FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

| | |
|---|---|
| **Název:** | Poopeek - klientská část aplikace |
| **Student:** | Dominik Fryč |
| **Vedoucí:** | Ing. Filip Glazar |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2020/21 |

## Pokyny pro vypracování

Webový portál Poopeek propojuje kapely a pořadatele různorodých kulturních akcí. Cílem této práce je navrhnout a implementovat klientskou část pro tuto webovou aplikaci pomocí frameworku Angular. Aplikace umožňuje vytvořit profil pro kapelu nebo pořadatele a tyto dva subjekty spolu propojit formou poptávky a nabídky. Implementace bude využívat serverovou část aplikace, která je implementovaná jako REST API.

1. Analyzujte funkcionalitu backendu a jeho návrh.
2. Navrhněte, alespoň část uživatelského rozhraní.
3. Implementujte prototyp aplikace, který bude obsahovat vámi navržené části aplikace.
4. Navrhněte a realizujte continuous integration pro nasazení aplikace.
5. Spusťte produkční verzi aplikace.
6. Na základě údajů z reálného běhu aplikace navrhněte, či rovnou implementujte potřebné změny.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 3. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

# Poopeek – klientská část aplikace

***Dominik Fryč***

Katedra softwarového inženýrství
Vedoucí práce: Ing. Filip Glazar

4. června 2020

# Poděkování

Děkuji Ing. Filipu Glazarovi za vedení a konzultace při realizaci této práce. Dále děkuji spoluzakladatelům firmy Poopeek za motivaci a pomoc při návrhu systému. Také bych rád poděkoval rodině za podporu během psaní práce.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). 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 zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. června 2020 . . . . . .

### Odkaz na tuto práci

Fryč, Dominik. *Poopeek – klientská část aplikace*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

# Abstrakt

Bakalářská práce představuje návrh a implementaci klientské části webové aplikace, vytvořené pro start-up Poopeek. Aplikace slouží k propojení kapel a pořadatelů, kteří zde mohou vystavovat nabídky a vzájemně vyjednávat o podmínkách. Analýza existujícího serverového řešení, na které se aplikace napojuje, hovoří o použitých technologiích a základních entitách. Návrh uživatelského rozhraní a grafického designu se skládá z uživatelského výzkumu, popisu aplikace pomocí wireframů a následnou tvorbou prototypů. V rámci implementace je představen použitý framework Angular včetně jeho součástí, vytvořené komponenty a komunikace mezi nimi. Po sestavení aplikace následuje automatická integrace a dodatečné optimalizace. Výsledkem práce je produkční verze aplikace včetně nasazení, která zajišťuje komplexní systém pro booking kapel a pořadatelů.

**Klíčová slova** hudební booking, kapela, pořadatel, webová aplikace, front-end, návrh uživatelského rozhraní, Angular

# Abstract

The bachelor thesis presents the design and implementation of the front-end web application created for the start-up Poopeek. The goal of the app is connecting bands and organizers, who can create offers and negotiate conditions with each other. The analysis of the existing server solution, which application connects to, speaks of used technologies and basic entities. The design of the user interface and graphic design consists of user research, description of the application using wireframes, and subsequent creation of prototypes. Within the implementation, the thesis presents the Angular framework and its parts, created components, and communication between them. After the application build, automatic integration and additional optimizations follow. The result of the work is a production version of the application, including deployment, which provides a complex system for booking bands and organizers.

**Keywords** music booking, band, organizer, web application, front-end, user interface design, Angular

# Obsah

# Seznam obrázků

# Seznam výpisů kódu

# Úvod

Digitální technologie dnes ovládají celý svět. Většina služeb je poskytovaná přes internet, každá má svoji aplikaci a jiným způsobem už ji uživatelé ani nevyužívají. Pomocí aplikace si objednáváme jídlo, ubytování, půjčujeme dopravní prostředky nebo nakupujeme věci od cizích lidí. Přesto však stále existují odvětví, ve kterých tyto moderní způsoby neplatí. Jedním z nich je booking v hudebním průmyslu.

Drtivá většina pořadatelů rezervuje kapely přes e-mail na základě svých kontaktů. To způsobuje spoustu problémů a nepříjemností. Jedním z nich může být komunikační šum. Často se stává, že v jednu chvíli není jasné, na jakých podmínkách je pořadatel s kapelou domluvený, jelikož se jejich konverzace vleče už přes desítky zpráv v několika vláknech. Dalším problémem může být uzavřenost vůči vlastním kontaktům, kdy pořadatel jen těžce objevuje nové kapely, a tak na jeho akcích hraje pořád stejná sestava. A právě to jsme se rozhodli s dvěma kamarády změnit.

Založili jsme start-up Poopeek, jehož cílem je vybudovat komplexní softwarové řešení pro propojování kapel a pořadatelů. Poskytne možnost vystavování nabídek, domlouvání podmínek vystoupení a pomůže i s následnou administrativou. Cílovou skupinou jsou hudebníci, pořadatelé různorodých akcí, agentury i kulturní služby měst. Na trhu existují různé databáze kapel, ale žádný nástroj, který by zjednodušil celý proces bookingu. Zároveň bude přinášet edukační přínos pro začínající interprety, kteří ještě nejsou v hudebním světě zběhlí a třeba neví, které dokumenty organizátor akce potřebuje.

Uživatel se bude moci přihlásit a vytvořit profil kapely či pořadatele. Bude moct vystavit nabídku nebo listovat v nabídkách opačné strany. Jakmile dojde k reakci, otevře se vyjednávání o podmínkách vystoupení. Formou chatu se domluví podrobnosti akce. V momentě, kdy obě strany přijmou stanovené podmínky, vznikne dohoda a uživatelé obdrží potřebné dokumenty. V aplikaci budou integrované notifikace a peněženka, ve které uživatel nalezne provedené transakce a možnost dobít kredity na svůj účet.

Serverová část aplikace je již hotová a mým úkolem je nyní navrhnout a implementovat část klientskou. Musím navrhnout uživatelské rozhraní tak, aby se správně zobrazovalo na mobilních i desktopových zařízeních, dodržovalo firemní identitu a umožňovalo uživateli jednoduše ovládat celou aplikaci. Dalším cílem je implementovat prototyp za pomoci znovupoužitelných komponent, aby bylo snadné aplikaci upravovat a rozšiřovat. V neposlední řadě mám za úkol realizovat produkční verzi včetně jejího nasazení.

V první části práce se zaměřuji na analýzu serverového řešení. Popisuji architekturu rozhraní a použité technologie pro server a databázi. Představuji základní entity, vztahy mezi nimi a metody komunikace, které jsem následně využil v implementaci. Z výsledků analýzy jsem vytvořil návrh jednotlivých částí aplikace a jejich funkcí a navrhl grafický design šablony webu. Dále řeším samotnou implementaci prototypu. Zejména použité technologie, jednotlivé moduly a komponenty včetně komunikace se serverem a následnou prací s daty. Po dokončení prototypu přecházím na tvorbu produkční verze. Zaměřuji se na optimalizaci, sestavení aplikace a automatické integrace. Nakonec zmiňuji použité nástroje pro odchytávání chyb a sběr dat pro analytiky.

Kapitola 1

# Analýza serverové části

Aplikace je rozdělena na dva celky. První je část serverová (dále označovaná jako back-end), která je již hotová a je připravena k použití. Druhá část je klientská (dále označovaná jako front-end), tedy webová aplikace pro obsluhu serveru, která je předmětem této práce. V první kapitole se věnuji analýze serverové části.

## 1.1 Technologie serveru

Celá aplikace je postavená na základě tzv. MEAN stack [1]. Jedná se o open-source sadu softwaru pro tvorbu moderních webových aplikací. Název je odvozen od jednotlivých komponent, tedy MongoDB, Express, Angular a Node.js. Síla vybraného řešení je v tom, že klientská i serverová část je napsána ve stejném jazyce a tím je JavaScript.

Back-end tvoří samostatný server, jehož cílem je poskytovat a zpracovávat data od klienta. Komunikace mezi klientem a serverem probíhá pomocí rozhraní API, které umožňuje snadnější vývoj více klientských aplikací. Webové a mobilní aplikace pak komunikují se serverem právě přes jedno společné rozhraní. Existuje více druhů architektur, ovšem mezi nejznámější patří REST a GraphQL. Back-end naší aplikace používá právě REST.

### REST API

REST je sada principů pro návrh rozhraní API s využitím protokolu HTTP. Mezi jeho klíčové vlastnosti patří architektura klient-server, bezstavovost, efektivní *kešování* a systém, rozdělený do více vrstev. Systémy dodržující tyto principy se označují jako RESTful API [2].

Podle [3] je rozhraní definováno jako kolekce URI adres (často označováno jako endpointy), na které klienti posílají své požadavky. HTTP požadavek je tvořen hlavičkou, která o něm nese základní informace, a tělem, ve kterém se mohou nacházet data. Právě díky metodě a adrese cíle, které nalezneme

v hlavičce, server pozná, kterou operaci má vykonat. Mezi základní metody, včetně jejich běžného chování, patří:

**GET** – získání konkrétního zdroje nebo kolekce dat,

**POST** – vytvoření nového zdroje,

**PUT** – nahrazení konkrétního zdroje,

**DELETE** – smazání konkrétního zdroje.

Server požadavek zpracuje a pošle klientovi zpět odpověď. Součástí odpovědi mohou být data nebo chyba, která při zpracování nastala. Odpověď je definovaná číselným návratovým kódem, který označuje její stav. Číselné skupiny návratových kódů se dělí na informační, úspěšné, přesměrovávací, chyby u klienta nebo chyby na serveru.

Endpointy našeho rozhraní REST mají po adrese serveru prefix `/api/v0`, přičemž druhá část označuje verzi API. Prefix nebudu nadále u jednotlivých endpointů uvádět. Vzhledem k tomu, že jsou obě aplikace napsané v jazyce JavaScript, rozhodli jsme se, aby server ve výchozím stavu vracel odpověď ve formátu JSON, což usnadní práci s daty na obou stranách.

### Node.js a Express

Back-end je napsaný v JavaScriptu [4]. Jedná se o skriptovací jazyk, který se používá na většině webových stránek. Je postaven na standardu ECMAScript, který definuje jeho funkčnost. JavaScript je dynamický, multiplatformní, běží na jednom vlákně a podporuje objektově orientované, imperativní a deklarativní styly. Nemusí však běžet pouze v prohlížeči, díky prostředím jako je Node.js je možné ho provozovat jako samostatný server.

Node.js [5] je open-source *runtime* prostředí pro jazyk JavaScript. Běží na jádře V8 od společnosti Google, které je využito například v prohlížeči Chrome [6]. Jedná se o asynchronní prostředí, které je řízeno akcemi. Jelikož neběží na více vláknech, není možné, aby došlo k časové chybě, kdy se novější požadavek zpracuje dříve než předchozí. Procesy nic neblokuje, a tak je možné aplikaci jednoduše škálovat.

Pro vybudování API byl zvolen Express [7]. Jedná se o populární webový framework pro Node.js, který umožňuje vytvořit solidní HTTP server. Sám o sobě je ale minimalistický, neobsahuje žádné nástroje k propojení s databází, autorizaci nebo šablonovací systém. Tyto úlohy umožňuje třetím stranám zprostředkovat pomocí *middlewaru*, který lze použít během zpracování požadavku a je možné ho libovolně napojovat. Díky tomu se Express skvěle hodí pro webové API.

### MongoDB

Pro ukládání dat na serveru běží NoSQL databáze [8]. Ta umožňuje uložit specifické datové modely s flexibilním schematem. Nabízí vysoký výkon společně s možností škálování a oproti běžným relačním databázím neukládá data do tabulky, ale do vybraných kolekcí. Nejčastějšími typy jsou klíč-hodnota, databáze dokumentová, sloupcová nebo grafová.

Náš server využívá dokumentovou NoSQL databázi MongoDB [9]. Data ukládá do dokumentů s JSON strukturou. Atributy se mezi dokumenty mohou měnit a nemusí dodržovat stejnou strukturu. Dokumenty je možné vnořovat nebo odkazovat na jejich reference pomocí unikátního hexadecimálního klíče *ObjectId*. Server k propojení s databází využívá knihovnu Mongoose, která slouží k mapování databázových modelů na objektovou reprezentaci v kódu.

### Autentizace a bezpečnost

Komunikace mezi klientem a serverem je šifrovaná pomocí protokolu HTTPS. Zabrání se tak odposlechu posílaných dat, zejména přihlašovacích a osobních údajů. Uživatelé se přihlašují standardně pomocí e-mailu a hesla. Do databáze se neukládá heslo samotné, ale pouze jeho *hash* vytvořený *hashovací* funkcí `bcrypt`. Aby dvě stejná hesla v databázi neměla totožný hash, před vytvořením se heslo zkombinuje s tzv. „solí“, tedy náhodně vygenerovaným řetězcem. V rámci přihlášení server nejprve vytvoří hash kombinace přijatého hesla od klienta a soli z databáze a následně porovná, jestli se shoduje s hashem z databáze. Pokud jsou stejné, přihlášení bylo úspěšné.

Aby bylo možné udržovat relaci přihlášeného uživatele, server používá otevřený standard JWT [10]. V podstatě jde o způsob pro zabezpečený přenos dat mezi dvěma stranami pomocí zašifrovaného objektu JSON. Tyto informace mohou být verifikovány, jelikož jsou digitálně podepsány pomocí tajného klíče. Token tvoří tři části oddělené tečkou:

**Header** obsahující většinou typ tokenu a použitý algoritmus,

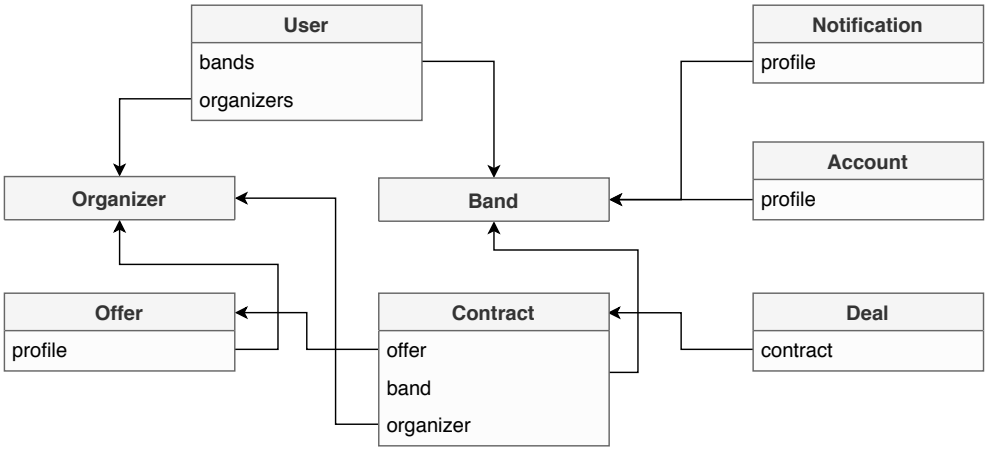
**Payload** neboli přenášená data o entitě (nejčastěji identifikátor uživatele),

**Signature** představující podpis tokenu vytvořený z předchozích dvou zašifrovaných částí, tajného klíče a zvoleného algoritmu z hlavičky.

Po úspěšném přihlášení klient od serveru obdrží JWT token. Ten uloží do prohlížeče po dobu platnosti tokenu. Při HTTP požadavku na endpoint vyžadující autentizaci či autorizaci klient připojí token do speciální hlavičky `x-access-token`. Server po přijetí ověří, že se v hlavičce token nachází a je validní. Na základě toho zjistí, o kterého uživatele se jedná a zda má oprávnění přistoupit k požadovaným datům. Server je bezestavový, takže je potřeba token přikládat ke každému požadavku. Po vypršení platnosti tokenu je nutné ho obnovit opětovným přihlášením.

## 1.2 Základní entity a vztahy

Každý uživatel může mít více profilů typu kapela nebo pořadatel. Jednotlivé profily mohou vystavovat nabídky, na které reagují opačné profily. Reakcí na nabídku vznikne vyjednávání a pokud se strany mezi sebou domluví, vznikne dohoda. Notifikace a účty se vážou k profilům. K lepší představě entit a vztahů poslouží obrázek 1.1:




Obrázek 1.1: Diagram základních entit a vztahů mezi nimi

### User

Základní entitou aplikace je uživatel. Slouží zejména pro autentizaci a správu jednotlivých profilů. Entita reprezentuje reálného uživatele, obsahuje tedy atributy jako jméno, příjmení, e-mail a další osobní údaje. Ukládá identifikátory profilů, které uživatel spravuje. Dále nese informaci o tom, kterým profilem je v dané chvíli uživatel přihlášen, protože může v jeden okamžik obsluhovat pouze jeden profil.

### Band a Organizer

Band (kapela) a Organizer (pořadatel) jsou dva druhy profilů, které může uživatel spravovat. Do budoucnosti se počítá s možným rozšířením. Oba druhy mají několik společných atributů. Mezi veřejně dostupné atributy, které mohou vidět ostatní uživatelé, patří název, popis, fotografie, sociální sítě, datum založení a další obecné údaje. Údaje typu kontaktní osoba, fakturační údaje nebo dokumenty nejsou veřejné, zobrazí se až při vzniku dohody. Pořadatel je definován oblastí působení, typem a komunikačními jazyky. Dokumenty tvoří

logo nebo jiná grafika a stageplan pódia. Kapela má ještě hudební žánry, počet členů a původ. V dokumentech má navíc oproti pořadateli inputlist a playlist.

### Offer

Jedná se o entitu představující nabídku kapely nebo pořadatele. Vždy je určený termín akce a délka vystoupení. Kapela vybírá časové okno v rámci jednoho dne, ve kterém nabízí své vystoupení, zatímco pořadatel určuje konkrétní datum a čas s délkou vystoupení. Autor může zvolit preference, jaké profily pro svojí nabídku hledá. Kapely si mohou vybrat typ pořadatele a oblast, kde chtějí hrát. Pořadatelé vybírají typy kapel a hudební žánry. Uvádějí také konkrétní místo konání akce. Místo je specifikované názvem, typem a adresou. Nabídka má určenou dobu platnosti, po kterou se bude zobrazovat ostatním uživatelům. Obsahuje také odkaz na profil, ke kterému se váže a stav, ve kterém se nachází. Ten může být koncept, zveřejnění nebo archivace.

Důležitou součástí nabídky jsou také požadavky na pořadatele (ze strany kapely) nebo podmínky pro kapelu (ze strany pořadatele). Jedná se o seznam atributů, které si autor libovolně zvolí a později se mohou měnit v rámci vyjednávání. Patří sem například honorář a cestovní náklady, počet jídel, postelí, parkovacích míst či jiné technické požadavky na zvukaře nebo osvětlení.

### Contract

Contract slouží k vyjednávání mezi kapelou a pořadatelem. Vzniká v momentě, kdy cizí profil zareaguje na nabídku. Obsahuje tedy odkaz na původní nabídku, kapelu a pořadatele, kteří jsou součástí vyjednávání. Proces vyjednávání probíhá formou chatu, avšak strany mezi sebou nekomunikují formou textových zpráv, ale pomocí definovaných podmínek. Původní podmínky jsou převzaté z nabídky, účastníci pak mohou upravovat hodnoty nebo je mazat a přidávat nové. Vyjednávání je možné kdykoliv ukončit nebo s danými podmínkami souhlasit. Poté, co podmínky odsouhlasí obě strany, vznikne dohoda.

### Deal

Úspěšným stavem vyjednávání označujeme dohodu. Skládá se z odkazů na zúčastněné profily, původní nabídku a také vyjednávání, ze kterého dohoda vzešla. Dohoda zobrazuje dohodnuté podmínky vystoupení a základní informace o místě, kde se akce koná. Dále se navzájem zobrazí kontaktní údaje a dokumenty, které jsou pro pořadatele důležité. Strany se v tuto chvíli mohou kontaktovat mimo náš systém z důvodu nenadálých změn. Back-end také nabízí možnost vygenerovat a stáhnout právně ověřený návrh smlouvy mezi kapelou a pořadatelem ve formátu PDF. To, jestli strany smlouvu podepíší, už není součástí aplikace.

### Account

Každý profil má přiřazen svůj kreditový účet. Účet obsahuje odkaz na profil majitele, aktuální zůstatek a disponibilní zůstatek kreditů. Kredity jsou považovány jako virtuální metoda úhrady jednotlivých zpoplatněných akcí. V tuto chvíli se jedná pouze o poplatek za vystavení nabídky a servisní poplatek za zprostředkování dohody. V účtu se také nachází seznam provedených transakcí. Každá z nich je definována časem, popisem, změnou počtu kreditů a stavem, ve kterém se aktuálně nachází. Transakce může čekat na vyřízení, být dokončena, nezdařena, vrácena, zrušena nebo blokována.

### Notification

Notifikace jsou taktéž důležitou součástí systému. Informují o změnách stavu jednotlivých entit a podporují dynamiku celé aplikace. Příkladem může být notifikace ohledně reakce na nabídku nebo změně ve vyjednávání. Každá notifikace obsahuje profil, ke kterému je přiřazena, typ, ke které entitě se váže, a viditelnost, zda si ji uživatel zobrazil. Jejím obsahem je titulek, popis, libovolná data a čas, kdy byla odeslána. Front-end podle typu a dat notifikace pozná, kam má po kliknutí uživatele v rámci aplikace přesměrovat.

Kapitola 2

# Návrh uživatelského rozhraní

V této kapitole se zabývám postupem tvorby grafického uživatelského rozhraní. Nejprve zmiňuji zásadní přístupy a principy, zkoumám cílové skupiny včetně person a zpracovávám požadavky do uživatelských scénářů. Na mapě aplikace ukazuji strukturu stránek, které následně podrobně popisuji. Určuji jejich cíle, informační přínos a konverzní akce. K dispozici přikládám také náhledy vytvořených wireframů a ukázky výsledného designu.

## 2.1 Úvodní analýza

Grafické uživatelské rozhraní neboli GUI je systém interaktivních prvků umožňující uživateli ovládat software [11]. V rámci webové aplikace uživatelské rozhraní představují zejména textové a grafické ovládací prvky a jejich rozmístění. Cílem je zpracovávat vstupy uživatele a zároveň prezentovat data aplikace pomocí grafické podoby. Výhodou tohoto přístupu je oddělení prezentační a aplikační vrstvy, což umožňuje jednodušší vývoj aplikací pro více různých platforem (např. pro mobilní zařízení).

Uživatel komunikuje přímo s grafickým rozhraním, které ovlivňuje, jak se mu bude aplikace používat a jestli mu umožní dosáhnout akce, kterou chce pomocí aplikace udělat. Zkoumáním chování uživatele a navrhování změn používání aplikace se zabývá metodika *User Experience Design*. Snaží se o správné navrhnutí GUI s ohledem na design, který slouží jako vizuální forma funkcí aplikace, ale také na samotnou použitelnost. Úkolem je odstraňování nedostatků v ovládání aplikace a postupné vylepšování přístupnosti na základě uživatelské zpětné vazby.

### Nelsonova heuristika

Při návrhu uživatelského rozhraní aplikace se doporučuje řídit se heuristickou analýzou podle Jakoba Nielsena [12]. Vychází z poznatků dlouhodobého pozorování chování uživatelů v aplikacích. Jedná se o soubor desíti pravidel,

které pomáhají pochopit, jak jsou uživatelé zvyklí s aplikacemi pracovat a co od nich očekávají. Jejich použitím můžeme dosáhnout lepší použitelnosti a intuitivního ovládání aplikace. Patří sem:

1) **Visibility of system status** Systém by měl vždy viditelně informovat uživatele o aktuálním stavu aplikace. Například signalizovat odesílání dat formuláře nebo načítání nové stránky.

2) **Match between system and the real world** Aplikace by měla s uživatelem komunikovat lidským jazykem raději než systémovými pojmy. Držet se konvencí z reálného světa a dodržovat přirozené pořadí.

3) **User control and freedom** V případě, že uživatel omylem vyvolá nějakou nechtěnou akci, měl by mít vždy možnost z tohoto stavu jednoduše odejít. Zejména pomocí možností zpět nebo vpřed.

4) **Consistency and standards** Aplikace by měla být konzistentní se standardními ovládací prvky, aby uživatel nebyl zmaten a různé akce neplnily stejnou funkci.

5) **Error prevention** Systém by měl předcházet chybovým stavům a uživatele upozornit případným varováním ještě před uskutečněním akce. Příkladem může být upozornění na povinná nevyplněná pole formuláře.

6) **Recognition rather than recall** Zredukovat množství informací, které si uživatel musí pamatovat. Aplikace by měla zobrazovat objekty, akce a možnosti, aby byly pro uživatele viditelné nebo lehce dostupné.

7) **Flexibility and efficiency of use** Systém musí být jednoduše ovladatelný, zároveň by však měl nabízet možnosti pro pokročilé uživatele. Často prováděné akce by měly být zjednodušeny pro zvýšení efektivity.

8) **Aesthetic and minimalist design** Aplikace by měla zobrazovat minimum informací a voleb, které nejsou zásadní, aby byla práce snadnější a uživatel nemusel tolik přemýšlet.

9) **Help users diagnose and recover from errors** Chybové zprávy musí být psané srozumitelně bez použití chybových kódů. Měli by informovat, jaký problém se stal, a navrhnout možné řešení.

10) **Help and documentation** Přestože by mělo být možné systém používat bez dokumentace, nápověda by měla být přítomna nebo lehce dohledána. Instrukce by měly být psány krátce v jednotlivých krocích.

## 2.2 Uživatelský výzkum

Firma Poopeek s.r.o. je start-up specializovaný na vývoj komplexního softwarového řešení pro booking v hudením průmyslu. Podnikatelským záměrem je vybudovat službu pro interprety a pořadatele akcí, která jim usnadní hledání nabídek a následné domlouvání podmínek vystoupení. Cílem produktu je úspora času, zvýšení přehlednosti vyjednávání a edukace začínajících umělců.

Naše analýza konkurence ukázala, že na trhu není nástroj, který by řešil hudební booking od začátku do konce. Konkurenční aplikace končí u vystavování nabídek s kontaktem na majitele. Naší největší výhodou je standardizace procesu vyjednávání. Probíhá formou chatu, avšak místo textových zpráv si strany vyměňují definované podmínky. Díky tomu obě strany přesně ví, na jakých podmínkách se domlouvají.

### Cílová skupina zákazníků

Nejprve je potřeba segmentovat trh a určit, kdo je naším zákazníkem a jaké má skutečné potřeby. Cílíme na celou Českou republiku, díky tomu je aplikace dostupná v češtině. Naší cílovou skupinou jsou muži i ženy v rozmezí přibližně od 20 do 50 let. S vyšším věkem klesá motivace používat nové technologie a tedy šance stát se naším zákazníkem. Nezáleží na vzdělanosti v hudebním průmyslu, aplikace je navržena tak, aby byla srozumitelná i pro začátečníky.

První skupinou jsou interpreti. Patří sem kapely, zpěváci a další hudebníci. Cílíme hlavně na poloprofesionální hudebníky, kteří hrají pravidelně a za provizi. Kapel je však mnoho a většinou nemají kde hrát. Hledají nové kluby a události, kterých se mohou zúčastnit. Na to potřebují manažera, který jim bude nové akce hledat. Toho si však ne každá kapela může dovolit, a tak jeho funkci zastává jeden z členů. Naší motivací je tomuto člověku usnadnit hledání nových příležitostí a ušetřit čas strávený manažerstvím kapely.

Druhou skupinu tvoří pořadatelé a organizátoři hudebních akcí. Patří sem kluby a hospody, hudební agentury, nezávislý pořadatelé i kulturní služby měst a obcí. Tato skupina je menší, avšak míst, kde je možné vystoupit, je mnoho. Problémem pořadatelů bývá malá databáze kontaktů, což vede k střídání stejné sestavy kapel na pravidelných akcích. Naše aplikace pořadatelům umožňuje objevování nových kapel, jejich porovnání a filtrování. Zároveň usnadňuje vyjednávání a následnou administrativu s pořádáním akce. Vytvořená dohoda obsahuje veškeré potřebné dokumenty a kontakty druhé strany.

### Persony

Persony jsou fiktivní postavy představující konkrétní zákazníky z cílové skupiny [13]. Jedná se o typické uživatele, kteří mohou být relevantním a významným obrazem budoucích zákazníků. Modelují motivaci k používání aplikace a pochopení produktu. Pro přesnou představu by měla persona mít jméno,

věk a lokalitu, práci související s naším podnikání a potřeby, které může náš produkt splnit. Díky personám je snadnější zaměřit se na relevantní problémy uživatelů a najít jejich řešení. Vytvořil jsem proto tři klíčové persony:

**Petr** – Petrovi je 32 let a provozuje středně velký klub v Plzni. Kapely u něj hrají pravidelně každý týden. Petrovi chodí spousty e-mailů s nabídkami od kapel a neustále mu zvoní mobil. Stává se mu, že má nepořádek ve vyjednáváních a neví, na čem se s kterou kapelou domluvil.

**Matěj** – Matěj je student vysoké školy a hraje na kytaru v menší kapele pocházející kousek od Prahy. Kapela nemá manažera, o vše se stará právě Matěj. Propagace a veškerá administrativa okolo kapely mu zabírá spoustu času, raději by se věnoval muzice.

**Hana** – Hana žije v Ostravě, pracuje v kulturních službách města a je jí 46 let. Ročně pořádají tři větší hudební akce, kde se však poslední roky střídají stejné kapely. Hana by ráda oslovila nové kapely, ale nedaří se jí žádné najít. Ocenila by místo, kde by si mohla vybrat.

### Funkční požadavky

Na základě vypracovaných person jsou viditelné potřeby typických zákazníků. V tuto chvíli je možné navrhnout základní očekávané funkce, které by měla aplikace mít. Vycházím samozřejmě i z návrhu serverové části, kde jsou již samotné operace vypracované. Zaměřuji se však spíše na návrh uživatelského rozhraní, aby použití jednotlivých funkcí dávalo uživateli smysl. Patří sem i uživatelské scénáře, které popisují konkrétní potřeby uživatele končící konverzní akcí. Mezi hlavní požadavky řadím:

**Autentizace** – registrace a přihlášení uživatele, odhlášení a obnova hesla;

**Profily** – vytvoření kapely nebo pořadatele, možnost editace, zobrazení detailu, listu ostatních profilů, možnost přepínat mezi vlastními profily;

**Nabídky** – vytvoření a editace nabídky, zobrazení detailu, listu svých i cizích nabídek s filtrováním, možnost reakce na nabídku opačného profilu;

**Vyjednávání** – list svých vyjednávání, možnost přijmout nebo upravit podmínky, ukončit vyjednávání, načtení nových podmínek při změně;

**Dohody** – list svých uzavřených dohod, kontakty a dokumenty protistrany;

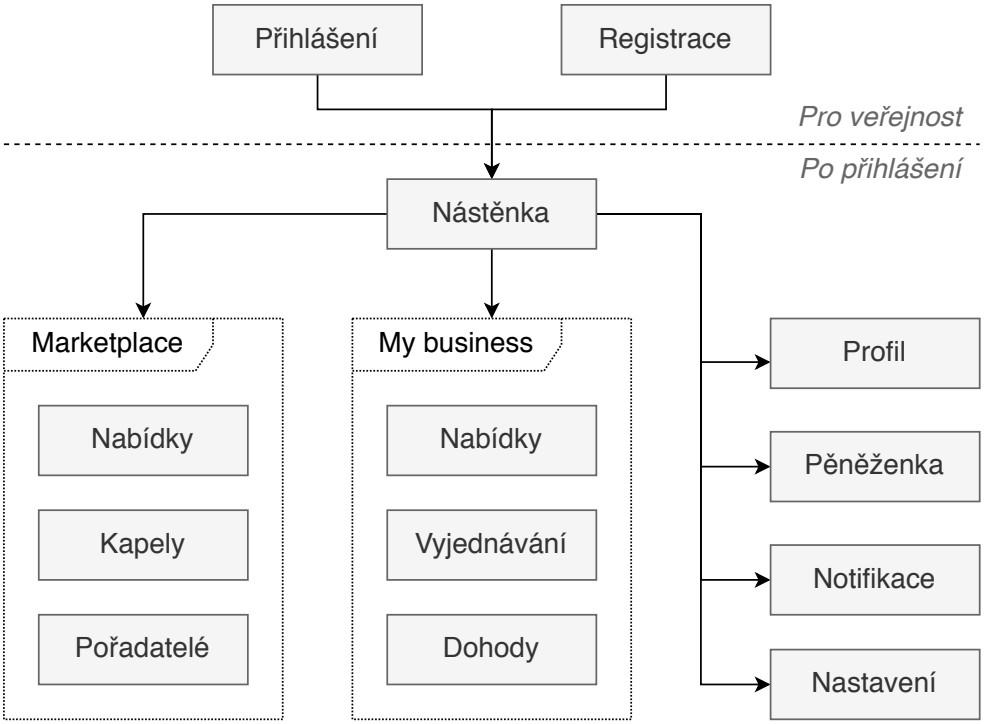
**Nastavení** – úprava dat uživatele, změna hesla, e-mailových preferencí;

**Peněženka** – zobrazení počtu kreditů, stavu předplatného a seznamu transakcí, možnost dobití kreditů, aktivace předplatného;

**Notifikace** – list nepřečtených i všech notifikací, možnost označit notifikaci jako přečtenou, automatické načtení nových notifikací.

## 2.3 Popis aplikace

Podle jednotlivých funkčních požadavků jsem navrhl mapu obsahu aplikace zobrazenou na obrázku 2.1. Stránky pro veřejnost tvoří přihlášení a registrace. Po přihlášení se uživatel dostane na nástěnku, která slouží jako rozcestník. Sekce *Marketplace* představuje tržiště a obsahuje skupinu stránek určených pro objevování a následné obchodování. Patří sem nabídky, kapely a pořadatelé. *My business* je sekcí pro správu vlastních nabídek, vyjednávání a dohod. Aplikace je dále tvořena stránkami profil, peněženka, notifikace a nastavení.




Obrázek 2.1: Mapa jednotlivých částí aplikace

Jednotlivé části aplikace popisuji pomocí *wireframů*. Jedná se o drátěné modely uživatelského rozhraní, které představují strukturu jednotlivých stránek aplikace [13]. Popisují základní rozvržení, jakým bude obsah uspořádán včetně ovládacích prvků. Cílem návrhu wireframů je také naznačit akce a vztahy mezi stránkami. Neobsahují konkrétní data, ale pouze výplňové texty a obrázky. Patří mezi *low-fidelity* (nízkoúrovňové) modely, které se snaží maximálně zjednodušovat a vynechávat detaily.

Podle Jana Řezáče [14] je důležité u každé stránky stanovit, jaký je její cíl, informační přínos, který chceme uživateli předat, a akce, která by se měla na stránce vykonat. Díky tomu podpoříme význam jednotlivých stránek a bu-

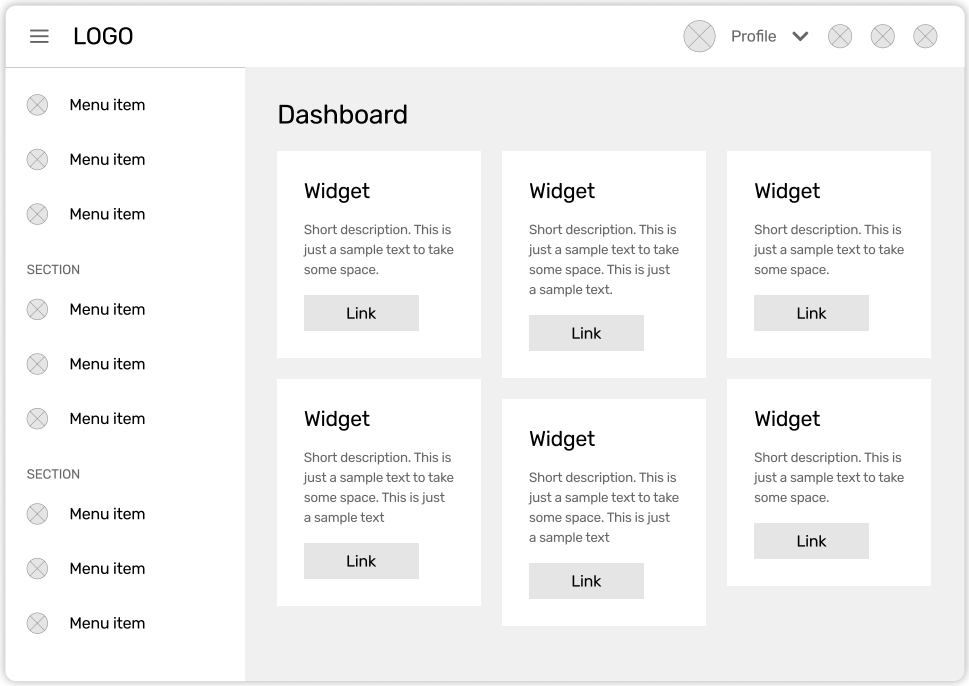
deme si jistější, že plní svůj úkol. Některé stránky mohou nabízet více různých akcí, jiné zase slouží pouze pro smysluplný přechod na další. V následující části popisuji návrhy jednotlivých stránek a uvádím některé ukázky drátěných modelů. Kompletní přehled vytvořených návrhů je v přílohách této práce.

### Autentizace

Mezi veřejné stránky patří přihlášení, registrace a obnova zapomenutého hesla. Všechny tyto stránky tvoří jeden hlavní formulář s jedinou logickou akcí. Návrh je co nejjednodušší, aby neodváděl uživatelovu pozornost a minimalizoval riziko opuštění stránky ještě před přihlášením do aplikace.

### Nástěnka

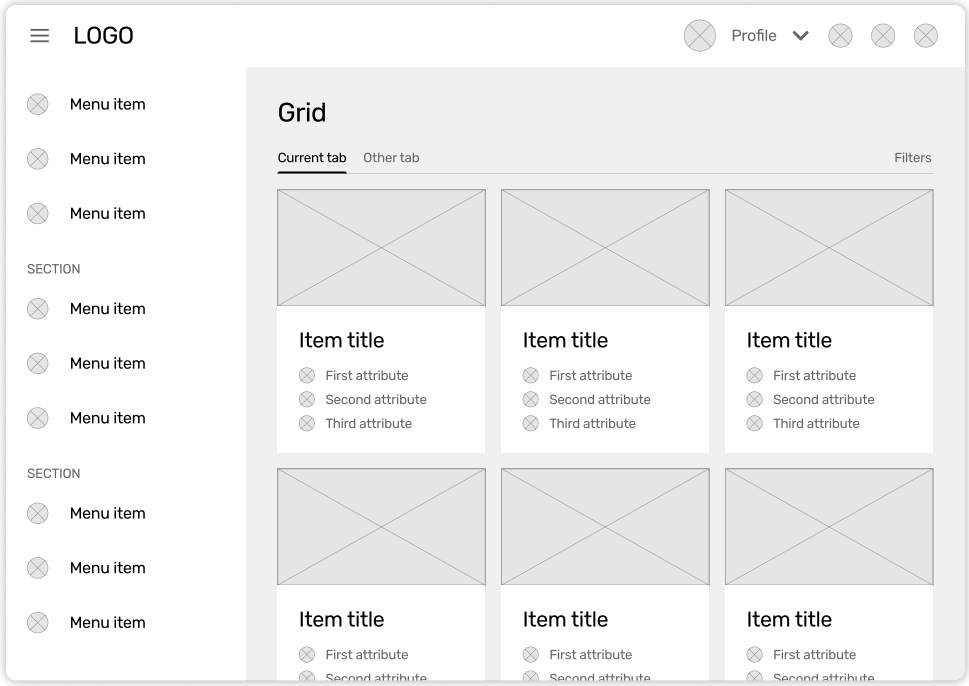
Po přihlášení se uživatel ocitne na nástěnce. Nástěnka slouží jako přehled důležitých informací a novinek. Tvořena je jednotlivými widgety, které slouží k informaci uživatele a rychlému přístupu k vybraným funkcím. Na obrázku 2.2 je vidět, že se widget skládá z nadpisu, těla a tlačítka pro akci. Tělo může představovat text, strukturovaný seznam nebo jiná libovolná data. Příkladem konkrétního widgetu může být zobrazení počtu kreditů s rychlým přístupem k peněžence nebo seznam posledních článků z firemního blogu.




Obrázek 2.2: Wireframe nástěnky

### Marketplace

Sekce *Marketplace* se skládá z nabídek, kapel a pořadatelů. V podstatě představuje tržiště, které má uživateli prezentovat jednotlivé nabídky a profily, se kterými může vyjednávat. Všechny tyto stránky umožňují filtrování a řazení pro snadnější objevování na základě vlastních preferencí. Seznam je realizován pomocí *gridu* neboli mříže, ve které se nachází odkazy na jednotlivé konkrétní stránky. Z obrázku 2.3 je vidět, že každý prvek obsahuje titulní fotografii dané entity, nadpis a tři hlavní atributy.

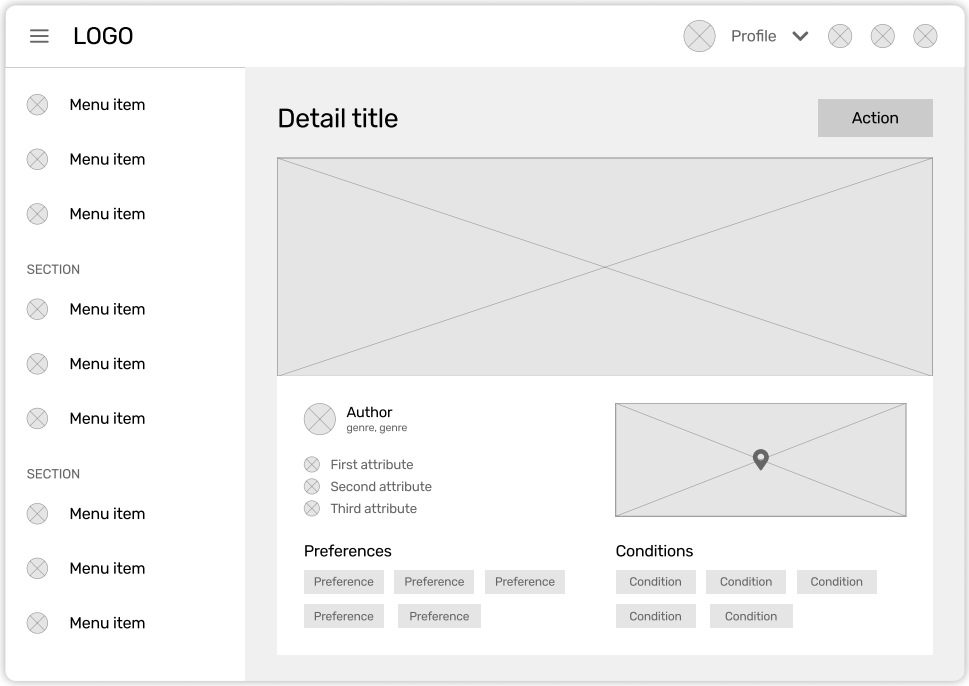



Obrázek 2.3: Wireframe seznamu nabídek

Nabídky je možné řadit od nově přidaných nebo nejbližšího termínu. Kapela může nabídky filtrovat podle typu pořadatele nebo typu místa konání, pořadatelé naopak dle typu nebo žánru kapely. Oba mohou specifikovat rozsah termínů. Prvky vždy obsahují titulek nabídky, název místa, adresu a datum konání. Seznam kapel i pořadatelů se řadí od nově přidaných nebo podle abecedy. Filtrovat můžeme pomocí stejných parametrů, jako u nabídek. V kartách se zobrazuje název profilu, umístění, v případě kapel žánry a počet členů, u pořadatelů typ a komunikační jazyky.

Cílem těchto stránek je přechod na konkrétní nabídku nebo profil. Detail nabídky (na obrázku 2.4) zobrazuje podrobnější informace o nabídce, preference autora a požadavky na pořadatele nebo podmínky pro kapelu. Konverzní akcí na této stránce je zahájení vyjednávání. U kapel a pořadatelů uživatel na-

lezne podrobné informace o profilu včetně popisu, sociálních sítí a případného promo videa. Pod profilem se nachází odkazy na vystavené nabídky. Pořadatelé mají možnost přímo oslovit kapelu soukromou nabídkou.




Obrázek 2.4: Wireframe detailu nabídky

## My business

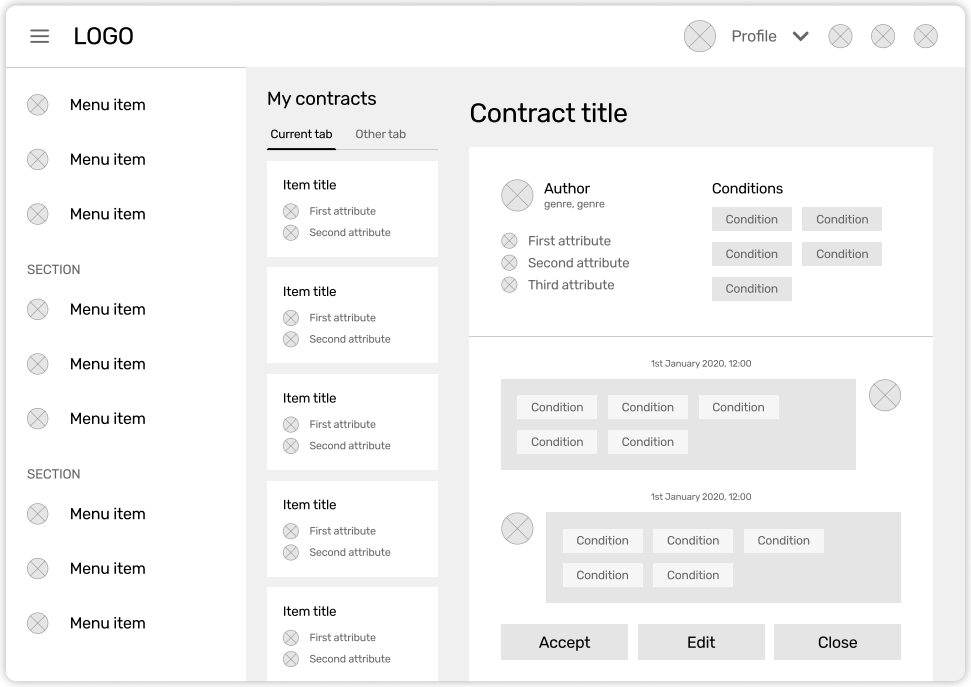
Sekce *My business* se soustředí na vlastní nabídky, vyjednávání a dohody. Návrh jednotlivých stránek v této sekci se skládá vždy ze dvou částí (obrázek 2.5). V levé části se nachází panel se seznamem prvků, které se po vybrání otevřou v pravé části stránky. V mobilní verzi nejsou obě části viditelné najednou, scénář přechodu je rozdělen na dvě samostatné stránky.

V panelu nabídek je umístěno tlačítko pro vytvoření nové nabídky. Každou z nich je možné odstranit nebo duplikovat. Duplikace nevytvoří přímo novou nabídku, ale pouze předvyplní formulář stejnými daty, takže je možné upravit atributy včetně data a času. Platné nabídky, které zatím nevypršely, je možné upravit. Detail nabídky byl již popsán v předchozí sekci.

Vyjednávání jsou rozdělena na probíhající a ukončené. Není možné vytvořit nové vyjednávání přímo z této sekce, pouze se zde zobrazují existující, které vznikly pomocí reakce na nabídku. Každé vyjednávání obsahuje titulek, informace o vystoupení včetně odkazu na profil protistrany a seznam podmínek z původní nabídky.

Pod informační částí se nachází konverzace mezi kapelou a pořadatelem formou *chatu*. Zprávy představují iterace změn podmínek. Podmínky, které se v rámci dané iterace nezměnily, jsou graficky odlišeny větší průsvitností, aby zvýraznily změněné. Podobně ty, které byly odstraněny, se zobrazí přeškrtnuté, aby si toho všimla druhá strana a nedošlo k přehlédnutí. V probíhajícím vyjednávání se pod konverzací nachází tři tlačítka pro hlavní akce: přijetí nebo úprava podmínek a ukončení vyjednávání. Pokud vyjednávání skončilo dohodou, nachází se zde na ni odkaz.

Tlačítko pro přijetí podmínek otevře vyskakovací okno. Zde se nachází přehledný souhrn podmínek, aby uživatel viděl, s čím přesně souhlasí, protože přijetí podmínek je závazné. Při volbě úpravy podmínek se otevře krátký formulář s předvyplněnými atributy z poslední zprávy. Ukončení vyjednávání je nutné potvrdit ve vyskakovacím okně.




Obrázek 2.5: Wireframe detailu vyjednávání

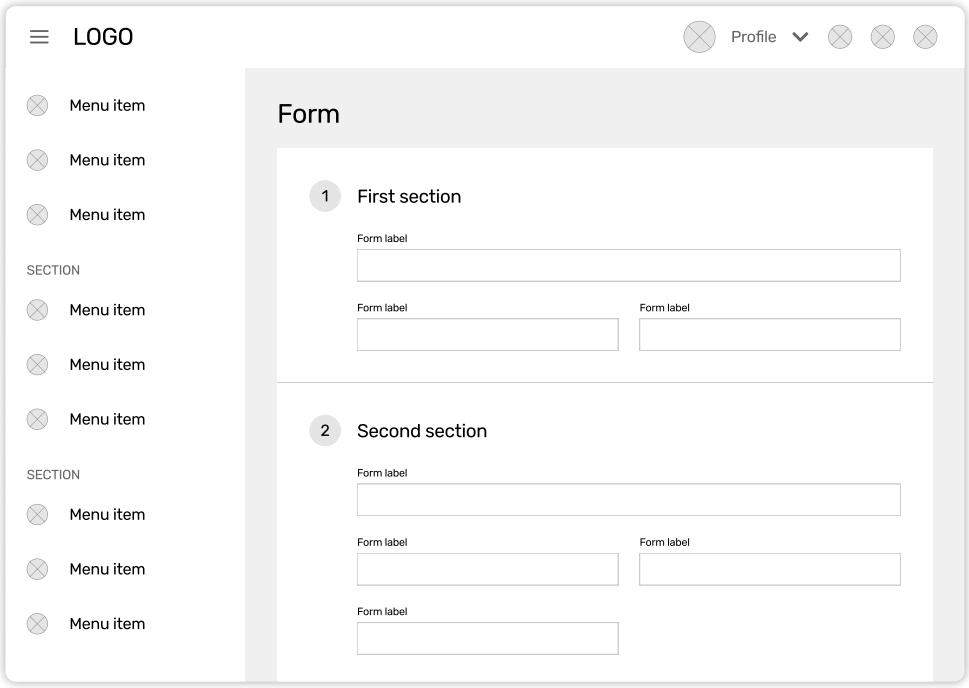
Seznam dohod je rozdělen na aktivní, jejichž termín se ještě nekonal, a archiv již proběhlých. Každá dohoda obsahuje podrobné informace včetně dat, časů, místa konání a odkazů na oba profily. Viditelné jsou i konečné podmínky vystoupení, na kterých se dohodly v rámci vyjednávání. Obě strany vzájemně vidí e-mail a telefonní číslo na kontaktní osobu. Dále jsou zde ke stažení důležité dokumenty obou profilů. Nechybí možnost uložení vygenerovaného návrhu smlouvy ve formátu PDF. Vytvořenou dohodu nelze upravovat.

### Profil

Profil kapely nebo pořadatele byl popsán v sekci Marketplace. V tomto případě se jedná o vlastní profil, kterým je uživatel v aplikaci přihlášen. Vlastník má možnost profil upravit a nebo smazat. Smazání profilu je nutné potvrdit a je nevratné. Smažou se veškeré nabídky a uzavřou se probíhající vyjednávání.

K tvorbě nebo úpravám profilů a nabídek slouží formulář uvedený na obrázku 2.6. Formulář je vždy členěn do oddělených logických celků označených nadpisy. Při tvorbě jsou zobrazeny i čísla jednotlivých sekcí. Každé formulářové pole má svůj viditelný popisek. Povinná pole jsou označena hvězdičkou. Varování validačních pravidel se zobrazují pod jednotlivými poli.

Standardní pole pro zadávání textu a čísel nejsou ničím výjimečné. Zmínit však mohu pole s napovídáním textu (například u měst) nebo speciální pole s nabídkou časů a dat, která uživatel vybírá přímo v kalendáři.




Obrázek 2.6: Wireframe formuláře

### Nastavení

Na této stránce se zobrazují jednotlivá nastavení, která se vztahují k uživateli, nikoliv k profilu. Patří sem úprava dat uživatele, změna hesla nebo nastavení e-mailových preferencí. Jednotlivé části jsou oddělené, každou tvoří samostatný formulář. Do budoucna se počítá s možným rozšířením.

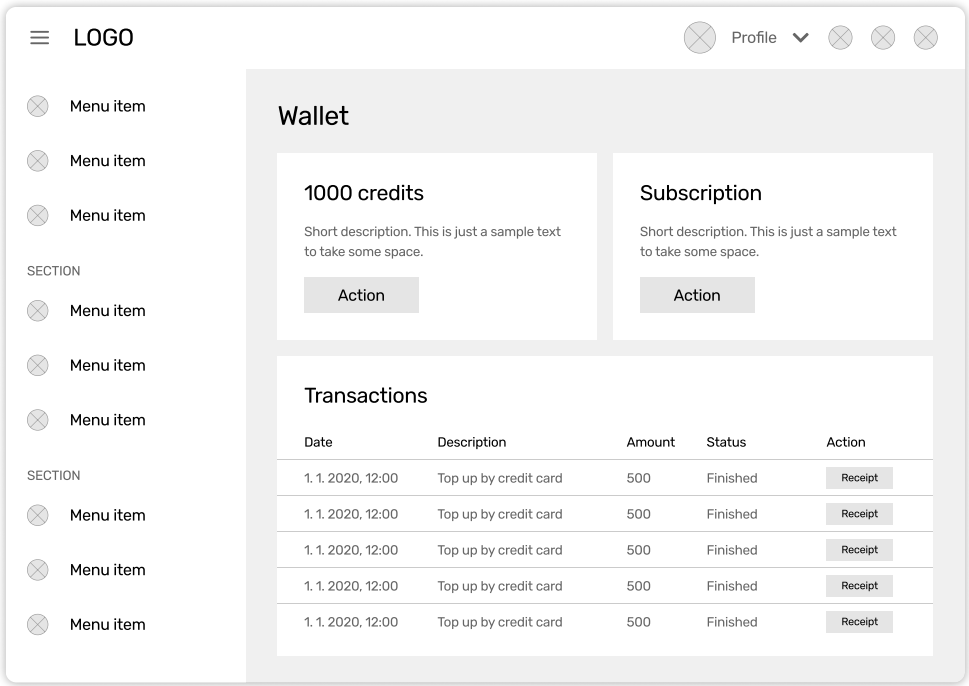
### Notifikace

V pravé části hlavičky stránky se nachází ikona pro přístup k notifikacím. U ikony je také indikace nepřečtených zpráv. Po kliknutí se zobrazí okno s nepřečtenými notifikacemi. Ve spodní části okna je odkaz na seznam všech notifikací. Každá notifikace má svoji ikonu představují její typ. Uveden je titulek, textový popis a čas vytvoření. Notifikace může odkazovat na danou stránku aplikace podle přidaných dat. Je zde možnost označit notifikaci jako přečtenou bez nutnosti vyvolání akce notifikace. Přečtené zprávy se od nepřečtených graficky odlišují barvou ikony.

### Peněženka

První widget zobrazuje zůstatek kreditů a možnost dobití. Ve vyskakovacím okně uživatel uvede počet kreditů k dobití a typ platební metody. Při výběru bankovního převodu se zobrazí údaje k platbě včetně QR kódu. Druhou možností je platba kartou, kdy dojde k přesměrování na platební bránu.

Druhý widget slouží ke správě předplatného. Pokud je aktuálně profil v bezplatné variantě, může si zažádat o VIP účet. Po zvolení je možné vybrat způsob měsíčního placení. Spodní část stránky tvoří transakce. Každá obsahuje datum, popis, počet kreditů, stav a případně akci. Tou může být opakování zaplacení nebo zobrazení platebních údajů.




Obrázek 2.7: Wireframe peněženky

## 2.4 Grafický návrh

Vizuální styl aplikace představuje volby barev, ikon, písma, animací a vzhledu jednotlivých komponent. Grafický design je složitá disciplína, protože rozhoduje o tom, jak se uživateli aplikace líbí a jak se mu používá. Líbivost však nelze považovat za indikátor úspěchu návrhu. Efektivitu zpracování ukáže až analýza chování návštěvníků nebo testování grafického vzhledu. Design lze podle Jana Řezáče [14] posuzovat různě:

- splňuje aktuální grafické trendy;
- dodržuje principy grafického designu a typografie;
- podporuje pozici a důvěryhodnost značky;
- předává zprávu, kterou si má uživatel odnést;
- stanovené konverzní akce fungují.

Mimo to by měl dodržovat základní pravidla pro použitelnost a přístupnost. Obsah a nadpisy by měly být správně strukturovány, v rámci všech komponent by měl být dodržován barevný kontrast písma s pozadím a velikost písma včetně řádkování by měla zaručit čitelný text. Každý formulář a jeho prvky by měly být srozumitelně popsány a na mobilu by měly být dostatečné velké pro ovládání dotykem. Takových pravidel je spousta a je dobré na ně při tvorbě grafického vzhledu nezapomínat.

Pro reprezentaci grafického vzhledu před implementací slouží *high-fidelity* (vysokoúrovňové) prototypy [13]. Jedná se v podstatě o propojení kostry webu s vizuálním vzhledem a interakčním designem. Propojuje jednotlivé stránky webu a věrohodně interpretuje vlastnosti výsledného produktu včetně designu, interakcí a obsahu. Funkční prototypy lze použít pro uživatelské testování.

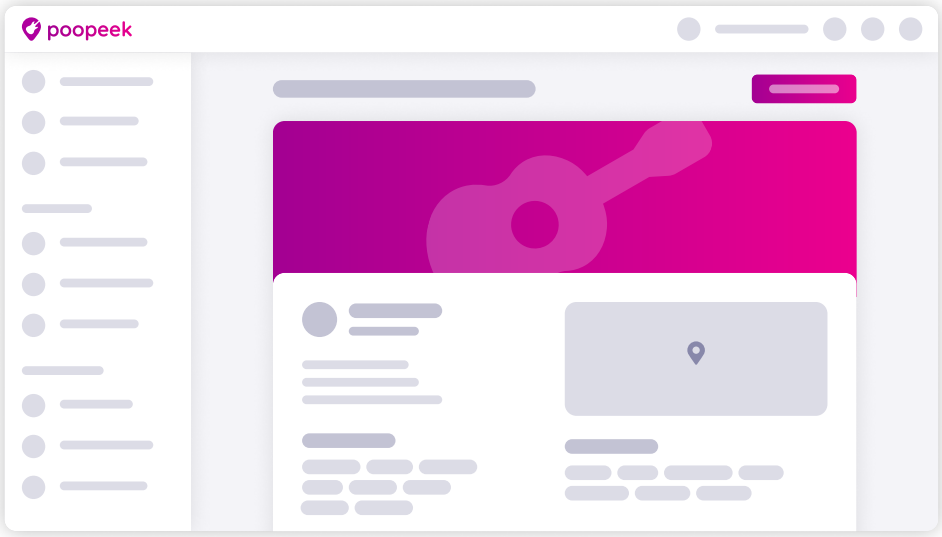
Prototypy aplikace jsem tvořil v nástroji Figma [15]. Slouží k tvorbě UX a UI prototypů a umožňuje vytvářet wireframy nebo celý grafický design. Aplikaci jsem si vybral zejména kvůli tomu, že je webová a tedy dostupná odkudkoliv, podporuje tvorbu *layoutů* a znovupoužitelných komponent a zjednodušuje práci v týmu pomocí komentářů a sdílení změn v reálném čase. Front-end developeři ocení export všech grafických podkladů ve formátech PNG, PDF a SVG a také možnost zobrazení CSS stylů jednotlivých prvků. Hlavní výhodou je také pohodlná práce s vektory.

Vektorová grafika oproti rastrové používá k vykreslení designu matematicky definované křivky [16]. Realizovány jsou pomocí bodů spojených do křivek (vektorů), které se skládají dohromady a mohou tvořit různé tvary. Díky tomu je možné vektorový obrázek libovolně zvětšovat bez jakékoliv ztráty kvality. Výhodou je i malá velikost výsledného souboru, pokud neobsahuje příliš mnoho komplikovaných objektů. Proto je ideální zejména pro grafické návrhy aplikací a naopak není určený pro fotografie.

### Vizuální styl

Náš start-up v době psaní této práce nemá oficiální grafický manuál, ze kterého bych mohl čerpat přesné normy. Při tvorbě vzhledu aplikace jsem vycházel z grafických materiálů, které tvoří naší identitu. Jedná se o různé prezentace pro partnery, letáky z našich akcí nebo grafika na sociálních sítích. Návrhy jsme testovali v rámci týmu a průběžně jsem zapracovával dohodnuté změny.

Inspiroval jsem se principy systému Material Design [17]. Jednotlivé styly a komponenty vypadají přirozeně a jsou podobné prvků z reálného světa. Přestože je design dvourozměrný, dodržuje vlastnosti objektů jako jsou seskupování nebo překládání vrstev. Čím výše se vrstva nachází, tím větší vrhá stín. Nejen díky tomu začnou být komponenty konzistentní napříč aplikací.




Obrázek 2.8: Základní layout aplikace

Základní *layout* (rozložení) aplikace šel odvodit již z přiložených drátěných modelů. Na obrázku 2.8 jsou zobrazené hlavní tři části:

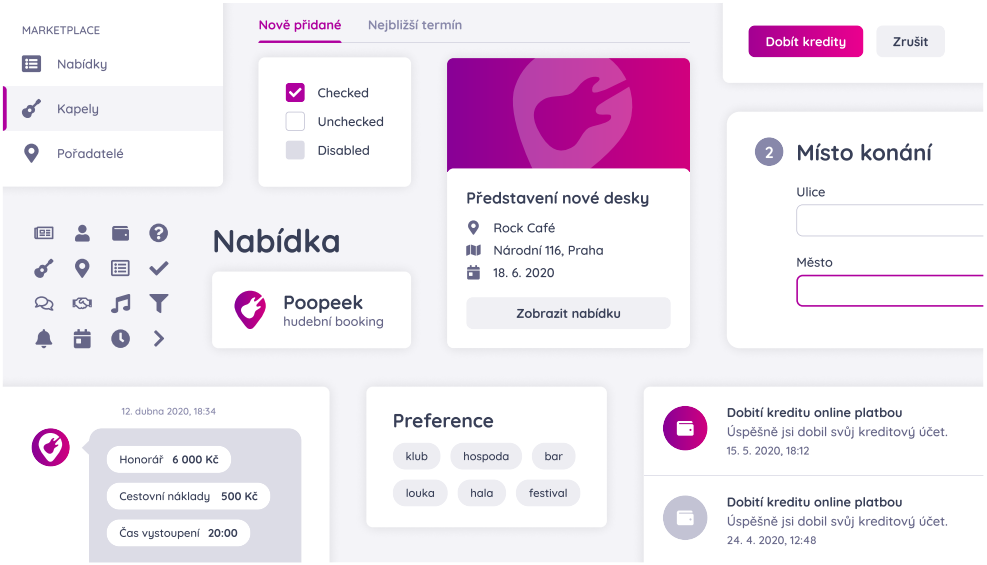
**Top bar** – Horní lišta obsahující v levé části logo a možnost skrytí/zobrazení navigace, v pravé části možnost přepnout profil, zobrazit notifikace, nápovědu, nastavení a možnost ohlášení.

**Navigation** – Levý panel slouží pro navigaci, jednotlivé sekce jsou odděleny nadpisem a každá stránka má svoji unikátní ikonu. V mobilní verzi je navigace skrytá a při zobrazení vklouzne zleva.

**Content** – Hlavní obsah se nachází v pravé části od navigace. V principu je tvořen nadpisem první úrovně a velkou kartou zobrazující obsah stránky. V pravé horní části se může nacházet tlačítko pro konverzní akci.

Barevné schéma obsahuje primárně purpurovou barvu. Využívá přechody v odstínech barvy *magenta*, zjednodušeně od fialové až po tmavě červenou. Varianty této barvy umožňují odlišit významné prvky od ostatních a upoutat pozornost očí uživatele. Pozadí je světle šedé s jemným modrým odstínem. Díky tomu perfektně vyniknou karty a widgety, jejichž barva povrchu je čistě bílá. Nadpisy, texty a ikony jsou tmavě šedé, taktéž se slabým odstínem modré barvy. Při použití barev je kladen důraz na dostatečný kontrast mezi barvou písma a pozadím elementu.

Hlavním fontem aplikace je písmo Quicksand převzaté z Google Fonts [18]. Přestože je font variabilní, používám hlavně 3 řezy: *Semi-bold*, *Medium* a *Regular*. Nejsilnější variantu pro nadpisy, středně silnou pro odkazy a zvýraznění a obyčejnou pro zbytek aplikace. Jako výchozí jednotku jsem zvolil `rem`, který určuje základní velikost písma. Podle něj se dále odvíjí velikosti nadpisů, elementů a dokonce i celého rozvržení. Řádkování se drží doporučeného 1,5 násobku velikosti písma. Velikosti se dále responzivně škálují, aby byl text čitelný i na menších obrazovkách.

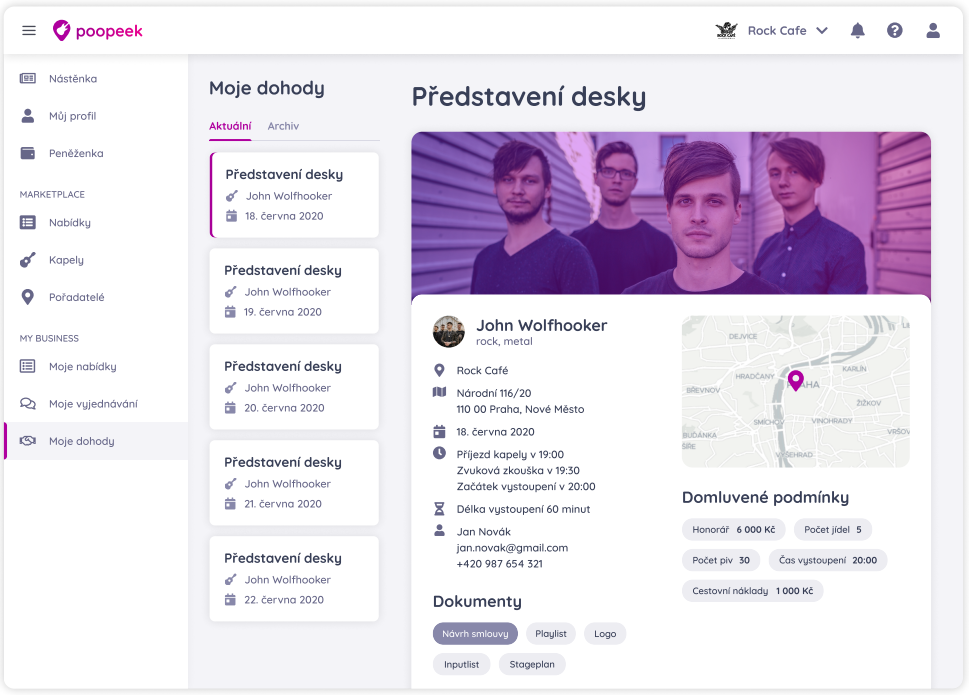



Obrázek 2.9: Ukázky designu komponent

Ikony aplikace jsou použité z knihovny Font Awesome [19]. Využívám styly *Solid* (plné) a *Regular* (obrysové). Knihovnu jsem zvolil na základě toho, že obsahuje všechny potřebné ikony pro použití v aplikaci (např. kytaru a spoustu charakteristických ikon pro podmínky vyjednávání). V rámci celé aplikace se snažím držet oblých tvarů a zakulacených rohů. Míra zakulacení se odvíjí od velikost elementu. Animace komponent jsou co nejvěrohodnější a v přijatelné míře, aby nerušily uživatele. Využívám způsob *ease-in-out*, který umožňuje zrychlení a následné zpomalení elementu místo konstantní rychlosti.

### Prototypy vzhledu aplikace

Grafické návrhy obrazovek vychází z vytvořených wireframů a vizuálního stylu. Nemusí však plně odpovídat původním návrhům, jsou v nich již obsaženy změny na základě testování, které nepropaguji zpětně do vytvořených wireframů. Cílem prototypů je naznačit styl základních obrazovek s unikátními prvky, podle kterých lze odvodit i další stránky, které nejsou součástí návrhu. Všechny vytvořené prototypy můžete nalézt v přílohách. Pro demonstraci přikládám obrázek 2.10 s ukázkou prototypu dohody.




Obrázek 2.10: Prototyp dohody

Kapitola 3

# Implementace prototypu

V první řadě je nejdůležitější seznámit se s použitými technologiemi. Důkladně rozebírám zvolený framework Angular a všechny části, které s ním souvisí. Následně popisuji strukturu celé aplikace a tvorbu komponent, které jsem implementoval. Zmíním také způsoby, jakými pracuji s daty a stavy jednotlivých komponent. Výsledkem této části je funkční prototyp.

## 3.1 Použité technologie

Klasická webová stránka se skládá z několika součástí. Základem je HTML, představující kostru stránky včetně obsahu, a CSS, starající se o vzhled a grafickou podobu. Často bývá přítomný JavaScript, ať už jen pro analytické nástroje, nebo pro dodání dynamiky uživatelského rozhraní.

Moje řešení využívá framework Angular, který mi umožňuje vytvořit moderní *single-page* aplikaci. Ten je psaný v jazyce TypeScript, který se překládá do JavaScriptu, aby ho mohl prohlížeč zpracovat. Dále využívám principy reaktivního programování, konkrétně knihovnu RxJS, která je s frameworkem Angular úzce spjata. V této sekci popisuji zmíněné technologie.

### HTML a CSS

HTML je jazyk popisující strukturu webových stránek [20]. Umožňuje strukturovat dokument do nadpisů a odstavců, vkládat tabulky, obrázky a jiná multimédia přímo do dokumentu, vytvářet formuláře pro práci s daty a také vzájemně propojovat dokumenty pomocí hypertextových odkazů. Struktura stránky je popsána značkami, které představují jednotlivé elementy (např. `<table>`, `<img>` nebo `<button>`). Objektově orientovaná reprezentace struktury dokumentu je označována jako rozhraní DOM.

Pro popis vzhledu stránky slouží jazyk CSS [20]. Jednotlivým elementům přiřazuje styly, které tvoří atributy jako barva, velikost, font a spousta dalších. Umí také základní animace při interakci uživatele (např. při najetí kurzorem

myši na element). Dokáže navrhnout rozvržení stránky pro malá mobilní zařízení, velké desktopové obrazovky nebo tiskárny. Oddělení HTML od CSS umožňuje jednodušší správu stránek a také sdílení stylů mezi více stránkami.

Styly se po čase mohou stát rozsáhlé a těžko udržovatelné, pomoct však může preprocesor. Rozhodl jsem se použít Sass [21], který do CSS přidává nové funkce. Zejména jde o vnořování, funkce, proměnné, dědičnost nebo moduly. Psaní stylů je pak jednodušší, kód zabírá méně místa, pomáhá zamezovat opakování kódu a je možné ho členit do modulů. Před použitím stylů je nutné SCSS soubory přeložit do běžného CSS, které umí prohlížeč vykreslit.

### TypeScript

Přesto, že je JavaScript velmi populární, má spoustu nedostatků, které mohou ztížit vývoj rozsáhlých aplikací. Právě proto vznikl TypeScript [22], jazyk vyvíjený společností Microsoft, který JavaScript rozšiřuje a přidává mu chybějící funkce. Mezi zásadní přínosy tohoto jazyka patří typová kontrola, statické datové typy a pokročilý koncept objektově orientovaného programování.

To umožňuje chytřejší napovídání při psaní kódu a odhalení některých chyb už při kompilaci, zatímco v klasickém JavaScriptu by se projevily až při běhu aplikace. Před spuštěním probíhá *transpilace* neboli překlad kódu, díky čemuž je výsledný kód kompatibilní s webovými prohlížeči podporující obyčejný JavaScript. TypeScript je součástí frameworku Angular a s pomocí nástroje Angular CLI se překlad děje automaticky během sestavování aplikace.

### RxJS

Paradigma reaktivního programování tkví v propagaci změn pomocí datových proudů (streamů). RxJS je knihovna pro reaktivní programování usnadňující tvorbu programů založených na asynchronních akcích. Podle [23] kombinuje myšlenky návrhových vzorů *Observer* a *Iterator* společně s prvky funkcionálního programování. Mezi základní koncepty knihovny patří:

**Observable** představující stream dat,

**Observer** obsahuje kolekci *callback* funkcí, které určí, co se bude dít s přijatými daty ze streamu,

**Subscription** reprezentuje samotné připojení ke streamu dat, využívá se hlavně pro pozdější možnost odpojení,

**Operators** jsou funkce využívající styl funkcionálního programování k práci s kolekcemi (patří sem například `map`, `filter`, `reduce`, atd.),

**Subject** je ekvivalentem *emitoru událostí*, tedy poskytuje možnost, jak poslat hodnotu nebo událost více posluchačům najednou.

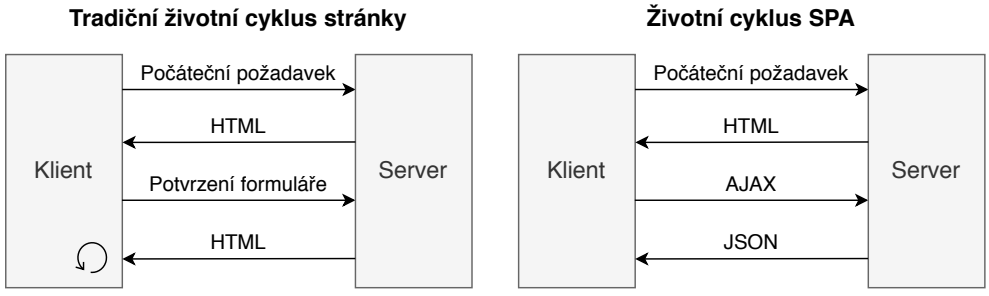
Knihovna je přímo integrovaná v Angularu, což umožňuje její využití i pro nativní nástroje tohoto frameworku. Příkladem může být *HttpClient*, který se stará o volání HTTP požadavků a odpověď vrací jako *Observable*. V místě, kde chceme s daty pracovat, zavoláme na objektu metodu `subscribe`. V atributu předáme callback funkci, která obdrží přijatá data.

### Single-page application

Dříve se stránky generovaly na serveru a posílaly zpět klientovi sestavené a připravené k vykreslení. Výhoda tohoto řešení spočívá hlavně v indexaci vyhledávači, jelikož robot, který stránky prochází, ihned po obdržení souborů ze serveru dokáže přečíst obsah webu. Důležitá je i rychlost, protože po vykreslení prohlížeč už nemusí čekat na jinou odpověď pro získání dat. Nevýhodou však je, že daný cyklus stažení celé stránky se opakuje po každé, kdy uživatel přejde na další stránku a to může být poměrně neefektivní.

Řešení, které se stalo v poslední době velmi populární, se nazývá SPA neboli *single-page application* [24]. Aplikaci tvoří pouze jeden HTML dokument a JavaScript běžící na straně klienta (v prohlížeči). Při prvním navštívení webu se stáhne kostra webu společně se styly a balíky JavaScriptu. Po obdržení souborů klient aplikaci sestaví, vykreslí a pošle požadavek na API pro získání dat. Po obdržení se data vykreslí a načítání je dokončeno. Doba trvání prvního načtení je proto nevýhodou tohoto řešení.

Hlavním důvodem oblíbenosti single-page aplikací je pokračování komunikace, tedy přechod na další stránku webu. Zatímco u serverově generovaných webů by následovalo opětovné stažení hotové stránky včetně stylů a skriptů, SPA již další zdroje nestahuje, pouze pošle dotaz na API pro získání požadovaných dat nové stránky. Díky tomu je přechod plynulý bez zbytečného překreslení celého uživatelského rozhraní. Nevýhodou je však indexace vyhledávačů roboty, kteří nečekají na dokončení zpracování JavaScriptu, protože jim se zobrazí pouze kostra webu bez obsahu.




Obrázek 3.1: Životní cyklus tradiční stránky oproti SPA (překresleno z [24])

## 3.2 Angular framework

Pro tvorbu aplikace jsem použil Angular [25]. Jedná se o populární framework a platformu pro vývoj front-end aplikací vyvíjený společností Google. Je napsaný v TypeScriptu a hlavní součástí je knihovna RxJS. Umožňuje tvořit nejen progresivní webové aplikace, ale i nativní pomocí technologií Cordova, Ionic nebo NativeScript. Framework také umí *server-side rendering* díky Angular Universal, takže výkon a zároveň optimalizace SEO není problém. Součástí jsou také nástroje Karma pro testování komponent a Protractor pro *end-to-end* testování scénářů používání celé aplikace. Zapomenout nesmím také na podporu přístupnosti a více cizojazyčných verzí.

Jednou z hlavních výhod je nástroj Angular CLI. Používá se v příkazové řádce a umí spoustu důležitých funkcí pro zvýšení produktivity. Uživatelům pomůže hlavně s vytvořením projektu jedním příkazem, spouštěním a sestavováním aplikace nebo generováním jednotlivých komponent. Podporuje také *code splitting*, který rozdělí aplikaci na jednotlivé moduly, které se načítají podle potřeby. Výhodou tohoto řešení je snížení velikosti celé aplikace a načítání pouze částí, které uživatel v danou chvíli využívá. Angular má také silnou komunitu a spoustu integrací do jiných nástrojů, které v editorech chytře napovídají doplňování kódu, hlídají syntaxi a zabraňují chybám, které je možné odhalit přímo při psaní kódu.

Architektura Angular aplikace se skládá z několika klíčových částí [26]. Celou strukturu si můžeme představit jako strom modulů, které jsou do sebe vnořeny a obsahují jednotlivé komponenty. Vrchol stromu tvoří *root module*, který je potřebný pro spuštění aplikace. Níže rozeberu jednotlivé části:

**Modules** Moduly usnadňují vývoj a škálovatelnost aplikace pomocí rozdělení částí kódů na samostatné celky. Každý modul umí import funkcionality z jiných modulů a samozřejmě také export svých definovaných částí. V rámci modulu definujeme vytvořené komponenty, které se tak stanou součástí aplikace a je možné je použít. Komplexním aplikacím pomáhá *lazy-loading*, který umožňuje načítat jiné moduly až v případě potřeby.

**Components** Komponenty jako takové také tvoří strom, který se následně spojí s hierarchií DOM při vykreslení stránky prohlížečem. Každou komponentu tvoří TypeScript soubor obsahující třídu, která se stará o data a logiku. Pro vykreslení komponenty slouží soubor HTML, ve kterém je definovaná šablona, společně se souborem stylů, které se však vážou pouze k dané komponentě a není možné je použít v jiné části aplikace.

**Templates** Šablona komponenty je definovaná pomocí HTML a Angular direktiv. Ty určují logiku a napojování dat na DOM. Vázat lze události (např. kliknutí na element) nebo atributy (např. vypsání proměnné do odstavce). Angular podporuje obousměrnou datovou vazbu, tedy

změny v DOM jsou reflektovány do dat aplikace a opačně. Pro efektivnější práci s funkcemi je možné použít *pipes* nebo vlastní direktivy.

**Services** Služby, které nejsou přímo spjaty s konkrétní komponentou a používají se na více místech napříč aplikací, se nazývají *services*. Před definicí třídy se nachází dekoratér `@Injectable()`, který ji umožňuje ostatním třídám použít jako závislost. Tomuto principu se říká *dependency injection* a jeho cílem je odlehčit logiku komponent a přenechat ji jiné službě, která může obsluhovat více komponent najednou. Je možné ji použít jako efektivní způsob komunikace mezi komponenty.

**Routing** Vestavěný Angular *Router* je v podstatě služba, která pomáhá s navigací mezi stavy a komponenty aplikace. Splňuje standardní chování navigace v aplikaci, tedy při přechodu na jinou stránku se mění URL adresa v prohlížeči, kliknutím na odkaz dojde k přesměrování a veškeré cesty se ukládají do historie prohlížeče, takže je možné jít zpět nebo vpřed. Navigace se tvoří z definovaných pravidel, které určují cestám jednotlivé komponenty a další atributy včetně přístupových práv.

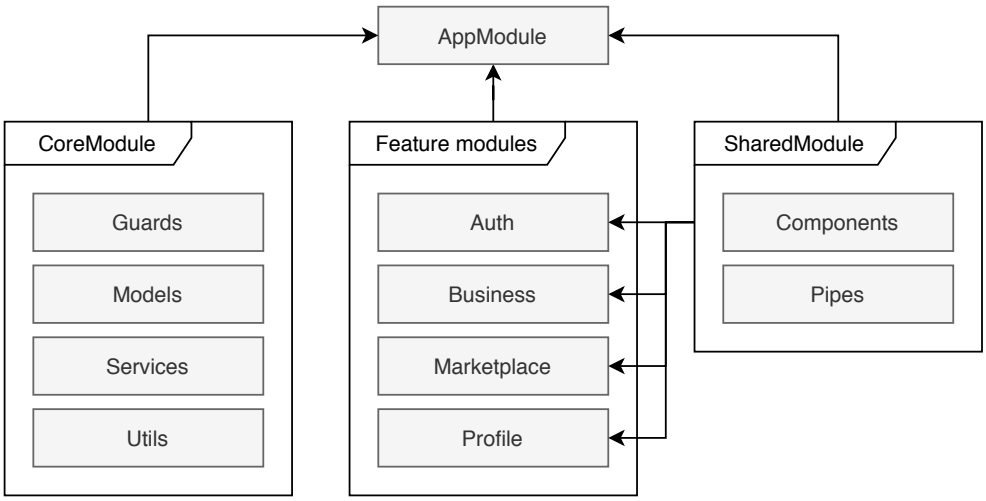
Důležité je také zmínit životní cyklus komponenty. Angular rozhoduje o prvotním vykreslení komponenty a jejích potomcích, dále pak novému vykreslení, pokud dojde ke změně dat vázaných se šablonou, a také zničení komponenty před odstraněním z DOM. Díky tomu, že se změny propagují do všech potomků, je vhodné je hlídat. Jakmile se aplikace rozroste, může dojít k problémům s výkonem, jelikož se změn bude provádět až příliš a prohlížeč bude muset překreslovat DOM pořád dokola. Řešením je rozdělení komponent do chytrých, které se budou starat o logiku, a prezenčních, které pouze převezmou data od rodiče a vykreslí je bez pomocí jiné datové služby.

Zpracovávat uživatelský vstup je součástí většiny aplikací. Formuláře slouží k získávání textu nebo souborů, které se dále zpracovávají. Angular poskytuje dva druhy formulářů. Prvním jsou jednoduché šablonové formuláře, které lze využít pro jednoduché vstupy bez možnosti většího škálování. Druhým typem jsou reaktivní formuláře, které jsou znovupoužitelné, dají se testovat a poskytují spoustu možností pro validaci a rozšíření. V zásadě se formuláře tvoří ze skupin, ve kterých mohou být prvky, pole prvků nebo vnořené skupiny.

Nakonec bych rád přidal srovnání s ostatními frameworky [27]. React od společnosti Facebook je nejrozšířenější knihovnou pro tvorbu front-end aplikací. Oproti Angularu používá virtuální DOM, který může dosahovat lepších výkonů. Komponenty jsou psány jako JSX, které rozšiřuje syntaxi JavaScriptu, ve výchozím stavu však nepodporuje TypeScript a samotná knihovna nenabízí moc oficiálních nástrojů. Další populární knihovnou je Vue.js, která vyniká hlavně malou velikostí. Šablony jsou podobné Angularu a lehce se integrují knihovny třetích stran. Nevýhodou je menší komunita, množství materiálů a flexibilita při tvorbě velkého projektu.

## 3.3 Architektura aplikace

Celá aplikace je postavena na základě stromu modulů, které obsahují komponenty. Vrcholem tohoto stromu je `AppModule`, který se stará o spuštění aplikace a importuje ostatní moduly. Prvním z nich je `CoreModule` definující základní služby, které jsou dostupné napříč aplikací. Dále importuje jednotlivé funkční moduly, které představují samostatné celky aplikace. Patří sem například moduly pro autentizaci, části *My Business* a *Marketplace* a zbylé stránky. Posledním důležitým modulem je `SharedModule`, v rámci kterého jsou definované znovupoužitelné komponenty. Tento modul je pak importován ve všech modulech využívající tyto komponenty.




Obrázek 3.2: Architektura stromu modulů

Návrh architektury vychází z oficiálních doporučení tvůrců Angularu [28] a mých vlastních zkušeností. Pro pochopení rozdělení modulů je vhodné uvést základní princip, jak framework s moduly pracuje při startu aplikace.

Při navštívení aplikace uživatel obdrží stránku `index.html`. V těle této stránky je pouze jedna značka `<app-root></app-root>` a import potřebných skriptů. Na tomto místě se při startu aplikace vytvoří *root* komponenta s názvem `AppComponent` v rámci modulu `AppModule`. Zde se nachází základní prvky vzhledu včetně značky `<router-outlet></router-outlet>`. Jedná se o místo, kam *Router* vykreslí jednotlivé stránky funkčních modulů definovaných v `AppRoutingModule`. Může se jednat o přímo použité komponenty, jejichž moduly se načtou ihned při spuštění aplikace, a nebo moduly využívající tzv. *lazy-loading*, které se načtou až v případě potřeby nebo při vstupu na danou URL podle použité *preloading strategy*.

### CoreModule

Jedná se o hlavní modul, který poskytuje modely a služby napříč aplikací. Od `SharedModule` se liší tím, že obsahuje objekty, které není nutné explicitně definovat v rámci modulu pro export a následný import v jiných modulech. Patří sem modely entit, globální služby, *guards* pro oprávnění stránek a pomocné funkce, kam patří animace nebo validátory formulářů.

**Models** Modely představují entity popsané v první kapitole. V podstatě kopírují schémata použitá na serveru včetně názvů a datových typů. Některé entity jsou z důvodu zabránění duplikace kódu rozloženy do několika modelů, které se používají ve více entitách (například lokace). Tvořeny jsou pomocí TypeScript tříd a rozhraní. Patří sem například model Account, Conditions, Deal, Payment nebo Venue.

**Services** Služby jsou znovupoužitelné funkce, které je možné použít v kterékoliv komponentě. Pro komunikaci s API slouží `ApiService`. Využívají ji všechny služby, které přímo pracují s entitami. Další důležitou službou je `AuthService`, která uchovává aktuálně přihlášeného uživatele a profil. Tato služba využívá `TokenService` pro jednoduchou práci s tokenem uloženým na straně klienta (výpis kódu 3.1). Poslední speciální službou je `CodeService`, která poskytuje statická data, jako jsou například žánry, typy, jazyky nebo města.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class TokenService {
  getToken(): String {
    return window.localStorage['token'];
  }

  saveToken(token: String): void {
    window.localStorage['token'] = token;
  }

  destroyToken(): void {
    window.localStorage.removeItem('token');
  }
}
```

Výpis kódu 3.1: Služba pro správu tokenu

**Guards** Jedná se o funkce sloužící k autentizaci a případné autorizaci uživatele před vstupem na stránku. Umožňují tedy vymezit části aplikace, které jsou dostupné bez přihlášení a naopak. Pro tyto účely používám `NoAuthGuard` (například na stránce s přihlášením) a `AuthGuard` (ve většině aplikace). Pokud funkce zjistí, že nemá uživatel na stránku přístup, přesměruje ho na hlavní stránku.

**Utils** Zde nalezneme animace, zejména pro otevření a zavření mobilní navigace. Dále tu jsou validátory formulářů, kam patří například kontrola minimálního a maximálního data. V neposlední řadě tu jsou také tzv. *interceptors* sloužící k odchytávání HTTP požadavků. `TokenInterceptor` přidává hlavičku `Content-Type` a `x-access-token`, kde se nachází autentizační token uživatele. `HttpErrorInterceptor` slouží k odchytávání chyb a jejich zpracování.

### Feature modules

Funkční moduly představují samostatné celky aplikace. Jednotlivé stránky se vykreslují pomocí značky komponenty nebo pomocí *Routeru* na základě URL adresy. Každý funkční modul je definován unikátní částí URL adresy, kterou mají všechny komponenty tohoto modulu stejnou (například `/bands/`). Níže popisuji jednotlivé skupiny funkčních modulů:

**Auth** Skupina modulů tvořící stránky pro autentizaci. Společnou vlastností těchto stránek je veřejný přístup bez autentizace. Patří sem stránky pro přihlášení, registraci uživatele, obnovení ztraceného hesla a verifikace e-mailové adresy.

**Business** Součástí sekce *My business* jsou stránky pro vytvořené nabídky, vyjednávání a dohody. Dodržují layout *master-detail*, tedy levý seznam položek a napravo detail konkrétního prvku. Nabídky je možné číst, vytvářet, upravovat, duplikovat nebo mazat. Vyjednávání umožňuje čtení a manipulaci s podmínkami. Dohody jsou finální, je možné je pouze číst.

**Marketplace** Jedná se o moduly sloužící k objevování nových nabídek a profilů kapel nebo pořadatelů. Jsou postaveny na stejném rozvržení *grid*, které zobrazují mřížku s jednotlivými prvky. Nechybí možnosti řazení, filtrování a stránkování. Po kliknutí se zobrazí detail konkrétní položky se všemi informacemi.

**Profile** Poslední sekce obsahující stránky vázající se k profilu uživatele. Patří sem detail profilu s možností editace a smazání, nástěnka tvořená samostatnými *widgety* a peněženka s možností dobití účtu, volbou předplatného a správou transakcí. Uživatel má k dispozici stránku se všemi notifikacemi a nastavením uživatele, hesla a e-mailových preferencí.

```
<app-loading-spinner *ngIf="loading"></app-loading-spinner>
<div class="card" *ngIf="!loading">
    <div class="card-header">
        <h1>Ověření</h1>
    </div>
    <div class="card-body">
        <ng-container *ngIf="!error">
            <i class="fas fa-check-circle success icon"></i>
            Ověření e-mailové adresy proběhlo úspěšně.
        </ng-container>
        <ng-container *ngIf="error">
            <i class="fas fa-times-circle danger icon"></i>
            Ověření e-mailové adresy se nezdařilo.
        </ng-container>
    </div>
</div>
```

Výpis kódu 3.2: Šablona stránky pro verifikaci e-mailové adresy

```
@Component({
  selector: 'app-verify',
  templateUrl: './verify.component.html',
  styleUrls: ['./verify.component.scss'],
})
export class VerifyComponent implements OnInit {
  loading = true;
  error = null;

  constructor(
    private api: ApiService,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    const token = this.route.snapshot.paramMap.get('token');
    this.api.post(`/auth/verify/${token}`).subscribe(
      () => (this.loading = false),
      (error) => { this.error = error; this.loading = false; }
    );
  }
}
```

Výpis kódu 3.3: Komponenta stránky pro verifikaci e-mailové adresy

Výpisy kódu 3.2 a 3.3 představují detail stránky pro verifikaci e-mailové adresy. Server vygeneruje unikátní token, který pošle formou odkazu na e-mail uživatele. Odkaz směřuje na tuto stránku, která se stará o verifikaci. Nejprve se zobrazí načítání, během kterého dojde ke zkopírování tokenu z URL adresy stránky. Tento token je následně odeslán na server, který ověří, zda je validní. Jakmile dorazí odpověď, skončí načítání a vypíše se informace o úspěšném nebo nezdařeném ověření adresy.

### SharedModule

V rámci tohoto modulu jsou definovány a exportovány znovupoužitelné komponenty, které se používají ve více různých modulech. Modul neexportuje pouze komponenty, ale i samotné moduly (např. `RouterModule`).

**Atomic components** Atomické komponenty jsou také označovány jako prezenční, protože nepracují s datovými službami, ale pouze zobrazují data, která dostanou na vstupu. Typickým příkladem je indikátor načítání, jednotný výpis podmínek vystoupení nebo komponenta notifikace.

**Form controls** Formulářové prvky představují sjednocený styl, popisek a výpis validační chyby. Patří sem pole pro text, datum a čas, výběr ze seznamu, zaškrtávací možnosti nebo našeptávací pole.

**Layout components** Prvky vzhledu slouží zejména pro tvorbu rozvržení. Definovaná je horní lišta, panel s navigací, patička webu a zmiňovaný *master-detail* layout pro sekci *My business*.

**Pipes** Operátor *pipe* slouží jako funkce, která se volá v šabloně komponenty. Angular jich má spoustu definovaných (např. *date* pro formátování data) a k tomu jsem vytvořil několik vlastních (např. ve výpisu kódu 3.4).

```
@Pipe({ name: 'qrpayment' })
export class QRPaymentPipe implements PipeTransform {
  transform(payment: PaymentDetails): any {
    return payment
      ? `SPD*1.0*` +
        `ACC:${payment.iban}+${payment.swift}*` +
        `AM:${payment.amount}.00*` +
        `CC:${payment.currency}*` +
        `X-VS:${payment.payment_vs}`
      : null;
  }
}
```

Výpis kódu 3.4: Funkce generující obsah QR platby

Kapitola 4

# Produkční verze

V poslední kapitole se zaměřuji na vytvoření produkční verze a její nasazení. Nejprve rozebírám průběh sestavení Angular aplikace, typy kompilace a strategie načítání modulů. Po sestavení se přesouvám k nasazení pomocí automatické integrace. Rozebírám použité nástroje a postup nasazení. V závěru kapitoly se věnuji dodatečným optimalizacím včetně vytvoření progresivní aplikace a přidání analytických nástrojů.

## 4.1 Sestavení aplikace

Angular aplikace se skládá z modulů a komponent. Těm však prohlížeč přímo nerozumí, takže je nutné aplikaci předem sestavit. Překladač převede šablony HTML a TypeScript komponenty do srozumitelného JavaScriptu, který už prohlížeč umí vykreslit. K dispozici máme dva druhy sestavení [29]. Aplikaci je možné kompilovat až při běhu podle potřeby a nebo před samotným nasazením a spuštěním v prohlížeči.

### Druhy překladu

Prvním druhem kompilace je typ *Just-in-Time* (JIT), který sestavuje aplikaci až během spuštění v přímo v prohlížeči. Tato metoda byla ve výchozím stavu používána až do Angular verze 8. Díky tomu, že se kód kompiluje při běhu aplikace a ne předem, je velikost balíku skriptů mnohem větší a prohlížeči tak vykreslení stránky trvá mnohem déle. Na druhou stranu si udržuje mapu kompilovaného kódu, takže se hodí na vývoj během implementace pro usnadnění hledání chyb přímo za běhu aplikace.

Druhou metodou je *Ahead-of-Time* (AOT) kompilace. Jedná se o výchozí typ sestavení Angular aplikace od verze 9. Celý proces sestavení probíhá před nasazením aplikace a výsledkem jsou hotové skripty, které už prohlížeč nemusí převádět. Tato metoda se hodí zejména pro produkční verzi aplikace a to z několika důvodů. Díky provedenému procesu předem je vykreslení

stránky mnohem rychlejší, celková velikost skriptů je menší, chyby v kódu se mohou projevit už při sestavení a díky tomu je kód také bezpečnější. Proces kompilace je rozdělen do tří fází:

1. V první fázi dochází k analýze kódu. Překladač zpracovává zdrojové kódy a vytváří jejich reprezentaci. V tuto chvíli se nesnaží interpretovat jednotlivá *metadata*, ale pouze zjišťovat chyby a kontrolovat syntaxi.

2. V druhé fázi se generuje výsledný kód. Překladač převádí reprezentaci vytvořenou v předchozím kroku a interpretuje všechna *metadata*.

3. Ve třetí fázi probíhá typová kontrola. TypeScript kompilátor je schopný validovat nekompatibilní datové typy a špatnou vazbu výrazů uvnitř šablon. Tato fáze není pro sestavení povinná, ale může objevit řadu chyb.

Společně s metodou AOT stojí za zmínku také překladač a *runtime* Ivy [30]. Jedná se kompletně přepsaný *render engine*, který se stará o sestavení a vykreslení aplikace. Umožňuje větší nezávislost překládaných komponent, díky tomu je možné přeložit pouze komponenty, které se změnily místo celé aplikace. Zaměřuje se také na *tree-shaking*, což je proces zjištění, které knihovny se v jakou chvíli používají a na základě toho odstraňuje nepoužité kusy kódu. Výsledkem je mnohem menší výsledný kód a zrychlení celé aplikace.

### Konfigurace sestavení

Často má aplikace více verzí: vývojová, *staging* pro testování a produkční. Všechny tyto druhy mohou mít různé konfigurace sestavení. K tomu slouží *environments*, které představují vybraná prostředí. Definice se nachází ve složce `/src/environments` a každý soubor reprezentuje jiné prostředí. Specifikovat můžeme údaje, zda je jedná o produkční sestavení, jaká je adresa API a jiné.

Čím více se aplikace rozrůstá, tím větší jsou výsledné soubory. Angular umožňuje nastavení omezení velikosti při sestavení. Během kompilace se tedy může vyvolat upozornění, že aplikace překročila limity velikosti. Možné je sledovat velikost celé aplikace, jednotlivých modulů nebo samostatných skriptů. Další konfigurace se upravují v rámci souboru `angular.json` [31].

Důležité je také myslet na kompatibilitu mezi prohlížeči. Je možné ji upravovat podle potřeb pomocí souboru `browserslist`. Některé starší prohlížeče nepodporují moderní funkce JavaScriptu a proto je nutné přidat *polyfills* pro jejich dodefinování [32]. Přidáním však může o dost narůst celková velikost. Tento problém řeší Angular CLI pomocí strategie *differential loading*, která při kompilaci vytvoří dvě samostatné verze aplikace. První obsahuje moderní ES2015 syntaxi a díky tomu je výsledný balík menší. Druhá verze je o poznání větší a obsahuje kód staré ES5 syntaxe se všemi potřebnými *polyfills*. Aplikace při běhu pozná, kterou verzi použít a podporuje tak všechny prohlížeče bez nutnosti načítat nepotřebný kód.

## 4.2 Automatická integrace

Sestavení a nasazení produkční aplikace je možné automatizovat. Tomuto procesu se říká průběžná integrace a nasazení (CI/CD). Hlavní výhodou je, že celý proces probíhá automaticky mimo počítač vývojáře, takže není závislý na prostředí a minimalizuje se tak riziko chyb zejména při práci v týmu. Umožňuje sestavení, testování a nasazení aplikace s minimální nebo bez interakce uživatele kdykoliv, kdy nastane změna v kódu. Automatického nasazení mé aplikace se skládá ze tří kroků:

1. Proces začíná odesláním změn do *master* větve v rámci repozitáře.
2. Jakmile to server detekuje, spustí se sestavení produkční verze.
3. Po úspěšném sestavení se soubory odešlou do služby Firebase Hosting.

Celý proces je monitorován, takže pokud dojde v nějaké části k chybě, správce je ihned informován a proces dále nepokračuje. Automatické testy nebyly součástí zadání práce, ale do budoucna se počítá s pokrytím celé aplikace a rozšířením automatické integrace.

Služba Firebase nabízí spoustu nástrojů pro vývoj moderních *real-time* aplikací. Pro hostování aplikace jsem použil Firebase Hosting [33]. Slouží k hostování statických souborů, což pro účely mé aplikace naprosto stačí. Hlavní výhodou je obrovská síť CDN serverů po celém světě, která zaručí rychlé doručení souborů nezávisle na tom, kde se uživatel nachází. Nabízí SSL certifikáty pro vlastní domény zdarma, takže připojení je vždy zabezpečené přes HTTPS. Ovládání služby je uživatelsky přívětivé a nasadit aplikaci lze jedním příkazem, což je naprosto ideální pro automatickou integraci.

Pro správu zdrojových kódu jsme se rozhodli použít systém GitLab [34]. Umožňuje vytvářet a spravovat repozitáře a nabízí kvalitní vlastní nástroje pro CI/CD. Vznikl jako otevřená alternativa služby GitHub a je možné ho hostovat na vlastním serveru nebo využít servery GitLab. Podporuje spoustu konfigurací, nástrojů pro práci v týmu a bezpečné způsoby, jak monitorovat vývoj celého projektu.

K nastavení a konfiguraci GitLab CI/CD slouží soubor `.gitlab-ci.yml`, který je nutné mít v kořenové složce projektu. V tomto souboru se definuje sekvence skriptů a příkazů, které se mají spouštět, dále ukládání závislostí, výběr prostředí či zda se mají příkazy vykonávat automaticky nebo manuálně. Výsledný soubor zpracovává nástroj GitLab Runner, což je program umožňující vykonávat jednotlivé příkazy.

Na začátku konfigurace jsem zvolil Node.js jako obraz systému, na kterém se budou příkazy provádět. Následuje nastavení *kešování* složky závislostí `node_modules`. Po instalaci závislostí se spustí první fáze, při které se aplikace sestaví a uloží vytvořené soubory. Ty se použijí ihned ve druhé fázi, kdy dojde k nasazení aplikace na Firebase Hosting. Ve skriptu je použitý API klíč, který je uložený a zašifrovaný přímo v administraci GitLab repozitáře.

## 4.3 Dodatečné optimalizace

Po spuštění produkční verze bylo potřeba provést několik optimalizací. Prvním krokem bylo přidání nástrojů pro vyhodnocování stavu aplikace. Bez efektivního sběru dat nejsme schopni analyzovat výkon a navrhovat potřebné úpravy. Rozhodl jsem se nasadit tyto 4 nástroje:

**Google Analytics** Osvědčený nástroj pro analytiku návštěvnosti. Umožňuje pozorovat uživatele, jejich chování v aplikaci, demografická data a používané technologie. Umí také nastavit konverzní akce pro sledování výkonu marketingových kampaní a cílů webu [35].

**Smartlook** Tento nástroj nabízí spoustu unikátních funkcí. Hlavním důvodem použití je nahrávání chování uživatelů, díky kterému je možné objevit problémy s používáním aplikace a nedokončování konverzních akcí. Dále umí tvorbu *heatmap* a *trychtýřů* průchodu aplikací [36].

**Sentry** Stará se o odchytávání chyb během používání aplikace. To umožňuje vývojáři objevit problémy, se kterými se na svých zařízeních ještě nesetkal. Každá chyba obsahuje informace o prohlížeči, operačním systému a seznam akcí, které uživatel provedl než chyba nastala [37].

**Firebase Performance Monitoring** Součástí služby Firebase je i nástroj pro sledování výkonu aplikace. Vyhodnocuje dobu načtení a vykreslení stránky, společně s dobu odpovědi sítě. Dají se tím odhalit pomalé části webu, které je potřeba optimalizovat [38].

Z nasbíraných dat od uživatelů jsme zjistili, že aplikace je nejčastěji navštěvovaná na mobilních zařízeních. V tuto chvíli nemáme prostředky na vývoj mobilních aplikací, proto jsme se rozhodli vyladit responzivní design a spustit progresivní webovou aplikaci (PWA).

### Progresivní webová aplikace

PWA představuje webovou aplikaci využívající moderní API prohlížeče k dosažení možností, spolehlivosti a instalace podobné jako nativní aplikace [39]. Nabízí fungování offline, *push* notifikace, zobrazení na celé obrazovce a spoustu dalších možností. Na rozdíl od nativních aplikací nezabírají tolik místa, jsou přístupné přes URL adresu a o aktualizace se stará sama aplikace.

Nástrojem Angular CLI jsem přidal PWA funkcionality pomocí jediného příkazu `ng add @angular/pwa` [40]. Ten vytvoří *service worker*, který zajišťuje *kešování* statických souborů, dále soubor *manifest* obsahující základní konfiguraci instalace, přidá odkaz na tento soubor do `index.html`, *meta* značku `theme-color` pro určení barvy a vytvoří výchozí ikony do složky `src/assets`. Konfigurace jsem upravil podle potřeb a přidal vlastní ikony. V budoucnu bychom rádi umožnili používat aplikaci offline.

# Závěr

Bakalářská práce popisuje tvorbou webové aplikace pro start-up Poopeek, sloužící k propojování kapel a pořadatelů. Hlavním cílem bylo navrhnout a implementovat klientskou část, která se napojí na existující serverové řešení. Podařilo se realizovat moderní webovou aplikaci s responzivním designem, která splňuje funkční požadavky zadavatele. Projekt jsem postavil na bázi frameworku Angular, což mi umožnilo vytvořit jednostránkovou progresivní aplikaci, která je schopná nahradit nativní mobilní klienty.

V první části práce jsem udělal analýzu existujícího serverového řešení a jeho rozhraní API, na které se aplikace napojila. Zmínil jsem hlavní entity a vztahy mezi nimi, včetně způsobu autentizace. Dále jsem navrhl uživatelské rozhraní, popsal všechny části aplikace a určil jejich funkce. Výsledkem se stala grafická podoba šablony, kterou jsem následně implementoval. Vytvořil jsem strukturu prototypu včetně všech komponent a služeb, které se starají o prezenční i datovou vrstvu. Po implementaci jsem se zaměřil na tvorbu produkční verze a její nasazení. V poslední části je realizována automatická integrace pro usnadnění budoucího vývoje a nasazení klíčových nástrojů pro sběr dat a vyhodnocování stavu aplikace.

Výsledkem je komplexní systém pro hudební booking. Uživatel si vytvoří profil kapely nebo pořadatele. K dispozici má seznam nabídek, na které může reagovat a také může vytvořit svoji vlastní. Po reakci začne vyjednávání mezi kapelou a pořadatelem, které probíhá formou chatu pomocí definovaných podmínek vystoupení. Po souhlasu domluvených podmínek vznikne dohoda a oběma stranám jsou doručeny dokumenty pro uskutečnění akce.

Aplikaci plánuji dále rozvíjet a podílet se na aktivitách okolo ní. Máme v plánu přidat novou entitu připojující další odvětví hudebního průmyslu. Dále bychom rádi implementovali modul pro generování univerzálního *stageplanu* pódia a kapelám umožnili tvořit sekvenci návazných koncertů. V neposlední řadě chceme integrovat interaktivní mapy pro lepší uživatelský zážitek.

# Literatura


1. MEAN Stack. *IBM Cloud Learn Hub* [online]. 2019 [cit. 2020-04-01]. Dostupné z: https://www.ibm.com/cloud/learn/mean-stack-explained.
2. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network based Software Architectures* [online]. Irvine, 2000 [cit. 2020-04-02]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Disertační práce. University of California.
3. Understanding And Using REST APIs. *Smashing Magazine* [online]. 2018 [cit. 2020-04-03]. Dostupné z: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/.
4. About JavaScript. *MDN* [online] [cit. 2020-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
5. About Node.js. *Node.js* [online] [cit. 2020-04-06]. Dostupné z: https://nodejs.org/en/about/.
6. What is V8? *V8 JavaScript engine* [online] [cit. 2020-04-05]. Dostupné z: https://v8.dev/.
7. Express/Node introduction. *MDN* [online] [cit. 2020-04-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.
8. What is NoSQL? *AWS* [online] [cit. 2020-04-08]. Dostupné z: https://aws.amazon.com/nosql/.
9. What Is MongoDB? *MongoDB* [online] [cit. 2020-04-09]. Dostupné z: https://www.mongodb.com/what-is-mongodb.
10. Introduction to JSON Web Tokens. *JWT.IO* [online] [cit. 2020-04-10]. Dostupné z: https://jwt.io/introduction/.

11. Web design a uživatelská rozhraní. *Vysoká škola ekonomická v Praze: Katedra multimédií* [online] [cit. 2020-04-11]. Dostupné z: https://kme.vse.cz/o-katedre/granty-a-dalsi-aktivity/oppa/web-design-a-uzivatelska-rozhrani/.

12. NIELSEN, Jakob. Enhancing the Explanatory Power of Usability Heuristics. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Boston, Massachusetts, USA: Association for Computing Machinery, 1994, s. 152–158. CHI '94. ISBN 0897916506. Dostupné z DOI: 10.1145/191666.191729.

13. MAČEK, Pavel. *Návrh uživatelského rozhraní aplikace pro správu úkolů*. Brno, 2010. Dostupné také z: https://is.muni.cz/th/uzjlq/thesis.pdf. Diplomová práce. Masarykova univerzita, Fakulta informatiky.

14. ŘEZÁČ, Jan. *Web ostrý jako břitva: návrh fungujícího webu pro webdesignery a zadavatele projektů*. Vydání druhé. Brno: House of Řezáč, 2016. ISBN 978-80-270-0644-1.

15. FIGMA, Inc. *Figma: the collaborative interface design tool* [online] [cit. 2020-04-12]. Dostupné z: https://www.figma.com/.

16. What Are Vector Graphics? *Vectr* [online] [cit. 2020-04-13]. Dostupné z: https://vectr.com/tutorials/what-are-vector-graphics/.

17. Implementing your theme. *Material Design* [online] [cit. 2020-04-14]. Dostupné z: https://material.io/design/material-theming/implementing-your-theme.html.

18. PAGLINAWAN, Andrew. Quicksand. *Google Fonts* [online] [cit. 2020-04-15]. Dostupné z: https://fonts.google.com/specimen/Quicksand.

19. Icons. *Font Awesome* [online] [cit. 2020-04-16]. Dostupné z: https://fontawesome.com/icons?d=gallery%5C&s=regular,solid.

20. HTML & CSS. *W3C* [online] [cit. 2020-04-17]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss.

21. Sass Basics. *Sass* [online] [cit. 2020-04-18]. Dostupné z: https://sass-lang.com/guide.

22. TypeScript in 5 minutes. *TypeScript* [online] [cit. 2020-04-19]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

23. Introduction. *RxJS* [online] [cit. 2020-04-20]. Dostupné z: https://rxjs.dev/guide/overview.

24. What is a Single Page Application? *HUSPI* [online] [cit. 2020-04-21]. Dostupné z: https://huspi.com/blog-open/definitive-guide-to-spa-why-do-we-need-single-page-applications.

25. Features and benefits. *Angular* [online] [cit. 2020-04-22]. Dostupné z: https://angular.io/features.

26. Introduction to Angular concepts. *Angular* [online] [cit. 2020-04-23]. Dostupné z: https://angular.io/guide/architecture.

27. Angular vs React vs Vue: Which is the Best Choice for 2019? *Hacker Noon* [online] [cit. 2020-04-24]. Dostupné z: https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847.

28. Angular coding style guide. *Angular* [online] [cit. 2020-04-25]. Dostupné z: https://angular.io/guide/styleguide.

29. What are AOT & JIT Compiler in Angular? *DEV* [online] [cit. 2020-04-26]. Dostupné z: https://dev.to/imbilal1/what-are-aot-jit-compiler-in-angular-2-488h.

30. An introduction to Angular 9 Ivy Compiler. *DEV* [online] [cit. 2020-04-27]. Dostupné z: https://dev.to/eugeniolentini/an-introduction-to-angular-9-ivy-compiler-177n.

31. Building and serving Angular apps. *Angular* [online] [cit. 2020-04-28]. Dostupné z: https://angular.io/guide/build.

32. Browser support. *Angular* [online] [cit. 2020-04-29]. Dostupné z: https://angular.io/guide/browser-support.

33. Firebase Hosting. *Firebase* [online] [cit. 2020-05-01]. Dostupné z: https://firebase.google.com/products/hosting.

34. The DevOps Lifecycle with GitLab. *GitLab* [online] [cit. 2020-04-30]. Dostupné z: https://about.gitlab.com/stages-devops-lifecycle/.

35. Analytics Tools & Solutions for Your Business. *Google Analytics* [online] [cit. 2020-05-02]. Dostupné z: https://marketingplatform.google.com/about/analytics/.

36. Website analytics free. *Smartlook* [online] [cit. 2020-05-03]. Dostupné z: https://www.smartlook.com/website-analytics/.

37. Angular Error Tracking. *Sentry* [online] [cit. 2020-05-04]. Dostupné z: https://sentry.io/for/angular/.

38. Firebase Performance Monitoring. *Firebase* [online] [cit. 2020-05-05]. Dostupné z: https://firebase.google.com/docs/perf-mon.

39. What are Progressive Web Apps? *Web.dev* [online] [cit. 2020-05-06]. Dostupné z: https://web.dev/what-are-pwas/.

40. Create a Progressive Web App with the Angular CLI. *Web.dev* [online] [cit. 2020-05-07]. Dostupné z: https://web.dev/creating-pwa-with-angular-cli/.

Příloha A

# Seznam použitých zkratek

**AOT** Ahead-of-Time

**API** Application Programming Interface

**CDN** Content Delivery Network

**CI/CD** Continuous Integration and Continuous Deployment

**CLI** Command Line Interface

**CSS** Cascading Style Sheets

**DOM** Document Object Model

**GUI** Graphical User Interface

**HTML** Hypertext Markup Language

**HTTP(S)** Hypertext Transfer Protocol (Secure)

**JIT** Just-in-Time

**JSON** JavaScript Object Notation

**JWT** JSON Web Token

**MEAN** MongoDB, Express, Angular a Node.js

**PWA** Progressive Web Application

**REST** Representational State Transfer

**RxJS** Reactive Extensions for JavaScript

**Sass** Syntactically Awesome Style Sheets

**SEO** Search engine optimization

**SPA** Single-page application

**SSL** Secure Sockets Layer

**UI** User Interface

**URI** Uniform Resource Identifier

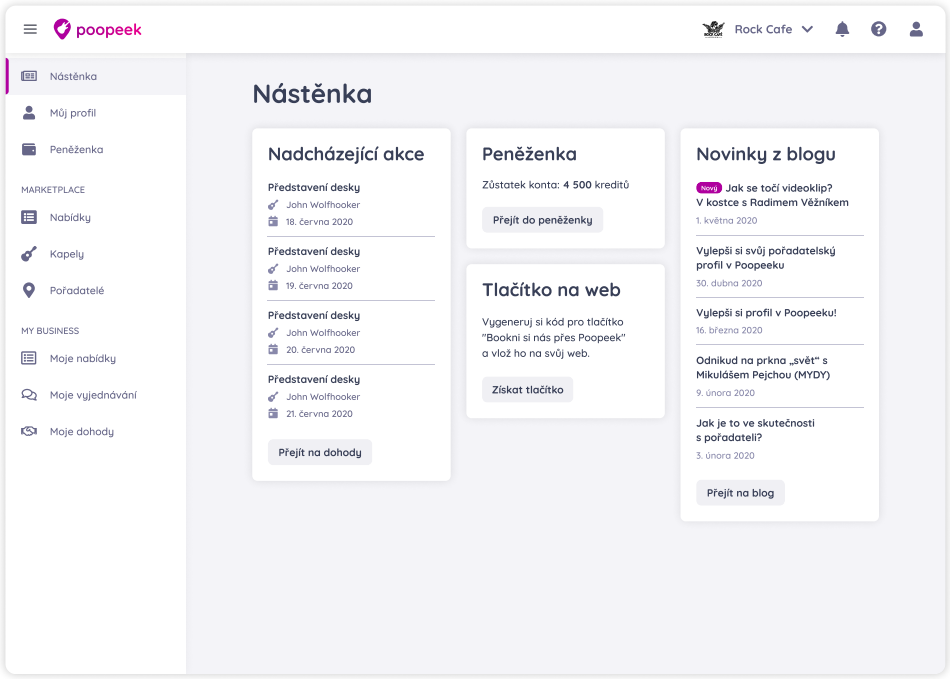
**URL** Uniform Resource Locator

**UX** User Experience

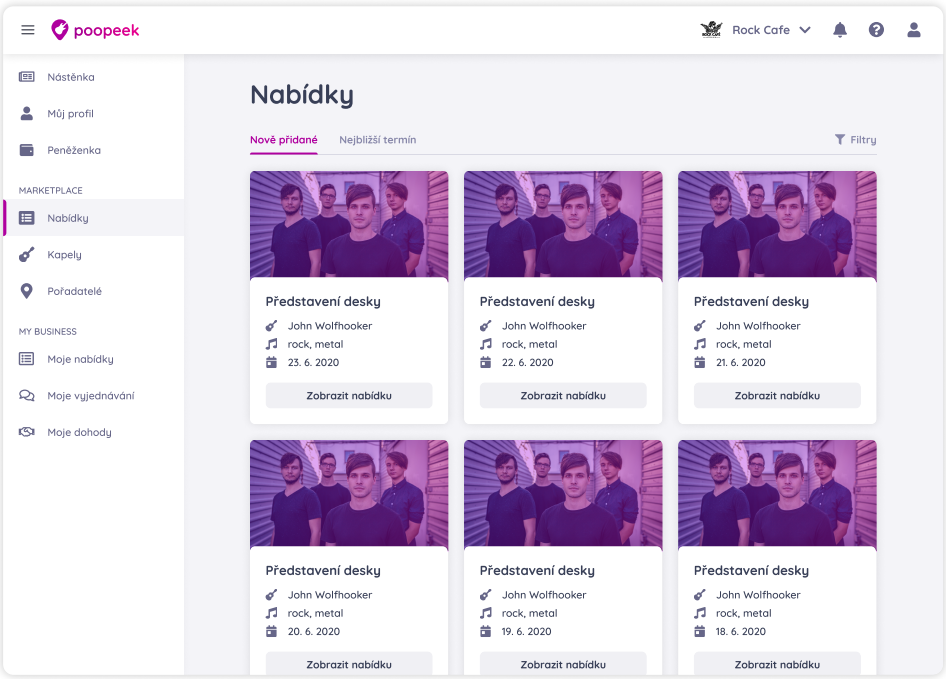
Příloha B

# Přílohy práce

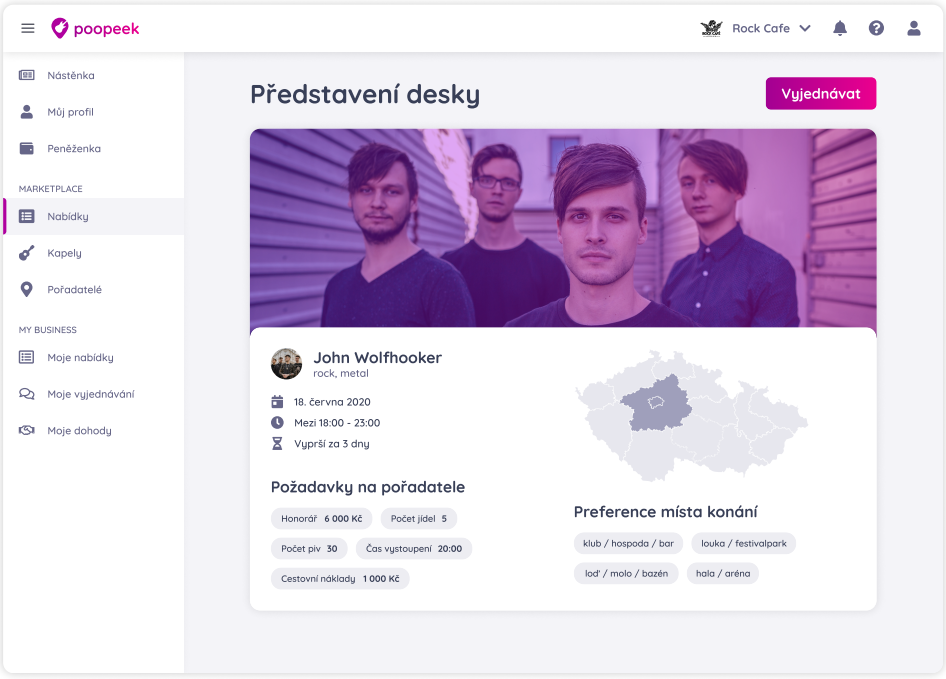
Přílohy zde nejsou kompletní, zbytek je k zobrazení na přiloženém médiu.



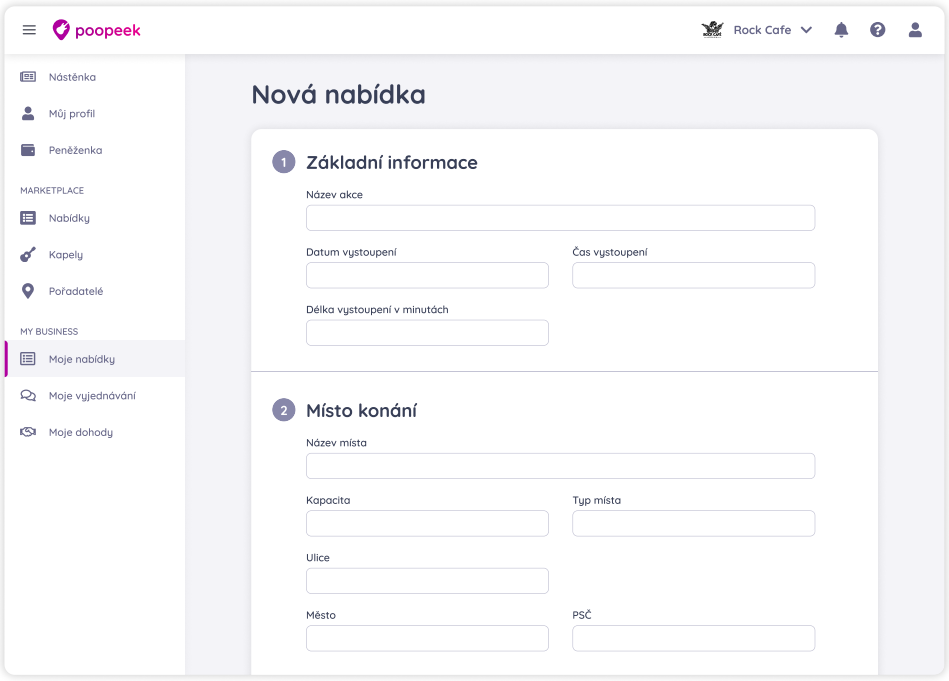

Obrázek B.1: Prototyp nástěnky



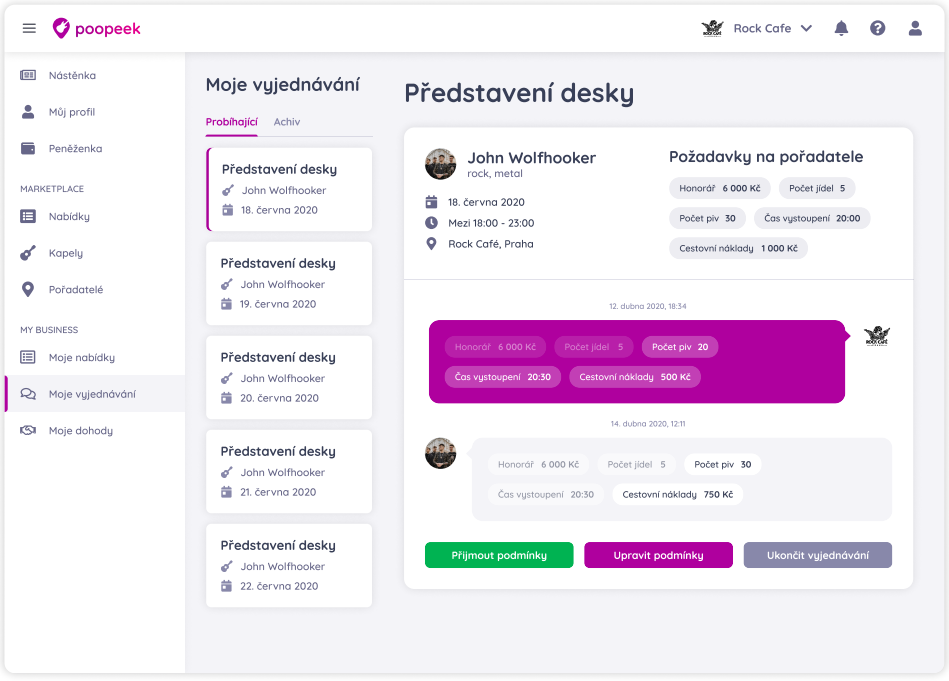

Obrázek B.2: Prototyp seznamu nabídek



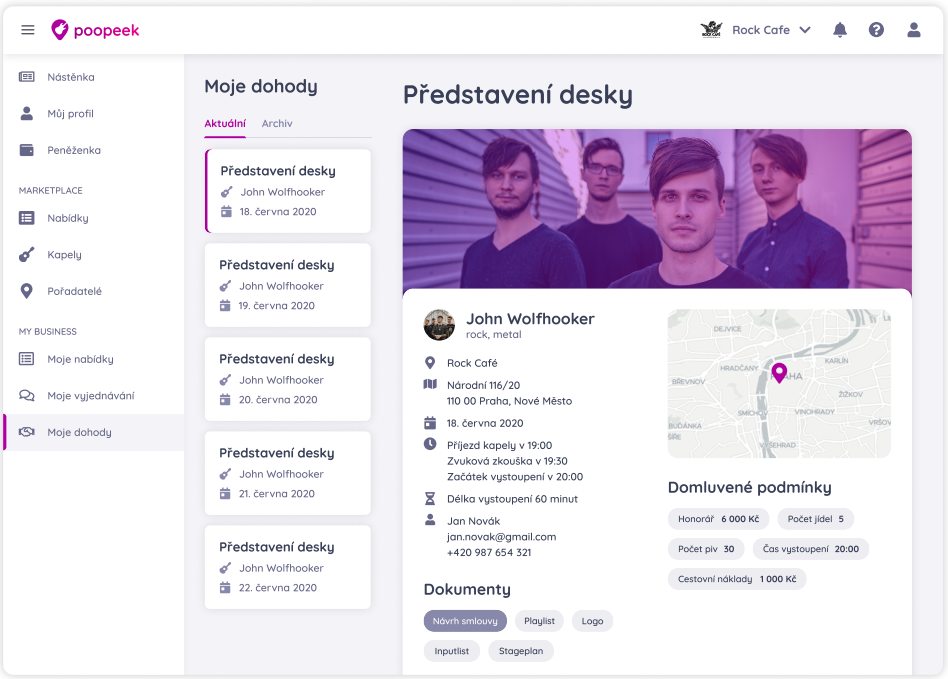

Obrázek B.3: Prototyp detailu nabídky



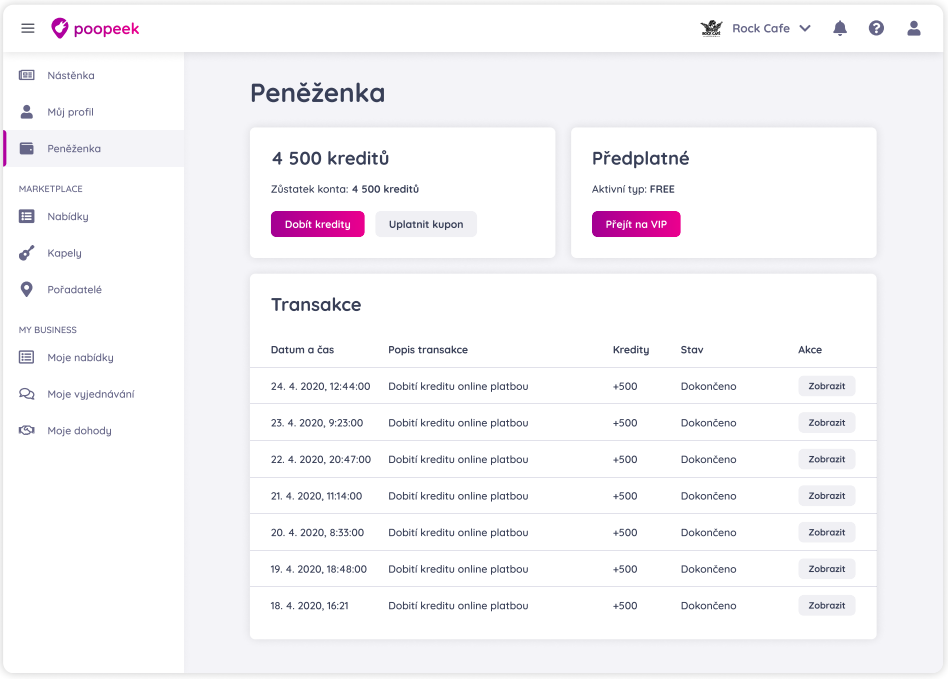

Obrázek B.4: Prototyp formuláře



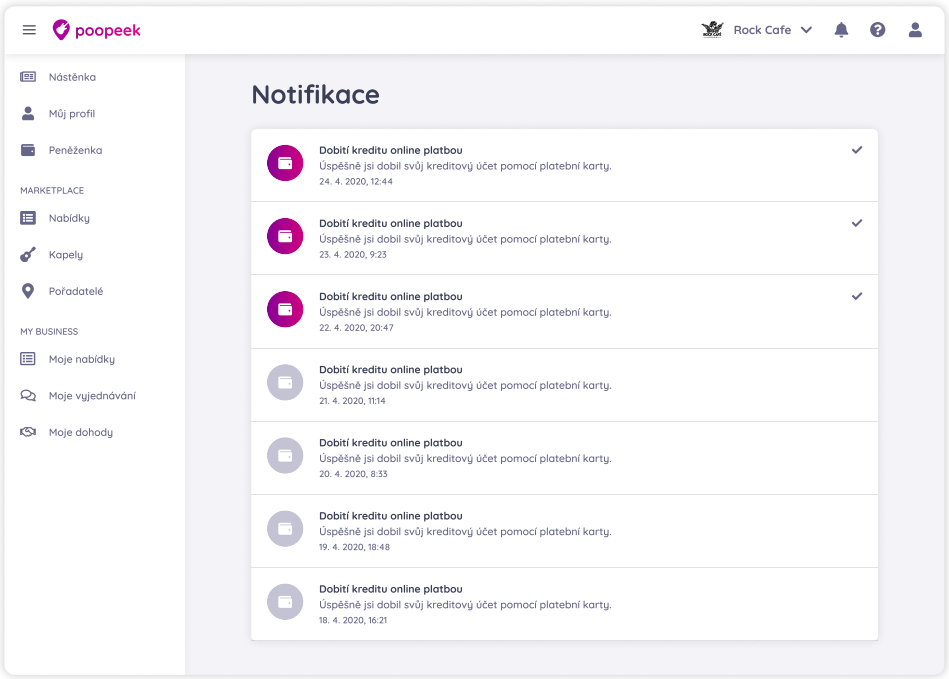

Obrázek B.5: Prototyp detailu vyjednávání



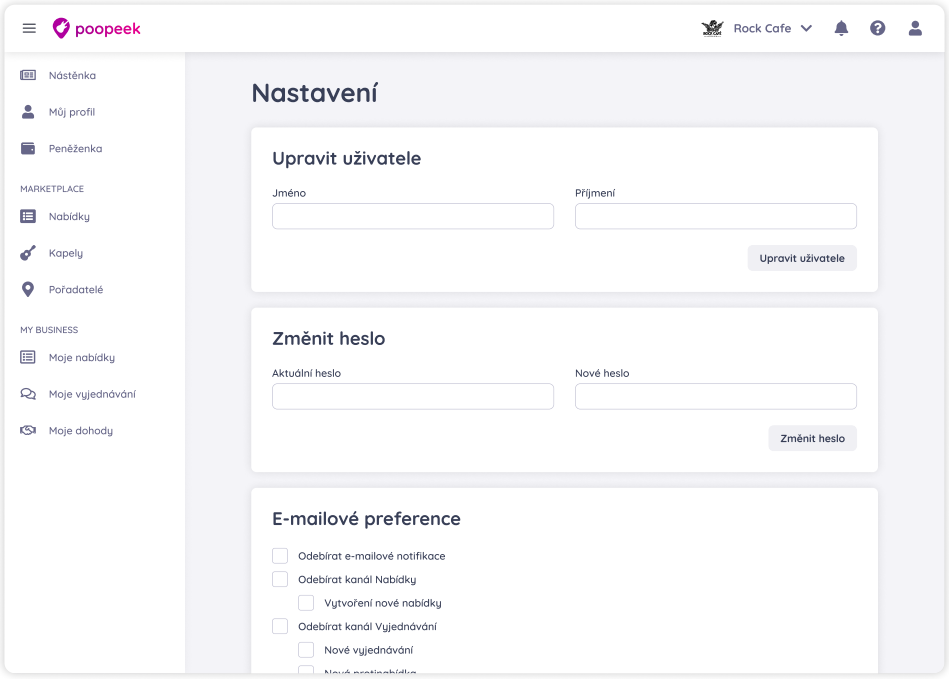

Obrázek B.6: Prototyp detailu dohody




Obrázek B.7: Prototyp peněženky




Obrázek B.8: Prototyp notifikací




Obrázek B.9: Prototyp nastavení

Příloha C

# Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu přiloženého média
- `src/`
  - `app/` ..... zdrojové kódy implementace
  - `wireframes/` ..... vypracované wireframy aplikace
  - `prototypes/` ..... vypracované prototypy aplikace
- `text/`
  - `src/` ..... zdrojové soubory LaTeX
  - `BP_Fryc_Dominik_2020.pdf` ..... text práce ve formátu PDF