k němu špatně hledají informace v případě potíží. Další problém, který tento přístup přinesl do práce, byla nutnost opravy chybějících závislostí některých balíčků. Jak již bylo zmíněno, každý balíček musí mít seznam svých závislostí a když nemá, tak sestavení selže. Takže je na každém vývojáři, aby tyto závislosti doplnil a to formou zápisu do konfiguračního *.yarnrc.yml* souboru, kde je v práci takto opraveno zhruba 30 závislostí. Výhodou pak je mnohem rychlejší instalace závislostí, sestavení aplikace a podpora monorepozitáře.

4.1.4 Express

Express je minimalistický webový framework pro Node.js. Jedná se o jednoduchý webový server, který dokáže obsluhovat standardní požadavky HTTP. Umožňuje, mimo jiné, jednotlivým cestám nastavit middleware, což jsou funkce, které jsou spuštěny před samotným vyhodnocováním funkce k dané cestě. Typickým využitím je autentizace uživatele, nebo zapisování do log souboru. V rámci aplikace byl použit jak pro poskytování backend, tak i frontend serveru. Backend server poskytuje GraphQL API a zajišťuje napojení na databázi, zatím co frontend server poskytuje Server Side Rendering (SSR) aplikace. Co znamená SSR je podrobněji vysvětleno v rámci samostatné sekce 4.4.1.

4.1.5 Apollo Server

Apollo Server je GraphQL server vyvíjený společností *Meteor Development Group, Inc.*, který je nezávislý na volbě webového serveru, v našem případě Express, který je takzvaně *unopinionated*, což znamená, že vývojáře netlačí nějakou specifickou cestou vývoje. Jeho hlavní výhodou je hlavně jednoduchost, ale i popularita, tedy větší komunita. Menším detailem, který potěší hlavně vývojáře, je automatické vystavování GraphQL Playground, což je webová aplikace umožňující simulování dotazů, které má funkce jako IDE, tedy našeptávání, kontrola typů, kontrola syntaxe, atd.

4.1.6 Apollo Client

Apollo Client je od stejné společnosti jako 4.1.5 a tyto dvě knihovny nejlépe fungují právě, když jsou pohromadě. Mezi hlavní výhody Apollo Client patří snadné cachování, velké množství dalších balíčků od komunity, nebo správa lokálního stavu aplikace, kterému se věnuje podrobněji sekce 4.4.2.

4.1.7 React

Konečná volba padla na React hlavně z důvodu největší popularity, a tedy i největší podpory ze strany komunity. V rámci této práce je již zakomponováno několik technologií, které nejsou zatím se stabilním stavu a je tedy mít nějaký styčný bod, na který je možný plně spoléhat. I z toho důvodu byla použita stabilní verze Reactu a ne *nightly* nebo *experimentální*, které sice poskytují některé pěkné funkcionality, např. *Concurrent Mode* zrychlující render aplikace nebo *Suspense* zjednodušující čekání na asynchronní úkoly, ale výsledná aplikace by byla nestabilní, což by pro vývoj znamenalo značné zdržení.

4.1.8 Storybook.js

Tato knihovna umožňuje vytvářet interaktivní dema k jednotlivým React komponentám. Každá komponenta může obsahovat story soubor, který specifikuje, jak se má daná komponenta zobrazit a případně nastavení k dalším pluginům. Tyto story soubory jsou následně načteny, seřazeny do stromové struktury a z toho je následně vytvořena interaktivní dokumentace. Tato dokumentace je velmi přehledná a zároveň nutí vývojáře izolovat komponenty, což opět vede implementaci Single responsibility principu. Jednotlivé pluginy rozšiřují možnosti například pro generování dokumentace z JSDoc komentářů nebo aby jednotlivé story byly součástí jednotkových testů.

4.1.9 Jest

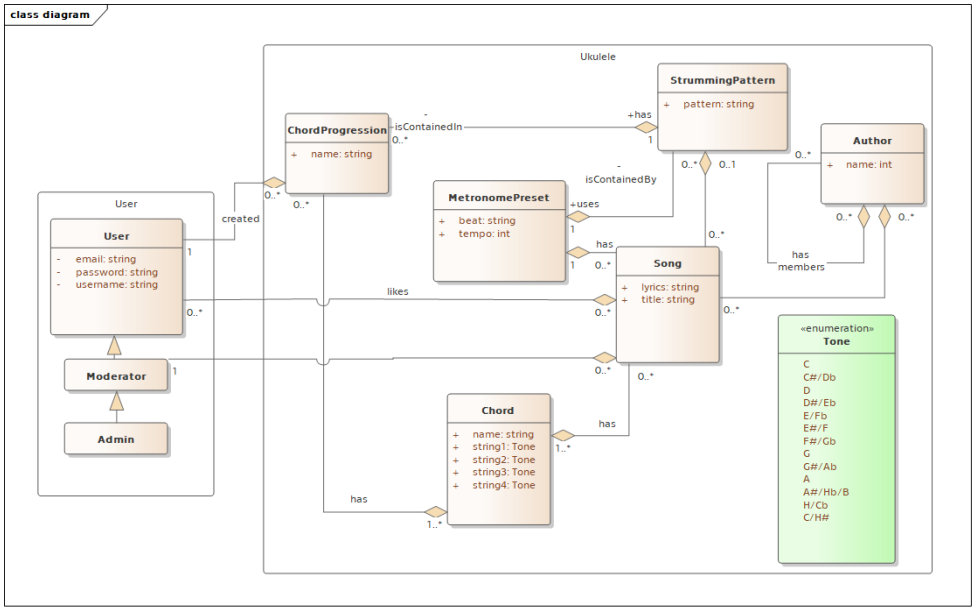
Jest [33] je testovací knihovna pro JavaScript vyvíjená společností Facebook. Umožnuje psát jednotkové testy, simulovat data, generovat reporty pokrytí kódu ale hlavně, má ze všech testovacích knihoven pro JavaScript nejlepší podporu testování React komponent.

4.2 Architektura systému

Jak již bylo zmíněno v nefunkčních požadavcích, aplikace má být lehce rozšiřitelná. Tomuto požadavku se dá vyhovět několika způsoby, avšak ideologii JavaScriptu se nejvíce blíží je modularizace [34]. A pro lepší přehlednost kódu a zjednodušení testování samotných modulů, se jednotlivé moduly řídí metodikou Clean Architecture.

4.2.1 Modularizace

Aplikace byla modularizována tak, že nejprve byla rozdělena na jednotlivé domény, které je možné vidět na obrázku 4.1. Následně tyto domény byly rozděleny na tři části, backend, frontend a společné.



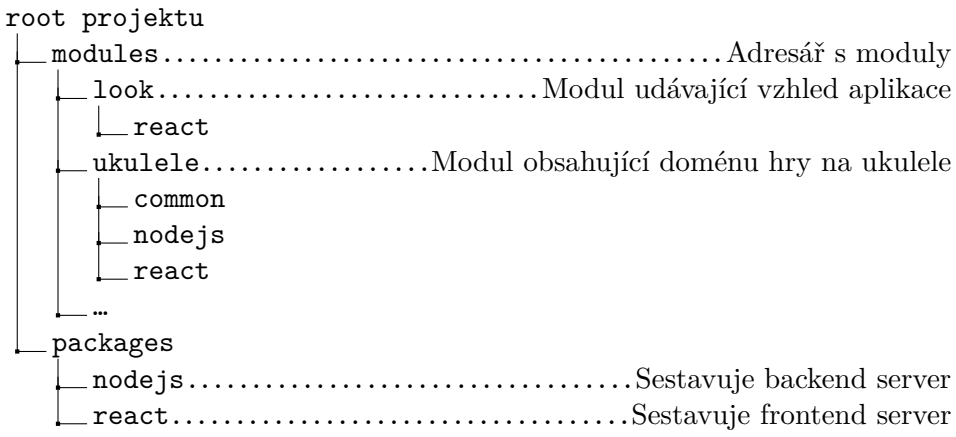
Obrázek 4.1: Diagram tříd znázorňující domény

Backend část specifikuje veškerou práci s daty, která se odehrává na serveru, od validace až po vystavování mutací a query pro GraphQL server. Tato část smí být závislá jen na společné části domény a modulu autorizace. Díky tomu se zajistí striktní oddělení domén a zjednoduší testování, jelikož jednotkové testy pak testují jen skutečně jednu funkcionalitu nezávisle na všech ostatních. Každý takovýto balíček exportuje jednu funkci, která jako parametr dostane nastavení a vrací objekt obsahující *seed* pro inicializaci aplikace, mutace a query pro GraphQL server.

V části frontend jsou vytvořeny jednotlivé komponenty, zase pouze pro jednu danou doménu. Tato část je závislá pouze na společné části domény, na modulu autorizace a na modulu *look*, který udává vzhled aplikaci, tím že obsahuje základní komponenty, jako například tlačítko nebo základní vstupní pole. Takovýto balíček exportuje dané komponenty.

Takto rozdělené části tvoří balíčky. Tyto balíčky na sobě závisí, tak jak bylo popsáno výše a tyto závislosti jsou spravovány právě pomocí Yarn v2 jako monorepo. Jednotlivé balíčky mají unifikované jména, a to takto: *uls@<doména>-<react|nodejs|common>* kde uls je zkratka pro *ukulele learning site*, což je pracovní název aplikace.

Jednotlivé balíčky ovšem netvoří výslednou aplikaci, proto je potřeba mít ještě další dva speciální balíčky, jeden pro backend a druhý pro frontend. Backend balíček importuje všechny inicializační funkce, vytvoří inicializační parametry, vytvoří jednotlivé moduly a následně je registruje do GraphQL serveru. Frontend balíček nejen, že importuje všechny komponenty, ale následně



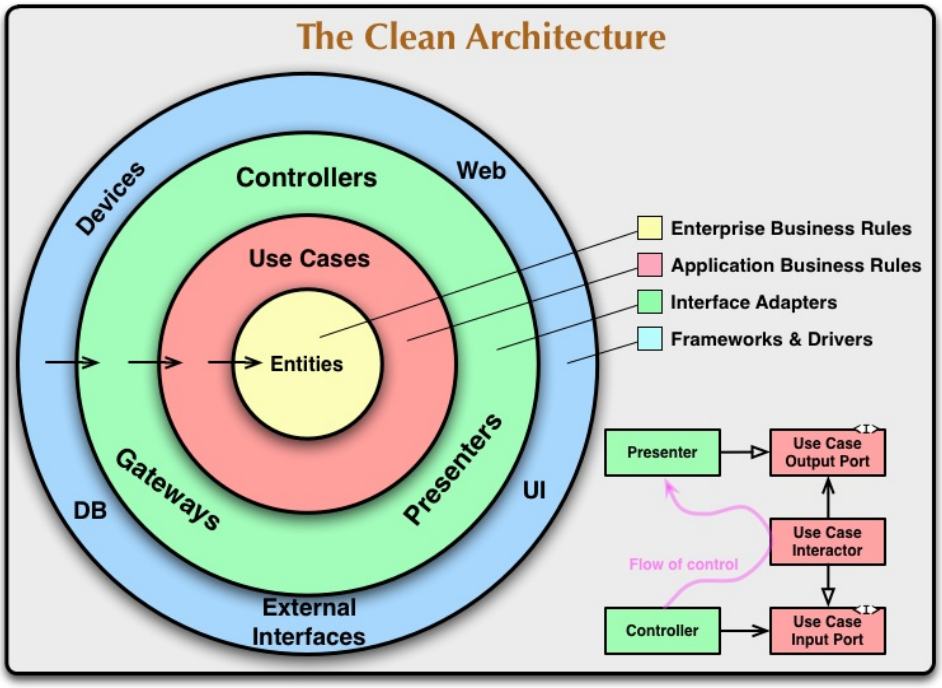
Obrázek 4.2: Zjednodušená adresářová struktura

je skládá na stránky. Jednotlivé komponenty tedy neví, v jakém kontextu a kde budou zobrazeny, a proto je potřeba aby byly správně zapouzdřené a komunikovali jen pomocí vstupních parametrů. Top level balíček, který je nadřazený všem, vytváří pouze testovací prostředí a slouží jako root balíček pro všechny moduly tak, aby mohlo správně fungovat monorepo.

4.2.2 Clean Architecture

Tato metodika, prezentovaná ve stejnojmenné knize od R. C. Martina [35], popisuje jak rozdělit aplikaci na jednotlivé vrstvy. Říká, že aplikace by měla být rozdělena na *entity*, *use case*, *presenters, controllers* a *frameworks, drivers*, znázorňující obrázek 4.3. Entity definují základní byznys pravidla dané domény a jejich reprezentace může být například sada funkcí, nebo objekt s metodami nebo bez. V rámci této práce jsou entity reprezentovány pomocí objektů bez metod, přesněji TypeScript interfaces, a to hlavně z důvodu snadné serializace a deserializace. Vrstva use case provadí operace specifické pro aplikaci nad danými entitami. V případě této práce to jsou *intereactory*, třídy které manipulují s entitami. Následující vrstva už převádí dané entity na objekty vhodné pro další vrstvu frameworků, jedná se tedy o interface. V této práci je tato vrstva Mongoose modely, se kterými následně pracuje databáze a zároveň ze kterých se generuje GraphQL API, kterým backend předává informace frontendu. A poslední vrstvou jsou samotné frameworky či databáze, v případě této práce se jedná o React, resp. MongoDB, které informace zobrazují, resp. ukládají.

Výhodou tohoto rozdělení je přehlednost a snadné testování. Při testování dané vrstvy, se vždy přistupuje z vrstvy nadřazené, test tedy simuluje operace, které by dělala vrstva nadřazená a mockuje se první podřazená vrstva. Je tedy vždy jasné, jak daný test funguje a jaká data testuje. V případě těchto testů mluvíme o testech jednotkových, nikoliv smoke nebo integračních.



Obrázek 4.3: Clean Architecture [36]

4.3 Backend

Realizace samotného backendu nebyla náročná, právě díky vhodně zvoleným technologiím. Většina potřeb byla vyřízena pouhým definováním mongoose schémat, ze kterých se následně vytvořili modely a z nich GraphQL schéma. Úprav modelů bylo minimum, jako příklad takové úpravy by mohlo být smazání pole hesla z modelu uživatele při vracení daného modelu. Tato úprava je vidět na ukázce kódu 4.4, kde je k metodě `toJSON`, přidaný transformátor, který vytvoří instanci `UserInteractor` a zavolá `ui.stripUser()`. Tato metoda pouze vrátí kopii původního uživatele bez hesla.

```
UserSchema.set('toJSON', {
    transform: function(doc: any, ret: any, opt: any) {
        const ui = new UserInteractor(ret);
        return ui.stripUser();
    },
});
```

Obrázek 4.4: Odebrání hesla z modelu uživatele

Největší problém bylo provázání modulů, tak aby na sobě byly nezávislé ale zároveň poskytovali všechny funkcionality. Konkrétně se jednalo o modul uživatele a výuky, například kvůli FP24, který vyžaduje propojení jednotlivých modulů. Tento problém byl vyřešen pomocí inicializačního kontextu.

Každý modul je při spuštění aplikace inicializován s inicializačním kontextem. Inicializační kontext je objekt, který modulům dodává potřebné objekty pro správnou inicializaci. Tento kontext jednotlivé moduly mohou modifikovat, jelikož JavaScript pro neprimitivní datové používá ukazatele, tedy změny ve vstupním parametru jsou zachovány. První je tedy inicializován modul uživatele, který uloží do `creatorModel` vytvořený mongoose model uživatele, se kterým další moduly mohou pracovat.

```
interface ServerModuleOptions<STypeComposer = any> {
    // Objekty vyjímek (validace, autorizace)
    errors: ServerModuleErrors;
    // Funkce která z~objektu vytvoří autorizační token,
    // který je odeslán
    tokenCreator: TokenCreator;
    // Hash funkce
    hashFunction: HashFunction;
    // Objekt poskytující metody pro vytváření umělých dat
    seedFaker: SeedFaker;
    // type composer z~graphql-compose
    creatorModel?: STypeComposer;
}
```

Obrázek 4.5: Inicializační kontext

Každý modul vystavuje inicializační metodu, která má jako parametr inicializační kontext a jako výsledek vrací pole ServerModuleModel. Tento model musí být generický, jelikož tento interface je definovaný v balíčku *core-nodejs*, který má minimum závislostí.

Inicializace a spuštění backend aplikace může probíhat dvěma způsoby. První způsob je vytvoření dokumentů a vložení inicializačních dat (tzv. seed databáze) a druhý způsob je spuštění samotného serveru a poskytování GraphQL API. V obou případech se nejprve inicializují moduly, nejprve uživatelský, následně ostatní. V případě seedu databáze se pak vytvoří spojení s databází a postupně se do dokumentů vkládají inicializační data (`ServerModuleModel::seed`). Jakmile jsou data vložena, tak se aplikace ukončí. V případě druhém se po inicializaci modulů začne vytvářet GraphQL endpoint. Nejprve se projdou všechny inicializované *modely* a přidají se mutace a query. Následně se vytvoří poslední část samotného endpointu, vyhledávací query, server se připojí k databázi a začne vystavovat GraphQL API.

```
interface ServerModuleModel<
    // datový typ seedovacích dat
    TSeed = any,
    // datový typ graphql resolveru
    RResolver = any,
    // datový typ graphql-compose type composeru
    STypeComposer = any
> {
    // jméno databázové entity
    name: string;
    // pole mutací
    mutation: { [key: string]: RResolver };
    // pole query
    query: { [key: string]: RResolver };
    // výsledek seedovací funkce
    seed: TSeed;
    // type composer z~graphql-compose
    typeComposer: STypeComposer;
    // voitelná vyhledávací query
    searchQuery?: RResolver;
}
```

Obrázek 4.6: Server module model

Jelikož je potřeba vyhledávat mezi moduly, ale ne všechny entity a nejednotným způsobem, tak pro každou entitu může být specifikovaná vyhledávací query. Výsledná souhrnná vyhledávací query pak jen paralelně provolá všechny vyhledávací query z modelů, spojí výsledky do pole a vrátí výsledek. Tento přístup umožňuje specifikovat jaké entity a případně jakým způsobem se mají vyhledávat. Další výhoda je, že na sobě moduly nijak nezávisí a další rozšíření je velmi jednoduché.

4.4 Frontend

Na rozdíl od backendu, frontend moduly se nijak neinicializují, jelikož vystavují už hotové komponenty a GraphQL mutace a query. Na druhou stranu frontend dělá více, než jen skládání mutací a query do jednoho endpointu. Proto následující sekce probírají podrobněji důležité a specifické aspekty aplikace. Funkce jako správa uživatelů nejsou ničím specifické, a tedy i přesto, že jsou implementované, nemá smysl se nimi zabývat v rámci této práce.

4.4.1 Server Side Rendering

Největším oříškem, co se týče nastavení projektu a výběru technologií, je funkce zvaná Server Side Rendering (SSR). Některé projekty se tímto vůbec nezabývají a to hlavně jelikož to pro ně nemá moc přínos, ale na druhou stranu to přidá hodně práce. Hlavní důvody, proč implementovat SSR jsou dva, první je SEO optimalizace a druhá je optimalizace tzv. First Contentfull Paint (FCP), tedy času od začátku požadavku než uživatel uvidí něco užitečného. Tato *užitečná* informace může být i třeba jen zobrazení načítání, ale je lepší uživateli zobrazit nejprve načítání a až pak interaktivní aplikaci, než čekat stejnou dobu na interaktivní aplikaci a mezitím mít zobrazenou bílou stránku.

Pro funkci SSR však musíme mít další Node.js server, který obsluhuje http požadavky. Tento server pro vybrané, nebo pro všechny cesty vyrenderuje aplikaci a vrátí html soubor, které již danou aplikaci obsahuje. Pokud takto obsluhuje jen některé cesty, tak pro zbylé vrací standardní *prázdný* html soubor, bílou stránku, a render aplikace pak nechává na klientovi. Pokud byla aplikace renderovaná na serveru, tak klient musí provést ještě jeden krok, zvaný hydratace, který celou aplikaci projde, vytvoří virtuální DOM strukturu, se kterou potom vnitřně pracuje a aplikaci oživí.

Tato aplikace implementuje SSR kvůli NP03 SEO. Aplikace bez SSR vrací klientovi html soubor s špatně nastavenou hlavičkou a web crawleři většinou nemají podporu JavaScriptu, takže ani nikdy neuvidí vyrenderovanou aplikaci. Správné nastavení hlavičky zajišťuje knihovna *react-helmet*. který při renderování na straně serveru extrahuje title a meta tagy, které jsou následně poslané v hlavičce výsledku.

4.4.2 Lokalní správa stavu aplikace

Aplikace potřebuje mít nějakou správu globálního stavu, a to hlavně aby komponenty mohli reagovat na to jestli je uživatel přihlášený, nebo ne. Pro tuto funkcionalitu se běžně v aplikacích používá velmi oblíbená knihovna *redux*. Správu lokálního stavu však umožňuje i knihovna Apollo Client, která již je v aplikaci kvůli provolívání backend serveru.

Apollo Client přistupuje ke správě lokálního stavu aplikace odlišně, než jiné knihovny. Místo upravováního nějakého globálního storu pomocí funkcí, reducerů nebo něčeho podobného, je lokální stav upravovaný GraphQL mutacemi a čten pomocí query. Tyto mutace a query se však musí nějak odlišit od normálních požadavků a z toho důvodu jsou označeny anotací `@client`. Takový požadavek Apollo Client zachytí a zpracuje pomocí lokálních resolverů. Tato data jsou ukládány a čteny z stejné paměti, kterou Apollo Client používá pro ukladání cache požadavků.

4.4.3 CSS in JS

Je mnoho možností, jak přistupovat k stylování komponent, čisté CSS, preprocesory SASS či LESS a CSS modules. Další možností je CSS in JS, které kombinuje všechny uvedené možnosti a navíc zlehčuje interakci s JavaScriptem. Místo standardních souborů dedikovaných jen pro stylování, jsou styly psané v rámci javascriptových nebo typescriptových souborů ve formě funkcí, objektů nebo template literals. Dvě nejpoužívanější knihovny poskytující tuto funkcionalitu jsou *styled-components* a *emotion*. Emotion má menší velikost balíčku, lepší podporu SSR, je rychlejší [37], lépe optimalizovaný pro tree shaking [3]. Volba právě této knihovny umožnila rychlý a jednoduchý vývoj s podporou více vzhledů aplikace.

4.4.4 Code splitting

Code splitting je metoda rozdělení sestavené aplikace na více souborů. Jde o optimalizaci ve formě zmenšení prvotního načítání. Aplikace je teda rozdělena na části, které se načítají postupně, jak uživatel prochází aplikací. Při první návštěvě se tedy ke klientovi stahuje soubor *main.js*, který obsahuje nutné části aplikace a pak další soubor, podle toho jakou stránku navštívil. Při přechodu na další stránku se donačte skript pro tu danou stránku. Tím se docílí snížení čas FCP. Navíc některé stránky, třeba administrátorské, běžný uživatel nikdy neuvidí, tak nepotřebuje stahovat jejich skripty.

Možností jak docílit efektivního rozdělování kódu je několik [38]. První možností je na úrovni webpacku, což ale z dlouhodobého hlediska není udržitelné. Dále tuto funkcionalitu poskytuje *React.lazy* a *Suspense*, který ale zatím nepodporuje SSR [38]. Poslední možností jsou externí knihovny, jako například *react-loadable*. Tato knihovna s minimem nastavení umožňuje rozdělovat kód buď podle cest, nebo přímo komponent a její použití je velmi jednoduché, viz ukázka kódu 4.7

```
// standardní import komponenty
import HomePage from './homepage';

const LoadableHopePage = Loadable({
    // import funkce
    loader: () => import('./homepage'),
    // komponenta zobrazená místo té načítané v~průběhu načítaní
    loading: Loading,
});
```

Obrázek 4.7: Code splitting pomocí react-loadable

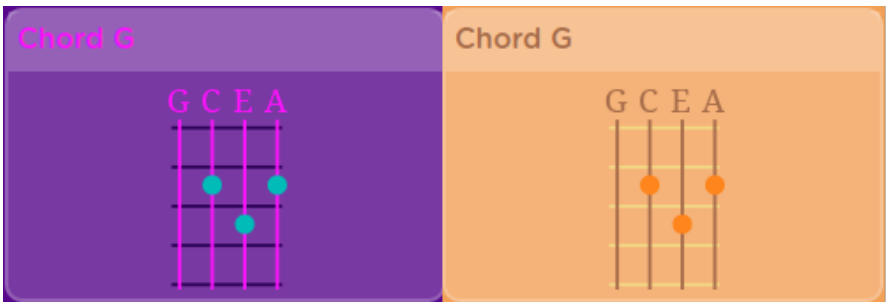
[3] Odstranění nepoužité části knihovny z výsledného balíčku

4.4.5 Ukládání písní

Ukládání textů písní a akordů je netriviální záležitost. Je potřeba ukládat texty písní přesně na řádky a k nim akordy na správné pozice. Tento problém je vyřešen tak, že píseň má text uložený ve speciálním formátu. Tento formát je pole řádků a každý řádek se skládá z textu a pole objektů, které reprezentují akord a jeho pozice od začátku řádku. Následně je text písně zobrazen neproporciálním písmem a akordy nad ním, také neproporciálním písmem. Tím jsou zaručené jednotné odstupy písmen a akordy jsou tedy na správných místech.

4.4.6 Vykreslování akordů

Komponenta akordů je další specifická funkce této aplikaci, proto musela být vytvořena na míru. Alternativní možností bylo aplikaci připojit na API [39], které vrací vykreslené akordy. Tento proces by práci sice zjednodušil, ale vygenerované obrázky jsou černobílé, což barevně nezapadá do konceptu aplikace a navíc nepodporují více barevných schémat. Z toho důvodu je tato komponenta implementovaná ručně. Také to je jediná komponenta, která pracuje přímo s grafikou. Vzhledem k jednoduchosti této grafiky a požadavku na co nejvyšší rychlost, tak je vykreslování implementováno přímo pomocí Canvas API a není pro to využita žádná knihovna. Tento přístup přinesl lehce konfigurovatelnou komponentu, která se škáluje bez ztrát a podporuje více barevných schémat, na rozdíl od rastrového obrázku z API. Výsledná komponenta je vidět ve dvou barevných provedeních na obrázku 4.8



Obrázek 4.8: Příklad komponenty akordu

Kapitola 5

Testování

Je několik různých typů testů, které se navzájem doplňují nebo překrývají. V rámci této práce byly vytvořeny automatické testy jednotkové a integrační, za pomoci nástroje Jest. V následujících sekcích jsou probírané jednotkové a integrační testy, další možnosti testování a finální zhodnocení aplikace a budoucí vývoj.

5.1 Jednotkové testy

Jednotkový test je typ testu, který testuje jednotlivé třídy, metody a funkce nezávisle na zbytku aplikace. Jak již bylo zmíněno v sekci Clean Architecture, rozdělení na vrstvy ulehčí testování. V rámci práce jsou takto testovány intereactory, které provádí změny nad entitami. Tím je zajištěno že všechny operace s entitami, jak na backendu, tak na frontendu jsou validní.

5.2 Integrační testy

Na rozdíl od jednotkového testu, který testuje jednotlivé nezávislé funkce čí metody, tak test integrační testuje více částí aplikace najednou. Zjistí se tím funkčnost celého systému. Bohužel, kvůli volbě verzi Yarn v2 nebylo možné správně nakofigurovat řádný integrační test. První pokus bylo rozšíření pro Storybook.js *storyshots*, který při spuštění vytvoří obrázky daných komponent a při dalších spuštěních je srovnává s předchozí verzí a případně notifikuje vývojáře. Další pokus byla konfigurace knihovny *Cypress*, která umožňuje vytvářet End To End (E2E) testovací scénáře, které simulují průchody aplikací. Oba pokusy selhaly kvůli špatné integraci knihovny *babel*, která transpiluje typescriptové soubory na soubory javascriptové a vytváří cesty k externím knihovnám, což byl práě důvod neúspěchu.

5.3 Ostatní testy

Samozřejmě existuje více druhů testů než pouze jednotkové a integrační. V rámci této práce by nás mohli zajímat hlavně testy akceptační a smoke. Akceptační test je v pravém slova smysl test klientem, kterému předáváme software. V rámci této práce by se za akceptační test dal považovat test uživateli, který ale nemohl být uskutečněn z důvodu absence dat. Smoke testy jsou testy nasazení. Po nasazení se spustí jednoduchá testovací sada, která má za úkol ověřit, zdali se aplikace správně nasadila a odpovídá.

5.4 Budoucí vývoj

Aplikace je nyní ve fázi funkčního prototypu, jsou implementovány všechny základní požadavky, ale stále je zde prostor na zlepšení. V následujících sekcích jsou rozebrány některé takové faktory.

5.4.1 Dopsání testů

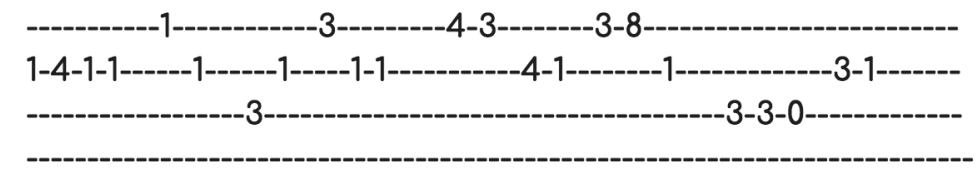
Asi největším nedostatkem aplikace je absence integračních testů, což koncového uživatele může, ale nemusí postihnout. Tento nedostatek půjde vyřešit ve chvíli, kdy vývojáři knihovny Babel plně dokončí integraci nové verze Yarn. Do té doby bohužel není možnost takové testy psát, protože všechny testovací frameworky, které umožňují více než jen jednotkové testy, na této knihovně závisí.

5.4.2 Data

To co nejvíce aplikaci schází v tuto dobu jsou funkční data. Tím jsou myšleny písně s texty, akordy atd. Tato data se dají jednoduše najít, ale pro osobní použití. Bohužel není žádné API poskytující seznam písní s textem a akordy pro ukulele. Tato data by se dala získat buď přepisem z jiných stránek či jejich scrapovaním, což by už mělo přesah do autorského zákona. Další možnost je kontaktovat přímo vydavatele a uzavřít spolupráci, která avšak nejspíš bude velmi finančně náročná. Poslední možnost by byla uzavřít spolupráci s již existujícími aplikacemi a získat data od nich.

5.4.3 Úpravy aplikace

Hlavním neduhem stávající aplikace je hlavně design. Aplikace je sice ve dvou barevných schématech, ty však nejsou zcela vyladěné. Samotná úprava barev a rozložení nebude náročná, díky dělení na komponenty a použití globálně dostupných barevných schémat. Zároveň je možné v budoucnu přidat další barevná schémata na vrch dvou stávajících.



Obrázek 5.1: Noty pro vybrnkávání

5.4.4 Překlad

Důležitou součástí aplikace, která míří na mezinárodní trh, tak je co největší jazyková dostupnost. Další krokem úpravy aplikace by tedy mělo být překlad do dalších jazyků, pro přilákání co největší skupiny uživatelů.

5.4.5 Podpora vybrnkávání

Vybrnkávání je styl hraní na ukulele, kde se nehraje přejížděním přes všechny struny, ale o některé se brnká. Tento styl má úplně jiný styl zobrazení. Jsou zobrazeny jednotlivé struny a k nim čísla pražců. Tato čísla jsou rozdělena do sloupců, které odpovídají době. Příklad takové tabulatury je na obrázku 5.1.

5.4.6 Nastavení ladění

Tato funkcionalita je téměř implementována. Komponenta vykreslovaného akordu dostává na vstupu mimo akordu k vykreslení i aktuální ladění, které je vždy nastaveno na *gcea*. Rozšíření by tedy spočívalo v přidání možnosti uživateli si toto ladění změnit, případně uložit pro příští návštěvy.

Kapitola 6

Kontinuální integrace a nasazení

Kontinuální integrace, dodání a nasazení jsou přístupy k automatizaci jednotlivých procesů, které jsou mezi vývojem a nasazením aplikace. V rámci této práce si pouze nastíníme, co jednotlivé výrazy znamenají.

Kontinuální integrace se zabývá automatickou validací kódu pomocí automatizovaných testů. Tyto testy jsou spuštěny ve chvíli kdy programátor nahraje do systému správy verzí (SCM). Tyto automatické testy ověří funkčnost, správnost a další požadavky na aplikaci a následně zpětně informují vývojáře, resp. tým o výsledcích testů. V praxi se to běžně využívá před zapojením nové části kódu do celé aplikace. Výhodou tohoto přístupu je minimalizování chyb a zvýšení produktivity programátora. V této práci kontinuální integrace byla realizována pomocí *GitHub Actions*. Jelikož je celá aplikace open source a uložena na serveru github.com, tak má nárok na využití systému GitHub Actions, který právě umožňuje automatické spouštění testů. Konkrétní testy prováděné automaticky jsou jednotkové a následně smoke build storybooku, který pouze ověří, zdali lze sestavit, tedy zdali aplikace lze sestavit. [40, s. 7]

Kontinuální dodání a nasazení jsou pojmy velmi podobné. Kontinuální dodání vytvoří balíčky aplikace připravené k nasazení a kontinuální nasazení je rovnou i nasadí. Jak dodání, tak nasazení je spouštěno automaticky po úspěšném provedení kontinuální integrace, nebo jiných krocích. V praxi to může být manuální spuštění na konci sprintu nebo automaticky po zařazení kódu do hlavní větve. [40, s. 18] Při tvorbě této aplikace bylo využito obojího. Při každé změně k hlavní větvi jsou spuštěny dva nasazovací skripty. První zajišťuje automatické sestavení dokumentace a statické verze storybooku a nasazení na *GitHub Pages*. Tato dokumentace je veřejně dostupná a hostovaná na serveru GitHub[4]. Druhý sestavuje výslednou aplikaci, vytváří docker kontejnery a následně spustí aktualizaci na vzdáleném serveru. Docker kontejnery jsou vytvořeny dva, jeden pro backend a druhý pro frontend a jsou uloženy

[4] danbalarin.github.io/ukulele-learning-site/

na veřejném docker repozitáři[5,6]. Následně se skript připojí přes ssh k vzdálenému serveru, spustí jednoduchý skript, který zastaví běžící instance, smaže, nahraje nové a spustí je. Jako poslední krok je spuštěn jednoduchý smoke test 6.1, který ověří, zdali frontend odpovídá na portu 80, a backend odpovídá na portu 4000.

```
curl -sSL --max-time 5 -D - $URL:$PORT -o /dev/null
```

Obrázek 6.1: Smoke test ověřující funkčnost http serveru

[5] hub.docker.com/repository/docker/kenny11/uls-react
[6] hub.docker.com/repository/docker/kenny11/uls-nodejs

Závěr

Cílem práce bylo vytvořit aplikaci pro podporu výuky hry na ukulele v souladu s metodami softwarového inženýrství. Ve zkratce jsem uvedl, co je potřeba znát pro hraní na ukulele. Prozkoumal jsem již existující aplikace a zvážil jejich výhody a nevýhody. Dále jsem uvedl dostupné technologie a cílovou skupinu, pro kterou je aplikace tvořena. Na základě těchto poznatků jsem stanovil funkční a nefunkční požadavky, ze kterých jsem vytvořil případy užití, wireframy a diagram tříd. Následně jsem zvolil technologie, které jsem využil k vytvoření aplikace. Popsal jsem metodiky použité k návrhu architektury aplikace a určil logické členění částí aplikace. Na základě návrhu jsem implementoval prototyp aplikace a automatické testy aplikace. Poté jsem vytvořil testovací a nasazovací skripty, které pracují zcela automaticky.

Prototyp aplikace je plně funkční, umožňuje vyhledávat akordy, strumming patterny, písně i autory. Uživatel má možnost se přihlásit a tím zpřístupnit nastavování písní jako oblíbené. Moderátor může přidávat, či upravovat písně a autory a administrátor může měnit role vytvořených účtů. Aplikace je dostupná v anglickém jazyce.

Aplikaci lze rozšířit o další funkcionality, třeba vybrnkávání, či nastavování vlastního ladění ukulele. Vhodné by bylo aplikaci rozšířit o další jazyky či vylepšit grafické pojetí. Dobrým nápadem je z responzivní aplikace udělat progresivní webovou aplikaci.

Hlavní přínos práce spočívá v přehlednosti a jednoduchosti aplikace, z čehož benefitují hlavně uživatelé méně zkušení s počítačem. Zvolené technologie umožnily rychle a efektivně vytvořit aplikaci, kterou je možné jednoduše rozšířit, či upravit. Jediná nevýhoda zvolených technologií je špatná kompatibilita, což znemožnilo integrační testování. Tento problém by však měl v budoucnu s rostoucí komunitou vymizet a aplikaci tedy bude možné o dané testy rozšířit a tím zajistit konzistenci kvality.

Bibliografie

1. SUTTON, Christopher. *The 5 Easiest Instruments Perfect for Adult Learners* [online]. 2016 [cit. 2020-05-22]. Dostupné z: `https://takelessons.com/blog/easiest-instrument-for-adults`.
2. RICHARDSON, Ged. *Top 5 Easiest Instrument to Learn for Adults & Children* [online]. 2019 [cit. 2020-05-22]. Dostupné z: `https://zingintruments.com/easiest-instrument-to-learn/`.
3. ARLOW, Jim; NEUSTADT, Ila. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Computer Press, 2007. ISBN 8025115038.
4. VYSLOUŽIL, Jiří. *Hudební slovník pro každého*. Lipa–A.J. Rychlík, 1995. ISBN 9788090119901.
5. ŠÁREK, Ondřej. *Škola hry na ukulele*. G + W, 2008. ISBN 9790706509709.
6. GUITARS, Cordoba. *Ukulele* [online]. 2020 [cit. 2020-05-23]. Dostupné z: `https://www.cordobaguitars.com/ukuleles/up100-ukulele-pack/`.
7. PUCHMAYR, Jakob. *Complete Ukulele Beginner Course* [online] [cit. 2020-05-22]. Dostupné z: `https://www.udemy.com/course/complete-ukulele-beginner-course/`.
8. GRUBBE, Lasse. *Online metronome* [online]. 2019 [cit. 2020-05-22]. Dostupné z: `https://www.musicca.com/metronome`.
9. NIEDT, Douglas. *FREE ONLINE METRONOME* [online]. 2019 [cit. 2020-05-22]. Dostupné z: `https://douglasniedt.com/metronomeonline.html`.
10. NETSCAPE COMMUNICATIONS CORPORATION. *Press Release* [online]. 1995 [cit. 2020-05-22]. Dostupné z: `https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html`.

11. STACK EXCHANGE, INC. *Stack Overflow Developer Survey 2019* [online]. 2019 [cit. 2020-05-22]. Dostupné z: https://insights.stackoverflow.com/survey/2019%5C#most-popular-technologies.

12. JETBRAINS S.R.O. *Demographics & Methodology 2019 - The state of Developer Ecosystem in 2019 Infographic* [online]. 2019 [cit. 2020-05-22]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2019/demographics/.

13. FACEBOOK, INC. *React* [online]. 2013 [cit. 2020-05-22]. Dostupné z: https://reactjs.org/.

14. GOOGLE LLC. *Angular* [online]. 2016 [cit. 2020-05-22]. Dostupné z: https://angular.io/.

15. YOU, Evan. *Vue.js* [online]. 2014 [cit. 2020-05-22]. Dostupné z: https://vuejs.org/.

16. YOU, Evan. *Comparison with Other Frameworks - Vue.js* [online]. 2014 [cit. 2020-05-22]. Dostupné z: https://vuejs.org/v2/guide/comparison.html#React.

17. POTTER, John. *NPM Trends: Compare NPM package downloads* [online]. 2020 [cit. 2020-05-11]. Dostupné z: https://www.npmtrends.com/@angular/core-vs-react-vs-vue.

18. RITCHIE, Rene. App Store Year Zero: Unsweet web apps and unsigned code drove iPhone to an SDK [online]. 2018 [cit. 2020-05-22]. Dostupné z: https://www.imore.com/history-app-store-year-zero.

19. FACEBOOK, INC. *React Native* [online]. 2015 [cit. 2020-05-22]. Dostupné z: https://reactnative.dev/.

20. SEPULVEDA, Christian. Share Code between React and React Native Apps [online]. 2017 [cit. 2020-05-22]. Dostupné z: https://hackernoon.com/code-reuse-using-higher-order-hoc-and-stateless-functional-components-in-react-and-react-native-6eeb503c665.

21. GOOGLE LLC. *Lighthouse* [online]. 2019 [cit. 2020-05-22]. Dostupné z: https://developers.google.com/web/tools/lighthouse.

22. FIGMA, INC. *Figma* [online]. 2019 [cit. 2020-05-22]. Dostupné z: https://www.figma.com/.

23. METEOR DEVELOPMENT GROUP, INC. *Apollo GraphQL* [online]. 2019 [cit. 2020-05-22]. Dostupné z: https://www.apollographql.com/.

24. EXPRESS CONTRIBUTORS. *Express* [online]. 2017 [cit. 2020-05-22]. Dostupné z: https://expressjs.com/.

25. BRITO, Gleison; VALENTE, Marco Tulio. *REST vs GraphQL: A Controlled Experiment*. 2020. Dostupné z arXiv: 2003.04761 [cs.SE].

26. PORCELLO, Eve; BANKS, Alex. *Learning GrapQL: declarative data fetching for modern web apps*. O'reilly, 2018. ISBN 9781492030713.
27. MONGODB, INC. *MongoDB* [online]. 2019 [cit. 2020-05-22]. Dostupné z: `https://www.mongodb.com/`.
28. AUTOMATTIC, INC. *Mongoose ODM* [online]. 2011 [cit. 2020-05-22]. Dostupné z: `https://mongoosejs.com/`.
29. AUTOMATTIC, INC. *Mongoose: Schemas* [online]. 2020 [cit. 2020-05-22]. Dostupné z: `https://mongoosejs.com/docs/guide.html`.
30. FACEBOOK, INC. *Yarn* [online]. 2019 [cit. 2019-09-11]. Dostupné z: `https://yarnpkg.com/`.
31. LERNA CONTRIBUTORS. *Lerna* [online]. 2015 [cit. 2020-05-25]. Dostupné z: `https://lerna.js.org/`.
32. JOYENT, INC. Node Resolution Algorithm. In: *Node.js Documentation* [online]. 2016 [cit. 2020-03-12]. Dostupné z: `https://nodejs.org/api/modules.html%5C#modules_all_together`.
33. FACEBOOK, INC. *Jest* [software]. 2017. Verze 25.5.4 [cit. 2020-05-13]. Dostupné z: `https://jestjs.io/`.
34. BEVACQUA, Nicolas. *Mastering Modular JavaScript*. O'Reilly Media, 2018.
35. MARTIN, Robert Cecil. *Clean Architecture : a craftsman's guide to software structure and design*. Prentice Hall, 2018.
36. MARTIN, Robert Cecil. *Clean Coder Blog* [online]. 2019 [cit. 2019-11-19]. Dostupné z: `https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html`.
37. SHEHET, Gregory. *CSS IN JS Benchmarks* [online]. 2020 [cit. 2020-04-30]. Dostupné z: `https://github.com/A-gambit/CSS-IN-JS-Benchmarks`.
38. FACEBOOK, INC. *Code-Splitting – React* [online]. 2018 [cit. 2020-04-30]. Dostupné z: `https://reactjs.org/docs/code-splitting.html`.
39. *Ukulele Chords API* [online]. 2011 [cit. 2020-05-22]. Dostupné z: `https://ukulele-chords.com/api`.
40. ROSSEL, Sander. *Continuous Integration, Delivery, and Deployment*. Packt Publishing, 2017. ISBN 9781787286610.

Příloha A

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `build` ..... adresář se spustitelnou formou implementace
  - `react` ..... adresář se spustitelným frontend serverem
  - `nodejs` ..... adresář se spustitelným backend serverem
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF